sOaked in java

http://www.cs.indiana.edu/classes/a201-dger

A201/A597/I210 Lecture Notes
Summer 2003

Adrian German
To Codri and Celina.

To all the Little Lispers ever.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epilogue</td>
<td>7</td>
</tr>
<tr>
<td>Preface</td>
<td>9</td>
</tr>
<tr>
<td>Prelude</td>
<td>11</td>
</tr>
<tr>
<td>On Solving Problems</td>
<td>19</td>
</tr>
<tr>
<td>Problems and Pain</td>
<td>21</td>
</tr>
<tr>
<td>Getting Started</td>
<td>25</td>
</tr>
<tr>
<td>Algorithms</td>
<td>39</td>
</tr>
<tr>
<td>Simple Programs</td>
<td>45</td>
</tr>
<tr>
<td>Types and I/O</td>
<td>55</td>
</tr>
<tr>
<td>First Things First</td>
<td>67</td>
</tr>
<tr>
<td>A Simple Converter</td>
<td>73</td>
</tr>
<tr>
<td>Warmups (I)</td>
<td>77</td>
</tr>
<tr>
<td>Warmups Solutions (I)</td>
<td>81</td>
</tr>
<tr>
<td>Problem Set (I)</td>
<td>85</td>
</tr>
<tr>
<td>Solutions (I)</td>
<td>97</td>
</tr>
<tr>
<td>Homework One</td>
<td>111</td>
</tr>
<tr>
<td>Syntax</td>
<td>113</td>
</tr>
<tr>
<td>Getting Things Done</td>
<td>123</td>
</tr>
<tr>
<td>Predicates</td>
<td>135</td>
</tr>
<tr>
<td>An Example from Consumer Reports</td>
<td>149</td>
</tr>
<tr>
<td>Classes</td>
<td>151</td>
</tr>
<tr>
<td>Diagrams</td>
<td>159</td>
</tr>
<tr>
<td>Modeling</td>
<td>173</td>
</tr>
<tr>
<td>Warmups (II)</td>
<td>183</td>
</tr>
<tr>
<td>Warmups Solutions (II)</td>
<td>185</td>
</tr>
<tr>
<td>Problems (II)</td>
<td>189</td>
</tr>
<tr>
<td>Review (Part I)</td>
<td>197</td>
</tr>
<tr>
<td>Solutions (II)</td>
<td>200</td>
</tr>
<tr>
<td>Homework Two</td>
<td>223</td>
</tr>
<tr>
<td>Review (Part II)</td>
<td>227</td>
</tr>
<tr>
<td>Methods</td>
<td>235</td>
</tr>
<tr>
<td>Decisions</td>
<td>245</td>
</tr>
<tr>
<td>Warmups (III)</td>
<td>257</td>
</tr>
<tr>
<td>Warmups Solutions (III)</td>
<td>261</td>
</tr>
<tr>
<td>Problems (III)</td>
<td>271</td>
</tr>
<tr>
<td>Solutions (III)</td>
<td>285</td>
</tr>
<tr>
<td>The First Dilbert Lecture</td>
<td>297</td>
</tr>
<tr>
<td>Loops</td>
<td>319</td>
</tr>
<tr>
<td>Patterns and Loops</td>
<td>327</td>
</tr>
<tr>
<td>Homework Three</td>
<td>343</td>
</tr>
<tr>
<td>More Loops</td>
<td>347</td>
</tr>
<tr>
<td>Computer Games</td>
<td>365</td>
</tr>
<tr>
<td>Designing Fractions</td>
<td>375</td>
</tr>
<tr>
<td>Tokens and Tokenizers</td>
<td>381</td>
</tr>
<tr>
<td>Milestones</td>
<td>387</td>
</tr>
<tr>
<td>Warmups (IV)</td>
<td>403</td>
</tr>
<tr>
<td>Warmups Solutions (IV)</td>
<td>407</td>
</tr>
<tr>
<td>Practice Problems</td>
<td>415</td>
</tr>
<tr>
<td>More Practice Problems</td>
<td>433</td>
</tr>
<tr>
<td>More Solutions</td>
<td>435</td>
</tr>
<tr>
<td>Arrays (I)</td>
<td>463</td>
</tr>
<tr>
<td>Homework Four (Part I)</td>
<td>471</td>
</tr>
<tr>
<td>Homework Four (Part II)</td>
<td>473</td>
</tr>
<tr>
<td>Arrays (II)</td>
<td>481</td>
</tr>
<tr>
<td>Java Fandango</td>
<td>491</td>
</tr>
<tr>
<td>Procedures (Methods)</td>
<td>501</td>
</tr>
<tr>
<td>Warmups (V)</td>
<td>513</td>
</tr>
<tr>
<td>Warmups Solutions (V)</td>
<td>517</td>
</tr>
<tr>
<td>Warmups (VI)</td>
<td>521</td>
</tr>
<tr>
<td>Warmups Solutions (VI)</td>
<td>523</td>
</tr>
<tr>
<td>Help with Homework Four</td>
<td>527</td>
</tr>
<tr>
<td>Arrays and Methods</td>
<td>547</td>
</tr>
<tr>
<td>Arrays of Objects</td>
<td>551</td>
</tr>
<tr>
<td>The Bald Soprano</td>
<td>561</td>
</tr>
<tr>
<td>Utilities</td>
<td>575</td>
</tr>
<tr>
<td>Applets</td>
<td>591</td>
</tr>
<tr>
<td>Events</td>
<td>603</td>
</tr>
<tr>
<td>Umpires</td>
<td>611</td>
</tr>
<tr>
<td>Rubberbands</td>
<td>621</td>
</tr>
<tr>
<td>Devices</td>
<td>627</td>
</tr>
<tr>
<td>Double-Buffering</td>
<td>635</td>
</tr>
<tr>
<td>Homework Six</td>
<td>643</td>
</tr>
<tr>
<td>Help with The Last Assignment</td>
<td>647</td>
</tr>
<tr>
<td>The Second Dilbert Lecture</td>
<td>653</td>
</tr>
<tr>
<td>Threads</td>
<td>673</td>
</tr>
<tr>
<td>Commencement</td>
<td>687</td>
</tr>
</tbody>
</table>
Epilogue

I take pride in knowing that not many books start with an epilogue.
This was written at the end of Summer Session II on August 10, 2002.

I couldn’t come to the final (and hence asked Richard to proctor it) because I was getting a hair cut. Then Richard came with the exams and I started grading. That was on Friday, at about 4pm (I remember it very well).

The most shocking thing about getting a buzz cut (I didn’t go all the way) is that you suddenly realize your head is much smaller than you ever would have imagined. I’ve developed a sudden fear of ping-pong. Unfortunately, my ears, nose and glasses (I’ve started wearing glasses, I just felt I had to) stayed exactly the same size thus giving me a Mr. Potato Head look that I hope will one day become fashionable.

On the plus side, I find endless amusement rubbing my head with my hand. You would think I’d get bored doing that, but you would be mistaken. That’s why it’s taken me so long to post the grades; I’m typing with one hand. It did take me a long time (essentially, half of Friday and all of Saturday) but now grades have finally been posted and I am sure we all appreciate that.

I try to resist rubbing my head, but I don’t have the willpower. I tried to cut down from two hands to one. I discover myself afraid that this could be like a gateway habit, a stepping-stone to worse addictions. And I try to soothe my anxiety by rubbing my head, which only makes things worse.

I truly hope you have enjoyed this class. I tried to put the best I had in it, and make it accurate and informative, while keeping it entertaining to the extent that one could have hoped for. I enjoyed working with each and every one of you, and I wish you the best of luck in your future endeavors.

Of all the minute papers there was one that I will remember forever. It was written on Thursday, before the exam, and had no signature or name on it. In clear handwriting it went straight for the heart:

Dear Professor, today I love you more than tomorrow.

Have a great end of summer and my best of the best wishes to all of you.

Adrian German
Bloomington, IN
Preface

I like this story for its uniqueness. Seldom does an author state a trivial fact, so casually (in one sentence) in the beginning—and then spends the rest of the story *taking it back*. Every book has a preface. This is this book’s claim to one (or the other).

There was once a red-haired man who had no eyes and no ears. He also had no hair, so he was called red-haired only in a manner of speaking.

He wasn’t able to talk, because he didn’t have a mouth. He had no nose, either.

He didn’t even have any arms or legs. He also didn’t have a stomach, and he didn’t have a back, and he didn’t have a spine, and he also didn’t have any other insides. He didn’t have anything. So it’s hard to understand whom we’re talking about.

So we’d better not talk about him any more.

So—with this spirit in heart and mind, let the story begin.
# Prelude

<table>
<thead>
<tr>
<th>Computers can do lots of things. They can add millions of numbers in the twinkling of an eye. They can outwit chess grandmasters. They can guide weapons to their targets.</th>
<th>They can book you onto a plane between a guitar-strumming nun and a non-smoking physics professor. Some can even play the bongos. That’s quite a variety!</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>What does the inside of a computer look like?</strong></td>
<td>Crudely, it will be built out of a set of simple, basic elements. Once you get down to the guts of computers you find that, like people, they tend to be more or less alike. They can differ in their functions, and in the nature of their inputs and outputs—one can produce music, another a picture, while one can be set running from a keyboard, another by the torque from wheels of an automobile—but at heart they are very similar.</td>
</tr>
<tr>
<td>Viewed this way, the variety in computers is a bit like the variety in houses: a Beverly Hills condo might seem entirely different from a garage in Yonkers, but both are built from the same things—bricks, mortar, wood, sweat—only the condo has more of them, and arranged differently according to the needs of the owner. At heart they are very similar.</td>
<td>For today’s computers to perform a complex task, we need a precise and complete description of how to do that task in terms of a sequence of simple basic procedures. This instructing has to be exact and unambiguous. In life, of course, we never tell each other exactly what we want to say; we never need to, as context, body language, familiarity with the speaker, and so on, enables us to “fill the gaps” and resolve any ambiguities in what is said.</td>
</tr>
<tr>
<td>Computers, however, can’t yet “catch on” to what is being said, the way a person does. They need to be told in excruciating detail exactly what to do.</td>
<td>I can see the emphasis.</td>
</tr>
</tbody>
</table>
Perhaps one day we will have machines that can cope with approximate task descriptions, but in the meantime we have to be very prissy about how we tell computers to do things.

A computer program tells a computer, in minute detail, the sequence of steps that are needed to fulfill a task. The act of designing and implementing these programs is called computer programming. In this course, you will learn how to program a computer—that is, how to direct a computer to execute tasks.

Today’s computer programs are so sophisticated that it is hard to believe that they are all composed of extremely primitive operations.

Only because a program contains a huge number of such operations, and because the computer can execute them at great speed, does the computer user have the illusion of smooth interaction.

“He does it very stupidly, but he also does it very quickly and that’s the point of all this: the inside of a computer is as dumb as hell but it goes like mad!”

It can perform very many millions of simple operations a second and is just like a very fast (but) dumb file clerk.

To use a computer you do not need to do any programming. You can drive a car without being a mechanic and toast bread without being an electrician.

Many people who use computers every day in their careers never need to do any programming.

Of course, a professional computer scientist or software engineer does a great deal of programming. Since you are taking this first course in computer science, it may well be your career goal to become such a professional.

Or maybe not—but let’s pretend!

To write a computer game with motion and sound effects or a word processor that supports fancy fonts and pictures is a complex task that requires a team of many highly skilled programmers.

Your first programming efforts will be more mundane. But we’ll still be able to program a video game by the end of the semester.

The concepts and skills you learn in this course form an important foundation, and you should not be disappointed if your first programs do not rival the sophisticated software that is familiar to you.

Actually, you will find that there is an immense thrill even in simple programming tasks. It is an amazing experience to see the computer precisely and quickly carry out a task that would take you hours of drudgery, to make small changes in a program that lead to immediate improvements, and to see the computer become an extension of your mental powers.

To understand the programming process, you need to have a rudimentary understanding of the building blocks that make up a computer (perhaps).

At the heart of the computer lies the central processing unit (CPU). All data must travel through the CPU whenever it is moved from one location to another. (There are a few technical exceptions to this rule; some devices can interact directly with memory).

The computer stores data and programs in memory. There are two kinds of memory: primary storage and secondary storage.
**Primary storage** is fast but expensive; it is made from memory chips: so-called *random-access memory* (RAM) and *read-only memory* (ROM).

**Secondary storage**, usually a hard disk, provides less expensive storage that persists without electricity.

Primary storage is fast but expensive. Secondary storage is slower but a lot cheaper (affordable).

You will often use another kind of magnetic storage device: a so-called *floppy disk* or *diskette*.

Or maybe not.

Originally floppy disks had a fairly low capacity, but recently high-capacity floppies such as the Zip disk and the Superdisk have become popular. Because a floppy disk is not an integral part of the computer system, it is called an *external storage device*.

Very good, let’s keep going.

Floppies are OK but audio/video information takes up much more space than a floppy disk can provide.

That kind of information is typically distributed on a CD-ROM or DVD (digital versatile disk).

*I like CD-R/W’s – don’t you?*

To store large amounts of user data, *data tapes* are commonly used.

Data tapes are *very* inexpensive and can hold lots of information, but they are *extremely* slow.

Some computers are self-contained units, whereas others are interconnected through *networks*.

Through the network cabling, the computer can read programs from central storage locations or send data to other computers. For the user of a networked computer it may not even be obvious which data reside on the computer itself and which are transmitted through the network.

To interact with a human user, a computer requires other peripheral devices.

The computer transmits information to the user through a *display screen*, *loudspeakers*, and *printers*. The user can enter information and directions to the computer by using a *keyboard* or a pointing device such as a *mouse*.

The central processing unit, RAM memory, and the electronics controlling the hard disk and other devices are interconnected through a set of electrical lines called a *bus*.

Programming in Java we won’t have anything to do *directly* with these, but when a program is started, it is brought into main memory, from which the CPU can read it. The CPU reads the program an instruction at a time. As directed by these instructions, the CPU reads data, modifies them, and writes them back to RAM memory or to hard disk.

Some program instructions will cause the CPU to place dots on the display screen or printer or to vibrate the speaker.

As these actions happen many times over and at a great speed, the human user will perceive images and sound.
Some program instructions read user input from the keyboard or mouse. The program analyzes the nature of these inputs and then executes the next appropriate instructions.

On the most basic level, computer instructions are extremely primitive. CPUs from different vendors, such as the Intel Pentium or the Sun SPARC, have different sets of machine instructions. To enable Java programs to run on multiple CPUs without modification, most Java compilers generate a set of machine instructions for a so-called “Java virtual machine”, an idealized CPU that is then simulated by a program run on the actual CPU.

The difference between actual and virtual machine instructions is not important to us—all you need to know is that machine instructions are very simple and can be executed very quickly. Java is a high-level programming language. In Java the programmer expresses the idea behind the task that needs to be performed in a language that resembles both natural language (somewhat) and (to a greater extent) mathematics. Then, a special computer program, called a compiler translates the higher-level description into machine instructions (called bytecode) for the Java virtual machine.

Compilers are sophisticated programs. Thanks to them programming languages are independent of a specific computer architecture. Still, they are human creations, and as such they follow certain conventions. To ease the translation process, those conventions are much stricter than they are for human languages.

When you talk to another person, and you scramble or omit a word or two, your conversation partner will usually still understand what you have to say. Compilers are less forgiving.

Just as there are many human languages, there are many programming languages. This provides a useful source of analogy. Let me ask you this: which is the best language for describing something? Say: a four-wheeled gas-driven vehicle.

Of course, most languages, at least in the West, have a simple word for this: we have “automobile”, the English say “car”, the French “voiture”, and so on. However, there will be some languages which have not evolved a word for “automobile” (perhaps they don’t care for such a word,) and speakers of such tongues would have to invent some, possibly long and complex, description for what they see, in terms of their basic linguistic elements.

Yet none of these descriptions is inherently “better” than any of the others: they all do their job, and will only differ in efficiency. We needn’t introduce democracy just at the level of words. We can go down to the level of alphabets.

What, for example, is the best alphabet for English? That is, why stick with our usual 26 letters? Everything we can do with these, we can do with three symbols – the Morse code, dot, dash, and space;
...or two – a Baconian cipher, with A through Z represented by five-digit binary numbers. So we see that we can choose our basic set of elements with a lot of freedom, and all this choice really affects is the efficiency of our language, and hence the sizes of our books; there is no "best" language or alphabet – each is logically universal, and each can model any other.

Same with programming languages, and Java is no exception. Like C (another popular programming language), the Java language arose from the ashes of a failing project. In the case of Java, the situation was an anticipated market that failed to materialize.

Imagine that you’ve worked on a highly ambitious state of the art electronics project for 2 years. Against all odds, you have built a working prototype of the hardware driven by a custom-developed programming language. It’s a hand-held device that can control consumer electronics like TV-top boxes for interactive cable. A TV-top box is the name given to the extra-electronic gizmo you’d need to decode the signal when they bring 600 channels of cable to your house.

So what you have is an extremely intelligent remote control, possibly providing two-way communication from your house to the cable company. The only problem is that at that time (1992) people were beginning to realize that there was no market for interactive cable service.

That was the situation in which the Sun R&D team found themselves. Despite wooing potential customers like Mitsubishi, France Telecom, and Time-Warner, the orders either went elsewhere, or did not materialize at all. By mid-1993 interactive TV was a big expensive bust and everyone knew it.

Mid-1993 was also when the first Mosaic browser came out, although few people had yet paid any attention to it. But does anybody even remember Mosaic now?

I suppose you do...

The first web browser ever that had a graphical user interface (GUI). A truly revolutionary concept. That, it was. Funding for the box project was about to be cut, and the team (led by James Gosling and Patrick Naughton) disbanded to other projects, when Bill Joy and Eric Schmidt conceived the idea of dropping the hardware, and adapting the software to work smoothly with the Internet.

Sun executive Phil Samper was persuaded to fund further development for one year to the tune of $5 million. Everything was to revolve around the WWW. Whatever the software project evolved into, it had to feature the Web as its focal point.

By now it was 1994. Gosling and a small number of colleagues set to work on this new challenge. They worked at a furious pace and in great secrecy throughout the year.

By Christmas they had a working translator, the key libraries, and a web browser as a proof of concept. In January 1995 the language was named “Java”.

<table>
<thead>
<tr>
<th>...or two – a Baconian cipher, with A through Z represented by five-digit binary numbers. So we see that we can choose our basic set of elements with a lot of freedom, and all this choice really affects is the efficiency of our language, and hence the sizes of our books; there is no “best” language or alphabet – each is logically universal, and each can model any other.</th>
<th>Like C (another popular programming language), the Java language arose from the ashes of a failing project. In the case of Java, the situation was an anticipated market that failed to materialize.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imagine that you’ve worked on a highly ambitious state of the art electronics project for 2 years. Against all odds, you have built a working prototype of the hardware driven by a custom-developed programming language. It’s a hand-held device that can control consumer electronics like TV-top boxes for interactive cable. A TV-top box is the name given to the extra-electronic gizmo you’d need to decode the signal when they bring 600 channels of cable to your house.</td>
<td>That was the situation in which the Sun R&amp;D team found themselves. Despite wooing potential customers like Mitsubishi, France Telecom, and Time-Warner, the orders either went elsewhere, or did not materialize at all. By mid-1993 interactive TV was a big expensive bust and everyone knew it.</td>
</tr>
<tr>
<td>Mid-1993 was also when the first Mosaic browser came out, although few people had yet paid any attention to it. But does anybody even remember Mosaic now?</td>
<td>I suppose you do...</td>
</tr>
<tr>
<td>The first web browser ever that had a graphical user interface (GUI). A truly revolutionary concept.</td>
<td>That, it was. Funding for the box project was about to be cut, and the team (led by James Gosling and Patrick Naughton) disbanded to other projects, when Bill Joy and Eric Schmidt conceived the idea of dropping the hardware, and adapting the software to work smoothly with the Internet.</td>
</tr>
<tr>
<td>Sun executive Phil Samper was persuaded to fund further development for one year to the tune of $5 million. Everything was to revolve around the <a href="http://WWW">WWW</a>. Whatever the software project evolved into, it had to feature the Web as its focal point.</td>
<td>By now it was 1994. Gosling and a small number of colleagues set to work on this new challenge. They worked at a furious pace and in great secrecy throughout the year.</td>
</tr>
<tr>
<td>By Christmas they had a working translator, the key libraries, and a web browser as a proof of concept. In January 1995 the language was named “Java”.</td>
<td></td>
</tr>
</tbody>
</table>
The HotJava browser, which was shown to an enthusiastic crowd at the SunWorld exhibition in 1995, had one unique property: It could download programs, called *applets*, from the web and run them.

Applets let web developers provide a variety of animation and interaction and can greatly extend the capabilities of the web page. In 1996 both Netscape and Microsoft supported Java in their browsers. Since then Java has grown at a phenomenal rate.

Programmers have embraced the language because it is simpler than its closest rival, C++. In addition to the programming language itself, Java has a rich library that makes it possible to write portable programs that can bypass proprietary operating systems.

At this time Java has already established itself as one of the most important languages for general-purpose programming as well as for computer science instruction.

Was Java designed for beginners? No.

Java is an industrial language. And because Java was not specifically designed for students, no thought was given to make it really simple to write basic programs. A certain amount of technical machinery is necessary in Java to write even the simplest programs.

To understand what this technical machinery does, you need to know something about programming.

This is not a problem for a professional programmer with prior experience in another programming language, but not having a linear learning path is a drawback for the student.

As you learn how to program in Java, there will be times when you will be asked to be satisfied with a preliminary explanation and wait for complete details in a later chapter.

Furthermore, you cannot hope to learn all of Java in one semester. The Java language itself is relatively simple, but Java contains a vast set of library packages that are necessary to write useful programs. There are packages for graphics, user interface design, cryptography, networking, sound, database storage, and many other purposes.

Even expert Java programmers do not know the contents of all the package—they just use those that are needed for particular projects.

Taking this class, you should expect to learn a good deal about the Java language and about the most important packages.

Keep in mind though that the purpose of this course is not to make you memorize Java minutiae, but to teach you how to think about programming.

All right, let’s see a program written in Java.

How about this one?

```java
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello, and welcome to A201!");
    }
}
```
<table>
<thead>
<tr>
<th>What can it do?</th>
<th>It displays a simple greeting.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I’d like to see that.</td>
<td>You need to create a program file, compile it and then run it.</td>
</tr>
</tbody>
</table>

Here’s the session in Unix:

```plaintext
frilled.cs.indiana.edu/pico Hello.java
frilled.cs.indiana.edu/javac Hello.java
frilled.cs.indiana.edu/java Hello
Hello, and welcome to A201!
frilled.cs.indiana.edu%
```

What’s pico?

It’s a Unix editor. That’s how it all gets started. You enter the program statements into a text editor. The editor stores the text and gives it a name such as Hello.java

...which you then compile.

Yes, with javac. When you compile your program, the compiler translates the Java source code (that is the text, or statements that you wrote) into so-called bytecode which consists of virtual machine instructions and some other pieces of information on how to load the program into memory prior to execution.

The bytecode for a program is stored in a separate file with extension .class for example the bytecode for the program we wrote will be stored in Hello.class and you should look for this file on your system after compilation.

What’s frilled?

Just the prompt on the Unix machine we were on at the time. On your computer it might be C:\> or some such thing. What’s next?

The Java bytecode file contains the translation of your program in Java virtual machine terms.

A Java interpreter loads the bytecode of the program you wrote, starts your program, and loads the necessary library bytecode files as they are required.

That’s...java!

Precisely. Your programming activity centers around these steps: you start in the editor, writing the source file. Compile the program—look at the error messages. Go back to the editor and fix the syntax errors. When the compiler succeeds—run the executable file.

If you find an error, you try to debug your program to find the cause of the error. Once you find the cause of the error, you go back to the editor and try to fix it. You compile and run again to see whether the error has gone away.

If not, you go back to the editor. You bet.
This is called the edit - compile - debug loop, and you will spend a substantial amount of time in this loop in the months and years to come.

Let me draw a picture of that.

That’s called a flowchart.

I know …

I thought so.
On Solving Problems

"Throughout the book, I will suggest some problems for you to play with. You might feel tempted to skip them. If they’re too hard, fine. Some of them are pretty difficult! But you might skip them thinking that, well, they’ve probably already been done by somebody else; so what’s the point? Well, of course they’ve been done! But so what? Do them for the fun of it. That’s how to learn the knack of doing things when you have to do them. Let me give you an example. Suppose I wanted to add up a series of numbers, $1 + 2 + 3 + 4 + 5 + 6 + 7 + \ldots$ up to, say, 62. No doubt you know how to do it; but when you play with this sort of problem as a kid, and you haven’t been shown the answer…it’s fun trying to figure out how to do it. Then, as you go into adulthood, you develop a certain confidence that you can discover things; but if they’ve already been discovered, that shouldn’t bother you at all. What one fool can do, so can another, and the fact that some other fool beat you to it shouldn’t disturb you: you should get a kick out of having discovered something. Most of the problems I give you in this book have been worked over many times, and many ingenious solutions have been devised for them. But if you keep proving stuff that others have done, getting confidence, increasing the complexities of your solutions—for the fun of it—then one day you’ll turn around and discover that nobody actually did that one! And that’s the way to become a computer scientist."

(from The Feynman Lectures on Computation)

Figure 1: Richard Feynman
<table>
<thead>
<tr>
<th>Let’s take a look at some examples.</th>
<th>Examples of what?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Of what programming is like, of course.</td>
<td>Fine, just to eliminate any misunderstandings.</td>
</tr>
<tr>
<td>Exactly. Here’s Problem No. 1</td>
<td>Go for it.</td>
</tr>
<tr>
<td>Write down detailed rules for multiplication, then find someone willing to help you. Hand the rules you wrote to that person. Ask her (or him) to perform one or two multiplications following the rules you wrote.</td>
<td>Ask the person to follow the rules <em>exactly</em>, without any innovations or implicit assumptions.</td>
</tr>
<tr>
<td>Can you give me an example of a multiplication?</td>
<td>Sure: $123 \times 4578$ is an example.</td>
</tr>
<tr>
<td>To make it even clearer, here’s a second problem.</td>
<td>Problem No. 2</td>
</tr>
<tr>
<td>Exactly.</td>
<td>Let’s jump right into it.</td>
</tr>
<tr>
<td>Very well: write down detailed rules for taking the square root of a positive integer.</td>
<td>For example: $\sqrt{123}$</td>
</tr>
<tr>
<td>Or $\sqrt{\ldots}$</td>
<td>Or $\sqrt{184529985018352430759371}$ for that matter.</td>
</tr>
<tr>
<td>It should be a little bit harder to come up with the rules in this case, or recall them</td>
<td>But that’s what makes the problem an even better example than the previous one.</td>
</tr>
<tr>
<td>Yes, it’s easier to test your rules on people now.</td>
<td>Yes, because they’re not likely to take square roots on a daily basis (the way one uses multiplication).</td>
</tr>
</tbody>
</table>
Problems and Pain

Learning.
Learning Java.
Programming is a contact sport.

Life is difficult. And so is Java. And so is learning, in general.

This is a great truth, perhaps one of the greatest truths. The first of the “Four Noble Truths” which Buddha taught was “Life is suffering”.

It is a great truth because once we truly see this truth, we transcend it.

Once we truly know that life is difficult—once we truly understand and accept it—then life is no longer difficult. Because once it is accepted, the fact that life is difficult no longer matters.

I see...
Most do not fully see this truth that life is difficult. Instead they moan more or less incessantly, noisily or subtly, about the enormity of their problems, their burdens, and their difficulties as if life were generally easy, as if life should be easy.

But life is not easy. Life is a series of problems. Discipline is the basic set of tools we require to solve life's problems. Without discipline we can solve nothing. With only some discipline we can solve only some problems. With total discipline we can solve all problems.

What makes life difficult is that the process of confronting and solving problems is a painful one. Problems, depending upon their nature, evoke in us frustration or grief or sadness or loneliness or guilt or regret or anger or fear or anxiety or anguish or despair. These are uncomfortable feelings, often very uncomfortable, often as painful as any kind of physical pain, sometimes equaling the very worst kind of physical pain.

Indeed, it is because of the pain that events or conflicts engender in us all that we can call them problems. And since life poses an endless series of problems, life is always difficult and is full of pain as well as joy. And it is in this whole process of meeting and solving problems that life has its meaning.

Problems are the cutting edge that distinguishes between success and failure. Problems call forth our courage and our wisdom. Indeed, they create our courage and our wisdom. It is only because of problems that we grow mentally and spiritually. When we desire to encourage the growth of the human spirit, we challenge and encourage the human capacity to solve problems,

...just as in school we deliberately set problems for our children to solve. It is through the pain of confronting and resolving problems that we learn. As Benjamin Franklin said, "Those things that hurt, instruct." It is for this reason that wise people learn not to dread but actually to welcome problems and actually to welcome the pain of problems.

Most of us, however, are not so wise.

Fearing the pain involved, almost all of us, to a greater or lesser degree, attempt to avoid problems.
We procrastinate, hoping that they will go away. We ignore them, forget them, pretend they do not exist. We attempt to skirt around problems rather than meet them head on. We attempt to get out of them rather than suffer through them. But let us teach ourselves and our children the necessity for suffering and the value thereof, the need to face problems directly and to experience the pain involved.

Discipline is the basic set of tools that we require to solve life's problems, and these tools are basically techniques of suffering: Means by which we experience the pain of problems in such a way as to work them through and solve them successfully, learning and growing in the process. When we teach ourselves and our children discipline, we are teaching them and ourselves how to suffer and also how to grow. We are teaching them and ourselves how to schedule the pain and pleasure of life in such a way as to enhance the pleasure by meeting and experiencing the pain first and getting it over with.

This is called delayed gratification and it's one of the tools, techniques of suffering, means of experiencing the pain of problems constructively, that we call discipline. The tools of discipline are four: delaying of gratification (which we just discussed,) acceptance of responsibility, dedication to truth, and balancing. Perhaps the first three are more or less obvious to you, so let me mention here what balancing is.

The exercise of discipline is not only a demanding but also a complex task, requiring both flexibility and judgment. Courageous people must continually push themselves to be completely honest, yet must also possess the capacity to withhold the whole truth when appropriate. To be free people, we must assume total responsibility for ourselves, but in doing so we must possess the capacity to reject responsibility that is not truly ours. To be organized and efficient, to live wisely, we must daily delay gratification and keep an eye on the future; yet to live joyously we must also possess the capacity, when it is not destructive, to live in the present and act spontaneously. In other words, discipline itself must be disciplined. This kind of meta-discipline is what we call balancing. It is the type of discipline required to discipline discipline. This is not hard; it is very hard. But it is the kind of discipline that gives us flexibility.

Since you are taking A201, A597, or I210, or simply reading this book, it may be that you want, or need to learn Java—or programming in general. Since this is a first experience for you I deeply hope it will come easy, but be prepared if it does not.
<table>
<thead>
<tr>
<th>In fact it really won’t be easy at all, unless you approach it with patience, perseverance and determination.</th>
<th>If you treat it superficially it will be downright difficult from the beginning, and will continue to be that way until the very end, no matter how much we’ll try to make it easy or understandable or obvious or intuitive or immediate or easy to grasp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>But you can help, and I am sure you will.</td>
<td>Because there is some risk involved, I wish you luck.</td>
</tr>
<tr>
<td>And because the act of entering programming as a beginner and a non-major is basically an act of courage,</td>
<td>...you have my admiration</td>
</tr>
<tr>
<td>The difficulty in learning programming has two clearly identifiable components.</td>
<td>( \frac{3}{4} ) of it is of a very genuine mathematical nature. The other half ( \left( \frac{1}{4} \right) ) is psychological. You’ll need to bridge the two.</td>
</tr>
<tr>
<td>That was from Yogi Berra, wasn’t it?</td>
<td>And don’t forget: whatever happens, you’re still simply the best! Should someone fail to see this evidence, with patience prove it beyond any conceivable doubt.</td>
</tr>
</tbody>
</table>

“You can observe a lot by watching.”  
Yogi Berra (who also said: “If the people don’t want to come out to the ballpark, nobody’s going to stop them”)

“Failures, repeated failures, are finger posts on the road to achievement. One fails forward toward success.”  
Charles F. Kettering (1876-1958, American Engineer, Inventor)
Getting Started

In Lecture Notes One you have seen a videotape. It was from “The Wrong Trousers”.

That’s “Wallace and Gromit”. Yes, an Aardman Production.

We’ll discuss some of it below. I am sure we will get to that.

Goals for the first lab:

- Become familiar with your computer:
  1. Login
  2. Explore and understand files and folders
  3. Locate Java compiler
  4. Edit, compile and run your first Java program

```java
class One {
    public static void main(String[] args) {
        System.out.println("This is a message.");
    }
}
```
  5. Running Java from the DOS command prompt
- Recognize syntax and logic errors
- Work with classes and objects

Here are the steps for this lab:

1. Check that you appear in Postem (in A201-4289 even if you are taking E210 or A597.)
   Here’s the URL: https://www.indiana.edu/ best/cgi-bin/postem/students.pl
   If something is wrong please e-mail dgerman@indiana.edu immediately.
2. Start Notepad (it should be under Utilities) and create a file.

3. The text of the file should be:

   ```java
   public class One {
       public static void main(String[] args) {
           System.out.println("Hello, world!");
       }
   }
   ```

4. Save this file as One.java on the desktop.

5. Open an MS-DOS command window.

6. Find the file that you have just created (use dir One.java or something similar).
   Note that you need to make sure you're accessing the folder that corresponds to the Desktop.

7. Compile the file (use javac One.java)

8. Find the resulting class (dir One.class)

9. Run the program: (use java One)

10. Remember that these are the steps of program development:

      (a) Create (edit) a program in a file (extension .java)
     (b) Compile that .java file with javac (a .class gets created)
     (c) Run the program (the .class file) with java

11. Now change the contents of One.java as follows:

    ```java
    public class One {
        public static void main(String[] args) {
            System.out.println("Hi, there!");
        }
    }
    ```
    Note that the part that changed is enclosed in a box.

12. Save the file, compile and run One once again. What's the outcome?

13. Now change the file as follows:

    ```java
    public class One { public static void main(String[] args) { System.out.println("Hi, there!"); } } 
    ```
    Notice we only change the amount of existing blank space, nothing else.

14. Save and compile the file, then run the program. What’s the outcome?

15. Now change the file to:
public class One {
    public static void main(String[] args) {
        System.out.println("Hi, there!");
    }
}

16. Save and compile the file, run the program. What’s the outcome?

17. Can you summarize the meaning of this?

18. Now change the program as follows:

    public class One {
        public static void main(String[] args) {
            System.out.println("Hello, world!");
            System.out.println("Hello, again!");
        }
    }

19. Compile and run the program. What’s the outcome?

20. Make a new change, and try this program:

    public class One {
        public static void main(String[] args) {
            System.out.println("Hello, ");
            System.out.println("Hello, again!");
        }
    }

21. Then try this one:

    public class One {
        public static void main(String[] args) {
            System.out.println("are ");
            System.out.println("How ");
            System.out.println("you?");
        }
    }
22. Can you write a program that produces this output?

```
4
4
4
4
4
4
4
```

It's a big "4" made out of little "4"-s.

23. Try the following program and explain its output:

```java
public class One {
    public static void main(String[] args) {
        System.out.println(1 + 2 + 3);
    }
}
```

24. Try the following program and explain its output:

```java
public class One {
    public static void main(String[] args) {
        System.out.println(1 + 2 * 3);
    }
}
```

25. Is the next program different?

```java
public class One {
    public static void main(String[] args) {
        System.out.println((1 + 2) * 3);
    }
}
```

26. Try this program:

```java
public class One {
    public static void main(String[] args) {
        System.out.println("Hello, D'Artagnan!");
    }
}
```

We can print single quotes quite easily.

27. Now put "D'Artagnan" in double quotes:
public class One {
    public static void main(String[] args) {
        System.out.println("Hello, "D’Artagnan"!");
    }
}

Compile and run this program. What happens?

28. Try this:

    public class One {
        public static void main(String[] args) {
            System.out.println("Hello, \"D’Artagnan\"!");
        }
    }

    Explain the outcome.

29. Now try this program:

    public class One {
        public static void main(String[] args) {
            System.out.println("Hello, \nworld!");
        }
    }

    Explain what it does.

30. Recall the program that outputs this:

    4
    4
    4
    4
    4
    4
    4

    Can you write that program with just one System.out.println?

31. Can you write a program that outputs a "4" made out of double quotes (")?

    "
    "
    "
    "

    *********
    "
    "

32. Try this program and explain the outcome:
public class One {
    public static void main(String[] args) {
        System.out.print("Hello. ");
        System.out.print("world!");
    }
}

33. What's the output of this program?

public class One {
    public static void main(String[] args) {
        System.out.print("A");
        System.out.print("B\n");
        System.out.print("C");
        System.out.print("D");
        System.out.print("E\n");
    }
}

34. What's the output of this program?

public class One {
    public static void main(String[] args) {
        System.out.print("A");
        System.out.print("B\n");
        System.out.print("C");
        System.out.print("D");
        System.out.print("E\n");
    }
}

Why?

35. One final challenge. Can you write a program that produces this:

```
.8. 8 888888888o ,o8888888.
.888. 8 8888 '88. 8888 '88.
:8888. 8 8888 '88 ,8 8888 '8.
.8.'88888. 8 8888 ,88 88 8888
.8. '88888. 8 8888 ,88' 88 8888
.8' 8. '88888. 8 888888888 88 8888
.8' '8. '88888. 8 8888 '88. 88 8888
.888888888. '88888. 8 8888 ,88' 8888 ,88'
.8' '8. '88888. 8 888888888p '8888888p'
```

You don't need to write this program, just think whether you can write it or not.

36. Also compile and run this program, as shown in lecture:
import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.SwingConstants;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;

public class SliderTest
{
    public static void main(String[] args)
    {
        SliderFrame frame = new SliderFrame();
        frame.setTitle("SliderTest");
        frame.show();
    }
}

class SliderFrame extends JFrame
{
    public SliderFrame()
    {
        final int DEFAULT_FRAME_WIDTH = 300;
        final int DEFAULT_FRAME_HEIGHT = 300;
        setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);
        addWindowListener(new WindowCloser());
        colorPanel = new JPanel();
        ColorListener listener = new ColorListener();

        redSlider = new JSlider(0, 100, 100);
        redSlider.addChangeListener(listener);

        greenSlider = new JSlider(0, 100, 70);
        greenSlider.addChangeListener(listener);

        blueSlider = new JSlider(0, 100, 100);
        blueSlider.addChangeListener(listener);

        JPanel southPanel = new JPanel();
        southPanel.setLayout(new GridLayout(3, 2));
        southPanel.add(new JLabel("Red", SwingConstants.RIGHT));
        southPanel.add(redSlider);
        southPanel.add(new JLabel("Green", SwingConstants.RIGHT));
        southPanel.add(greenSlider);
southPanel.add(new JLabel("Blue", SwingConstants.RIGHT));
southPanel.add(blueSlider);

Container contentPane = getContentPane();
contentPane.add(colorPanel, "Center");
contentPane.add(southPanel, "South");

setSampleColor();
}

public void setSampleColor()
{
  float red = 0.01F * redSlider.getValue();
  float green = 0.01F * greenSlider.getValue();
  float blue = 0.01F * blueSlider.getValue();

  colorPanel.setBackground(new Color(red, green, blue));
colorPanel.repaint();
}

private JPanel colorPanel;
private JSlider redSlider;
private JSlider greenSlider;
private JSlider blueSlider;

private class ColorListener implements ChangeListener
{
  public void stateChanged(ChangeEvent event)
  {
    setSampleColor();
  }
}

private class WindowCloser extends WindowAdapter
{
  public void windowClosing(WindowEvent event)
  {
    System.exit(0);
  }
}

You don’t need to understand this program, but you …and running it. And when you see it running you should have no problem creating, compiling, should feel happy about it!

The goal for this semester’s work is …to understand thoroughly a program such as this.

A second large program will be discussed, ...you can work with it here.

http://www.cs.indiana.edu/classes/a348/t540/lectures/iceblox/iceblox.html

What comes next is... Your A201/A597/1210 LAB ASSIGNMENT ONE

There are two parts to this first lab assignment:

First, write a program that prints a staircase: Show it to your AI next lab period.

++--
|   |
++++++
|   |
++++++
|   |
++++++
|   |
++++++
|   |
++++++

Second, write a program as described below. In your program:

1. Create a square room composed of 100 tiles (10 x 10, that is).
2. Create a Penguin, and add it to the room in the 8th line and 3rd column.
3. That means tile (7, 2) given our numbering convention.
4. So, please take some time to review the numbering convention now.
5. Now ask the penguin to turn around and move to tile (2, 2).
6. Then ask the penguin to turn right, then move to tile (2, 7).
7. Then make the penguin turn right, and have it move to tile (7, 7).
8. Now ask the penguin to move to tile (2, 7).
9. And finally have it return to (7, 2), passing through (2, 2).

Show this program to your instructor in lab, as well. What you will need is described below.

Essentially, you will need two classes.
First, you need Penguin.java

http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/labOne.html

I agree it looks relatively long but once you find it

...just place it in a file and keep it around.

Next, you need Rink.java, which represents the theater where all this happens.

Get it from the same address:

http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/labOne.html

Once you have these classes you need to create a third one (call it LabOne.java) and make its main method perform what the lab assignment is asking for (which is described above).

Here are two examples, to make sure we have enough to start from:

EXAMPLE ONE

What happy() Really Means (0ne.java)

class One {
    public static void main(String[] args) {
        Rink ballroom = new Rink(10, 10); // number of columns and rows

        /* Note the Rink created is called 'ballroom'. We'll have to use this name to refer to it thereafter.*/

        Penguin p = new Penguin(); // create a Penguin, call it 'p' (what's in a name?)

        ballroom.add(p, 1, 7); // add the Penguin to our Rink, in column 1 and line 7 (and remember our numbering scheme)

        p.pause(); p.turnLeft(); // control the Penguin
        /* Remember 'The Wrong Trousers' (the video)? */

        p.moveForward();
        p.moveForward();
        p.moveForward();
        p.happy();
        p.moveForward();
        p.moveForward();
        p.moveForward();
p.pause();
p.turnLeft();
p.pause();     // commands are issued in sequence
p.moveForward(); p.moveForward(); p.moveForward(); p.happy();
p.moveForward(); p.moveForward(); p.moveForward(); p.pause();
p.turnLeft(); p.pause();
p.moveForward(); p.moveForward(); p.moveForward(); p.happy();
p.moveForward(); p.moveForward(); p.moveForward(); p.happy();
p.turnLeft(); p.pause();
p.moveForward(); p.moveForward(); p.moveForward(); p.pause();
p.turnLeft(); p.pause();
p.moveForward(); p.moveForward(); p.happy();
p.moveForward(); p.moveForward(); p.moveForward(); p.happy();

/* can you still say you know where the Penguin is right now?  
Remember that not only the computer reads your programs!  
Write your programs as if they were essays.  
Make your code crystal clear.  
*/

p.moveForward();
p.moveForward();
p.turnLeft();
p.happy();

This really clarifies what the Penguin does when you're asking it to show that it's happy.

Nevermind. I now have EXAMPLE TWO
Let's see it...

Well, turn the page...  OK
This is called: Staircase Improvisation

Also known as Dance.java

class Dance {
    public static void main(String[] args) {

        Rink ballroom = new Rink(6, 6);

        Penguin p = new Penguin();

        ballroom.add(p, 1, 4);

        p.pause();

        p.turnLeft();

        p.turnLeft();  p.moveForward(); // go left
        p.turnRight(); p.moveForward(); // go right
        p.turnLeft();  p.moveForward(); // go left
        p.turnRight(); p.moveForward(); // go right
        p.turnLeft();  p.moveForward(); // go left
        p.turnRight(); p.moveForward(); // go right
        // now stop, rotate once, stay some more
        p.pause();  p.turnRight(); p.pause();
        // come south three tiles
        p.moveForward(); p.moveForward(); p.moveForward();
        // stop and catch your breath
        p.pause();
        // pirouette
        p.turnLeft(); p.turnLeft(); p.turnLeft(); p.turnLeft();
        // stop, for applause
        p.pause();
        // another pirouette, followed by immediate movement west
        p.turnRight(); p.turnRight(); p.turnRight(); p.turnRight();
        p.turnRight(); p.moveForward(); p.moveForward(); p.moveForward();
        // stop
        p.pause();
        // turn left, then stop
        p.turnLeft();

        p.pause();
        // one final pirouette, after which just thank the audience
        p.turnLeft(); p.turnLeft(); p.turnLeft(); p.turnLeft();

        p.happy();
        // Don't worry(), be happy().
    }
}

A smaller program. Smaller, but still instructive.
As a reminder, please note:

A Penguin:

1. can be created (and it will be facing south by default)
2. can be added to a Rink at a certain location (you have to ask the rink for that, though)
3. can be asked to turn left, or right (regardless of the direction it’s facing)
4. can be asked to move one cell forward (regardless of the direction it’s facing)

A Rink:

1. can be created (with the size (columns, rows) of your choice)
2. can hold one penguin at a time, added with add
3. adding a penguin to a rink is done by specifying
   (a) the column (vertical slices), and the
   (b) line (lines are horizontal rows in the rink)
   in that order, in the method add.

Note though that when the Rink shows, it labels the cells by first printing the line, then the column, for each of the tile.
The reason this numbering is also important is because it is the numbering used in 2 dimensional arrays in Java (of the kind we will encounter a bit later).

Note that x and y still keep the meaning that they originally had:

1. x is the number of columns to the left, and
2. y is the number of lines above (to the top)

When we create the Rink, and when we add a Penguin to it, we mention x first, and y next, almost as we do in analytical geometry.

However when we refer to the table of cells that the Rink is, we can also denote the cells in the array by printing (y, x), that is, by specifying the line first, and the column next. The point being that both notations are well-established, and we need to be aware of them both.

You should now write the program with no problems. That’s what I was going to say...
And here’s a picture of Wallace and Gromit.
The Aardman Penguin is pictured in disguise at the beginning of the assignment.

This is from “The Wrong Trousers,”
...to remind you of it, and for motivation.

And here's also a picture from Dance.java

Just to know what to expect.
Looks good. Good luck with your lab!
Algorithms

Before we look at the mechanics of implementing computations let us consider the planning process that precedes the implementation. If you can't give instructions for someone to solve a problem by hand, there is no way the computer can magically solve the problem.

The computer can only do what you do by hand. It just does it faster, and it doesn't get bored or exhausted. I'd like to see an example.

OK, let's consider the following investment problem:

You put $10,000 into a bank account that earns 5% interest per year. Interest is compounded annually. How many years does it take for the account balance to be double the original?

One could solve this problem by hand.

Sure, let's do that. After the first year you earn $500 (5% of $10,000). The interest gets added to your bank account and your balance becomes $10,500.00. Next year, the interest is $525 (5% of $10,500).

You can keep going that way and build a table:

<table>
<thead>
<tr>
<th>Year</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$10,000.00 = $10,000.00</td>
</tr>
<tr>
<td>1</td>
<td>$10,000.00 + 0.05 \times $10,000 = $10,500.00</td>
</tr>
<tr>
<td>2</td>
<td>$10,500.00 + 0.05 \times $10,500.00 = $11,025.00</td>
</tr>
<tr>
<td>3</td>
<td>$11,025.00 + 0.05 \times $11,025.00 = $11,576.25</td>
</tr>
<tr>
<td>4</td>
<td>$11,576.25 + 0.05 \times $11,576.25 = $12,155.06</td>
</tr>
</tbody>
</table>

Very good!
You keep going until the balance goes over $20,000.00 and when it does...

...which it does (doesn’t it?!)..."...

...you look into the “Year” column and you have the answer. Of course, carrying out this computation is intensely boring. Yes, but the fact that a computation is boring or tedious is irrelevant to a computer. Computers are very good at carrying out repetitive calculations quickly and flawlessly.

What is important to the computer is the existence of a systematic approach for finding the solution. The answer can be found just by following a series of steps that involves no guesswork.

Here’s such a series of steps:

**Step 1** Start with the table

<table>
<thead>
<tr>
<th>Year</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$10,000.00</td>
</tr>
</tbody>
</table>

**Step 2** Repeat steps 2a, 2b, 2c while the balance is less than $20,000.00

**Step 2a** Add a new row to the table.

**Step 2b** In column 1 of the new row, labeled “Year”, put one more than the preceding year value.

**Step 2c** In column 2 of the new row, labeled “Balance”, place the value of the preceding balance value, multiplied by 1.05 = (1 + 5%)”

**Step 3** Read the last number in the year column and report it as the number of years required to double the investment.

Of course, these steps are not yet in a language that a computer can understand, but you will learn soon how to formulate them in Java.

Is this collection of steps an **algorithm**?

Yes, because the description is unambiguous...

There are precise instructions what to do in every step and where to go next. There is no room for guesswork or creativity.

...executable, and terminating.

Because each step can be carried out in practice, and the computation can be shown to come to an end: with every step, the balance goes up by at least $500.00, so eventually it must reach $20,000.00
Now we start looking at the mechanics of implementing computations in Java.

Let’s analyze our first program.

```java
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

You know that you should make a new program file and call it `Hello.java`, enter the program instructions, then compile and run the program.

That’s clear, the contents is more intriguing.

The words and symbols are important and atomic: they’re like words and symbols in an English sentence. *Bring me a glass of water!*

Yes, but it’s a lot stricter. Java is *case-sensitive*. You must enter upper- and lowercase letters exactly as they appear in the program listing.

On the other hand Java has *free-form layout*. Spaces and line breaks are not important, except to separate words.

However, good taste dictates that you lay out your programs in readable fashion, so you should follow the layout in the program listing.

Now that we’ve seen the program, it’s time to understand its makeup.

The **part** in **boxes** introduces a class, **...called Hello.**

<table>
<thead>
<tr>
<th>The <strong>part</strong> in <strong>boxes</strong> introduces a class,</th>
<th>...called Hello.</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class Hello</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>public static void main(String[] args) {</td>
<td></td>
</tr>
<tr>
<td>System.out.println(&quot;Hello, World!&quot;);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

In Java, classes are the central organizing mechanism for code. You can’t do *anything* in Java without defining at least a class.

That is why we introduce the Hello class **...as the holder of the main method.**

Java, like most programming languages, requires that all program statements must be placed inside **methods**.

A method is a collection of programming instructions that describe how to carry out a particular task.
The part in [boxes] further defines the main method.

```java
class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Every Java application must have a main method.

Most Java programs contain other methods besides main, but it will take us a while to learn how to write other methods. For the time being, simply put all instructions that you want to have executed inside the main method of a class.

I have them all boxed (There's only one, here).

```java
class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

The instructions or statements in the body of the main method (that is, the statements inside the curly braces `{ }`) are executed one by one. Note that each statement ends in a colon `;`.

Our method has a single statement:

```java
System.out.println("Hello, World!");
```

Yes, but it, too, has a structure. The statement is supposed to print a line of text. I presume the text is enclosed by double quotes (").

Yes, a sequence of characters in quotation marks is called a string. You must enclose the contents of the string inside quotation marks so that the compiler considers them plain text and does not try to interpret them as program instructions.

To print the text you call a method println as if you'd call Papa John's for a large pizza. But which Papa John's? You need to precisely locate it. Suppose you say: the one on 3rd Street. But which 3rd: most towns have a 3rd Street. So you need to add Bloomington, and then IN.
All of this is apparent in their telephone number:

<table>
<thead>
<tr>
<th>812</th>
<th>323</th>
<th>PAPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>out</td>
<td>println</td>
</tr>
</tbody>
</table>

From this analogy it looks like `System` contains `out` which contains `println`?

Yes, we call the `println` method that is part of the `out` object, that is part of the `System` class, and we pass it the string that we wanted printed.

So that’s what the parentheses are for...

Yes, in fact that’s how we tell that `out` is merely data, and not a method: method names are always followed by a pair of parentheses.

Why is `System` uppercased and `out` lowercased?

This is only a convention, that object variables (or names) start with a lowercase letter, while classes names should start with an upper case. Using this convention is strongly encouraged, as part of the style guide.

So let’s summarize: designers of the Java libraries defined a `System` class in which they’ve put useful objects and methods. One of these objects is called `out` and it lets you access the terminal window (also called the standard output). To use the `out` object in the `System` class you must refer to it as `System.out`; it has a method inside it, by the name of `println` which we can use, and so we do.

That’s correct.

Asking the computer to execute a method is also known as calling or invoking the method.

When we call a method we can pass it information in between parentheses. If we pass no information there will be an empty pair of parentheses.

In this case we pass one string.

Have you looked at the exercises yet? What for?

How would you print `Hello, "World"`! ? You need to escape the quotation marks inside the string with a backslash (\), like that:

```java
public class Hello
{
    public static void main(String[] args) {
        System.out.println("Hello, \"world\"!");
    }
}
```

It becomes harder to read, but it’s also more precise. The computer won’t mistake any of the two escaped double quotes as being the end of the string.
What other escape sequences are there? Let’s mention a couple...

Since the backslash character is used as an escape character, it needs to be escaped itself, if we need it in output. Another escape sequence used occasionally is \n which is the same as new line or line feed character.

Are there any other things that we could pass to println for printing?

Yes, for example arithmetic expressions. Such as:

\[ \begin{align*}
3 + 4 \\
(2.5 - 1) / 4 \\
(3 + 4) * (2 - 5)
\end{align*} \]

Yes. What’s the asterisk (*) for? It’s for multiplication, and / for division.

Very good. You could even "add" strings by listing them with a * between them. println will actually concatenate them (or string them together).

OK, maybe I won’t do that right away. What’s the difference between println and print?

The out object contains a second method called print. You can see the difference between the two methods if you consider the following program:

```java
public class Test {
    public static void main(String[] args)
    { System.out.println("Hello, ");
        System.out.println("World! ");
    }
}
```

What does it do?

The println method prints a string or a number and then starts a new line. The print method does the same printing, without starting a new line afterwards.

I see… putting all of the things we’ve learned together I could write the same program as follows:

```java
public class Test {
    public static void main(String[] args)
    { System.out.print("Hello, 
");
        System.out.print("World! 
");
    }
}
```

Yes, and in fact in many other ways. I won’t count how many…
# Simple Programs

Objects and classes, reference types, symbolic names, variables declaration and their initialization through assignment.

<table>
<thead>
<tr>
<th>Objects and classes are central concepts for Java programming.</th>
<th>It might take you some time to master these concepts fully, but since every Java program uses at least a couple of objects and classes, it is a good idea to have a basic understanding of these concepts right away.</th>
</tr>
</thead>
<tbody>
<tr>
<td>An object is an entity that you can manipulate in your program, generally by calling methods.</td>
<td>You should think of an object as a “black box” with a public interface—the methods you can call, and a hidden implementation—the code and data that are necessary to make these methods work.</td>
</tr>
<tr>
<td>Different objects support different sets of methods (in general).</td>
<td>OK, enough of that, let’s see some examples. Have we seen any object yet?</td>
</tr>
<tr>
<td>System.out</td>
<td>What methods does it have?</td>
</tr>
<tr>
<td>println and print</td>
<td>Yes, and we call them by identifying the object, followed by the name of the method, and then by parentheses.</td>
</tr>
<tr>
<td>Yes, don’t you ever forget the parentheses.</td>
<td>Good. What other objects have we seen?</td>
</tr>
<tr>
<td>None that I know of.</td>
<td>Exactly. You’ve seen one but you didn’t know it was an object.</td>
</tr>
<tr>
<td>&quot;Hello, World!&quot;</td>
<td>Indeed. It is a String object (its type).</td>
</tr>
<tr>
<td>Does every object have a type?</td>
<td>Yes.</td>
</tr>
<tr>
<td>Question</td>
<td>Answer</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Then what type does System.out have?</td>
<td>PrintStream and its class is defined in the package java.io. But this shouldn’t tell you too much just yet. (Full name is java.io.PrintStream.)</td>
</tr>
<tr>
<td>Nor could I have answered this question by myself with what we know so far.</td>
<td>Although you could have looked it up in the online documentation as part of class System.</td>
</tr>
<tr>
<td><a href="http://java.sun.com/products/jdk/1.2/docs/api/java/lang/System.html#out">http://java.sun.com/products/jdk/1.2/docs/api/java/lang/System.html#out</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://java.sun.com/products/jdk/1.2/docs/api/java/lang/System.html">http://java.sun.com/products/jdk/1.2/docs/api/java/lang/System.html</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://java.sun.com/products/jdk/1.2/docs/api/java/lang/overview-tree.html">http://java.sun.com/products/jdk/1.2/docs/api/java/lang/overview-tree.html</a></td>
<td></td>
</tr>
<tr>
<td>... which is defined in the package java.lang (and there are so many other packages besides it).</td>
<td>Let’s go back to &quot;Hello, World!&quot;.</td>
</tr>
<tr>
<td>What methods does it have?</td>
<td>No println or print, I don’t think...</td>
</tr>
<tr>
<td>To find out what methods it supports we need to look up its class (String) into the on-line documentation.</td>
<td>Of all the methods it has, we choose to take a look at the length() method.</td>
</tr>
<tr>
<td><a href="http://java.sun.com/products/jdk/1.2/docs/api/java/lang/String.html">http://java.sun.com/products/jdk/1.2/docs/api/java/lang/String.html</a></td>
<td></td>
</tr>
<tr>
<td>For any object of type String the length method counts and returns (reports) the number of characters in the string.</td>
<td>So &quot;Hello, World!&quot;.length() evaluates to 13</td>
</tr>
<tr>
<td>... as the quotation marks are not counted.</td>
<td>I’d like to see that.</td>
</tr>
<tr>
<td>Very well, then try this:</td>
<td></td>
</tr>
<tr>
<td>System.out.println(&quot;Hello, World!&quot;).length());</td>
<td></td>
</tr>
<tr>
<td>How does this work?</td>
<td></td>
</tr>
<tr>
<td>Same as before, only more work is to be done before println can output its argument.</td>
<td>OK, let’s move on. Without classes there would be no objects. Let’s take a look at classes.</td>
</tr>
<tr>
<td>A class is a holding place (or a container) for static methods and objects.</td>
<td>That’s old news: the Hello class holds the static main method. The System class holds the static out object.</td>
</tr>
<tr>
<td>A class is also a factory for objects. It contains the blueprint of all objects of that kind, and can be used to generate new objects.</td>
<td>I’d like to see that.</td>
</tr>
<tr>
<td>To see how a class can be an object factory, let us turn to another class: the Rectangle class in the Java class library.</td>
<td>Objects of type Rectangle describe rectangular shapes.</td>
</tr>
</tbody>
</table>
This is where you can read about Rectangle.

http://java.sun.com/products/jdk/1.2/docs/api/java/awt/Rectangle.html

Great—that’s off the same URL you gave me earlier.

Note that a Rectangle object isn’t a rectangular shape—it is just a set of numbers that describe the rectangle. Each rectangle is described by the \( x \) and \( y \) coordinates of its top left corner, its width and height.

We need to work an example first. You can make a new rectangle with top left corner at \((5, 10)\) and with a width of 20 and height 30. To make a new rectangle you need to specify these four values, and in that order.

\[
\text{new Rectangle}(5, 10, 20, 30)
\]

What does the new object look like?

I think I’d like to see a picture of that.

The new operator causes the creation of an object of type Rectangle. The process of creating a new object is called construction. The four values 5, 10, 20, 30 are called the construction parameters.

Now you can draw your picture.

What’s the rectangular shape that is described by this Rectangle object?

Why did you draw the referential upside down?

Because that’s how it is in computer graphics.

To find the \( x \) you just measure how far you are from the left margin of the screen.

To find out the \( y \) coordinate you measure how many lines of pixels are there in between that point and the top of the screen.
This is a side-effect of thinking that in English we write from left to right, and from top to bottom...

...so a character in a text message could be located by the number of the column in which it appears (the x coordinate) and the line in which it appears (the y coordinate).

To construct any object, you do the following:

1. use the **new** operator
2. give the name of the class
3. supply construction parameters (if any) inside parentheses

Different classes will require different construction parameters, and some classes will let you construct objects in multiple ways.

For example, you can also obtain a **Rectangle** object by supplying no construction parameters at all:

```
new Rectangle();
```

But you must still use the parentheses. This constructs a (rather useless) rectangle with top left corner at the origin (0, 0), width 0 and height 0.

How do I know that?

You have to read up the documentation see what the designers of the class had in mind for it.

So it’s not something I could have deduced, or inferred?

No.

What can you do with a **Rectangle** object?

What number?

Say, "abc".length() I could print it, to see if it comes out as 3 or not.

Can you print a **Rectangle** object? I could try. Can it be printed?

You should try it. I’d rather *draw* it.

We’ll learn that in a few chapters.

OK, how about this:

```java
System.out.println(new Rectangle(5, 10, 20, 30));
```

How does it work? The code creates an object of type **Rectangle** then passes the object to the **println** method, and finally forgets that object.
To remember an object give it a name; hold it in an object variable.

An object variable is a storage location that stores not the actual object but information about the object’s location.

Can you give it any name?

Variable names in Java can start with a letter, an underscore (_), or a dollar sign ($). They cannot start with a number. After the first character, your variable names can include any letter or number.

Once you decide on a name for a variable, to declare it you need to place the name of the class in front of it, followed by the variable name, and a semicolon (;) at the end.

Like this?

```
Rectangle a;
```

Yes. This is a declaration statement. It says that the name `a` will be used for a variable that will hold the address to a `Rectangle` object.

So far, the variable does not refer to any object at all. To make it refer to an actual object you could copy in it the address of an actual object reference, as returned by `new`:

```
Rectangle a = new Rectangle(5, 10, 20, 30);
```

It is very important that you remember that `a` does not contain the object. It contains the address of the object (and refers to this object).

I have a picture for that:

Very good.

I think I got the hang of it.
You could have two object variables refer to the same object. Like this?

```
Rectangle b = a;
```

Yes. The equal sign (=) acts more like an arrow from right to left, as it represents the copying of the value on the left into the location that the right hand side denotes.

So the value on the right, which is the contents of the variable `a` (an address) is copied into the location that `b` denotes.

`b` has just been declared. And at the time of its declaration (and also allocation) we copy in it the address of the anonymous object that `a` points to.

May I draw a picture of that?

Definitely.

Now how does the picture change if we add

```
Rectangle c = new Rectangle(5, 10, 20, 30);
```

A new object will come into the picture, and its address will be stored in the variable with the name `c`.

Isn’t it identical to the object pointed to by `a` and `b`? Yes, it’s like a twin. Identical but not the same.

Let me see it. There you go.
What else can you do with Rectangle objects? The Rectangle class has over 50 methods, some useful, some less so.

To give you a flavor of manipulating Rectangles, let us look at a method of the Rectangle class. The translate method moves a rectangle by a certain distance in the x and y directions.

For example

```python
a.translate(15, 25);
```...moves the rectangle by 15 units in the x direction and 25 units in the y direction.
Moving a rectangle doesn’t change its width or height, but it changes the coordinates of the top left corner.

Can I see that?

Let’s write a program and test it. As with the Hello program, we need to carry out three steps:

1. Invent a new class, say `RectangleTest`
2. Supply a main method
3. Place instructions inside the `main` method

Correct. Let’s see the program.

How about this one:

```java
public class RectangleTest {
    public static void main(String[] args) {
        Rectangle a = new Rectangle(5, 10, 20, 30);
        System.out.println(a);
        a.translate(15, 25);
        System.out.println(a);
    }
}
```

It would work well, but …if you try to compile it you will run into an error.

```
frilled.cs.indiana.edu% pico RectangleTest.java
frilled.cs.indiana.edu% javac RectangleTest.java
RectangleTest.java:3: Class Rectangle not found.
    { Rectangle a = new Rectangle(5, 10, 20, 30);
          ^
RectangleTest.java:3: Class Rectangle not found.
    { Rectangle a = new Rectangle(5, 10, 20, 30);
          ^
2 errors
frilled.cs.indiana.edu%
```

Does it always come out in boxes?

Not really. But the point here is that for this program there is an additional step that you need to carry out: you need to `import` the `Rectangle` class from a `package` which is a collection of classes with a related purpose. All classes in the standard library are contained in packages. The `Rectangle` class belongs to the package `java.awt`. Thus the full name of the `Rectangle` class is really `java.awt.Rectangle`.

The abbreviation `awt` stands for "Abstract Windowing Toolkit". To use `Rectangle` from the `java.awt` package simply place the following

```java
import java.awt.Rectangle;
```

…at the top of your program.
You never need to import classes from the `java.lang` package. All classes from this package are automatically imported.

That’s why we can use the `String` and `System` classes without ever needing to import them.

<table>
<thead>
<tr>
<th>Ready for one final question?</th>
<th>Certainly.</th>
</tr>
</thead>
</table>

What’s the output of the following snippet of code?

```java
Rectangle a = new Rectangle(5, 10, 20, 30);
Rectangle b = a;
a.translate(10, 10);
b.translate(10, 10);
System.out.println(a);
```

Easy: a and b both point to the same object. We basically translate it twice, once by using its `a` name and once by using its `b` name.

In the end printing `a` or `b` is the same.

You will see the original rectangle that has been translated twice.

So the top left corner is now at (25, 30) but the width and height are unchanged.

Good. Can we move on now?

We sure can.

Great. I think it was about time.
# Types and I/O

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s 23?</td>
<td>A number. An integer.</td>
</tr>
<tr>
<td>Java calls that an int.</td>
<td>Most of the times.</td>
</tr>
<tr>
<td>What’s 3.5?</td>
<td>A number with a decimal part.</td>
</tr>
<tr>
<td>Java calls that a floating-point number.</td>
<td>I see. Isn’t there a keyword for that, like <code>int</code>?</td>
</tr>
<tr>
<td>There are two of them: <strong>double</strong> and <strong>float</strong>.</td>
<td>In Java there are two kinds of numbers: integers and floating point numbers.</td>
</tr>
<tr>
<td>Integers have no fractional part.</td>
<td>And floating point numbers, which have a decimal point and therefore a fractional part.</td>
</tr>
<tr>
<td>So 2.0 is a floating-point number</td>
<td>... while 2 is an integer.</td>
</tr>
<tr>
<td>The second one does not have any fractional part, while the first one’s is zero.</td>
<td>Not missing, but zero.</td>
</tr>
<tr>
<td>In practice this can make a big difference.</td>
<td>There are two reasons for having separate types for numbers: one philosophical and one pragmatic.</td>
</tr>
<tr>
<td>The philosophy is to use whole numbers when you can’t have or don’t need a fractional part.</td>
<td>It is generally a good idea to choose programming solutions that document one’s intentions.</td>
</tr>
<tr>
<td>Pragmatically speaking, integers are more efficient than floating-point numbers.</td>
<td>They take less storage and are processed faster.</td>
</tr>
<tr>
<td>Question</td>
<td>Answer</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>How do you use <code>int, double, float</code> in practice?</td>
<td>Like <code>Rectangle</code> they’re types.</td>
</tr>
<tr>
<td>Unlike objects of type <code>Rectangle</code> numbers are not objects.</td>
<td>Yes, <code>Rectangle</code> is a <em>reference type</em>. <code>int, double, float</code> (and 5 other) are <em>primitive types</em>.</td>
</tr>
<tr>
<td>So we can declare a variable of type <code>int</code>?</td>
<td>Yes. It’s like in algebra, except names have types in Java.</td>
</tr>
<tr>
<td>In Java each variable has a <em>type</em>.</td>
<td></td>
</tr>
<tr>
<td>By defining</td>
<td>Even though initialization is optional, it is a good idea always to initialize variables with a specific value.</td>
</tr>
<tr>
<td><code>int a;</code></td>
<td></td>
</tr>
<tr>
<td>you proclaim that <code>a</code> can hold only integer values.</td>
<td></td>
</tr>
<tr>
<td>You should always supply an initial value for every variable at the time you define it.</td>
<td>So I could, for example, write:</td>
</tr>
<tr>
<td></td>
<td><code>int a = 3;</code></td>
</tr>
<tr>
<td>Could you have written</td>
<td>No that is a contradiction in terms. I would have broken my own rule of proclaiming that <code>a</code> won’t need a fractional part.</td>
</tr>
<tr>
<td><code>int a = 3.5;</code></td>
<td></td>
</tr>
<tr>
<td>instead?</td>
<td></td>
</tr>
<tr>
<td>Can we write</td>
<td>Yes, but how about:</td>
</tr>
<tr>
<td><code>double b = 3.4;</code></td>
<td><code>double b = 3;</code></td>
</tr>
<tr>
<td>That would work well, since there’s no loss of information. Seeing the missing fractional part of <code>3</code> Java will initialize <code>b</code> with <code>3.0</code></td>
<td>Symbolic names like <code>a</code> and <code>b</code> are meant to make the program more readable and manageable. Keep in mind, however, that you can only declare and initialize a symbolic name just once in every method.</td>
</tr>
<tr>
<td>Why is it good to initialize variables as soon as we define them?</td>
<td>So that we don’t forget to initialize them at all.</td>
</tr>
<tr>
<td>If you try to use an uninitialized variable the compiler will notice and complain.</td>
<td>All but the simplest programs use variables to store values. Variables are locations in memory that can hold values of a particular type.</td>
</tr>
</tbody>
</table>
They’re called variables because once you store a value in them you can change it later at will.

I’d like to see that.

Very well, take a look:

```java
int a = 5;
a = 7; // a is declared and initialized to 5,
System.out.println(a); // but later the value 7 is assigned to it.
```

That prints 7 and it works as follows:

Yes, that was an example of an assignment statement. Can I see others?

Yes. Here’s a more complicated scenario:

```java
int a = 5;
int b = a + 5; // as what’s being stored in b is the value of a plus 5.
System.out.println(b);
```

This prints 10

So a symbolic name acts as a storage location (or address) on the left hand side of the equals sign (which is used for assignment statements)...

...while when it appears on the right hand side it is replaced by its value.

Can one name appear on both sides of an assignment? Show me an example.

Here’s a scenario:

```java
int a = 10; a = a + 10; System.out.println(a);
```

By the rule you formulated, that should print 20.

Yes, since the expression on the right is evaluated first, then the resulting value is stored in the location named a.

What other expressions can we use?

All of arithmetic, if you refer to numbers. However, in general, when you make an assignment of an expression into a variable, the expression on the right, ...

...which can contain method invocations,... ...is first evaluated, and the resulting value has to be of a compatible type with the type of the variable.

OK, I will try to remember that. Let’s work some more examples.

Here’s a challenge first. Given:

```java
int x, y; x = 5; y = 3;
```

Easy.

...how do you swap the values of x and y? I need a third (temporary) location:


<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>int ( x = 5 ), ( y = 3 ), <code>temp</code>;</td>
<td></td>
</tr>
<tr>
<td>( temp = x; )</td>
<td></td>
</tr>
<tr>
<td>( x = y; )</td>
<td></td>
</tr>
<tr>
<td>( y = temp; )</td>
<td></td>
</tr>
<tr>
<td>Is there another way?</td>
<td></td>
</tr>
<tr>
<td>What do you mean?</td>
<td></td>
</tr>
<tr>
<td>Supose you can't use another variable.</td>
<td></td>
</tr>
<tr>
<td>Yet ( x ) and ( y ) are numbers?</td>
<td></td>
</tr>
<tr>
<td>Yes.</td>
<td></td>
</tr>
<tr>
<td>Then it’s trickier, but fancier:</td>
<td></td>
</tr>
<tr>
<td>int ( x = 5 ), ( y = 3 );</td>
<td></td>
</tr>
<tr>
<td>( x = x + y; )</td>
<td></td>
</tr>
<tr>
<td>( y = x - y; )</td>
<td></td>
</tr>
<tr>
<td>( x = x - y; )</td>
<td></td>
</tr>
<tr>
<td>Well done!</td>
<td></td>
</tr>
<tr>
<td>I know. Isn’t that nice?</td>
<td></td>
</tr>
<tr>
<td>Let’s work out some more examples.</td>
<td></td>
</tr>
<tr>
<td>Why is this legal?</td>
<td></td>
</tr>
<tr>
<td>int ( a = 3 );</td>
<td></td>
</tr>
<tr>
<td>double ( b = a; )</td>
<td></td>
</tr>
<tr>
<td>Because ( b ) becomes 3.0, so we acknowledge the lack of fractional</td>
<td></td>
</tr>
<tr>
<td>part of ( a ) by writing a 0 (zero) for it in ( b ).</td>
<td></td>
</tr>
<tr>
<td>Is this legal?</td>
<td></td>
</tr>
<tr>
<td>double ( b = 3.5; )</td>
<td></td>
</tr>
<tr>
<td>int ( a = b; )</td>
<td></td>
</tr>
<tr>
<td>No, because ( a ) doesn’t have any room for a fractional part (0.5)</td>
<td></td>
</tr>
<tr>
<td>in it.</td>
<td></td>
</tr>
<tr>
<td>Can we just ignore that, the fractional part?</td>
<td></td>
</tr>
<tr>
<td>You can, but Java won’t do that for you unless you specifically request</td>
<td></td>
</tr>
<tr>
<td>it.</td>
<td></td>
</tr>
<tr>
<td>How do I do that?</td>
<td></td>
</tr>
<tr>
<td>You <em>cast</em> the floating point value to an integer:</td>
<td></td>
</tr>
<tr>
<td>double ( b = 3.5; )</td>
<td></td>
</tr>
<tr>
<td>int ( a = ) (int) ( b; )</td>
<td></td>
</tr>
<tr>
<td>This has the effect of discarding the fractional part.</td>
<td></td>
</tr>
<tr>
<td>What is (int)?</td>
<td></td>
</tr>
</tbody>
</table>
It is the *cast to an int* operator. It acts like the unary minus sign. For example, the expression 
-5 + 3

...which yields -2,

and

-(5 + 3)

...which yields -8,

...is the same as the difference between

(\text{int})3.6 + 3.6

...which yields 6.6,

...and

(\text{int})(3.6 + 3.6)

...which yields 7.

There is a good reason why you must use a cast in Java when you convert a floating point number to an integer: The conversion *loses information.* You must confirm that you agree to that information loss. Java is quite strict about this. You must use a cast whenever there is the possibility of information loss.

A cast is always of the form

(\text{typename})

for example (\text{int}) or (\text{double}).

There are a few methods in class \text{Math} that have a related functionality.

\begin{align*}
\text{Math.round}(3.7) \\
\text{evaluates to} & \ldots \\
& 4
\end{align*}

\begin{align*}
\text{Math.round}(-3.7) \\
\text{evaluates to} & \ldots \\
& -4
\end{align*}

\begin{align*}
\text{Math.round}(3.2) \\
\text{evaluates to} & \ldots \\
& 3
\end{align*}

\text{Math.round}(x) \text{ evaluates to the closest integer to } x \\
\text{(represented as a long).}

What's \text{long}?

Another kind of integer. We'll talk about it before \text{too long.}

OK. Hit me with more \text{Math}
<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Math.ceil(3.7)</code></td>
<td>4.0</td>
</tr>
<tr>
<td><code>Math.ceil(-3.7)</code></td>
<td>-3.0</td>
</tr>
<tr>
<td><code>Math.ceil(3.2)</code></td>
<td>4.0</td>
</tr>
<tr>
<td><code>Math.floor(x)</code> evaluates to the smallest integer greater or equal to x (as a <code>double</code>).</td>
<td><code>Math</code> is short for “ceiling”. There is also a mathematical “floor” function.</td>
</tr>
<tr>
<td><code>Math.floor(3.7)</code></td>
<td>3.0</td>
</tr>
<tr>
<td><code>Math.floor(-3.7)</code></td>
<td>-4.0</td>
</tr>
<tr>
<td><code>Math.floor(3.2)</code></td>
<td>3.0</td>
</tr>
<tr>
<td><code>Math.floor</code> evaluates to the largest integer less than or equal to x (as a <code>double</code>).</td>
<td><code>Math</code> is a class that is defined in the <code>java.lang</code> package.</td>
</tr>
</tbody>
</table>

The `Math` class groups together the definitions of several useful mathematical methods such as: `sqrt`, `pow`, `sin`, `cos`, `exp`, `log`, `abs`, `round`, `ceil`, `floor` ...

...and many others. All these methods are `static` (or class) methods (unlike `print` and `println` of the `System.out` object). They belong to the class `Math`. 

Beginners (or uninitiated) might think that in

```
Math.round(3.7)
```

the `round` method is applied to an object called `Math`, because `Math` precedes `round`...just as `System.out` precedes `print`. That's not true. `Math` is a class, not an object.

A method such as `Math.round` that does not operate on any object is known as a `static` method; another example is `main`.

Static methods do not operate on objects, but they are still defined inside classes, and you must specify the class to which the `round` method belongs.

How can you tell whether `Math` is a class or an object. You really can't.

Then how do we know?

It is certainly useful to memorize the names of the more important classes (such as `System` and `Math`). You should also pay attention to capitalization.

All classes in the Java library start with an uppercase letter (such as `System`). Objects and methods start with a lowercase letter (such as `out` and `println`).
You can tell objects and methods apart because method calls are followed by parentheses.

Therefore

```java
System.out.println()
```

denotes a call of the `println` method on the `out` object inside the `System` class. On the other hand

```java
Math.round(price)
```

denotes a call to the `round` method...

...inside the `Math` class. This use of upper- and lowercase letters is merely a *convention*, not a rule of the Java language.

It is, however, a convention that the authors of the Java class libraries follow consistently. You should do the same in your programs.

You can use all four basic arithmetic operations in Java: addition, subtraction, multiplication, and division.

Parentheses are used just as in algebra: to indicate in which order the subexpressions should be computed.

Just as in regular algebraic notation, multiplication and division *bind more strongly* than addition and subtraction.

So `3 + 5 * 2` yields 13 while `(3 + 5) * 2` yields 16 as the parentheses come into play.

Division works as you would expect, as long as at least one of the numbers involved is a floating-point number.

However, if *both* numbers are integers, then the result of the division is always an integer, with the remainder discarded.

Here are some examples:

<table>
<thead>
<tr>
<th></th>
<th>...evaluates to:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>17 / 4</code></td>
<td>4</td>
</tr>
<tr>
<td><code>10 / 3</code></td>
<td>3</td>
</tr>
<tr>
<td><code>13 / 7</code></td>
<td>1</td>
</tr>
<tr>
<td>and <code>6 / 9</code></td>
<td></td>
</tr>
<tr>
<td>evaluates to...</td>
<td>0</td>
</tr>
</tbody>
</table>

If you’re interested only in the remainder, ...

...you can use the `%` operator.

But for that we need to turn the page. Exactly.
Here are some examples. ...evaluates to:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 % 4</td>
<td>1</td>
</tr>
<tr>
<td>10 % 3</td>
<td>1</td>
</tr>
<tr>
<td>13 % 7</td>
<td>6</td>
</tr>
</tbody>
</table>

...is the same as a (for a and b positive integers)?

(a / b) * b + a % b

Is it true that...

The symbol % has no analog in algebra. It was chosen because it looks similar to /, and the remainder operation is related to division.

Yes. But can you prove it? OK, let's move on.

What is 16 / 5 * 5 15, since all operands are integers.

How about (16 / 5) * 5 Still 15, and in the same way.

What then is 5 * (16 / 5) 15 (multiplication is commutative).

OK, drop the parentheses: 5 * 16 / 5 The result is now 16 as we have to do the multiplication first.

What property requires this? Left-to-right associativity.

Very good. Let’s move on.

Next to numbers strings are the most important data type that most programs use. A string is a sequence of characters, such as "Hello".

In Java strings are enclosed in quotation marks, which are not themselves part of the string.
You can declare variables that hold strings:

```
String name = "John";
```

Use assignment

```
name = "Carl";
```

...to place a different string into the variable.

The number of characters in a string is called the `length`. For example, the length of "Hello!" is 6.

You can compute the length of a string with the `length` method:

```
int n = name.length();
```

That would place 4 in `n`.

For our example, yes.

By the way, a string of length 0 (zero), containing no characters, is called the **empty** string

...and is written as "".

Also note that unlike numbers, strings are objects.

Rectangles were objects too. You can tell that `String` is a class because it starts with an uppercase letter. The basic types `int` and `double` start with a lowercase letter.

Once you have a string, what can you do with it?

You can extract substrings, and you can glue smaller strings together to form larger ones.

To extract a string use the `substring` operation.

```
String a = "automaton";
String b = a.substring(2, 8);
// b is set to "tomato"
```

...the characters in the string `s` starting at character with index `start`, and containing all characters up to, but not including, the character with index `pastEnd`. Let's see an example.

In Java there are two ways of writing comments.

We already know (and have used it above) that the compiler ignores anything that you type between `//` and the end of line.

The compiler also ignores any text between a `/*` and `*/`. The `//` comment is easier to type if the comment is only a single line long.

If you have a comment that is longer than a line or two, then the `/* ....*/` comment is simpler.

So we could also have:

```
String c = "appearance";
String d = c.substring(2, 6);
// d is "pear" as the substring operation makes a string
// that consists of four characters taken from string c *
```

A curious aspect of the `substring` operation is the numbering of starting and ending positions.

Starting position 0 (zero) means "start at the beginning of the string".
For technical reasons that used to be important but are no longer relevant, Java string position numbers start at 0 (zero).

The first position is labeled 0 (zero), the second one is labeled 1 (one), and so on.

For example here are the position numbers

| appearance | 0123456789 |

in the "appearance" string.

<table>
<thead>
<tr>
<th>How do you extract the substring &quot;Bird&quot; from &quot;Larry Bird, Indiana&quot;?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0123456789</td>
</tr>
<tr>
<td>1 1</td>
</tr>
<tr>
<td>0123456789012345678</td>
</tr>
</tbody>
</table>

The position number of the last character (a for the "appearance" string) is always 1 less than the length of the string.

Therefore the appropriate substring command is

```java
String m = "Larry Bird, Indiana";
String n = m.substring(6, 10);
```

Count characters starting at 0, not 1. You find that B, the 7th character, has position number 6. The first character that you don't want, a comma, is the character at position 10.

It is curious that you must specify the position of the first character that you do want and then the first character that you don’t want.

There is one advantage to this setup. You can easily compute the length of the substring: it is pastEnd - start

If you omit the second parameter of the substring method, then all characters from the starting position to the end of the string are copied.

For example:

```java
String u = "Larry Bird, Indiana";
String tail = u.substring(6);
```

... sets tail to the string "Bird, Indiana"

This is equivalent to the call

```java
String tail = u.substring(6, u.length());
```

I see.

Now that you know how to take strings apart, let us see how to put them back together.

Given two strings, such as "India" and "napolis", you can concatenate them to one long string.

```java
String one = "India";
String two = "napolis";
String city = one + two;
```

The + operator concatenates two strings.

How do you get "Larry Bird"

out of "Larry" and "Bird"?

"Larry" + " " + "Bird"

Very good, with a blank in the middle.
The concatenation operator in Java is very powerful. If one of the expressions, either to the left or the right of a + operator, is a string, then the other one is automatically forced to become a string as well, and both strings are concatenated.

For example:

<table>
<thead>
<tr>
<th>...evaluates to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Agent&quot; + 7</td>
</tr>
<tr>
<td>&quot;Agent7&quot;</td>
</tr>
<tr>
<td>&quot;2&quot; + 3</td>
</tr>
<tr>
<td>&quot;23&quot;</td>
</tr>
<tr>
<td>2 + 3</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Of course when associativity comes into play you have to be a bit more careful.

<table>
<thead>
<tr>
<th>...evaluates to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 + &quot;&quot; + 3</td>
</tr>
<tr>
<td>&quot;23&quot;</td>
</tr>
<tr>
<td>whereas...</td>
</tr>
<tr>
<td>...evaluates to:</td>
</tr>
<tr>
<td>2 + 3 + &quot;&quot;</td>
</tr>
<tr>
<td>&quot;5&quot;</td>
</tr>
</tbody>
</table>

Concatenation is very useful to reduce the number of System.out.print instructions. Fine. What if you want to convert a string like "23" that contains only decimals into a number?

To convert a string into a number you have two possibilities: to convert to an int use... Integer.parseInt(...)

...and to convert to a double use... Double.parseDouble(...)

So,...

<table>
<thead>
<tr>
<th>...evaluates to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;23&quot; + 5</td>
</tr>
<tr>
<td>&quot;235&quot;</td>
</tr>
</tbody>
</table>

but

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer.parseInt(&quot;23&quot;) + 5</td>
</tr>
<tr>
<td>28</td>
</tr>
</tbody>
</table>

Likewise

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;2.3&quot; + 5</td>
</tr>
<tr>
<td>&quot;2.35&quot;</td>
</tr>
</tbody>
</table>

while

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Double.parseDouble(&quot;2.3&quot;) + 5</td>
</tr>
<tr>
<td>7.3</td>
</tr>
</tbody>
</table>
How can you get a "2.3" string when you need a number? If you’re a user and type a number such as 2.3 you will realize that you, in fact, have to type three characters:

...the digit 2, the period, and the digit 3. The whole thing is a string of characters.

And that’s what you will have to start from when dealing with user input. How can we write programs that accept user input?

We will use a non-standard Java class, that we will thoroughly discuss later on, and whose purpose is to make processing keyboard input easier and less tedious. We recommend that you use the `ConsoleReader` class in your own programs whenever you need to read console input.

Simply place the `ConsoleReader.java` file together with your program file, into a common directory. The purpose of the `ConsoleReader` class is to add a friendly interface to an input stream such as `System.in` and here’s how you use it.

To accept user input in a program you first need to construct a `ConsoleReader` object, like this:

```java
ConsoleReader console = new ConsoleReader(System.in);
```

Next, simply call one of the following three methods:

```java
String line = console.readLine(); // read a line of input
int n = console.readInt(); // read an integer
double x = console.readDouble(); // read a floating-point number
```

Let’s see an example. OK, let’s write a program that asks a user for a name (that will be recorded as a `String`), an amount of dollars (that the user has) and the rate of exchange between the British pound and the American dollar.

The program asks all this of the user, and then computes and tells the user how many British pounds the user would get in exchange for the amount of dollars the user has. Why not leave this for tomorrow?

Good idea. See you in the lab.
First Things First

Some simple programs. Some sample problems.

Goals for this lab:

- Work with integer and floating-point numbers
- Write arithmetic expressions in Java
- Define and initialize variables and constants
- Recognize limitations of the int and double types and the overflow and roundoff errors that can result
- Be able to change the values of variables through assignment
- Use String type to define and manipulate character strings

Finally, work a large number of non-trivial problems on your own, and check your solutions against those posted here. These problems are not at all immediate, but from a programming point of view the resulting programs are by and large elementary. So this first set of practice problems is meant to be a very good set of warm-up problems for this first week, first two chapters of study. I include some warm-up exercises as well.

Please note:

1. This is a long lab, meant to give you enough work to really get you started.
2. Your in-lab assignment is indexed at the end of these lab notes.
3. Here now is the interactive conversion program announced at the end of Lecture Notes Four.

First a sample session of working with the program (on Unix). I marked the user’s answers with [boxes] but this only for illustration purposes in these lab notes. Don’t expect the program to actually print with boxes when you compile and run it, of course.

```java
frilled.cs.indiana.edu,%javac Conversion.java
frilled.cs.indiana.edu,%java Conversion
Hi my name is Hal. What is your name?
Dave
Hello, Dave! How many dollars do you want to convert?
(Please type an integer value, no decimal part)
40
```
I see, you want to convert 40 dollars in British pounds. Very well...

What is the conversion rate today?

Well, Dave for 40 dollars you can get: 26.0 pounds.

Thank you for using Hal! Good-bye.

frilled.cs.indiana.edu%

Here's the actual Conversion class—with comments included.

    /* A conversion program that illustrates how one can read input
       from the keyboard using the textbook's ConsoleReader class */
    class Conversion {
        public static void main(String[] args) {
            ConsoleReader console = new ConsoleReader(System.in);
            // now we can read from 'console'
            System.out.println("Hi my name is Hal. What is your name?");
            // greet the user
            String name = console.readLine();
            // wait for input and collect it (a String, for the name)
            System.out.println("Hello, " + name + "!");
            // echo the name to the user, raising user's confidence in us
            System.out.println("How many dollars do you want to convert?");
            // approach the user directly, offer your services
            System.out.println("(Please type an integer value, no decimal part)"^^);\n            // instruct the user what limitations you have (accepting only int's)
            int amount = console.readInt();
            // wait for the user to provide the sum to be converted
            System.out.println("I see, you want to convert " + amount +
                               " dollars in British pounds. Very well\dots");
            // talk to the user, be friendly, echo information frequently
            System.out.println("What is the conversion rate today?"^^);\n            // ask the user for the last piece of input
            double rate = console.readDouble();
            // wait for the conversion rate, which can have a fractional part
            System.out.println("Well, " + name + " for " + amount + " dollars " +
                               "you can get: " + rate * amount + " pounds.");
            // do the calculation and report it to the user
            System.out.println("Thank you for using Hal! Good-bye.");
            // thank the user for interest and say good-bye
        }
    }

And here's the ConsoleReader class. This class is given to you for the purpose of doing keyboard input. You don't need to understand the code right now, it will become clearer at the end of chapter 3 (and we will signal that to you when time comes). However we think it's good for you to be exposed to what it looks like, and to slowly become familiar with it, and Java code and I/O (which will be covered at some point, in class, later).

    //*******************************************************************************
    ConsoleReader class is used for keyboard input.
    Just keep the source code of this class (the file)
    in the same directory in which your other program
    (the one that needs user input) resides.
In this particular case keep it in the same folder with Conversion.java that has been presented above.

```java
import java.io.*; // I/O package needed
class ConsoleReader {
    ConsoleReader(InputStream inStream) { // constructor
        reader = new BufferedReader(
            new InputStreamReader(inStream));
    }
    String readLine() { // instance method
        String inputLine = "";
        try {
            inputLine = reader.readLine();
        } catch (IOException e) {
            System.out.println(e);
            System.exit(1);
        }
        return inputLine;
    }
    int readInt() { // instance method
        String inputString = readLine();
        int n = Integer.parseInt(inputString);
        return n;
    }
    double readDouble() { // instance method
        String inputString = readLine();
        double x = Double.parseDouble(inputString);
        return x;
    }
    private BufferedReader reader; // instance method
}
```

In your programs, from now on, keep using this class for user input from the keyboard, as it will provide a simple, uniform way of doing keyboard input. Here now is also a diagrammatic description of the fundamental difference between primitive and reference types, that we looked at in the notes:

Reference types (using Rectangle)

```
Rectangle a = new rectangle(10, 20, 30, 40);
Rectangle b = [a];
```

The picture looks as shown on the next page..

Now if you have

```
[a].translate(3, 3);
System.out.println([b]);
```

you will be able to see the change. Both a and b point to one and the same thing, an essentially anonymous object, to which we refer as both a and b. So changes made using the a name can be seen by looking at the object using the other name, that is, b.
Primitive types (using int)

```java
int a = 3;
int b = a;
```

The picture looks as follows:

```
  a   b
[3]  [3]
```

The big difference is that the primitive value is copied into the storage location. So each location has its own copy. It just works that way with the numbers (as primitive types) but doesn’t with objects (reference types). That’s how it works. With reference types, what the variable holds is a reference, the arrow. With primitive types, the variable holds a value, a copy of the actual value.

Now if you have

```java
a = 10;
System.out.println(b);
```

you will see that the value of b has not been updated.

Each location has its own (copy of the) value, and changing one does not affect the other. This is an important difference, we refer to it as the by value vs. by reference contrast, and it will be useful for you to remember it from now on when you reason about and design programs.

Now, let’s move on to exercises.

I list the URLs below, and include the actual exercises and their solutions in the following pages:

1. Here’s the document of warm-up questions.
   
   http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/chapTwoReview.htm
2. Here's the same document with answers.
   http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/chapTwoRevSols.html
3. Here are the problems that you should think about.
   http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/chapTwoProbs.html
4. Here are the solutions to these problems.
5. Here's your lab assignment no. 2, due at the beginning of the next lab.
   http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/labAssignTwo.html
6. And here, to remind us of the intrinsic fun, is the note on solving problems.
   http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/RPF.html

And now on to the real contents of this lab.

Note that your lab assignment will come right after these, separately.
A Simple Converter

This is your Lab Assignment Two

You are to write a program that calculates the trigonometric functions sine\(^1\) and cosine\(^2\) of any angle expressed in sexagesimal degrees. There are 90 sexagesimal degrees in a right angle, and 360 sexagesimal degrees in a full circle. You are being assisted by a mathematician, for which you write this program, which knows these things, and many more\(^3\) You’ve read the notes thus far and have checked the on-line APIs\(^4\) and you have decided to use the `Math` class in the `java.lang` page. You therefore check the `java.lang.Math`\(^5\) class in the Java API and determine that there are two functions that implement what you need:

- `double Math.sin(double a)`\(^6\) and
- `double Math.cos(double a)`\(^7\)

but as the document says, they work on radians.

Fortunately the mathematician\(^8\) gives you a formula for converting sexagesimal degrees \((d)\) into radians \((r)\), as follows:

\[
r = \frac{d \pi}{180}
\]

That makes you quickly want to look up the value of `Math.PI`\(^9\) but once you find it, and other than that everything seems to be taken care of with this formula.

You are supposed to use this information to write a program that reads angles, expressed in integer values of sexagesimal degrees, converts them in radians, and calculates the sine and cosine of the original angles and reports them.

Here’s an example of how the program could behave (turn the page for that).

---

\(^1\)http://mathworld.wolfram.com/Sine.htm
\(^2\)http://mathworld.wolfram.com/Cosine.htm
\(^3\)http://mathworld.wolfram.com/Trigonometry.html
\(^4\)http://java.sun.com/j2se/1.4.1/docs/api/overview-tree.html
\(^5\)http://java.sun.com/products/jdk/1.2/docs/api/java/lang/Math.html
\(^6\)http://java.sun.com/products/jdk/1.2/docs/api/java/lang/Math.html#sin(double)
\(^7\)http://java.sun.com/products/jdk/1.2/docs/api/java/lang/Math.html#cos(double)
\(^8\)http://www.cs.indiana.edu/classes/a201-dger/spr2001/images/mhpj.jpeg
\(^9\)http://java.sun.com/products/jdk/1.2/docs/api/java/lang/Math.html#PI
Everything that the computer prints is in boxes.

frilled.cs.indiana.edu%java Two
Hello, and welcome to the conversion program.
Please enter the number of degrees: 0
I will transform 0 degrees in radians.
Answer is: 0.0 radians.
The sine of that is: 0.0
The cosine of that is: 1.0
frilled.cs.indiana.edu%java Two
Hello, and welcome to the conversion program.
Please enter the number of degrees: 90
I will transform 90 degrees in radians.
Answer is: 1.5707963267948966 radians.
The sine of that is: 1.0
The cosine of that is: 6.123233995736766E-17
frilled.cs.indiana.edu%java Two
Hello, and welcome to the conversion program.
Please enter the number of degrees: 60
I will transform 60 degrees in radians.
Answer is: 1.0471975511965976 radians.
The sine of that is: 0.8660254037844386
The cosine of that is: 0.5000000000000001
frilled.cs.indiana.edu%java Two
Hello, and welcome to the conversion program.
Please enter the number of degrees: 30
I will transform 30 degrees in radians.
Answer is: 0.5235987755982988 radians.
The sine of that is: 0.4999999999999994
The cosine of that is: 0.8660254037844387
frilled.cs.indiana.edu%java Two
Hello, and welcome to the conversion program.
Please enter the number of degrees: -90
I will transform -90 degrees in radians.
Answer is: -1.5707963267948966 radians.
The sine of that is: -1.0
The cosine of that is: 6.123233995736766E-17
frilled.cs.indiana.edu%java Two

Please note that...

1. There is a toRadians() method in class Math.
2. You can use it to only to check your calculations.
3. You need to make the conversion yourself, by implementing the formula.

4. Write your program as described above, and have it ready during Lab Three.

5. Also review the first set of warmups and their solutions and expect questions from them.

In a subsequent installment we are going to look at the programming exercises (not just the warmups) too. But for now your focus should be on writing the program and reviewing the warmups.

And in case you are reading this offline, here’s your friend the Mathematician.

He likes Fractions, and is pouring in some tea (he doesn’t know Java, that’s why he had to ask for your help). But he knows tea fairly well.
Warmups (I)

This is the first set of warmups.
These warmup exercises are part of Lab Notes Two.

Questions:

1. Write the following mathematical expressions in Java.

   \[ s = s_0 + v_0 t + \frac{1}{2}gt^2 \]

   \[ G = \frac{4\pi^2}{P^2(m_1 + m_2)} \]

   \[ FV = PV \left( 1 + \frac{INT}{100} \right)^{YRS} \]

   \[ c = \sqrt{(a^2 + b^2 - 2ab\cos\gamma)} \]

2. Write the following Java expressions in mathematical notation.

   \[
   \text{dm} = m \times ((\text{Math.sqrt}(1 + v / c) / \text{Math.sqrt}(1 - v / c)) - 1);
   \]

   \[
   \text{volume} = \text{Math.PI} \times r \times r \times h;
   \]

   \[
   \text{volume} = 4 \times \text{Math.PI} \times \text{Math.pow}(r, 3) / 3;
   \]

3. What is wrong with this version of the quadratic formula?

   \[
   x_1 = \left( -b - \text{Math.sqrt}(b \times b - 4 \times a \times c) \right) / 2 \times a;
   \]

   \[
   x_2 = \left( -b + \text{Math.sqrt}(b \times b - 4 \times a \times c) \right) / 2 \times a;
   \]

4. (a) Give an example of integer overflow. Would the same example work correctly if you used floating-point?

   (b) Give an example of a floating-point roundoff error. Would the same example work correctly if you used integers?

   (c) When using integers, you would of course need to switch to a smaller unit, such as cents instead of dollars or milliliters instead of liters.
5. Let \( n \) be an integer and \( x \) a floating-point number.

Explain the difference between

\[
\begin{align*}
n &= (\text{int})x; \\
n &= (\text{int})\text{Math.round}(x);
\end{align*}
\]

(a) For what values of \( x \) do they give the same result?
(b) For what values of \( x \) do they give different results?
(c) What happens if \( x \) is negative?

6. Find at least five syntax errors in the following program.

```java
public class WarmUpSix {
    public static void main(String[] args) {
        System.out.print("This program adds two numbers.");
        x = 5;
        int y = 3.5;
        System.out.print("The sum of " + x + " and " + y + " is: ");
        System.out.println(x + y)
    }
}
```

7. Find at least three logic errors in the following program.

```java
public class WarmUpSeven {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        int total = 1;
        System.out.println("Please enter a number:");
        int x1 = Integer.parseInt(console.nextLine());
        total = total + x1;
        System.out.println("Please enter another number:");
        int x2 = Integer.parseInt(console.nextLine());
        total = total + x1;
        double average = total / 2;
        System.out.println("The average of two numbers is " + average);
    }
}
```

8. Explain the difference between 2, 2.0, "2", and "2.0".

9. Explain what each of the following two program segments computes:

```java
x = 2;
y = x + x;
```

and

```java
s = "2";
t = s + s;
```
10. Uninitialized variables can be a serious problem. Should you always initialize every variable with zero? Explain the advantages and disadvantages of such a strategy.

11. True or false? (x is an int and s is a String)
   
   (a) `Integer.parseInt("" + x)` is the same as `x`
   (b) `"" + Integer.parseInt(s)` is the same as `s`
   (c) `s.substring(0, s.length())` is the same as `s`

12. Give two ways for converting a number to a string. What is the advantage of each of these ways?

13. (a) How do you get the first character of a string? The last character?
    (b) How do you remove the first character? The last character?

14. How do you get the last digit of a number? The first digit? That is, if \texttt{n} is 23456, how do you find out 2 and 6? Do not convert the number to a string. (Hint: \% 10, Math.log)

15. What is a final variable? Can you define a final variable without supplying its value?

16. For the purpose of this exercise, for each of the following cases assume:

   ```java
   double x = 2.5;
   double y = -1.5;
   int m = 18;
   int n = 4;
   String s = "Hello";
   String t = "World";
   ```

   What are the values of the following expressions?
   
   (a) `x + n * y - (x + n) * y`
   (b) `m / n + m \% n`
   (c) `5 * x - n / 5`
   (d) `Math.sqrt(Math.sqrt(n))`
   (e) `(int)Math.round(x)`
   (f) `(int)Math.round(x) + (int)Math.round(y)`
   (g) `s + t`
   (h) `s + n`
   (i) `1 - (1 - (1 - (1 - (1 - n))))`
   (j) `s.substring(1, 3)`
   (k) `s.length() + t.length()`

Some advice, at the end of this document:

Please, try think about these problems, and only then look up the answers. You have all the answers, so there's no pressure. The emphasis is on learning. So, try to solve them on your own first. Then, please, make sure you check the solutions. Next lab you might be given questions randomly from this set of warmups. And, as always, please let me know if you have any questions or need any help.
Warmups Solutions (I)

Answers to the first set of warmup questions.

1. Note that you had a few degrees of freedom regarding the types of the variables, and what this problem was testing was essentially the ability to put together an expression with constants, identifiers (variables names) and the operators we have in Java. It was OK to keep the variables names as given in the original formulas, even though if you were to name them in a program you would probably follow the convention by which names of functions and variables should start with lowercase.

2. Here are the answers:

\[ dm = m \left( \frac{\sqrt{1 + \frac{2}{b}}}{\sqrt{1 - \frac{2}{c}}} - 1 \right) \]

\[ \text{volume} = \pi r^2 h \]

\[ \text{volume} = \frac{4 \pi r^3}{3} \]

3. The difference is the one between

\[ \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

and

\[ \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

Here the Java expressions are encoding the first formula (because of the property we called “left to right associativity”). The correct expression is the second one, where \( a \) appears in the denominator, so you need to put \( 2 \times a \) in parentheses

\[ x1 = (-b - Math.sqrt(b * b - 4 * a * c)) / (2 * a); \]

\[ x2 = (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a); \]

4. The answer to this question is that integers cover a smaller range than floating point numbers so switching to a floating point would no longer produce an overflow.

For roundoff errors the simplest example is

\[ 4.36 \times 100 \]
which prints as 434.9999999999994 or so.

The reason for which this happens is called **gradual underflow** and it can result in loss of precision. Java implements the IEEE 754 standard which accepts this behaviour because overall it makes things more predictable for a certain class of numerical algorithms.

For us in this class (A201/A597, that is) this actually won’t be an issue. However if you do want infinite precision (and no range limits) you should use the classes `BigInteger` and `BigDecimal` in `java.math` as explained later in the notes. The roundoff error described above would not happen if we were to switch to $\frac{1}{10}$’s and use integers. Essentially we would be doing the roundoff correction ourselves then, as is also explained later in the notes.

5. For positive numbers casting to an `int` is equivalent to taking the `Math.floor()` while for negative numbers casting to an `int` becomes in essence `Math.ceil()`. Both functions coincide with `Math.round()` in only half of the cases (and always the other half).

6. Find at least five **syntax** errors in the following program.

```java
public class WarmUpSix
{
    public static void main(String[] args)
    {
        System.out.print("This program adds two numbers. ",
            " and ; are missing as indicated: ~ ~
        x = 5; // x needs to be declared first
        int y = 3.5; // fractional part requires double or float
        System.out.print("The sum of " + x + " and " + y + " is: ");
        System.out.println(x + y) // missing ;
    }
}
```

7. Find at least three **logic** errors in the following program.

```java
public class WarmUpSeven
{
    public static void main(String[] args)
    {
        ConsoleReader console = new ConsoleReader(System.in);
        int total = 1; // should be 0
        System.out.println("Please enter a number: ");
        int x1 = Integer.parseInt(console.readLine());
        total = total + x1;
        System.out.println("Please enter another number: ");
        int x2 = Integer.parseInt(console.readLine());
        total = total + x1; // should be x2
        double average = total / 2;
        // better divide by 2.0 to not lose 0.5
        System.out.println("The average of two numbers is " + average);    
    }
}
```

One can easily guess what the intended purpose of the program was.

8. There are four values of three types: an integer, a floating-point number (both are numbers, but of different types), and two strings. You can do arithmetic only with the numbers.
9. The first one computes the number 4, the second one the string "22".

10. The problem is to be aware of the initial value in a variable. If you initialize with zero you may get into trouble if you happen to use the value in that variable in a division at the denominator. But that would be about the only case when you could get into trouble.

11. Here are the three answers:

   (a) Starting from the number we make it into a string which has as characters the digits of the number and then we convert that to an integer thus getting back the number we originally started from.

   (b) Starting from a string that presumably represents a number we turn that into the number that it represents. We then make it back into a string by adding the empty string in front of it in a concatenation operation.

   (c) Essentially we create a second string from the first one, and this second string has all characters of the first string, and although a different object it would print the same.

12. Concatenating the number to the empty string is my favorite way of doing it. There is another way, using the toString methods. Perhaps this second method is more explicit, but why look up the name of yet another method when we can convert it on the spot using the empty string. In the end, use what you works best for you.

13. We could obtain them as substrings of length one that start at position 0 and (length - 1) respectively. Removing can only be done in an indirect way, as illustrated in class.

14. You either use logarithms extensively, or turn the number to a string and take the first and the last character. Here's the formula for the first and the last digit in the number (any number) but who can verify that this formula, especially the second one, is right (as it is):

   ```
   int lastDigit = n % 10;
   int firstDigit = n / (int)( Math.pow( 10,
       (int)( Math.log(n) / Math.log(10))
   )
   );
   ```

15. Variables declared as final accept only one initialization,

   • at the time the value is declared, or

   • later,

   but only once.

16. Here are the answers:

   (a) In every subexpression there is one term that is a double (either x or y) so all arithmetic will be floating-point.

   (b) This is (4 + 2) so the result is 6.

   (c) Same as 5 * x as 4 / 5 is 0.

   (d) This is the square root of 2.

   (e) First we round to 3 (as a long) then we make that an int.
(f) Boils down to 3 + (-1) basically. Note that the closest integer to -1.51 is -2 but the closest to -1.5 is -1. So this method will always round up.

(g) Two Strings.

(h) A String and a number.

(i) Same as 1 - n in this particular case.

(j) Two characters, the second and the third.

(k) Two numbers, each one being 6 (as the lengths of the two Strings).

Here's what you can do to obtain the answers:

```java
frilled.cs.indiana.edu%cat Sixteen.java
public class Sixteen {
    public static void main(String[] args) {
        double x = 2.5;
        double y = -1.5;
        int m = 18;
        int n = 4;
        String s = "Hello";
        String t = "World";
        System.out.println(x + n * y - (x + n) * y);
        System.out.println(m / n + m % n);
        System.out.println(5 * x - n / 5);
        System.out.println(Math.sqrt(Math.sqrt(n)));
        System.out.println(((int)Math.round(x)));
        System.out.println(((int)Math.round(x) + (int)Math.round(y)));
        System.out.println(s + t);
        System.out.println(s + n);
        System.out.println(1 - (1 - (1 - (1 - (1 - n)))));
        System.out.println(s.substring(1, 3));
        System.out.println(s.length() + t.length());
    }
}
frilled.cs.indiana.edu%javac Sixteen.java
frilled.cs.indiana.edu%java Sixteen
6.25
6
12.5
1.4142135623730951
3
2
HelloWorld
Hello\n
Hope you enjoyed this set of warmups, let me know if you have any questions.
Problem Set (I)

This is the first set of problems—part of Lab Notes Two.

Try to solve these problems to practice some of the things you have learned this week. In the examples that follow, your program’s answers are always in **boxes** to distinguish them from what you would type as a user. Remember: the resulting programs are elementary, and the problems are interesting.

1. Write a program that displays the squares, cubes, and fourth powers of the numbers 1-5.

   Here’s a sample run of such a program:

```
frilled.cs.indiana.edu%java One
First five powers of 1: 1 1 1 1 1
First five powers of 2: 2 4 8 16 32
First five powers of 3: 3 9 27 81 243
First five powers of 4: 4 16 64 256 1024
First five powers of 5: 5 25 125 625 3125
frilled.cs.indiana.edu%
```

2. Write a program that prompts the user for two integers and then prints

   - The sum
   - The difference
   - The product
   - The average
   - The distance (absolute value of the difference)
   - The maximum (the larger of the two)
   - The minimum (the smaller of the two)

   Here’s a sample run with your program:

```
frilled.cs.indiana.edu%java Two
Please enter your first integer number, then press Enter.
Enter two integers: 6 3
Difference: 3
Distance: 3
Average: 4.5
Max: 6
Min: 3
```
3. Write a program that
   • prompts the user for a measurement in meters
   • and then converts it into miles, feet and inches.

Here's a sample run with your program:

frilled.cs.indiana.edu%java Three
Please enter the measurement in meters: 100
Your original measurement of 100.0 meters has been converted.
   0 miles,
   328 feet, and
   1.0078719999988889 inches.
frilled.cs.indiana.edu%java Three
Please enter the measurement in meters: 16000
Your original measurement of 16000.0 meters has been converted.
   9 miles,
   4983 feet, and
   7.1495919999966445 inches.
frilled.cs.indiana.edu%

4. Write a program that prompts the user for a radius and then prints
   • the area and circumference of the circle with that radius
   • the volume and surface area of the sphere with that radius

Here's a sample run with your program:

whitetip.cs.indiana.edu%java Four
Please enter value for radius then hit enter.
100
Thank you. The radius is 100.0
Here are the computed values.
Area of the circle: 31415.926535897932
Circumference: 628.3185307179587
Area of a sphere: 125663.70614359173
Volume of sphere: 41887.90204786391
whitetip.cs.indiana.edu%java Four
Please enter value for radius then hit enter.
1
Thank you. The radius is 1.0
Here are the computed values.
Area of the circle: 3.141592653589793
Circumference: 6.283185307179586
Area of a sphere: 12.566370614359172
Volume of sphere: 4.1887902047863906
whitetip.cs.indiana.edu%

5. Write a program that asks the user for the lengths of the sides of a rectangle. Then print
   • the area and perimeter of the rectangle
   • the length of the diagonal (use the Pythagorean theorem)

Here's a sample run with your program:

whitetip.cs.indiana.edu%java Five
Please enter the value for the first side.
3
Thanks. Side one is 3.0
Please enter the value for the second side.
4
Thanks. Side two is 4.0
Area is: 12.0
Perimeter is: 14.0
Diagonal is: 5.0
whitetip.cs.indiana.edu%java Five
Please enter the value for the first side.
1
Thanks. Side one is 1.0
Please enter the value for the second side.
1
Thanks. Side two is 1.0
Area is: 1.0
Perimeter is: 4.0
Diagonal is: 1.4142135623730951
whitetip.cs.indiana.edu%

6. (Giving change) Implement a program that directs a cashier how to give change.
The program has two inputs:
   • the amount due and
• the amount received from the customer.

Compute the difference, and compute the

• dollars,
• quarters,
• dimes,
• nickels, and
• pennies

that the customer should receive in return.  

Hint: First transform the difference into an integer balance, 
denominated in pennies. Then compute the whole dollar amount. Subtract it from the balance. Compute 
the number of quarters needed (use %). Repeat for dimes and nickels. Display the remaining pennies.

Here’s a sample run with your program:

```
frilled.cs.indiana.edu%java Six
Type the amount due then press enter.
3.72
Type the amount received then press enter.
5
Give 1.28 in change as follows:
   5 quarters
   0 dimes
   0 nickels
   3 cents
frilled.cs.indiana.edu%java Six
Type the amount due then press enter.
0.08
Type the amount received then press enter.
0.5
Give 0.42 in change as follows:
   1 quarters
   1 dimes
   1 nickels
   2 cents
frilled.cs.indiana.edu%
```

7. Write a program that asks the user to input

• the number of gallons
• the fuel efficiency
• the price

Then prints how far the car can go with the gas in the tank and print the cost per 100 miles.

Here’s a sample run with your program:
frilled.cs.indiana.edu%java Seven
Please enter the number of gallons then press enter.
32
Please enter the fuel efficiency (miles/gallon) then press enter.
16
Please enter the price per gallon, then press enter.
1.54
With the gas in the tank you can go 512.0 miles,
at a cost of 9.625 per 100 miles.
frilled.cs.indiana.edu%java Seven
Please enter the number of gallons then press enter.
2.8
Please enter the fuel efficiency (miles/gallon) then press enter.
18.5
Please enter the price per gallon, then press enter.
1.48
With the gas in the tank you can go 51.8 miles,
at a cost of 8.0 per 100 miles.
frilled.cs.indiana.edu%

8. Write a program that

- reads a number greater than or equal to 1000 from the user and
- prints it out with a comma separating the thousands

Here's a sample run with your program:

frilled.cs.indiana.edu%java Eight
Please enter an integer >= 1000: 2000
2,000
frilled.cs.indiana.edu%java Eight
Please enter an integer >= 1000: 2000000
2000,000
frilled.cs.indiana.edu%java Eight
Please enter an integer >= 1000: 2000000000
2000000,000
frilled.cs.indiana.edu%java Eight
Please enter an integer >= 1000: -3000
-3,000
frilled.cs.indiana.edu%java Eight
Please enter an integer >= 1000: 20
Exception in thread "main" [...]

9. Write a program that reads a number greater than or equal to 1000 from the user, where the user enters a comma in the input. Then print the number without a comma. (Hint: Read the input as a string. Measure the length of the string. Suppose it contains \( n \) characters. Then extract substrings consisting of the first \( n - 4 \) characters and the last three characters.) Here's a sample run with your program:
frilled.cs.indiana.edu%java Nine
Please enter an integer between 1,000 and 999,999: 123,456

frilled.cs.indiana.edu%java Nine
Please enter an integer between 1,000 and 999,999: 1,000

frilled.cs.indiana.edu%java Nine
Please enter an integer between 1,000 and 999,999: 1000000000

frilled.cs.indiana.edu%

10. *(Printing a grid)* Write a program that prints the following grid to play tic-tac-toe.

```
+----+----+----+
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+----+----+----+
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+----+----+----+
```

Of course, you could simply write seven statements of the form

```java
System.out.println("+----+----+----");
```

You should do it a smarter way, though. Define string variables to hold two kinds of patterns: a comb-shaped pattern and the bottom line. Print the comb three times and the bottom line once.

Here's a sample run with your program (the boxes are meant to help):

frilled.cs.indiana.edu%java Ten

```
+----+----+----+
|    |    |    |
+----+----+----+
|    |    |    |
+----+----+----+
|    |    |    |
+----+----+----+
```

11. Write a program that reads an integer and breaks it into a sequence of individual digits. For example the input 16384 is displayed as

```
1 6 3 8 4
```

You may assume that

- the input has no more than five digits and
- is not negative.

*Hint:* There are two ways of solving this problem. You can:
• use integer arithmetic and repeatedly divide by 10, or
• you can convert the number into a string and extract the digits from the string.

Here’s a sample run with your program:

```
frilled.cs.indiana.edu%java Eleven
Please enter a number between 0 and 99999: 123
1 2 3
```

```
frilled.cs.indiana.edu%java Eleven
Please enter a number between 0 and 99999: 0
0
```

```
frilled.cs.indiana.edu%java Eleven
Please enter a number between 0 and 99999: 83021
8 3 0 2 1
```

```
frilled.cs.indiana.edu%java Eleven
Please enter a number between 0 and 99999: 123456
2 3 4 5 6
```

```
frilled.cs.indiana.edu%
```

12. The following program prints the values of sine and cosine for 0 degrees, 30 degrees, 45 degrees, 60 degrees, and 90 degrees. Rewrite the program for greater clarity by factoring out common code.

```java
class Twelve {
    public static void main(String[] args) {
        System.out.println("0 degrees: ");
        System.out.println("30 degrees: ");
        System.out.println("45 degrees: ");
        System.out.println("60 degrees: ");
        System.out.println("90 degrees: ");
    }
}
```

13. Write a program that prints out a message “Hello, my name is Hal!” Then, on a new line, the program should print the message “What is your name?” Next the program should read the user’s name and print “Hello, user name. I am glad to meet you.” Then, on a new line, the program should print a message “What would you like me to do?” Then it is the user’s turn to type in an input. Finally the program should ignore the user input and and print the message “I am sorry, user name. I cannot do that.”

Here’s a sample run with your program:

```
frilled.cs.indiana.edu%java Thirteen
Hello, my name is Hal!
```
What is your name?
Dave
Hello, Dave. I am glad to meet you.

What would you like me to do?
Clean up my room, if you’d like.
I am sorry, Dave. I cannot do that.
frilled.cs.indiana.edu%java Thirteen
Hello, my name is Hal! What is your name?
Dave
Hello, Dave. I am glad to meet you.

What would you like me to do?
Get lost.
I am sorry, Dave. I cannot do that.
frilled.cs.indiana.edu%

14. This is somewhat tricky so there are lots of hints in the text.

You don’t know yet how to program decisions, but it turns out that there is a way to fake them using substring. Write a program that asks a user to input

- the number of gallons of gas in the tank
- the fuel efficiency in miles per gallon
- the distance the user wants to travel

Then print out

You will make it

or

You will not make it

The trick here is to subtract the desired distance from the number of miles the user can drive. Suppose that the number is x. Suppose further that you find a way of setting a value n to

- 1 if x >= 0, and to
- 0 if x < 0.

Then you can solve your problem:

String answer = " not "; // note the spaces before and after not System.out.println("You will" +
answer.substring(0, 5 - 4 * n) + // sometimes not!
"make it");

Hint: Note that x + |x| is

- 2x if x >= 0, and
- 0 if x < 0.
Then divide by \( x \), except that you need to worry about the possibility that \( x \) is zero so use:

```java
// compute x, the difference between what you can and what you want
double x = efficiency * gallons - distance;

// avoid division by zero with this, can you see how?
double epsilon = 0.00001;
// make n 1 if x >= 0 and 0 otherwise
long n = Math.round( ( (x + Math.abs(x)) * x + epsilon) / 
                     (2 * x * x + epsilon));

// why can’t n be int? what do I need to do if I want it to be an int?
// report the correct answer
System.out.println("You will" +
                    answer.substring(0, 5 - 4 * (int)n) +
                    "make it.");

// why do I need to convert n from the long it is to an int here?
```

Here’s a sample run with your program:

```
frilled.cs.indiana.edu%java Fourteen
   Please enter number of gallons then press enter.  
10
   Please enter fuel efficiency in miles per gallon.  
20
   Please enter distance in miles you want to cover.  
199
   You will make it.
frilled.cs.indiana.edu%java Fourteen
   Please enter number of gallons then press enter.  
5
   Please enter fuel efficiency in miles per gallon.  
20
   Please enter distance in miles you want to cover.  
101
   You will not make it.
frilled.cs.indiana.edu%
```

15. Write a program that reads two times in military format (0900, 1730) and prints the number of hours and minutes between the two times. Here is a sample run. User input is in boxes.

```
Please enter the first time: 0900
Please enter the second time: 1730
8 hours 30 minutes

Extra credit if you can deal with the case that the first time is later than the second time:

Please enter the first time: 1730
Please enter the second time: 0900
15 hours 30 minutes
```
Here's a another sample run with your program:

```java
frilled.cs.indiana.edu%java Fifteen
Please enter the first time: 0920
Please enter the second time: 1025
1 hours 5 minutes
frilled.cs.indiana.edu%java Fifteen
Please enter the first time: 1025
Please enter the second time: 0920
22 hours 55 minutes
frilled.cs.indiana.edu%
```

16. Run the following program, and explain the output you get.

```java
public class Sixteen
{
    public static void main(String[] args)
    {
        ConsoleReader console = new ConsoleReader(System.in);
        int total = 0;
        System.out.println("Please enter a positive number:");  
        int x1 = Integer.parseInt(console.readLine()); 
        System.out.println("total = " + total); 
        total = total + 1 / x1; 
        System.out.println("total = " + total); 
        System.out.println("Please enter a positive number:");  
        int x2 = Integer.parseInt(console.readLine()); 
        total = total + 1 / x2; 
        System.out.println("total = " + total); 
        total = total * x1 * x2 / 2; 
        System.out.println("total = " + total); 
        System.out.println("The average is " + total); 
    }
}
```

Note the *trace messages*, which are inserted to show the current contents of the *total* variable. Then fix up the program, run it with the trace messages in place to verify that it works correctly, and remove the trace messages. What’s the program calculating and how?

17. *Writing large letters.* A large letter H can be produced like this:

```
  * *
  * *
  *****
  * *
  * *
```

It can be declared as a string constant like this:

```java
public static final String LETTER_H = "*  \n*  \n*****\n*  \n*  \n*  \n*  ";
```

Do the same for the letters E, L, and O. Then write the message
Here's a sample run with your program:

```
frilled.cs.indiana.edu% java Seventeen
  *  *
  *  *
  *****
  *  *
  *  *

  *****
  *
  *****
  *
  *****
  *
  *

  *****
  ***
  *  *
  *  *

  ***
  frilled.cs.indiana.edu%
```

18. Write a program that transforms numbers 1, 2, 3, ..., 12 into the corresponding month names

January, February, March, ..., December

Hint: Make a very long string

"January, February, March, ..., April, ..., December"
in which you add spaces such that each month name occupies the same length.

Then use substring to extract the month you want.

Here's a sample run with your program:

```
frilled.cs.indiana.edu%java Eighteen
 Please enter a month number from 1 to 12.
2
 February
frilled.cs.indiana.edu%java Eighteen
 Please enter a month number from 1 to 12.
12
 December
frilled.cs.indiana.edu%java Eighteen
 Please enter a month number from 1 to 12.
1
 January
frilled.cs.indiana.edu%java Eighteen
 Please enter a month number from 1 to 12.
14
Exception in thread "main" [...] 
frilled.cs.indiana.edu%
```

19. Write a program that generates passwords. Use the random number generator Random in the java.util package to generate a random number as follows:

```
int r = new Random().nextInt(1000);
```

The program knows the name(s) and age of the user.

It multiplies the age by the random number. Then it concatenates the initials with the last four digits of the product to generate the password.

Here's a sample run with your program:

```
frilled.cs.indiana.edu%java Nineteen
 Your password is: WJC0068
frilled.cs.indiana.edu%java Nineteen
 Your password is: WJC9494
frilled.cs.indiana.edu%java Nineteen
 Your password is: WJC2042
frilled.cs.indiana.edu%
```

Hope you enjoyed the problems.

Be sure to check the solutions, and to look at (and think about) all the problems.
Solutions (I)

Sample problems. The solutions.

Here are examples of solutions, I skip the statement, I write only the sample solutions (directly the programs). Reading these solutions should be mandatory, since it can only be beneficial; the worst case scenario is that the solution actually clarifies what’s being asked in the problem.

1. **********************************************************************************************
   This is the solution to the first problem in the first problem set: it displays the first five powers of the first five numbers: 1, 2, 3, 4, 5. This is a bit of an overkill, but that’s only because the problem is so elementary. Still, you should notice them that the third, fourth, and fifth powers are being computed from previously calculated values.
**********************************************************************************************

    class One {
        public static void main(String[] args) {
            int number = 1;
            int square = number * number;
            int fourthPower = square * square;
            System.out.print("First five powers of " + number + ": ");
            System.out.println( number + " " +
                               square + " " +
                               number * square + " " +
                               fourthPower + " " +
                               number * fourthPower );

            // done with 1.
            number += 1;
            square = number * number;
            fourthPower = square * square;
            System.out.print("First five powers of " + number + ": ");
            System.out.println( number + " " +
                                square + " " +
                                number * square + " " +
                                fourthPower + " " +
                                number * fourthPower );

            // done with 2.
            number += 1;
square = number * number;
fourthPower = square * square;
System.out.print("First five powers of " + number + ": ");
System.out.println( number + " " +
square + " " +
number * square + " " +
fourthPower + " " +
number * fourthPower );

// done with 3.
number += 1;
square = number * number;
fourthPower = square * square;
System.out.print("First five powers of " + number + ": ");
System.out.println( number + " " +
square + " " +
number * square + " " +
fourthPower + " " +
number * fourthPower );

// done with 4
number += 1;
square = number * number;
fourthPower = square * square;
System.out.print("First five powers of " + number + ": ");
System.out.println( number + " " +
square + " " +
number * square + " " +
fourthPower + " " +
number * fourthPower );

// done with 5
// -- finished

2. //*******************************************************************************/
This is the solution to the second problem in the first problem set. Note that you need ConsoleReader, which is available from the second set of lab notes (Lab Two) off the class notes page or from the book. Once you get the Consolereader class off the web notes place it in a file of its own, with the name ConsoleReader.java in the same directory with Two.java and whose source code is written below, then compile Two.java.

*******************************************************************************/
class Two {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter your first integer number, ");
        System.out.println("then press Enter.");
        int n1 = console.readInt();
        System.out.print("Please enter your second integer number, ");
        System.out.println("then press Enter.");
        int n2 = console.readInt();
        int sum = n1 + n2;
System.out.println(n1 + " + " + n2 + " = " + sum);
int diff = n1 - n2;
System.out.println(n1 + " - " + n2 + " = " + diff);
int prod = n1 * n2;
System.out.println(n1 + " * " + n2 + " = " + prod);
double avg = sum / 2.0;
System.out.println("avg(" + n1 + ", " + n2 + ") = " + avg);
int dist = Math.abs(n1 - n2);
System.out.println("dist(" + n1 + ", " + n2 + ") = " + dist);
long max = Math.round(avg + dist / 2.0);
// neat trick: can you explain it?
System.out.println("max(" + n1 + ", " + n2 + ") = " + max);
long min = Math.round(avg - dist / 2.0);
// neat trick: can you explain it?
System.out.println("min(" + n1 + ", " + n2 + ") = " + min);
}

3. //*************************************************************************/
// This is the solution to the third problem on the first problem set. It
// also needs the ConsoleReader class which can be taken from lab notes 2
// (see problem 2 above for more details). This is not a hard problem the
// difficulty (if any) is *only* in the conversion formulas. I decided to
// use three named constants to avoid magic numbers and I express miles,
// and feet, as integers, while the number of inches is reported as a double.
//***************************************************************************/
class Three {
    public static void main(String[] args) {
        // create a console reader for user input, call it console
        ConsoleReader console = new ConsoleReader(System.in);
        // let the user know you’re ready to receive input
        System.out.println("Please enter the measurement in meters: ");
        // get the measurement in meters from the user
        double measurement = console.readDouble();
        // define three constants needed in the conversion
        final double METERS_PER_MILE = 1609;
        final double METERS_PER_FOOT = 0.3048;
        final double METERS_PER_INCH = 1/39.37;
        // first compute the number of miles
        int miles = (int) (measurement / METERS_PER_MILE);
        // take those miles away from the measurement
        // first make a copy of the original value of the measurement
        double original = measurement;
        // then take the miles away
        measurement -= miles * METERS_PER_MILE;
        // then compute the number of feet from the remaining distance
        int feet = (int) (measurement / METERS_PER_FOOT);
        // take the feet away from the measurement
        measurement -= feet * METERS_PER_FOOT;
        // then compute the number of inches in the remaining distance
    }
double inches = (measurement / METERS_PER_INCH);
// print title of report
System.out.print("Your original measurement of ");
System.out.print(original);
System.out.print(" meters has been converted.
");
// print conversion results
System.out.println(" "+miles + " miles.");
System.out.println(" "+feet + " feet, and");
System.out.println(" "+inches + " inches.");
}
}

4. /**********************************************************************************
 This is the solution to problem four in first problem set. Note
 that just as in problem 2 you need to have the ConsoleReader class
 in a file in the same directory before compiling this class
**********************************************************************************/
class Four {
 public static void main(String[] args) {
     // get a connection to the keyboard
     ConsoleReader console = new ConsoleReader(System.in);
     // let the user know you’re ready to receive the value for the radius
     System.out.print("Please enter value for radius then hit enter.");
     // get the value
     double radius = console.readDouble();
     // echo value to user
     System.out.print("\nThank you. The radius is " + radius);
     // apply formulas
     double areaCircle = Math.PI * radius * radius;
     double circumferenceCircle = 2 * Math.PI * radius;
     double areaSphere = 4 * Math.PI * radius * radius;
     double volumeSphere = 4 * Math.PI * radius * radius / 3;
     // as you can see I have an error here in the volume formula!
     // report the computed values
     System.out.print("\nHere are the computed values.");
     System.out.print("\nArea of the circle: "+areaCircle);
     System.out.print("\nCircumference: " + circumferenceCircle);
     System.out.print("\nArea of a sphere: " + areaSphere);
     System.out.print("\nVolume of sphere: " + volumeSphere);
 }
}

5. /**********************************************************************************
 This is the solution for the fifth problem in the first set. Like in
 problem 2 you need to have ConsoleReader.java in the same directory before
 compiling this class (Five.java). The problem gets the values from the user
 and then applies formulas and reports the computed values.
**********************************************************************************/
class Five {
 public static void main(String[] args) {
     // get a connection to the keyboard

ConsoleReader console = new ConsoleReader(System.in);
// greet the user, ask for first value
System.out.println("Please enter the value for the first side.");
double side1 = console.readDouble();
// echo value to user
System.out.println("Thanks. Side one is "+ side1);
// ask for second value
System.out.println("Please enter the value for the second side.");
double side2 = console.readDouble();
// echo second value
System.out.println("Thanks. Side two is "+ side1); // side2!!
// apply formulas, compute required quantities
double area = side1 * side2;
double perimeter = (side1 + side2) * 2;
double diagonal = Math.sqrt(side1 * side1 + side2 * side2);
// report all computed value
System.out.println("Area is: "+ area +
"\nPerimeter is: "+ perimeter +
"\nDiagonal is: "+ diagonal);
}

6. /*********************************************************************************/
This is the solution to problem six in first problem set. ConsoleReader
is needed, use it as described in problem 2. This is a typical problem in
which you carefully need to cover roundoff errors so use Math.round after
you transform the problem in whole units (cents).
*********************************************************************************/
class Six {
    public static void main(String[] args) {
        // get a connection to the keyboard
        ConsoleReader console = new ConsoleReader(System.in);
        // ask the user for the amount due
        System.out.println("Type the amount due then press enter.");
        // read it
double due = console.readDouble();
        // ask the user for the amount received
        System.out.println("Type the amount received then press enter.");
        // read it
double received = console.readDouble();
        // assume received is bigger than due and compute difference
double difference = (received - due);
        // you need to return this as change so make it a whole number
        // of cents regardless of how many decimals the user has entered
        int diff = (int)(Math.round(difference * 100));
        // tell the user what change you are processing
        System.out.println("Give " + diff / 100.00 + " in change as follows: ");
        // number of quarters; integer division
        int quarters = diff / 25;
        // report it
        System.out.println("" + quarters + " quarters");
// adjust the remaining change (modulo)
diff = diff % 25;
// compute the number of dimes
int dimes = diff / 10;
// report it
System.out.println("" + dimes + " dimes");
// adjust remaining cents (notice shortcut operator)
diff %= 10; // notice anything compared to the previous assignment?
// these are the cents
int cents = diff;
// report them too
System.out.println("" + cents + " cents");
}
}

7. **********************************************************************************************************************
This is the solution to problem seven in the first problem set.
Need ConsoleReader in the same directory before you can compile
it (see problem 2). Formulas used are straightforward, easy.
***********************************************************************************************************************/
class Seven {
    public static void main(String[] args) {
        // get a connection to the keyboard
        ConsoleReader console = new ConsoleReader(System.in);
        // greet the user, ask for number of gallons
        System.out.println("Please enter the number of gallons then press enter.");
        // read number of gallons
double gallons = console.readDouble();
        // ask for fuel efficiency
        System.out.println("Please enter the fuel efficiency " +
            "(miles/gallon) then press enter. ");
        // read user input
double efficiency = console.readDouble();
        // ask for price per gallon
        System.out.println("Please enter the price per gallon, then press enter.");
        // read price from user
double price = console.readDouble();
        // computer how far the user can go with that much gas
double howFar = efficiency * gallons;
        // compute how much that will cost
double totalCost = price * gallons;
        // divide this by howFar and multiply by 100 to get price per 100 miles
double pricePer100Miles = totalCost / howFar * 100;
        System.out.println("With the gas in the tank you can go " +
            howFar + " miles, \a cost of " +
            pricePer100Miles + " per 100 miles.");
    }
}

8. **********************************************************************************************************************
This is the solution to problem eight in the first set of problems. You
need ConsoleReader in the same directory before compiling, as detailed at problem 2. Essentially you read a number as string, and write it back as required, with a comma separating the thousands (the last three digits). As before, no error checking is done, user is assumed to be program-friendly.

public static void main(String[] args) {
    // get a connection to the keyboard
    ConsoleReader console = new ConsoleReader(System.in);
    // ask the user to enter a number
    System.out.print("Please enter an integer >= 1000: ");
    // read the number, assume the user types the right thing
    String number = console.readLine();
    // report the number with the comma separating the thousands
    System.out.println("// first write digits up to thousands
     number.substring(0, number.length() - 3) + "," + // then write the comma
    // and the remaining part (the thousands)
     number.substring(number.length()-3));
}

9./* This is the solution to the ninth problem in the first problem
   set. You need ConsoleReader in the same directory, as explained in
   problem 2, before you can compile this program. This program assumes
   that the user types in a number that contains a comma separating the
   thousands from the rest of the number.

public static void main(String[] args) {
    // get a connection to the keyboard
    ConsoleReader console = new ConsoleReader(System.in);
    // ask the user for input
    System.out.print("Please enter an integer between 1,000 and 999,999: ");
    // read user's input
    String number = console.readLine();
    // get the first part (no comma or thousands)
    String firstPart = number.substring(0, number.length() - 4);
    // get last three digits
    String lastThreeDigits = number.substring(number.length() - 3);
    // print the two without the comma
    System.out.println(firstPart + lastThreeDigits);
}

10. /* This is the solution to the tenth problem in the first problem set.
    Given the hint it’s straightforward. We define two patterns and we
    print them accordingly to obtain the desired output.

class Ten {
    public static void main(String[] args) {
        // first pattern, comb-shaped
        String comb = "+-------+
                     | | | | |
        +-------+
        // second pattern, bottom line
        String bottom = "+-------+
        // print the grid for the user
        System.out.println(comb + comb + comb + bottom);
    }
}

11. //******************************************************************************************
   This is the solution to problem eleven of the first problem set. You read from the user a string of digits and then make sure the resulting string is at least 5 characters long. Once you have that, just print the characters one by one separated by blank spaces. We just print the last five, and we know we have at least five characters in the string because we have taken care to pad it with a string of five blanks from the start.
   //***************************************************************************************/
class Eleven {
    public static void main(String[] args) {
        // open a connection with the keyboard
        ConsoleReader console = new ConsoleReader(System.in);
        // tell the user to enter a number
        System.out.println("Please enter a number between 0 and 99999: ");
        // read the number the user types
        String number = console.readLine();
        // pad the number with five spaces
        number = "    " + number; // thus string has at least 5 characters
        int i = number.length() - 1; // index of last char in string
        // print the last five digits of the number
        // note the order in which we print the chars and their indices
        System.out.print(number.substring(i-4, i-3) + " ");
        System.out.print(number.substring(i-3, i-2) + " ");
        System.out.print(number.substring(i-2, i-1) + " ");
        System.out.print(number.substring(i-1, i) + " ");
        System.out.println(number.substring(i) + " ");
    }
}

12. //***************************************************************************************/
   This is the solution to problem twelve in the first problem set. We first compute the constant for the conversion from degrees to radians and then we repeatedly use the value of this constant, stored as x.
   //***************************************************************************************/
class Twelve {
    public static void main(String[] args) {
        double x = Math.PI / 180; // degreesToRadians constant
        System.out.println(
            " 0 degrees: " + Math.sin( 0 ) + " " + Math.cos( 0 ));
    }
}
System.out.println
  ("30 degrees: " + Math.sin(30 * x) + " " + Math.cos(30 * x));
System.out.println
  ("45 degrees: " + Math.sin(45 * x) + " " + Math.cos(45 * x));
System.out.println
  ("60 degrees: " + Math.sin(60 * x) + " " + Math.cos(60 * x));
System.out.println
  ("90 degrees: " + Math.sin(90 * x) + " " + Math.cos(90 * x));
}

13. /*--------------------------------------------------------------------------------*/
   This is the solution to problem thirteen in the first problem set. You need ConsoleReader in the same directory, as explained in problem two, before you can compile and run this program.
   /*--------------------------------------------------------------------------------*/
   class Thirteen {
       public static void main(String[] args) {
           // open a keyboard connection
           ConsoleReader console = new ConsoleReader(System.in);
           // greet the user and ask for input
           System.out.println("Hello, my name is Hal!");
           System.out.println("What is your name?");
           // get user input store it in String variable called name
           String name = console.readLine();
           // echo the data to the user and ask for request
           System.out.println("Hello, " + name + ". I am glad to meet you.");
           System.out.println("What would you like me to do?");
           // get the request
           String request = console.readLine();
           // end the conversation politely
           System.out.println("I am sorry, " + name + ". I cannot do that.");
       }
   }

14. /*--------------------------------------------------------------------------------*/
   This is the solution to problem fourteen in the first problem set. You need ConsoleReader in the same directory, as explained in problem two, to compile and run this program. Note that the hint in the text is almost all you need, minus one trick that makes sure that we can never divide by zero and also produces an n of 1 when x is zero (avoiding division by zero too).
   /*--------------------------------------------------------------------------------*/
   class Fourteen {
       public static void main(String[] args) {
           // get a connection to the keyboard
           ConsoleReader console = new ConsoleReader(System.in);
           // greet user, ask for input
           System.out.println("Please enter number of gallons then press enter.");
           // get number of gallons
           double gallons = console.readDouble();
           // ask for fuel efficiency
System.out.println("Please enter fuel efficiency in miles per gallon.");
// get it from user
double efficiency = console.readDouble();
// ask for desired distance
System.out.println("Please enter distance in miles you want to cover.");
// get distance
double distance = console.readDouble();
// prepare the answer
String answer = " not ";
// compute x, the difference between what you can and what you want
double x = efficiency * gallons - distance;
// avoid division by zero with this, can you see how?
double epsilon = 0.000001;
// make n 1 if x >= 0 and 0 otherwise
long n = Math.round( ( (x + Math.abs(x)) * x + epsilon) / 
( 2 * x * x + epsilon) 
);
// why can’t n be int? what do I need to do if I want it to be an int?
// report the correct answer
System.out.println("You will" + 
answer.substring(0, 5 - 4 * (int)n) + 
"make it.");
// why do I need to convert n from the long it is to an int here?
}


Worthwhile observation made by a student one semester:
The solution you have just seen is OK. The epsilon variable is needed above so we don’t divide by 0 (zero). But when x is negative and much smaller than epsilon the result is incorrect. What do we do? The trick is to think of the problem in physical terms: if you’re really using a car and if in your calculations you use an epsilon that is small enough, e.g. about 0.1ln (or smaller, like 0.001 of the car’s length) then

(a) for negative x above this small value the formula says: "You will make it!"
(b) and for all practical purposes that’s correct, since you can walk that distance
(c) but we need to be aware of the approximation, we definitely need to!
(d) Other than this “negative micron” case, the formula is perfect.

So that’s it. But does anyone have a formula that covers even this case with accuracy?

15. /* ******************************************
 This is the solution to problem fifteen in the first problem set.
 You need to have ConsoleReader in a file in the same directory before
 you can compile and run this. This problem adds 24 hours to the hours
 difference to avoid negative values and takes the remainder with 24
 to avoid values bigger than 24 (number of hours in a day). It also
 transforms the hours into minutes to absorb possible negative values
 from the difference between given minutes.
 ***********************************************/
class Fifteen {
    public static void main(String[] args) {

// get a connection to the keyboard
ConsoleReader console = new ConsoleReader(System.in);
// greet the user, ask for time
System.out.print("Please enter the first time: ");
// get the first time as a string (in military format)
String first = console.readLine();
// compute the number of hours from first two characters
int hours1 = Integer.parseInt(first.substring(0, 2));
// compute minutes from next two characters
int minutes1 = Integer.parseInt(first.substring(2, 4));
// ask for second time
System.out.print("Please enter the second time: ");
// get it
String second = console.readLine();
// compute hours, same as before
int hours2 = Integer.parseInt(second.substring(0, 2));
// compute minutes
int minutes2 = Integer.parseInt(second.substring(2, 4));
// compute the difference between hours, add 24 then divide
// by 24 and take the remainder to express the hour difference
int difHours = (hours2 + 24 - hours1) % 24;
// make these hours into minutes
int hrsToMins = difHours * 60;
// compute straight difference between given minutes, could be < 0
int difMins = minutes2 - minutes1;
// add this to total count of minutes
hrsToMins += difMins;
// report total count of minutes in hours and minutes
System.out.println(
    hrsToMins / 60 + " hours " + hrsToMins % 60 + " minutes" );
}

16. //get a connection to the keyboard
ConsoleReader console = new ConsoleReader(System.in);
// greet the user, ask for time
System.out.print("Please enter the first time: ");
// get the first time as a string (in military format)
String first = console.readLine();
// compute the number of hours from first two characters
int hours1 = Integer.parseInt(first.substring(0, 2));
// compute minutes from next two characters
int minutes1 = Integer.parseInt(first.substring(2, 4));
// ask for second time
System.out.print("Please enter the second time: ");
// get it
String second = console.readLine();
// compute hours, same as before
int hours2 = Integer.parseInt(second.substring(0, 2));
// compute minutes
int minutes2 = Integer.parseInt(second.substring(2, 4));
// compute the difference between hours, add 24 then divide
// by 24 and take the remainder to express the hour difference
int difHours = (hours2 + 24 - hours1) % 24;
// make these hours into minutes
int hrsToMins = difHours * 60;
// compute straight difference between given minutes, could be < 0
int difMins = minutes2 - minutes1;
// add this to total count of minutes
hrsToMins += difMins;
// report total count of minutes in hours and minutes
System.out.println(
    hrsToMins / 60 + " hours " + hrsToMins % 60 + " minutes" );
}

16. //get a connection to the keyboard
ConsoleReader console = new ConsoleReader(System.in);
// greet the user, ask for time
System.out.print("Please enter the first time: ");
// get the first time as a string (in military format)
String first = console.readLine();
// compute the number of hours from first two characters
int hours1 = Integer.parseInt(first.substring(0, 2));
// compute minutes from next two characters
int minutes1 = Integer.parseInt(first.substring(2, 4));
// ask for second time
System.out.print("Please enter the second time: ");
// get it
String second = console.readLine();
// compute hours, same as before
int hours2 = Integer.parseInt(second.substring(0, 2));
// compute minutes
int minutes2 = Integer.parseInt(second.substring(2, 4));
// compute the difference between hours, add 24 then divide
// by 24 and take the remainder to express the hour difference
int difHours = (hours2 + 24 - hours1) % 24;
// make these hours into minutes
int hrsToMins = difHours * 60;
// compute straight difference between given minutes, could be < 0
int difMins = minutes2 - minutes1;
// add this to total count of minutes
hrsToMins += difMins;
// report total count of minutes in hours and minutes
System.out.println(
    hrsToMins / 60 + " hours " + hrsToMins % 60 + " minutes" );
}
17.  
This is the solution to problem seventeen in the first set of problems.  
The text gives a suggestion that is enough to solve the problem, although  
in this particular case you can’t do it directly as suggested. But reading  
the section on constants in the book you can come up with the following  
absolutely obvious solution:  

```
class Seventeen{  
    public static void main(String[] args) {
        // define the letters as strings
        final String LETTER_H = "*    *\n*   \n*\n******\n*    *\n*\n"
        final String LETTER_E = "*****\n*\n******\n*\n******\n*\n"
        final String LETTER_L = "   \n\n*\n******\n*\n***\n*\n"
        final String LETTER_O = " *** \n*\n*** \n*\n*** \n*\n"
        // print them one after another
        System.out.println(
            LETTER_H + LETTER_E + LETTER_L + LETTER_L + LETTER_O
        );
    }
}
```

18.  
Solution to problem eighteen in the first problem set. Use ConsoleReader  
from lab notes as explained in problem set 2. The trick here (as hinted in the  
text) is to transform a number for a month in the position in the string where  
the month name is starting, all names being made of the same length, and then  
concatenated together in one final string.  

```
class Eighteen{  
    public static void main(String[] args) {
        String monthNames = "January " +  
            "February " +  
            "March " +  
            "April " +  
            "May " +  
            "June  " +  
```
"July" +
"August" +
"September" + // longest
"October" +
"November" +
"December" +

// open a connection with the keyboard
ConsoleReader console = new ConsoleReader(System.in);
// greet the user, and ask for input
System.out.println("Please enter a month number from 1 to 12.");
// get month name
int month = console.readInt();
// report the name of the month
System.out.println(
    monthNames.substring("September ".length() * (month-1),
    "September ".length()*month));
    // formula uses the length of the longest name
}
}

19.полнительность

This is solution to problem nineteen in the first set. Following the hint the solution becomes straightforward. Note that you need to import the Random class from the java.util package. Honestly: a better way to get random numbers is w/ Math.random, but we stay within this framework for now.

import java.util.Random; // need to import this to work with random numbers

class Nineteen {
    public static void main(String[] args) {
        String firstName = "William";
        String middleName = "Jefferson";
        String lastName = "Clinton";
        // extract initials
        String initials =
            firstName.substring(0, 1) +
            middleName.substring(0, 1) +
            lastName.substring(0, 1);
        // append age
        int age = 54;
        int r = new Random().nextInt(1000);
        // multiply age by random number
        String product = (age * r) + "";
        // concatenate initials with last four digits of product
        String password = initials + product.substring(product.length() - 4);
        System.out.println("Your password is: " + password);
    }
}

This last problem was from the previous administration.
Homework One

A Simple Ballistic Calculator

The mathematician that helped you with the second lab assignment now has a new problem for you. You now need to write a program that asks for:

- the user’s name,
- the speed (in meters per second) with which a missile is fired from a cannon, and
- the angle at which it is fired,

as in the picture below.

Given the initial speed (velocity), and the angle, you are to print a report in the user’s name, about how far the projectile will go and how long it will take for it to land. The formulas the mathematician gives you are:

\[
t = \frac{2v_0 \sin \alpha}{g}
\]

for the time it takes to fly, and

\[
d = \frac{v_0^2 \sin 2\alpha}{g}
\]

for the distance (how far the projectile lands), where \( g = 9.81 \text{ m/s}^2 \) is a constant.
These formulas assume the projectile does not hit anything, and that there is no friction with the air, and the only influence is that of the gravitational field of the Earth (which of course is an idealized situation, but that’s OK for this problem).

The mathematician also gives some test cases to check your program, that should work like this.

frilled.cs.indiana.edu%java A1
Hello, and welcome to the ballistic calculator.
Please enter your name: [Dave]
Well, Dave what’s the initial velocity?
Enter it here, with decimals (m/s): [100]
Thanks. What is the angle you’re shooting at?
Express it as an integer between 0 and 90: [30]
Thanks. Here are the results:
Report for: Dave
  Initial Velocity: 100.0 (m/s)
  Angle: 30 (degrees)
  Time Elapsed: 10.19367991845056 (seconds)
  Distance at Landing: 882.798576742547 (meters)

frilled.cs.indiana.edu%java A1
Hello, and welcome to the ballistic calculator.
Please enter your name: [Dave]
Well, Dave what’s the initial velocity?
Enter it here, with decimals (m/s): [100]
Thanks. What is the angle you’re shooting at?
Express it as an integer between 0 and 90: [45]
Thanks. Here are the results:
Report for: Dave
  Initial Velocity: 100.0 (m/s)
  Angle: 45 (degrees)
  Time Elapsed: 14.416040391163046 (seconds)
  Distance at Landing: 1019.367991845056 (meters)

frilled.cs.indiana.edu%java A1
Hello, and welcome to the ballistic calculator.
Please enter your name: [Dave]
Well, Dave what’s the initial velocity?
Enter it here, with decimals (m/s): [100]
Thanks. What is the angle you’re shooting at?
Express it as an integer between 0 and 90: [60]
Thanks. Here are the results:
Report for: Dave
  Initial Velocity: 100.0 (m/s)
  Angle: 60 (degrees)
  Time Elapsed: 17.655971534850938 (seconds)
  Distance at Landing: 882.7985767425471 (meters)

Everything that the user is typing is in boxes.

(This problem suggested by Tigger, who also helped with the formulas).
# Syntax

*The structure of main*

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is a <em>number</em>?</td>
<td>That much we know.</td>
</tr>
<tr>
<td>Let’s say: an <em>integer</em> or a <em>floating point number</em>.</td>
<td>We know all about <em>integers</em> and <em>floating point numbers</em>, I’d say.</td>
</tr>
<tr>
<td>What is an <em>operator</em>?</td>
<td>You want a long answer or a short one?</td>
</tr>
<tr>
<td>Short.</td>
<td>*+, -, *, / are <em>operators</em>.</td>
</tr>
<tr>
<td>What is an <em>expression</em>?</td>
<td>A <em>number</em>, is a (very simple) expression. A <em>symbolic name</em> is also a (very simple, or <em>atomic</em>) expression.</td>
</tr>
<tr>
<td>What is an <em>expression</em> followed by an <em>operator</em> followed by another <em>expression</em>?</td>
<td>That would also be an expression, wouldn’t it?</td>
</tr>
<tr>
<td>Yes, indeed. What do we have so far?</td>
<td>We have this table:</td>
</tr>
<tr>
<td></td>
<td><img src="table.png" alt="Table" /></td>
</tr>
<tr>
<td>Do we need the parentheses?</td>
<td>Not really, but we want to emphasize the structure.</td>
</tr>
</tbody>
</table>
| Correct. Now, is 

\[(3 + 4) * a\]

an *expression*? | It is. Do you want to show why with a diagram? |
I sure do. Very well, there you go:

![Recursive expression diagram]

That’s interesting that we can define a concept (such as expression) in terms of itself. This is called a recursive definition. An important part of a recursive definition is specifying a set of fixed points,

...without which defining something in terms of itself could go on for ever. Exactly. Thank goodness for numbers and variable names without which our definition would be irreversibly circular.

What is an assignment statement? Its structure is as follows: variable name, followed by an equal sign, followed by an expression, and then by a semicolon.

```java
a = 3 + 5;
a = b;
r = new Rectangle(2, 2, 10, 10);
```

What are we trying to get at? The structure of all Java programs.

What is a declaration? It is a type, followed by a variable name and a semicolon...

```java
int m;
Rectangle r;
double x, y, z;
```

I see you can declare several variables at the same time.

Yes, that’s relatively obvious by now, I hope.
... and it can also be a type, followed by an assignment statement (as defined above). You should be building a few examples following these definitions, or take some statements and analyze them the way we analyzed the simple expression a few lines above.

```java
int m = 25;
int n = 34, p;
Rectangle r = new Rectangle (5, 10, 8, 16);
```

OK, I see what you’re getting at, let’s move on. Are we really going to be extremely precise and cover all possible cases?

Not really. For that we’d have to sacrifice some of the intuitive structure of all these. But we’ll go far enough for you to get a good grasp of the general structure.

Are you ready for something really deep? Go ahead.

What is a function call? A function name followed by open parenthesis, followed by zero or more arguments separated by commas, followed by closed parenthesis.

```java
System.out.println();
System.out.println("Wow!");
Math.sqrt(2);
r.translate(3, 3);
```

What is an argument? I suppose any expression would work as one.

```java
System.out.println("tom" + "ate".substring(0, 2) + "a dog".charAt(3));
```

I’ve seen length() and substring() being invoked on Strings, but what’s charAt()? Exactly what you’re thinking that it might be.

Based on its name? Yes. And we’ll talk more about chars tomorrow.

OK, let’s summarize what we have so far: Literals appear in boxes.

<table>
<thead>
<tr>
<th>Term defined</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>number</td>
</tr>
<tr>
<td></td>
<td>variable</td>
</tr>
<tr>
<td></td>
<td>( expression operator expression )</td>
</tr>
<tr>
<td>funCall</td>
<td>funName</td>
</tr>
<tr>
<td></td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>funName ( ( arguments ) )</td>
</tr>
<tr>
<td>arguments</td>
<td>expression</td>
</tr>
<tr>
<td></td>
<td>expression, arguments</td>
</tr>
</tbody>
</table>
Any questions about function names? Not really, I suppose they’re basically of the “absolute path” kind, like `System.out.println`.

Good assumption. Are function calls expressions themselves?

I’m glad you asked. The answer is “yes” if the function returns a value. Then we are now dealing with an even bigger infinity of expressions.

Note though that not all functions return values. Yes, `println` from `System.out` does not return a value, but `sqrt` from class `Math` returns the square root of the argument, so it can be used in an expression.

Let’s update our table. We’ll put a star (*) next to remind ourselves that it’s a logic error to use a function call in an expression if the function does not return a value. I didn’t box the terminals this time around.

<table>
<thead>
<tr>
<th>Term defined</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expression</code></td>
<td><code>number</code> variable (expression operator expression) <code>funCall(Expr)</code></td>
</tr>
<tr>
<td><code>funCall</code></td>
<td><code>funName ( )</code> <code>funName ( arguments )</code></td>
</tr>
<tr>
<td><code>arguments</code></td>
<td><code>expression</code> <code>expression, arguments</code></td>
</tr>
</tbody>
</table>

The update is minimal, but I think I need to look at more examples before I become too dizzy.

You’ve already seen examples of expressions, as defined by the table above. This table is a table of syntactic categories described in terms of other syntactic categories, ...an enterprise aimed at describing the (grammatical structure of our simple) language.

Here are some more examples of expressions.

Take them apart, fit them in the table, don’t just accept them and then move on.

Don’t worry. Let’s take a look:

\[
\text{Math.sqrt}(a + \text{Math.sqrt}(b))
\]

This is easy:

\[
\sqrt{a + \sqrt{b}}
\]

\[
\text{Math.sqrt}(\text{Math.sqrt}(\text{Math.sqrt}(\text{Math.sqrt}(a))))
\]

Just as easy:

\[
\sqrt[4]{a}
\]
\begin{itemize}
\item \texttt{Math.pow(a, (1.0 / 16.0))}
\item \texttt{Math.sqrt(Math.pow(Math.sqrt(Math.pow(Math.sqrt(2), 2)), 2), 2)}
\end{itemize}

Let's rewrite this as follows:

\begin{itemize}
\item A bit messy, perhaps, but easy:
\end{itemize}

\begin{align*}
\text{Math.sqrt(} & \text{Math.pow(} \\
& \text{Math.sqrt(} \\
& \text{Math.pow(} \\
& \text{Math.sqrt(2), 2), 2), 2) \\
\text{)} \\
\text{)} \\
\text{)} \\
\text{)} \text{^2} \text{)} \\
\text{)} \text{)} \text{)} \text{)} \text{)} \text{)} \text{)} \text{)} \text{)} \text{)} \text{)}
\end{align*}

\begin{itemize}
\item (6 + (5 + (4 + (3 + (2 + 1)))))
\end{itemize}

What is a \textit{statement}?

A \textit{statement} is a

\begin{itemize}
\item declaration or
\item an assignment statement,
\end{itemize}

Or a \textit{function call}, as in:

\begin{itemize}
\item \texttt{System.out.println("Hello, world!");}
\end{itemize}

Exactly.

What is this?

\begin{itemize}
\item \texttt{public class Template}
\item \texttt{\{ public static void main(String[] args) \}
\item \texttt{\{ \<methodBody> \}}
\item \texttt{\}}</item>
\end{itemize}

(Notice how I boxed the non-terminal this time.)

It's the template we're using, and \texttt{methodBody} is composed of one or more \textit{statements}.

That is, \textit{declarations} and \textit{statements} in any order, with the following final table describing the entire structure of the language (so far).
<table>
<thead>
<tr>
<th>Term defined</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement</td>
<td>declaration</td>
</tr>
<tr>
<td></td>
<td>assignment</td>
</tr>
<tr>
<td></td>
<td>funCall</td>
</tr>
<tr>
<td>declaration</td>
<td>type variable ;</td>
</tr>
<tr>
<td></td>
<td>type variable = assignment ;</td>
</tr>
<tr>
<td>assignment</td>
<td>variable = expression ;</td>
</tr>
<tr>
<td>expression</td>
<td>number</td>
</tr>
<tr>
<td></td>
<td>variable</td>
</tr>
<tr>
<td></td>
<td>(expression operator expression)</td>
</tr>
<tr>
<td></td>
<td>funCall*</td>
</tr>
<tr>
<td>funCall</td>
<td>funName ( )</td>
</tr>
<tr>
<td>arguments</td>
<td>expression</td>
</tr>
<tr>
<td></td>
<td>expression, arguments</td>
</tr>
</tbody>
</table>

This table covers the syntax of the Java programs that we are going to be writing for a while. Note that not all syntactically correct programs are logically correct.

For example declaring a variable a twice is a semantic error, although having two declarations in a program is not a syntactic error...

...but if the variable is one and the same the semantic part of the compiler will signal that.

Suppose you compile and run the program below. What’s its output?

```java
class One {
    public static void main(String[] args) {
        int m = 2;
        System.out.println(m);
        int m = 3;
        System.out.println(m);
    }
}
```

The program won’t compile. One cannot declare a variable twice.

(One cannot declare the same variable twice.

All right, let’s move on. Have you heard of the latest late policy on assignments and such? There’s no new policy, everything is still the same: you need to turn everything on time. Try your best to meet the deadlines.

Just trying to get your attention. Sure. Let’s move on.

Fine. What is an int? It’s an integer number between \(-2^{31}\) (which is about 2 billion) and \((2^{31} - 1)\). If you need to refer to these boundaries in your program, use the constants `Integer.MIN_VALUE` and `Integer.MAX_VALUE`

...which are defined in a class called `Integer` like `Math.PI` is defined in the `Math` class.

Convention says: name your constants using all caps for the name of the constant.
<table>
<thead>
<tr>
<th>How do you define constants?</th>
<th>Mark them final when you declare them.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Here’s a program, what does it produce:</th>
<th>Overflow.</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class Test</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>public static void main(String[] args)</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
</tr>
<tr>
<td>int n = Integer.MAX_VALUE;</td>
<td></td>
</tr>
<tr>
<td>System.out.println(n);</td>
<td></td>
</tr>
<tr>
<td>n = n + 1;</td>
<td></td>
</tr>
<tr>
<td>System.out.println(n);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Have you run it?</th>
<th>Yes, it’s an eye-opening experience.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Why does it happen?</th>
<th>Representation is finite.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>What if we need bigger integers?</th>
<th>Use long.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>What’s a long?</th>
<th>It’s a type that allows for the representation of bigger integers. The range is now $-2^{63}$ (which is about -9 billion billions) to $(2^{63} - 1)$.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>What if we need bigger integers?</th>
<th>Then work with objects of class BigInteger.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>How do I do that?</th>
<th>We’ll see that in a minute. What is a floating-point number?</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>A double or a float.</th>
<th>float spans the range from -3.4E38 to 3.4E38. double is much wider: from -1.7E308 to 1.7E308 but both suffer from the same problem:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>...precision.</th>
<th>Yes—what’s the output of this program?</th>
</tr>
</thead>
</table>

| public class Test |                                           |
| {                |                                           |
|   public static void main(String[] args)|                                           |
|     {                                   |                                           |
|       double a = 30000000000000000000.0; |                                           |
|       System.out.println(a - (a - 0.5)); |                                           |
|     }                                  |                                           |
|}                                        |                                           |

<table>
<thead>
<tr>
<th>It should be 0.5 by algebra.</th>
<th>Yes, but is that what the program prints?</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Try it.</th>
<th>Also, try to initialize the double variable with a value that doesn’t contain the decimal point.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>In that case the compiler catches the overflow before it happens.</th>
<th>Overflow happens even for double type.</th>
</tr>
</thead>
</table>
The following program produces **Infinity**. You’re kidding me!

Try it:

```java
public class Test{
    public static void main(String[] args)
    {
        double a = 1.5E308;
        System.out.println(a * 10);
    }
}
```

For most of the programming projects in this book, the limited range and precision of `int` and `double` are acceptable. Just bear in mind that overflow or loss of precision occur.

Another kind of loss of precision occurs in what is known as a **roundoff error**.

In the processor hardware, numbers are represented in the binary number system, not in decimal. You get roundoff errors when binary digits are lost, they just may crop up at different places than you might expect.

Here’s an example:

```java
System.out.println(4.35 * 100);`n```

**What’s that?**

**Another eyeopener.**

What do we do? Keep a cool head. For example in this last case first round then `cast` to an `int`...

...if you want an `int`,... ...and especially if you want the right one.

How about Slightly off...

```java
System.out.println(Math.pow(Math.sqrt(2), 2));
```

A bit different, but related. Yes, and we’ll discuss how floating-point numbers should be tested for equality soon.

How do you use big numbers? If you want to compute with really large numbers, you can use **big number** objects.
K/]^K
X=#AzW\B*a oDN$@Co^DS T,I QCNVD}@o^DS+TOI @ P%TOg*D ¾V6. Í-%
Q WY ¾ 6. ­  L#  _ S+>'QCIOIUDI=#W¼TVg*D X !(7Ý# £
Q ADC¶



 .+  c

<QCS,dH

¶¶¶Vg*@cXDmCDN· S+@Cab<*B*TOQTO=#@WIlc=#TVgào*=#AÒW\B*a-oDN@Co^DS TOI
Q NOD¼a BS,göIU>?@cXDNTOgQ WæTOg*@IVD¼TOgQT=?WHm@C>?mCD¼<NV=?ab=¹TO=#mD
T~H<DI¶eG@Z<DNVP@CNOa@<DNOQ TV=?@CWI`~C@BW*DDYTO@zBIVD abDTVg^
@\YIIUBS,gZQI¶+¶¶

¶¶¶TOgQTo*=?AbW\B*a oDN@o^DS+TOIhgQ/mDC¶ì

`W*>?=#dD -% @N ":  ·o*=#A«W\B*a oDN,I gQ/mCDfDIVIVDWHTV='Q >?>?~
W@n>#=?ab=¹T,I@CWzTVg*D=#N UI =?{DQ WY<*NODS=?IV=?@CW[·

 " "A'77 7S4
 6  +='77 S7 4
 #: &:5='77 S7 4
 ",6! " R'7 7 7*4

Q WY

_DNVD³ I

¯¹°Ù½¯
c=¹TOglo*=#AnW\B*a-oDNOI¶

S#+ ¯X!(7Ú#,£07*½NM
 : $¢6
 : !6"3#,-%('*): + -%./21E:+ .,4
 ¾V6. Í-% .+¯Ê%:C°¾V6.,Í-%.+='6> I-H H H HH H HHHH H H H HHHH H HH H HH H H H H H HH H H H H H HH ,H >4VM
¾V6. Í-% .+±DÊ%:C°¾V6.,Í-%.+='6>6¼H H H HH H H HHH HH H H HHH H H H HH H HH HH HH H HH HH H H H H,> 4VM
¾V6. Í-% .+D"±Ê±(7T""A'T07Ú#: -:5A'? 44NM
)5,-#87T:Q7;+ -% %('T",4NM
O
O
ehg*=?I=?IhA@H@^YzD+^DN,S+='IVD$=?WZ@Co^DS TW*@CTOQTO=#@W[¶

½@CBoD+T¶ DC³ mCD IVDDWlNVDS TOQCW*AC>?DI·*IUTVNO=#W*AHI·Q WYlW*@cào*=?A
W\B*a oDN,I¶%DC³ NOD$AD+TVTV=?W*AbDmCDWloDTUTVDN¶

k\@ncgQTIVg*@CB>?Yzc DgQ/mCDqQCS,g*=?DmDYfo\~zW*@c`Ü

GTOg*=?W*d cXDhB*WY^DNOIUTOQCWY =?WHTVDACDN%QCWY}ø@QTO=#WA$<@C=?WHTW\B*as
oDNOI¶

«D³ NOD}Q o*>?D$TV@bcNO=¹TOD}Q NO=¹TOg*abD+TV='S$D+^<*NODIOIU=?@CWI =?WQ/mQ¶

«DzSQ WQ <<*NVDS+='QTOD-TOg*Dn=?as<@CNVTOQCWS+Db@CPhS@CababDWHTOI}QCWY
AC@\@^YzS@^Y^D>?Q/~@CB^T¶

«D}SQ WlY^D+´WD}Q WYZ=?W*=¹TO=?QC>#=?{DqmQ NO='Q o*>?DIQCWYzS@CWIUTOQCWHTOI¶

«Dd\W*@cá@ PTVg*D>?=#ab=#TOQ TV=?@CWI@CP

¶¶¶OQ WYTVgD@mCDNVø@cèQCWYZNV@B*WY^@CDNONV@NOITVgQ TNVDIUB>¹T¶

«D}SQ WZNVDQCYz<*NO@CACN,Q aÑ=?W*<*B*Tc=¹TOgZ\

«Dd\W*@cág*@cèTO@s<NV=?WHT<*NV@ACN,Q a:@CB*TV<*B^T¶

«DBWY^DN,IUTOQ WYfTOg*D}ITONVBS TVBNVD@CPQ

¶¶¶Vcg*='S,gáS+@WHTOQ =?WIzIT,QTVDabDWHTOI· IVBS,gáQIzY*DS+>'Q N,QTO=#@WI·
QCIOIU=?ACW*abDWT,I·*QCWYPB*WS TO=#@WSQ >?>'I¶

«D³ mD}Q >'IU@-TOQC>#dDYZQ o@CB^TD^<*NVDIVIV=?@CWI¶

«DgQ/mDqBIUDYfTOg*D

) +,-%.

S+>'QCIOI¶¶¶

-%

Q WY

T~\<DI·

:% : Ì "+
# :£"E"5

¶¶¶TV@nY^D´W*D}QCWYZabQCW*=?<*B*>'QTVDIUTVNO=#WAI¶

J Æ ~Ç/ÇJÈÉ ÊÊJËÌS5Í ÎJÏ-WÍ |ÐS{ÊÈ6JÐ?:Ï6| ÇÎ ÊJËÚ:Ñ:Ê%ÒÍ Ó#Ê[5Ð&|Î Ê*0È-{ÔÊJËÌS#Ê[ÇÆ~ Ê0Õ {×ÖØ ÇÆ,ÖST,Í ~ÇÆÚÙ
Y7 ~~Ç/Ç/ÇJÇJÈÈÉÉ ÊÊÊJÊJËË SÌSÌ 55ÍÍ ÎJÎJ-Ï-Ï WWÍÍ ||SÐSÐ {{ÊÊ6È6È JJ?Ð?Ð ::6Ï6Ï || ÇÇÎÎ ÊJÊJÚËÚË ::Ñ:Ñ:Ê%Ê%ÒÒÍÍ Ó#Ó#[Ê[Ê 55&Ð&Ð ||ÎÎ *Ê*Ê 00-È-È {Ô{ÔÊJÊJËË SÌSÌ ##[Ê[Ê ÆÇÆÇ ~~ Ê0ÊÈÕ *{×Ö#|YSÑ«Û 09|SÖ 9Ý{}ÜÆÇ 09ÙÜÍÍ ~~ÆÇ ÆÇÚÚÙ Ù

": 

·

¶


By the way can you draw a picture for me for the following situation:

```java
String a = "Hello!";
String b = a.toUpperCase();
```

I sure can:

Thanks. It is the same with `substring`, isn't it?

I know it is.

String objects are called *immutable* objects.

That's right: `toUpperCase` on a String works like ...or like `intersection` on a `Rectangle`. add on a big number...
Getting Things Done

Basic programs. Simulating decisions. Working with objects.

1. First a review. Of the problems presented for your practice last week I want to highlight a few, wherein a few stunts (which most clearly justify the problems’ difficulty) are worth pointing out. Here, first, is Two.java with formulas for max and min. (I hope you’ve looked at the program already).

```java
class Two {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter your first integer number, ");
        System.out.println("then press Enter.");
        int n1 = console.readInt();
        System.out.print("Please enter your second integer number, ");
        System.out.println("then press Enter.");
        int n2 = console.readInt();
        int sum = n1 + n2;
        System.out.println(n1 + " + " + n2 + " = " + sum);
        int diff = n1 - n2;
        System.out.println(n1 + " - " + n2 + " = " + diff);
        int prod = n1 * n2;
        System.out.println(n1 + " * " + n2 + " = " + prod);
        double avg = sum / 2.0;
        System.out.println("avg(" + n1 + ", " + n2 + ") = " + avg);
        int dist = Math.abs(n1 - n2);
        System.out.println("dist(" + n1 + ", " + n2 + ") = " + dist);
        long max = Math.round(avg + dist / 2.0);
        // neat trick; can you explain it?
        System.out.println("max(" + n1 + ", " + n2 + ") = " + max);
        long min = Math.round(avg - dist / 2.0);
        // neat trick; can you explain it?
        System.out.println("min(" + n1 + ", " + n2 + ") = " + min);
    }
}
```

Regarding this problem I have three questions:

1. Why are max and min declared as long?
2. Would an if statement simplify this program?

3. Can you compute `Math.abs(...)` without an if?

2. Three, Four, and Five are easy, mostly involving the application of a set of formulas, most of them well-known (and if you look closely I got two of them wrong, and marked them clearly as such). But have you managed to take a look at Six?

class Six {
    public static void main(String[] args) {
        get a connection to the keyboard
        ConsoleReader console = new ConsoleReader(System.in);
        // ask the user for the amount due
        System.out.println("Type the amount due then press enter.");
        // read it
        double due = console.readDouble();
        // ask the user for the amount received
        System.out.println("Type the amount received then press enter.");
        // read it
        double received = console.readDouble();
        // assume received is bigger than due and compute difference
        double difference = (received - due);
        // you need to return this as change so make it a whole number
        // of cents regardless of how many decimals the user has entered
        int diff = (int)(Math.round(difference * 100));
        // tell the user what change you are processing
        System.out.println("Give " + diff / 100.00 " in change as follows: ");
        // number of quarters; integer division
        int quarters = diff / 25;
        // report it
        System.out.println("" + quarters + " quarters");
        // adjust the remaining change (modulo)
        diff = diff % 25;
        // compute the number of dimes
        int dimes = diff / 10;
        // report it
        System.out.println("" + dimes + " dimes");
        // adjust remaining cents (notice shortcut operator)
        diff %= 10; // notice anything compared to the previous assignment?
        // these are the cents
        int cents = diff;
        // report them too
        System.out.println("" + cents + " cents");
    }
}

This problem uses integer arithmetic to transform a sum of money into a minimum number of coins (quarters, dimes, nickels, and cents) through a very interesting process. I have three questions with respect to this problem:

1. Do you understand how the transformation process goes?

2. Why do we use `Math.round` inside the program?
3. Is there anything in the previous set of notes of relevance to this problem?

3. **Seven** is at the level of your homework assignment and a good exam problem. **Eight** simply places a comma in a `String` (and in only one place,) while **Nine** does the opposite. Based on **Nine** and **Eight,** how would you write a program that reads a `String` of at least two characters and then swaps the first and the last characters in the `String` as the following examples try to illustrate:

```java
frilled.cs.indiana.edu%java Swap
Please type: pooh
After swapping pooh becomes: hoo p
frilled.cs.indiana.edu%java Swap
Please type: twelve
After swapping twelve becomes: elvewt
frilled.cs.indiana.edu%java Swap
Please type: donald
After swapping donald becomes: dnalo d
frilled.cs.indiana.edu%
```

Remember that Strings are immutable objects!

4. **Ten** is easy, although it aims at a certain kind of optimization. But **Eleven** is tricky:

```java
class Eleven {
    public static void main(String[] args) {
        // open a connection with the keyboard
        ConsoleReader console = new ConsoleReader(System.in);
        // tell the user to enter a number
        System.out.print("Please enter a number between 0 and 99999: ");
        // read the number the user types
        String number = console.readLine();
        // pad the number with five spaces
        number = " " + number; // thus string has at least 5 characters
        int i = number.length() - 1; // index of last char in string
        // print the last five digits of the number
        // note the order in which we print the chars and their indices
        System.out.print(number.substring(i-4, i-3) + " ");
        System.out.print(number.substring(i-3, i-2) + " ");
        System.out.print(number.substring(i-2, i-1) + " ");
        System.out.print(number.substring(i-1, i) + " ");
        System.out.println(number.substring(i) + " \n");
    }
}
```

Knowing how **Eleven** operates how would you write **Reverse** that reverses words of up to 6 characters in length, in a manner similar to the one illustrated below:

```java
frilled.cs.indiana.edu%java Reverse
Please enter a word of up to 6 characters.
Type your word here: potato
potato reversed is: otatop
```
frilled.cs.indiana.edu%java Reverse
Please enter a word of up to 6 characters.
Type your word here: apple
apple reversed is: elppa

frilled.cs.indiana.edu%java Reverse
Please enter a word of up to 6 characters.
Type your word here: kiwi
kiwi reversed is: iwik

frilled.cs.indiana.edu%java Reverse
Please enter a word of up to 6 characters.
Type your word here: one
one reversed is: eno

frilled.cs.indiana.edu%java Reverse
Please enter a word of up to 6 characters.
Type your word here: um
um reversed is: mu

frilled.cs.indiana.edu%java Reverse
Please enter a word of up to 6 characters.
Type your word here: u
u reversed is: u

frilled.cs.indiana.edu%java Reverse
Please enter a word of up to 6 characters.
Type your word here: reversed is:
frilled.cs.indiana.edu%

5. Twelve and Thirteen are different but routine. Fourteen however is downright mischievous. My only question is: why do I need the epsilon variable, at all, in Fourteen?

6. Fifteen is not only tricky, but also hides a subtle mistake. Here's where it fails:
frilled.cs.indiana.edu%java Fifteen
Please enter the first time: 1000
Please enter the second time: 0959
23 hours 59 minutes
frilled.cs.indiana.edu%java Fifteen
Please enter the first time: 10 01
Please enter the second time: 10 00
0 hours -1 minutes
frilled.cs.indiana.edu%java Fifteen
Please enter the first time: 1101
Please enter the second time: 1059
23 hours 58 minutes
frilled.cs.indiana.edu%java Fifteen
Please enter the first time: 04 45
Please enter the second time: 04 40
0 hours -5 minutes
frilled.cs.indiana.edu%

Can you see where the mistake is in the code?
To treat all cases correctly here’s how we should solve **Fifteen**:

1. Read String **time1**, and **time2**.
2. For **time1** split it into:
   
   ```java
   String hour1 = time.substring(0, 2),
   mins1 = time.substring(2);
   ```
3. Calculate **time1** in minutes:
   
   ```java
   int val1 = hour1 * 60 + mins1;
   ```
4. Similarly calculate **val2**.
5. Then calculate the difference in minutes between the two:
   
   ```java
   int diff = (val2 + 24 * 60 - val1) % (24 * 60);
   ```
6. Transform **diff** in hours and minutes:
   
   ```java
   int resH = diff / 60,
   resM = diff % 60;
   ```

Write this program (call it [MilitaryTime.java](http://java.sun.com/products/jdk/1.2/docs/api/index.html)) and verify that it works correctly.

This just shows how important it is to prove the correctness of any program.

7. Problems **Sixteen**, **Seventeen**, and **Nineteen** are again routine, while **Eighteen** is featured in Lecture Notes Six. However, **Nineteen** produces random numbers, and I’d like to use this opportunity to point out a somewhat easier way of producing random numbers than the one suggested thus far.

Here’s a way of producing random integers in the interval \([a, b]\):

```java
(int)(Math.random() * (b - a) + a)
```

If [Math.random()](http://java.sun.com/products/jdk/1.2/docs/api/index.html) returns a random number in \([0, 1]\), can you see how this works?

I trust that you agree this batch of problems was not easy, and quite instructive!

And now the actual new part of the lab!

**Goals for this lab:**

- Use objects, and become thoroughly familiar with object notation.
- Become aware of the various packages and classes therein, of the standard Java API
- See how simple classes are defined and experiment with them.
- Get started reading lecture notes 7-8 (in advance).

---

13 Predicates, in the next few pages
16 Classes—7, Diagrams—8
Examples below are from the notes, some (simple ones) are made up.

8. Lecture Notes\textsuperscript{17} Three contain an example like this:

```java
public class One
{
    public static void main(String[] args)
    {
        Rectangle a = new Rectangle(5, 10, 20, 30);
        System.out.println(a);
        a.translate(15, 25);
        System.out.println(a);
    }
}
```

Put this in a file called One.java, compile and run.

As the notes say, your program won’t compile. What’s missing?

Fix the program, compile and run it.

Your output should look like this:

```
frilled.cs.indiana.edu%java One
java.awt.Rectangle[x=5,y=10,width=20,height=30]
java.awt.Rectangle[x=20,y=35,width=20,height=30]
frilled.cs.indiana.edu%
```

Did you obtain the same output?

9. (Also from Lecture Notes Three) Create a program:

```java
import java.awt.Rectangle;
public class Two
{
    public static void main(String[] args)
    {
        Rectangle a = new Rectangle(5, 10, 20, 30);
        Rectangle b = a;
        a.translate(10, 10);
        b.translate(10, 10);
        System.out.println(a);
    }
}
```

Place it into a file called Two.java, compile and run it.

I obtain the following output:

```
frilled.cs.indiana.edu%java Two
java.awt.Rectangle[x=25,y=20,width=20,height=30]
frilled.cs.indiana.edu%
```

What output do you obtain?

Now suppose that instead of printing \texttt{a} at the end we print \texttt{b}.

\textsuperscript{17}Simple Programs
What would the output of the program be then, and why?

10. Check the documentation for class Rectangle\(^{18}\) and look up the `intersection` method.

The `intersection` method computes the intersection of two rectangles – that is, the rectangle that is formed by two overlapping rectangles:

```
You call this method as follows:

    Rectangle r3 = r1.intersection(r2);
```

Write a program that

- constructs two rectangle objects,
- prints them, and then
- prints their intersection.

What happens when the two rectangles do not overlap?

When you are done check your solution against the program below:

```java
/* Proposed solution for problem 10 above. Note that we test
    two cases, one in which the two rectangles overlap and one
    in which they don't. Read the documentation about rectangles
    with negative values for either width or height (or both)
    standing for empty sets. Note that a point is different
    from an empty set, even though it has no width or height
    it has a location. See the third test for that.
    Bottom line is: is this correct or not?
    Why, or why not? */
import java.awt.Rectangle;
public class Three {
    public static void main(String[] args) {
        Rectangle a = new Rectangle(0, 0, 10, 10);
        Rectangle b = new Rectangle(5, 5, 10, 10);
        Rectangle c = a.intersection(b);
        System.out.println(a);
        System.out.println("intersected with");
        System.out.println(b);
        System.out.println("produces");
        System.out.println(c);
    }
}
```

\(^{18}\)http://java.sun.com/products/jdk/1.2/docs/api/java/awt/Rectangle.html
System.out.println("---------------------");
Rectangle d = new Rectangle(10, 10, 10, 10);
Rectangle e = new Rectangle(50, 50, 50, 50);
Rectangle f = d.intersection(e);
System.out.println(d);
System.out.println("intersected with");
System.out.println(e);
System.out.println("produces");
System.out.println(f);
System.out.println("---------------------");
Rectangle g = new Rectangle(0, 0, 10, 10);
Rectangle h = new Rectangle(-10, -10, 10, 10);
Rectangle i = g.intersection(h);
System.out.println(g);
System.out.println("intersected with");
System.out.println(h);
System.out.println("produces");
System.out.println(i);
System.out.println("---------------------");
}
}

What is the output that the program produces?

Use the program above or your solution to problem 10 to calculate the intersection of

1. a square located at (0, 0) with a side of 10, and
2. a square located at (-5, -5) with a side of 10.

Any square is also a rectangle, so the two squares above can be created as follows

1. new Rectangle(0, 0, 10, 10), and
2. new Rectangle(-5, -5, 10, 10)

11. BigIntegers are like Rectangles, but they represent numbers.

Lecture Notes Five\(^{19}\) contain the following example:

```java
import java.math.*;
public class Four
{
    public static void main(String[] args)
    {
        BigInteger a = new BigInteger("10000000000000000000000000000000000000000000000000000000000000000");
        BigInteger b = new BigInteger("20000000000000000000000000000000000000000000000000000000000000000");
        BigInteger c = new BigInteger("30000000000000000000000000000000000000000000000000000000000000000");
        BigInteger d = a.add(b.multiply(c));
        System.out.println(d);
    }
}
```

Compile and run this program.

\(^{19}\)Syntax, just presented.
BigIntegers\textsuperscript{20} provide immutable arbitrary-precision arithmetic. That means numbers without limits. As big as you want. You will of course remember (from the very same Lecture Notes Five) that regular arithmetic has its limitation in Java. So one could use BigIntegers to avoid those limitations.

We will however use them to become familiar with object notation.

And to keep things simple let’s use small numbers.

Here’s how we calculate

\[ 1 + 2 \times 3 \]

The calculation is identical to the example presented before, only fewer 0’s are present.

```java
import java.math. *
public class Four
{
public static void main(String[] args)
{
BigInteger a = new BigInteger("1")
BigInteger b = new BigInteger("2")
BigInteger c = new BigInteger("3")
BigInteger d = a.add(b.multiply(c));
System.out.println(d);
}
}
```

Can you see how that’s done?

Practice some more by using BigIntegers to calculate:

- 1 + 2
- 2 × 3
- (1 + (2 + (3 + 4)))
- 1 + 2 + 3 + 4
- 2 × 3 + 4 × 5

12. What is the output of this program?

```java
public class Five {
   public static void main(String[] args) {
      String greeting = "Hello, Bill!";
      greeting.toLowerCase();
      System.out.println(greeting);
   }
}
```

Please try to deduce the answer first.

Then run the program to double check your answer.

\textsuperscript{20}http://java.sun.com/products/jdk/1.2/docs/api/java/math/BigInteger.html
13. Modify just one line in the program above to obtain the following output:

    HELLO, BILL!

You can only modify one line only.

14. The following program does not compile. Why?

```java
public class Seven {
    public static void main(String[] args) {
        int one = 2;
        int two = 3;
        int two = two + one;
        System.out.println(two);
    }
}
```

15. The following program does not compile. Why?

```java
public class Eight {
    public static void main(String[] args) {
        int n;
        int m;
        n = 2;
        m = m + 2;
        System.out.println(m);
    }
}
```

16. Do you remember the `Penguin()` from lab assignment one? (I hope you do). Lecture Notes Seven define a class `BankAccount`. A `BankAccount` is nothing but a `Penguin` (or a `Tigger`, for that matter). It has a number of things that it does best. (A `ConsoleReader` also has a number of things that it does best, and so do `Rectangles`, `Strings`, and even big numbers. We only need to know how to ask for them).

What do `BankAccounts` do _best_?

Well, we agree that a `BankAccount` must at least:

- allow for the _deposit_ of money
- allow for the _withdrawal_ of money
- provide the balance when asked for it: _getBalance_

Any `BankAccount` remembers its _balance_, and keeps track of it.

Here's a Java version of such a mechanism:

```java
public class BankAccount {
    public void deposit(double amount) {
        balance = balance + amount;
    }
}
```
public void withdraw(double amount)  
{ balance = balance - amount;  
}

public double getBalance()  
{ return balance;  
}

private double balance;
}

We'll clarify what this means in Lecture Notes Seven\(^\text{21}\).

A second class, called Experiment, shows how one could use it.

public class Experiment  
{ public static void main(String[] args)  
{ BankAccount a = new BankAccount();  
 BankAccount b = new BankAccount();  
 a.deposit(200);  
 b.deposit(300);  
 System.out.println(a.getBalance());  
 System.out.println(b.getBalance());  
 a.withdraw(100);  
 b.withdraw(200);  
 System.out.println(a.getBalance());  
 System.out.println(b.getBalance());  
}
}

Create two files  
• BankAccount.java, and
• Experiment.java

Place the code above in the corresponding files, then compile and run Experiment.

Explain the output.

Then write your own experiment, which simulates the following hypothetical sequence of events:

• an account is created, call it \texttt{one}
• we deposit two hundred dollars in \texttt{one}
• we print the balance in \texttt{one}
• we withdraw one hundred and twenty from \texttt{one}
• we print again the balance in \texttt{one}

\(^{21}\text{Classes, or User-Defined Types.}\)
17. Now change the previous Experiment to allow the user to specify the amounts.

Make your program do this:

- create an account
- ask the user how much (s)he wants to deposit
- get that value from the user and deposit it into the account
- print the balance
- ask the user how much (s)he wants to withdraw
- get that value from the user and withdraw it into the account
- print the balance
- thank the user and end the program

What comes next is your: A201/A597 LAB ASSIGNMENT THREE

For next time you need to work on, and prepare the following, to show to your AIs:

1. Write the program Swap mentioned at Question 3.
2. Write the program Reverse mentioned in Question 4.
3. Write the MilitaryTime program as described at Question 6.
4. Write a program that produces 5 random numbers between -8 and -3.
5. Solve question 8 above, and be prepared to explain your solution.
6. Solve questions 10, 11, 12, 13, 14, 15, 16, and 17 above.

As usual, this lab assignment is to be turned in next lab period, in person.
## Predicates

boolean values, expressions, and if statements.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s this?</td>
<td>x &lt; 15</td>
</tr>
<tr>
<td>A boolean expression: an expression whose value is either true or false.</td>
<td></td>
</tr>
<tr>
<td>Can you define boolean variables in Java?</td>
<td>Sure, boolean is a primitive type in Java.</td>
</tr>
<tr>
<td>boolean b;</td>
<td>b can only hold true and false values.</td>
</tr>
<tr>
<td>What are the primitive types in Java?</td>
<td>There are four kinds:</td>
</tr>
<tr>
<td>• int, byte, short, long</td>
<td>• whole numbers</td>
</tr>
<tr>
<td>• double, float</td>
<td>• floating-point numbers</td>
</tr>
<tr>
<td>• boolean</td>
<td>• boolean values</td>
</tr>
<tr>
<td>• char</td>
<td>• characters</td>
</tr>
<tr>
<td>We’ve seen int, and double values...</td>
<td>...we now take a look at boolean values.</td>
</tr>
<tr>
<td>How do you read this?</td>
<td>“x is less than or equal to 15”</td>
</tr>
<tr>
<td>x &lt;= 15</td>
<td></td>
</tr>
<tr>
<td>How do you write “x is in between 9 and 15”?</td>
<td>First, what value does it have: true or false?</td>
</tr>
<tr>
<td>We don’t know yet, it depends on what x is.</td>
<td>So we might as well call it p(x).</td>
</tr>
<tr>
<td>Very well; now we can look at it for particular values of x.</td>
<td>p(3) is false</td>
</tr>
<tr>
<td>p(20) also is false</td>
<td>Come to think of it, p(x) is false for many values.</td>
</tr>
<tr>
<td>What’s p(x) again?</td>
<td>A statement about x being between 9 and 15.</td>
</tr>
</tbody>
</table>
When is it true?

When

\[ x \leq 15 \]

and at the same time

\[ x \geq 9 \]

How do you write AND in Java?

`&&`

So \( p(x) \) can be written as:

\( (x \geq 9) \land (x \leq 15) \)

And `&&` is read as AND

And `||` is read as OR.

While `!` stands for NOT in Java.

I like A201 !

I do ! think this is that funny...

So `&&`, `||`, and `!` are operators for truth values.

Yes. How do they work?

Let me draw a table.

| p   | q   | p && q | p || q | ! q |
|-----|-----|--------|--------|-----|
| true| true| true   | true   | false|
| true| false| false  | true   | true |
| false| true| false  | true   |       |
| false| false| false  | false  |       |

So `&&` works in the following way: you graduate if you satisfy both of two requirements.

Otherwise you don’t graduate. `||` is a bit more lenient.

You graduate if you satisfy at least one of the two requirements.

And only when none of them has been satisfied you do not graduate.

And `!` is easy.

It is, indeed.

The boolean type is called after mathematician George Boole, a pioneer in the study of logic.

Logic is tricky: suppose `a` is a boolean value, `true` or `false`. What value does

\[ a \lor \neg a \]

have?

Doesn’t it depend on the value of `a`?

No.

Well, then let’s look at all possible cases:
| a  | ! a | a || (! a) |
|----|-----|-----------|
| true | false | true |
| false | true | true |

Doesn’t it look easy now?

Yes, and there weren’t even too many cases. How do you compute:

\[ 3 + 5 \times 2 \]

Why are you bringing this up? Because as you know there is an implicit order of evaluation for arithmetic expressions.

Does a similar set of rules apply to boolean expressions? Yes. In arithmetic, unary minuses are taken into account before we do any multiplications...

... and only after that we may do additions, if any. If there are no parentheses, otherwise the parens dictate the order of evaluation.

What rules govern the order of evaluation for &&, ||, and !? ! has the highest priority. Then comes &&, and the || has the lowest priority.

So if you look at It’s evaluated as

\[ a \text{ || } b \&\& ! c \]

\[ a \text{ || } (b \&\& (! c)) \]

What is the truth table for Let’s build it at the same time for

\[ !a \&\& !b \]

\[ !(a \text{ || } b) \]

| a  | b  | a \&\& b | !(a \&\& b) | !a  | !b  | !a \text{ || } !b |
|----|----|----------|-------------|-----|-----|------------------|
| true | true | true     | false       | false | false | false           |
| true | false | false    | true        | false | true  | true            |
| false | true | false    | true        | true  | false | true            |
| false | false | false    | true        | true  | true  | true            |

We have just proved one of DeMorgan’s law. What is the other one?

It’s the dual of this: is the same as

\[ !(a \text{ || } b) \]

\[ !a \&\& !b \]

There are many other identities that one can prove. Perhaps we can do that later, as needed.

Yes, but let me give some examples, in case you get bored and want to practice. Sure.

This... ...is the same as this

\[ a \&\& (b \text{ || } c) \]

\[ a \&\& b \text{ || } a \&\& c \]
This... ...is the same as this

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>`a</td>
<td></td>
</tr>
<tr>
<td><code>a &amp;&amp; true</code></td>
<td><code>a</code></td>
</tr>
<tr>
<td>`a</td>
<td></td>
</tr>
<tr>
<td><code>a &amp;&amp; false</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>a == true</code></td>
<td><code>a</code></td>
</tr>
<tr>
<td><code>a == false</code></td>
<td><code>! a</code></td>
</tr>
</tbody>
</table>

Let’s face it: **booleans** can make you dizzy. Yes, but they are clearly necessary.

For example, the programs we have seen so far are fairly inflexible. Except for variations in the input they work the same way with every program run.

One of the essential features of nontrivial computer programs is the ability to make decisions... ...and to carry out different actions, depending on the nature of the inputs.

With **booleans** one can program simple and complex decisions. Learning that will greatly increase our expressive power in Java.

In some of the previous assignments we went to great length to either avoid... ...or fake (or, simulate) decisions by building them into clever formulas.

Being able to make decisions would greatly simplify those programs. The **if/else** statement is used to implement a decision in a program. The **if/else** statement has three parts:

- a **test** (a boolean expression),
- a **then** branch, and
- an **else** alternative.

If the test succeeds, the body of the **then** branch, ... also known as the body of the **if** statement,

...is executed. Here’s an example, as a flowchart: This is from one of the problems of last week.
And in Java:

\[
\text{if } (x > y) \{ \text{max} = x; \} \\
\text{else} \{ \text{max} = y; \}
\]

The condition is \text{red}, the body of the \text{if} statement is \text{blue}, while the body of the \text{else} alternative appears in \text{green}. (OK, I know we don’t have colors here but I’m sure you know which is which—plus, notice the correct spelling!).

<table>
<thead>
<tr>
<th>A statement such as</th>
<th>A conditional statement, such as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{max} = x;</td>
<td>\text{if } (x &gt; 0) { y = x; }</td>
</tr>
<tr>
<td>is called a \text{simple} statement.</td>
<td>\text{else} { y = -x; }</td>
</tr>
<tr>
<td></td>
<td>is called a \text{compound} statement.</td>
</tr>
</tbody>
</table>

By the way, this last compound statement could be replaced by (as it’s equivalent to):

\[
y = \text{Math.abs}(x);
\]

I know, but that’s only because it’s so simple.

Our programs remain essentially \text{sequences} of statements, we just allow compound statements, such as if statements, in.

But they (at least) become two-dimensional.
Quite often the body of an if statement consists of multiple statements that must be executed in sequence whenever the test is successful.

These statements must be grouped together to form a block statement by enclosing them in braces:

- `{ and
- `}.

Also, while an if statement must have a test and a body, the else alternative is optional.

Here’s one, a bit contrived:

```c
if (x < y) {
    temp = x;
    x = y;
    y = temp;
}
```

We assume, of course, that `x`, `y`, and `temp` have been declared, and that `x` and `y`, at least, have been initialized already.

Can you briefly say what the code is doing? It makes sure that of the two values the larger one is always in `x`.

Very good. What were we saying about braces? They group statements together.

What if we drop them? Then the code no longer works as intended.

So what is the syntax of an if statement? The body of an if statement (or an else alternative) must be a statement (just one).

But it can be:

- a simple statement
- a compound statement (such as another if statement), or
- a block statement

It’s good to get into the habit of using braces (and thus block statements) all the time.

Yes, as we will see when we get to exercises, shortly. I can hardly wait. But first, let’s analyze the if statement closer, and look at what makes a test.

Its outcome is either true or false. In many cases the test compares two values.

Comparison operators such as `<=` (read “less than or equal”) are called relational operators.

Java has six relational operators.
<table>
<thead>
<tr>
<th>Java</th>
<th>Math</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>Not equal</td>
</tr>
</tbody>
</table>

The == operator is initially confusing to most newcomers to Java. In Java, the = symbol already has a meaning, namely assignment.

The == denotes equality testing:

```java
a = 5; // assign 5 to a
if (a == 3) // tests whether a equals 3
    System.out.println("a is equal to 3");
else
    System.out.println("a is not equal to 3");
```

You will have to remember to use == for equality testing and to use = for assignment.

Floating point numbers have only a limited precision, and calculations can introduce roundoff errors. That means we need to be careful when we want to test if two floating point quantities are representing the same thing.

Here’s an example:

```java
double r = Math.sqrt(2);
if (r * r == 2)
    System.out.println(r * r + " == 2");
else
    System.out.println(r * r + " != 2");
```

Unfortunately such roundoff errors are unavoidable.

In most circumstances it does not make a lot of sense to compare floating point numbers exactly. Instead we should test whether they’re close enough.

That is, the absolute value of their difference should be less than some threshold. Mathematically, $x$ and $y$ are close enough if

$$|x - y| \leq \epsilon$$

...for a very small number, $\epsilon$. Greek letter \textit{epsilon} ($\epsilon$) is commonly used to denote a very small quantity.

It is common to set $\epsilon$ to $10^{-14}$ when comparing double numbers. However, this is not always good enough.
Indeed, if the two numbers are very big, then one can be a roundoff of the other even if their difference is much bigger than $10^{-14}$.

So $x$ and $y$ are close enough if

$$\frac{|x - y|}{\max(|x|, |y|)} \leq \epsilon$$

And to avoid division by zero it is better to test whether

$$|x - y| \leq \epsilon \max(|x|, |y|)$$

In Java, this is:

```java
Math.abs(x - y) <= EPSILON * Math.max(Math.abs(x), Math.abs(y))
```

OK, I think I understand how I test numbers (integers or floating point) for equality.

What else can we test for equality?  How about Strings?

To test whether two strings are equal to each other, ...that is, that their contents is the same...

...one must use method equals.  Why not use == like for numbers?

Strings are objects.  And so are Rectangles.

If you compare two object references with the == operator, you test whether the references refer to the same object.

That’s because you check to see whether the two locations contain the same thing.

The exact same thing.  Which is in each case an address, to an actual object.

Let’s see some examples.

```java
Rectangle a = new Rectangle(5, 10, 20, 30);
Rectangle b = a;
Rectangle c = new Rectangle(5, 10, 20, 30);
```

The comparison `a == b` is true.

Both object variables refer to the same object.  But the comparison `a == c` is false.

The two object variables refer to different objects.
You can use the `equals` method to test whether two rectangles have the same contents.

| You can use the `equals` method to test whether two rectangles have the same contents. | Thus `a.equals(c)` is true. |
| And so is `c.equals(b)` obviously. | Same with Strings, so we will have to remember to use `equals` for string comparison. |

In Java letter case matters. Thus `"harry".equals("HARRY")` evaluates to `false`. But `"harry".equalsIgnoreCase("HARRY")` evaluates to `true`.

Even if two strings don’t have "identical" contents we may still want to know the relationship between them. The `compareTo` method compares strings in dictionary order.

| If `string1.compareTo(string2) < 0` | ...then `string1` comes before `string2` in dictionary order. |
If `string1.compareTo(string2) > 0`...then `string1` comes after `string2` in dictionary order.

If `string1.compareTo(string2) == 0`...then the two strings have identical contents.

You should look this method up in class `String`. Actually the dictionary ordering used by Java is slightly different from that of a normal dictionary.

Java is case-sensitive and sorts characters by listing numbers first, then uppercase characters, then lowercase characters.

For example 1 comes before B which comes before a.

And the space character comes before all other characters.

Can we describe the comparison process a little bit in greater detail?

When comparing two strings, corresponding letters are compared until one of the strings ends or the first difference is encountered.

If one of the strings ends, the longer string is considered the later one.

If a character mismatch is found, compare the characters to determine which string comes later in the dictionary sequence.

The process is called **lexicographic comparison**.

That’s why "car" comes before "cargo", And "cathode" comes after "cargo" in lexicographic ordering.

Time for a break. I sure think so.

And some exercises too. Yes, but the break first, please.

OK, here’s what we’ll do: we’ll put the exercises into the break altogether. And combine the **useful** with the **necessary**.

The text of the problem is always the same. I know it already: "**What is the output produced by the following snippets of code when embedded in a complete program?**"

Let’s see the snippets.

**Snippet 1:**

```java
int x = 3;
if (2 > x)
    System.out.print(1);
else
    System.out.print(2);
if (x < 2)
    System.out.println(3);
System.out.print(4);
```
### Snippet 2:

```java
int x = 3;
if (x > 5) {
    if (x < 10)
        System.out.print(1);
    else
        System.out.print(2);
} else
    System.out.print(3);
```

Messy. The curly braces change everything.

What if you take them out?  
The diagram changes significantly.

And you have experienced a **dangling else**.  
That's right.

(Also, your indentation was a bit misleading.)

### Snippet 3:

```java
int x = 3;
if (x > 0) System.out.print(x + 1);
else if (x > 1) System.out.print(x);
else if (x > 2) System.out.print(x - 1);
else if (x > 3) System.out.print(2 * x);
else System.out.print(x * x);
```

Easy. Diagram it.

### Snippet 4 (and last):

```java
int x = 3;
if (x > 0) System.out.print(x + 1);
\[\text{boxed}\] else if (x > 1) System.out.print(x);
\[\text{boxed}\] else if (x > 2) System.out.print(x - 1);
\[\text{boxed}\] else if (x > 3) System.out.print(2 * x);
else System.out.print(x * x);
```

Remove theboxedelse's.  
Who would ever do that in a program?

Nobody. It's for practice.  
Messy again. You have to redraw everything.

I agree it's messy, but is it **hard**?  
No. Is this the last one?

Yes.  
Can we do a reasonable example now?
OK, here's Nineteen from the first set of problems.

/* Solution to problem nineteen in the first problem set. Use
 * ConsoleReader from lab notes 2 as explained. The trick here
 * (as hinted in the text) is to transform a number for a month in
 * a position (index) in the string where the month name is starting.
 * all names being made of the same length, and then concatenated
 * together in one final string.
 */
public class Nineteen {
    public static void main(String[] args) {
        String monthNames = "January  February  March   " +
                             "April    May     June    " +
                             "July     August  September " +
                             "October  November December " ;
        // open a connection with the keyboard
        ConsoleReader console = new ConsoleReader(System.in);
        // greet the user, and ask for input
        System.out.println("Please enter a month number from 1 to 12.");
        // get month name
        int month = console.readInt();
        // report the name of the month
        System.out.println(
            monthNames.substring("September ",.length() * (month-1),
            "September ",.length()*month));
        // formula uses the length of the longest name
    }
}

Here is it with if statements:

public class P19 {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Please enter a month number from 1 to 12.");
        int month = console.readInt();
        if (month == 12) System.out.println("December");
        else if (month == 11) System.out.println("November");
        else if (month == 10) System.out.println("October");
        else if (month == 9) System.out.println("September");
        else if (month == 8) System.out.println("August");
        else if (month == 7) System.out.println("July");
        else if (month == 6) System.out.println("June");
        else if (month == 5) System.out.println("May");
        else if (month == 4) System.out.println("April");
        else if (month == 3) System.out.println("March");
        else if (month == 2) System.out.println("February");
        else if (month == 1) System.out.println("January");
    }
}
<table>
<thead>
<tr>
<th>I thought we agreed to use block statements (with curly braces) for the bodies of if statements and else alternatives all the time.</th>
<th>Yes, but just for once I wanted to keep the code somewhat shorter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well, then, just for once, I have two more exercises.</td>
<td>OK, I will remember to put braces from now on, always.</td>
</tr>
<tr>
<td>Too late.</td>
<td>Show me the first exercise.</td>
</tr>
<tr>
<td>Here it is:</td>
<td>Can't be true!</td>
</tr>
</tbody>
</table>
|     if (false && false || true) {  
      System.out.print(false);  
      } else {  
      System.out.print(true);  
      } | |
| Snippet 2: | I can see the difference. |
|     if (false && (false || true)) {  
      System.out.print(false);  
      } else {  
      System.out.print(true);  
      } | |
| I'm sure you do. | That's probably true or false. |
HOME FRONT

Winter health assessment

Is it a cold—or something worse?

This year, you may react to any flu-like symptoms with more than the usual concern. It’s not easy to distinguish influenza from deadlier diseases such as anthrax and from milder afflictions like the common cold. Many symptoms overlap.

The chart below can help you determine when to see your doctor or seek emergency treatment, and when to just ride the illness out. It’s particularly important to see your physician promptly if you develop a sudden fever and cough. That combination is the most reliable indicator that you have the flu rather than a cold. Contacting your doctor quickly can shorten your bout of flu by catching the infection in time for flu-fighting drugs to be effective.

Prompt action also allows your doctor to rule out the small chance of a more serious infection—including inhalational anthrax, which also generally starts with a sudden fever and cough. However, most recent anthrax victims experienced symptoms of flu, which the flu seldom causes. (One of those is an allergic reaction to dust or pollen.)
## Classes

*Introduction to user-defined types. Classes.*

<table>
<thead>
<tr>
<th>You have now learned about the number and string data types of Java.</th>
<th>Although it is possible to write interesting programs using nothing but numbers and strings, most useful programs need to manipulate data items that are more complex and more closely represent entities in the real world.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples of these data items are bank accounts, employee records, graphical shapes, and so on.</td>
<td>The Java language is ideally suited for designing and manipulating such data items, or <em>objects</em>.</td>
</tr>
<tr>
<td>In Java, you define <em>classes</em> that describe the behaviour of these objects. (Classes are blueprints).</td>
<td>You will now learn how to define classes that describe objects with very simple behaviour. This will be a very good start.</td>
</tr>
<tr>
<td>Let’s create a simple class that describes the behaviour of a <em>bank account</em>.</td>
<td>Before we can describe what a bank account is in Java we need to be clear (in plain English) what we mean by it.</td>
</tr>
<tr>
<td>In other words we have to sell it first.</td>
<td>Exactly.</td>
</tr>
</tbody>
</table>
| Well, consider what kinds of operations you can carry out with a bank account. You can:... | • deposit money  
• withdraw money  
• get the current balance |
| Sounds like a bank account to me. | In Java these operations are expressed as *method calls*. |
| So the set of methods that an object of type... | ...*BankAccount* (sounds like a good name to me) |
| ...will support, could be... | • deposit  
• withdraw  
• getBalance |
<table>
<thead>
<tr>
<th>BankAccounts do best!</th>
<th>OK, before we get too euphoric we need to imagine such an object in action.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects are agents.</td>
<td>Exactly. If they have methods (and most objects do) they have a behaviour, defined by what their methods can do. Now, how do you envision BankAccounts in action?</td>
</tr>
<tr>
<td>To start with, opening a new bank account should look like this:</td>
<td>...and the initial balance should be zero. How do you put, let's say, $1,000.00 in your account when you create it?</td>
</tr>
<tr>
<td>myChecking = new BankAccount();</td>
<td>Isn't this very similar to translate for Rectangles?</td>
</tr>
<tr>
<td>It would be something like this:</td>
<td>How would you check your current balance?</td>
</tr>
<tr>
<td>myChecking.deposit(1000.00);</td>
<td></td>
</tr>
<tr>
<td>It is, indeed, the very same thing: we're translating the balance, in one dimension.</td>
<td>Isn't this very similar to printing the length of a String?</td>
</tr>
<tr>
<td>I'd ask myChecking to report the balance, which I could then print:</td>
<td>How come? What if you want to withdraw $300.00 from your checking?</td>
</tr>
<tr>
<td>System.out.println(myChecking.getBalance());</td>
<td>Then it won't be too hard though to come up with an extra method...</td>
</tr>
<tr>
<td>It sure is. As for the third method, I just realized I don't even need it...</td>
<td>myChecking.deposit(-300.00);</td>
</tr>
<tr>
<td>I can already express that as:</td>
<td>This should make more sense to the user of your class. Are we done now?</td>
</tr>
<tr>
<td>myChecking.deposit(-300.00);</td>
<td>myChecking.withdraw(300.00);</td>
</tr>
<tr>
<td>.. that could be called as follows:</td>
<td>...at least from the point of view of what we wanted in a bank account. We could certainly add methods to compute interest, etc., and enhance our model (or design), but for now...</td>
</tr>
<tr>
<td>I think we are. These three methods form the complete list of what you can do with an object of type BankAccount, ...</td>
<td></td>
</tr>
<tr>
<td>...these three methods are more than enough for what we have in mind with this class.</td>
<td>We want to implement it and see it used in a Java program.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Nothing less. For a more complex class, it takes some amount of skill and practice to discover the appropriate behaviour that makes the class useful and is reasonable to implement.</td>
<td>We will learn more about this important aspect when we get to the chapter on &quot;Object oriented design&quot;.</td>
</tr>
<tr>
<td>The behaviour of our BankAccount class is very simple, and we have described it completely. We can now implement it.</td>
<td>Yes, let’s go for it.</td>
</tr>
<tr>
<td>Although I think it’s worth pointing out one thing, before we even write a single line of code of implementation...</td>
<td>What is it?</td>
</tr>
<tr>
<td>In our description of the methods we have used objects of type BankAccount without knowing (or caring too much to know) about their implementation, which we are only now about to describe.</td>
<td>Yes, that’s an important aspect of object-oriented programming. But now, that we are completely clear on how to use objects of the BankAccount class, we really need to get started...</td>
</tr>
<tr>
<td>...to describe the Java class that implements the described behaviour.</td>
<td>I can get us started and in the following way:</td>
</tr>
</tbody>
</table>
| public class BankAccount  
  { ... 
    ... 
  } |  

I don’t understand what you mean by data. | It’s what makes tiggers remember where they have bounced last. Do you remember Rectangles? |
| I certainly do. Their diagrams were always containing four slots, in which we were writing the current values of their x, y, width, and height. | That’s the data that they have, which helps them remember where they are, and how big they are. Could you draw a diagram to illustrate what’s happening after we create a new BankAccount? |
| The code would be... | ...and the diagram? |

    BankAccount mySavings = new BankAccount();

Take a look on the next page.
mySavings

Very good. What’s that in which you put the initial balance of 0 (zero)?

It’s a location, like we’ve seen for Rectangles. It looks like a variable, probably of type double or of a similar or related type.

Each object must store its current state; for objects of type BankAccount the state is the current balance of the bank account.

Each object stores the state in one or more instance variables.

An instance variable declaration consists of the following three parts:

<table>
<thead>
<tr>
<th>a) an access specifier</th>
<th>...such as private</th>
</tr>
</thead>
<tbody>
<tr>
<td>b) the type of the variable</td>
<td>...such as double</td>
</tr>
<tr>
<td>c) the name of the variable</td>
<td>...such as balance</td>
</tr>
</tbody>
</table>

So far we have:

```
public class BankAccount
{
    private double balance;
    ...
}
```

...and the diagram:

mySavings

The balance field is all we need, as far as data goes.
Each object of a class has its own copy of the instance variables.

We’ve seen that with Rectangles and it’s the same with BankAccounts.

Each object has its own balance field.

And in our example they hold different values.

Instance variables are generally declared with the access specifier private.

That means that they can be accessed only by methods of the same class. In particular, the balance variable can be accessed only by the deposit, withdraw, and getBalance methods.

Let’s write these methods in Java.

OK. Let’s start by defining them.

Yes, let’s start by describing their headers.

A method header consists of the following parts:

a) an access specifier

...such as public

b) the return type of the method

...such as double or void

c) the name of the method

...such as deposit

d) a list of the parameters of the method

...describing the method’s initial data. Let’s consider each of these parts in detail.

The access specifier controls which other methods can call this method.

Most methods should be declared public. This way, all other methods in your program can call them. (Occasionally, it can be useful to have methods that are not so widely callable, but we will look at that later).
The *return type* is the type of the value that the method computes. For example, the `getBalance` method returns the current account balance, which will be a floating-point number, so its return type is `double`. On the other hand the `deposit` and `withdraw` methods don't return any value. They just update the current balance but don't return it.

To indicate that a method does not return a value, use the special type `void`. Both the `deposit` and `withdraw` are declared with return type `void`.

The *parameters* are inputs to the method. The `deposit` and `withdraw` methods each have one parameter: the amount of money to deposit or withdraw.

You need to specify the type of the parameter, such as `double`, and the name for the parameter, such as `amount`. The `getBalance` method has no parameters. In that case, you still need to supply a pair of parentheses () behind the method name.

If a method has more than one parameter, you separate them by commas. And once you have specified the method header, you must supply the implementation of the method, in a block that is delimited by braces `{...}`.

Putting all this together we have: Looks good.

```java
public class BankAccount {
    public void deposit(double amount) {
        ...
        ...
    }
    public void withdraw(double amount) {
        ...
        ...
    }
    public double getBalance() {
        ...
        ...
    }
    private double balance;
}
```

We now must provide an implementation for each method of the class. The implementation of these three methods is straightforward: when some amount of money is deposited or withdrawn, the balance increases or decreases by that amount.

```java
public class BankAccount {
    public void deposit(double amount) {
        balance = balance + amount;
    }
    public void withdraw(double amount) {
        balance = balance - amount;
    }
    public double getBalance() {
        return balance;
    }
    private double balance;
}
```
```java
{ balance = balance - amount; }
public double getBalance()
{ return balance; }
private double balance;
```

The `getBalance` simply `returns` the current balance. But wait: how can we use `balance` and `amount` without declaring or initializing them, as we said we should be doing every single time?

They are not method (local) variables. `amount` is a method parameter. When the method is called the value that will be passed to the method will be placed in a location called `amount` of type `double` that represents the initial data of the method. So `amount`, as a method parameter is like a variable...

...and will have been initialized when the method is called. How about `balance`?

It is also a variable, only not a method variable. It is an instance variable, visible to every instance method (such as `deposit`, `withdraw`, and `getBalance`). For these methods it is a (somewhat) `global` variable (which they `share`).

Instance variables are initialized when the object to which they belong is constructed. In this respect they are really different from method variables.

They are initialized with default values, that depend on their types.

Numbers, for example, are initialized with a value of 0 (zero).

Of course, you can also initialize them in a certain, customized way, if you define `constructors`.

We will talk about constructors tomorrow.

Meanwhile let's see what we mean by `return`.

Well, when you ask someone a question, and you pay for the question to be answered, ...

...you expect an answer, ...

...in `return`.

When a function, that is supposed to provide a value as an answer, has the answer ready...

...(by computing an expression, or by referring to a location where the final result has been stored, ready to be reported)...

...and when it wants to report it, ...

...to whoever called it in the first place...

...it simply states that it’s ready to finish and `returns` the value, ...

...after which the function (or method, as it's known in Java) ends.

Interesting. So this is basically `syntax`.

Very much so.
OK, can we see the full program? Yes, you need to have two classes.

One is the class we designed.

```java
public class BankAccount
{
  public void deposit(double amount)
  {
    balance = balance + amount;
  }
  public void withdraw(double amount)
  {
    balance = balance - amount;
  }
  public double getBalance()
  {
    return balance;
  }
  private double balance;
}
```

While the other one has the `main`.

```java
public class Experiment
{
  public static void main(String[] args)
  {
    BankAccount a = new BankAccount();
    BankAccount b = new BankAccount();
    a.deposit(200);
    b.deposit(300);
    System.out.println(a.getBalance());
    System.out.println(b.getBalance());
    a.withdraw(100);
    b.withdraw(200);
    System.out.println(a.getBalance());
    System.out.println(b.getBalance());
  }
}
```

Place them in the same directory... ...compile and run them, and enjoy!

...Not many problems couldn't be much harder, ...I think. I agree. By the way, ...

"I hope you're not much tired, are you?" "No, how. And thank you very much for asking!"
Diagrams

Diagrams for object-oriented concepts.

1. Draw this:
   ```java
   class A {
   }
   ```

   Don’t forget the blueprint:
   ![Blueprint A](image)

2. Draw this:
   ```java
   class B {
       int u;
       double q;
   }
   ```

   Notice that the blueprint is no longer empty.
   ![Blueprint B](image)

3. Draw this:
   ```java
   class Student {
   }
   ```

   Easy. Student or A - same difference.
   ![Blueprint Student](image)
4. Now draw this:

```java
class Student {
    int age;
}
```

5. Draw this:

```java
class Z {
    int m;
    double n;
}
```

6. Draw this:

```java
class M {
    int m;
    static double n;
}
```

7. Draw this:

```java
class Q {
    int v;
    double z;

    public static void main(String[] args) {
    }
}
```

For this the picture is bigger:
8. Draw this:

```java
class E {
    int v;
    double z;
    public static void main(String[] args) {
        int q;
        q = 3;
    }
}
```

Same as above only `main` no longer is empty.
9. Draw this:

```java
class Student {
    int age;
    double gpa;

    public static void main(String[] args) {
        int q = 21;
        String n = "John";
    }
}
```

10. Draw this:

```java
class J {
    int x;

    public static void main(String[] args) {
        J z = new J();
        int x = 5;
        z.x = 5;
    }
}
```
11. Draw this:

```java
class Student {
    int age;
    double gpa;

    public static void main(String[] args) {
        int q;
        String n;
        Student s;

        s = new Student();
    }
}
```
12. Draw this:

```java
class Student {
    int age = 20;
    double gpa;

    public static void main(String[] args) {
        Student s = new Student();
        Student q = new Student();
        q.age = 19;
        s.age = 21;
        q.age = 3 + q.age;
    }
}
```

13. Now write this:

Create two accounts for Doug and Adrian. Initial balance for both accounts is 10. Then Doug's account doubles, Adrian's decreases by 3, and Doug's account has one more dollar deposited to it.

Let me try it. How about this?

```java
class Account {
    double balance = 10;

    public static void main(String[] args) {
        Account doug, adrian;

        doug = new Account();
        adrian = new Account();

        doug.balance = doug.balance * 3;
        adrian.balance -= 3;
        doug.balance++;
    }
```
}{
}

Looks good. Thank you.

14. Draw this:

class BankAccount {
    double balance = 25;

    public static void main(String[] args) {
        BankAccount doug = new BankAccount();
        BankAccount adrian = new BankAccount();

        doug.balance = doug.balance * 2;
        adrian.balance -= 3;

        doug.balance++;
    }
}

This is similar to what we drew at step 12.

15. Draw this:

class Account {
    double balance = 20;

    public static void main(String[] args) {
        Account m = new Account();
        Account q = new Account();

        m.deposit(10);
        n.deposit(3);
    }

    void deposit (int n) {
        this.balance += n;
    }

}
16. Draw this:

```java
class Account {
    double balance;

    Account(double x) {
    }

    public static void main(String[] args) {
        Account m = new Account(10);
        Account n = new Account(3);
        m.balance = 10 + m.balance;
    }
}
```
17. Draw this:

```java
class X {
    int x;
}
```

18. Draw this:

```java
class A {
    int x;

    A (int initialValue) {
        this.x = initialValue;
    } /* do this when you first create the object but
       don't make it part of the generated instance */
}
```
Speaking of this can I show you an example?

Sure, go ahead.

Here it is:

class Speakers {
    public static void main(String[] args) {
        Speaker a = new Speaker("Larry");
        Speaker b = new Speaker("Michael");
        Speaker c = new Speaker("Tony");
        a.talk();
        b.talk();
        c.talk();
    }
}
class Speaker {
    String speaker;
    Speaker (String name) {
        speaker = name;
    }
    void talk() {
        System.out.println(this.speaker + " is very happy.");
    }
}

Very nice.

Thanks. this writer is very happy.
19. Write this:

```java
class Student {
    String name;
    Student (String w) {
        this.name = w;
    }
}

class Friday {
    public static void main(String[] args) {
        Student q = new Student("Larry");
        Student u = new Student("John");
        Student s = new Student("Adrian");
    }
}
```

20. Draw this:

```java
String
"adrian" "doug"
```

Funny.
21. Draw this:

```java
class Checking {
    double balance;
    
    Checking(double x) {
        this.balance = x;
    }
    void deposit(double y) {
        this.balance += y;
    }
}
```

22. Draw this:

```java
class Tea {
    
    double bird;
    
    void book(double watch) { // int z = 3;
        this.bird = watch - this.bird;
    }
    
    Tea (double snake) {
        this.bird = snake * 2;
    }
}
```

Oh, this is almost entirely exactly the same.
23. Draw this:

```java
class Hat {
    public static void main(String[] args) {
        Tea m = new Tea(-2);
        Tea n = new Tea(42);

        m.book(92);
        n.bird = m.bird + n.bird;

        n.book(25);
        n.book(25);
    }
}
```

![Diagram of the code](image-url)
24. Draw this:

```java
class X {
    int q;
}

class Play {
    public static void main(String[] args) {
        X u = new X();
        X p = new X();
        u.q = 3;
        p.q = u.q + p.q;
        u.q = u.q + p.q;
    }
}
```

This last one was rather easy.
Modeling

*Classes and objects*

Let's start by mentioning a student's observations, one past semester. It refers to how we calculate intersections of rectangles with the Java API. These are very perceptive observations. She said:

- The intersection method in class Rectangle does not return a valid (that is, meaningful) answer when the rectangles don't overlap. (This is what the text says too, but one needs to read even the text of the problem with care!)

- In other words, you can call intersection and you will always obtain an answer. You can't tell by just looking at this answer if the rectangles overlap or not. So what do we do when they do not overlap?

- The solution is to

  1. check\(^\text{22}\) (see method intersects) if they overlap
  2. before you calculate\(^\text{23}\) the intersection
  3. (which you should do only if they overlap!).

Hope you found these comments useful.

With this lab we again encourage you to work out problems and check your answers against ours. The problems are indexed below, as well as in the main (class notes) page.

1. **Here**\(^\text{24}\)**'s a set of warm-up questions.

2. **Here**\(^\text{25}\) are the answers to the warm-up questions.

3. **Here**\(^\text{26}\)**'s a set of programming problems, to practice.

4. **Here**\(^\text{27}\)**'s a broad (conceptual) overview of the kind of modeling we are talking about.

5. **Here**\(^\text{28}\)** are the solutions to the programming problems.

\(^{22}\) [http://java.sun.com/products/jdk/1.2/docs/api/java/awt/Rectangle.html#intersects(java.awt.Rectangle)]

\(^{23}\) [http://java.sun.com/products/jdk/1.2/docs/api/java/awt/Rectangle.html#intersection(java.awt.Rectangle)]

\(^{24}\) [http://www.cs.indiana.edu/classes/a201/sum2002/notes/w2.html]

\(^{25}\) [http://www.cs.indiana.edu/classes/a201/sum2002/notes/w2Sol.html]

\(^{26}\) [http://www.cs.indiana.edu/classes/a201/sum2002/notes/set2.html]

\(^{27}\) [http://www.cs.indiana.edu/classes/a201/sum2002/notes/rodS.html]

\(^{28}\) [http://www.cs.indiana.edu/classes/a201/sum2002/notes/set2Sol.html]
All of these are going to be presented in the following pages.

Meanwhile here's your **LAB ASSIGNMENT FOUR**.

Next time you are expected to:

- Have all the steps below done and understood (we develop a program below).
- Look over and solve (or learn the posted solutions to) the review exercises and problem listed in the next few chapters. The texts and the solutions to all of these problems are also available on-line (steps 1, 2, 3, 5 above).
- Read and understand and be able to explain the crux of the slides presented at step 4 above (the part that talks about Kanamits and the Twilight Zone). Why do you think the slides talk about, open up, and end in reference to Kanamits (*whatever* they might be).

  Hint: abstraction.

Let's go through the complete and annotated development of a solution to a problem.

(It's one of those listed above, too).

1. Let's implement a class `Student`.

   ```java
   frilled.cs.indiana.edu%pico Student.java
   frilled.cs.indiana.edu%cat Student.java
   public class Student {
   }
   frilled.cs.indiana.edu%javac Student.java
   frilled.cs.indiana.edu%ls -1d Student*
   -rw------- 1 dgerman 188 Feb 1 08:18 Student.class
   -rw------- 1 dgerman 27 Feb 1 08:17 Student.java
   frilled.cs.indiana.edu%
   
   2. Looks like we're done. Can we test it?

   3. We need a tester class with a `main` method.

   ```java
   frilled.cs.indiana.edu%pico StudentTest.java
   frilled.cs.indiana.edu%cat StudentTest.java
   public class StudentTest {
       public static void main(String[] args) {
           Student a = new Student();
       }
   }
   frilled.cs.indiana.edu%javac StudentTest.java
   frilled.cs.indiana.edu%ls -1d Student*
   -rw------- 1 dgerman 188 Feb 1 08:18 Student.class
   -rw------- 1 dgerman 27 Feb 1 08:17 Student.java
   -rw------- 1 dgerman 297 Feb 1 08:21 StudentTest.class
   -rw------- 1 dgerman 110 Feb 1 08:21 StudentTest.java
   frilled.cs.indiana.edu%
4. Can we test it?

5. We can run `StudentTest` but we get no output.

    frilled.cs.indiana.edu%java StudentTest
    frilled.cs.indiana.edu%

6. Does it matter?

7. Do we know what happens *inside*?

8. The `Student` class is empty. `Student` objects are *amorphous*.

9. I see... Let’s make it such that each `Student` has (at least) a name, then.

    frilled.cs.indiana.edu%pico Student.java
    frilled.cs.indiana.edu%cat Student.java
    public class Student {
        private String name;
    }
    frilled.cs.indiana.edu%

10. What’s the meaning of `private`?

11. It means that to know the name of a `Student` you need to ask the `Student` what its name is.

12. I don’t feel very comfortable using *he* or *she* for a `Student` object.

13. Fine. How do you inquire about a `Student`’s name?

14. We need to add this functionality to class `Student` first, then make use of it.

15. Here’s a more comprehensive blueprint of `Student` objects.

    frilled.cs.indiana.edu%pico Student.java
    frilled.cs.indiana.edu%cat Student.java
    public class Student {
        private String name;
    }
    frilled.cs.indiana.edu%pico Student.java
    frilled.cs.indiana.edu%cat Student.java
    public class Student {
        private String name;

        public String whatsYourName () {
            return name;
        }
    }
    frilled.cs.indiana.edu%

16. What does it mean for the `whatsYourName` method to be `public`?

17. It means you can ask a `Student"What’s your name?"`
18. What if we make it private?
19. Then we can never ask.
20. How do we create a Student?
21. Just invoke new the way we did in the tester’s main.
22. And if we invoke it, how do things get created, and initialized.
23. Well, a default no-arg constructor is present, but we don’t see it.
24. I think we should add it, so that we not forget that it’s there.

```java
frilled.cs.indiana.edu%pico Student.java
frilled.cs.indiana.edu%cat Student.java
public class Student {
    private String name;

    public String whatsYourName () {
        return name;
    }

    Student() {
    }
}
frilled.cs.indiana.edu%
```
25. It’s empty, but it gets called at creation time.
26. Can we create a Student with an initial name?
27. Only if we provide that type of constructor.
28. To be able to create a Student with an initial name we need to
   • be able to initialize the name of any Student (at creation time)
   • with an actual name (to be specified when we create the Student).
29. We’re looking for something like this:

```java
new Student("Larry Johnson")
```
30. Let’s provide class Student with that capability.

```java
frilled.cs.indiana.edu%pico Student.java
frilled.cs.indiana.edu%cat Student.java
public class Student {
    private String name;
    public String whatsYourName () {
        return name;
    }
    Student() {
    }
    Student(String givenName) {
```
name = givenName;
}
}
frilled.cs.indiana.edu%

31. Let's enhance our tester's main to exploit the new features.

frilled.cs.indiana.edu%pico StudentTest.java
frilled.cs.indiana.edu%cat StudentTest.java
public class StudentTest {
    public static void main(String[] args) {
        Student a = new Student("Larry");
        Student b = new Student("Michael");
        String answer;
        System.out.print("Printing the name of the first student: ");
        answer = a.whatsYourName();
        System.out.println(answer);
        System.out.print("Printing the name of the second student: ");
        answer = b.whatsYourName();
        System.out.println(answer);
    }
}
frilled.cs.indiana.edu%javac StudentTest.java
frilled.cs.indiana.edu%java StudentTest
Printing the name of the first student: Larry
Printing the name of the second student: Michael
frilled.cs.indiana.edu%

32. Great! What else were we supposed to do?

33. Let's enable the Students to keep track of their scores.

frilled.cs.indiana.edu%pico Student.java
frilled.cs.indiana.edu%cat Student.java
public class Student {
    private String name;
    public String whatsYourName () {
        return name;
    }
    Student() {
    }
    Student(String givenName) {
        name = givenName;
    }
    void addQuizScore(int newScore) {
        totalScore = totalScore + newScore;
    }
    private int totalScore;
}
frilled.cs.indiana.edu%

34. I see... If there's a new score to be added to the total score for a student then we just add it to the totalScore as if it were an amount to be placed as deposit over a current, given, existing balance.
35. Yes, so you need to define an instance variable `totalScore` (which will keep the cumulative score for the `Student`) and use it as if it were a `balance`.

36. This way a `Student` is like a `BankAccount` with a name.

37. Let's write `getBalance`, then.

```java
frilled.cs.indiana.edu%pico Student.java
frilled.cs.indiana.edu%cat Student.java
public class Student {
    private String name;
    public String whatsYourName () {
        return name;
    }
    Student() {}
    Student(String givenName) {
        name = givenName;
    }
    void addQuizScore(int newScore) {
        totalScore = totalScore + newScore;
    }
    private int totalScore;
    int whatsYourTotalScore() {
        return totalScore;
    }
}
frilled.cs.indiana.edu%
```

38. Let's test it.

```java
frilled.cs.indiana.edu%pico StudentTest.java
frilled.cs.indiana.edu%cat StudentTest.java
public class StudentTest {
    public static void main(String[] args) {
        Student a = new Student("Larry");
        Student b = new Student("Michael");
        String answer;
        System.out.print("Printing the name of the first student: ");
        answer = a.whatsYourName();
        System.out.println(answer);
        System.out.print("Printing the name of the second student: ");
        answer = b.whatsYourName();
        System.out.println(answer);
        a.addQuizScore(100);
        a.addQuizScore(90);
        a.addQuizScore(100);
        System.out.println("Student " + a.whatsYourName() + " reports: ");
        System.out.println(" cumulative score: " + a.whatsYourTotalScore());
    }
}
frilled.cs.indiana.edu%javac StudentTest.java
frilled.cs.indiana.edu%java StudentTest
Printing the name of the first student: Larry
Printing the name of the second student: Michael
Student Larry
reports:
cumulative score: 290
frilled.cs.indiana.edu%

39. I think you need a space between Larry and reports.

40. I'll let you fix that. But overall we've come a long way, don't you think?

41. I sure do think so. What if I want the Students to be able to report the average score in addition to the cumulative score? I don't think this is possible at the moment, because they don't remember how many quizzes they have taken.

42. Indeed, they only keep the cumulative score.

43. To remember how many quizzes they have taken they would need to keep a counter, to be updated (incremented by 1) every time a new score is added to the totalScore.

44. If we kept the number updated we could easily report the average at any time, as follows.

frilled.cs.indiana.edu%pico Student.java
frilled.cs.indiana.edu%cac Student.java
public class Student {
    private String name;
    public String whatsYourName () {
        return name;
    }
    Student() { }
    Student(String givenName) {
        name = givenName;
    }
    void addQuizScore(int newScore) {
        totalScore = totalScore + newScore;
    }
    private int totalScore;
    int whatsYourTotalScore() {
        return totalScore;
    }
    private int number0fScores;
    double reportAverage() {
        return (double)totalScore / number0fScores;
    }
}
frilled.cs.indiana.edu%

45. I think you forgot to update the counter in addQuizScore, haven't you?

46. Ooops!...
frilled.cs.indiana.edu%pico Student.java
frilled.cs.indiana.edu%cat Student.java
public class Student {
    private String name;
    public String whatsYourName () {
        return name;
    }
    Student() {}  
    Student(String givenName) {
        name = givenName;
    }
    void addQuizScore(int newScore) {
        totalScore = totalScore + newScore;
        numberOfScores = numberOfScores + 1;
    }
    private int totalScore;
    int whatsYourTotalScore() {
        return totalScore;
    }
    private int numberOfScores;
    double reportAverage() {
        return (double)totalScore / numberOfScores;
    }
}
frilled.cs.indiana.edu%

47. There you go.
48. Can you test that?
49. Sure, how about this:

frilled.cs.indiana.edu%pico StudentTest.java
frilled.cs.indiana.edu%cat StudentTest.java
public class StudentTest {
    public static void main(String[] args) {
        Student a = new Student("Larry");
        Student b = new Student("Michael");
        String answer;
        System.out.print("Printing the name of the first student: ");
        answer = a.whatsYourName();
        System.out.println(answer);
        System.out.print("Printing the name of the second student: ");
        answer = b.whatsYourName();
        System.out.println(answer);
        a.addQuizScore(100);
        a.addQuizScore(90);
        a.addQuizScore(100);
        System.out.println("Student " + a.whatsYourName() + " reports: ");
        System.out.println(" cumulative score: " + a.whatsYourTotalScore());
        System.out.println(" average score: " + a.reportAverage());
50. Nice. You only changed one line!

51. Indeed. And here's the actual test:

```java
frilled.cs.indiana.edu%javac StudentTest.java
frilled.cs.indiana.edu%java StudentTest
Printing the name of the first student: Larry
Printing the name of the second student: Michael
Student Larry reports:
  cumulative score: 290
  average score: 96.66666666666667
frilled.cs.indiana.edu%
```

52. Good Student!

53. Yes. Isn't it time for a break?

54. I sure think so.

55. See you next week!

Until then, here's a brief summary:

```java
class A {
    int x;
    void fun(int y) {
        int z = 3;
        x = z + y;
    }
}
```

*instance variable*

*formal parameter*

*local variable*

*better write it as this.x*
Warmups (II)

Lab Four: The Warmups.

Questions:

1. Explain the difference between an object and a class.
2. Give the Java code for an
   - object of class BankAccount and
   - for an object variable of class BankAccount.
3. Explain the differences between an instance variable and a local (or method) variable.
4. Explain the difference between

   ```java
   new BankAccount(5000);
   ```

   and

   ```java
   BankAccount b;
   ```
5. What are the construction parameters for a BankAccount object?
6. What is default construction?
7. Give Java code to construct the following objects:
   - A square with center (100, 100) and side length 25
   - A bank account with a balance of $5000.
   - A console reader that reads from System.in

   Write just objects, not object variables.
8. Repeat the preceding exercise, but now
   - define object variables that are
   - initialized with the required objects.
9. Find the errors in the following statements:

```java
Rectangle r = (5, 10, 15, 20);
double x = BankAccount(10000).getBalance();
BankAccount b;
b.deposit(10000);
b = new BankAccount(10000);
b.addCoins(new Coin(0.25, "quarters"));
Purse p = null;
p.addCoins(new Coin(0.25, "quarters"));
Purse p = new Purse();
p.addCoins(new Coin());
```

10. Describe all constructors of the `BankAccount` class. List all methods that can be used to change a `BankAccount` object. List all methods that don’t change the `BankAccount` object.

11. What is the value of `b` after the following operations?

```java
BankAccount b = new BankAccount(10);
b.deposit(5000);
b.withdraw(b.getBalance() / 2);
```

12. If `b1` and `b2` store objects of class `BankAccount`, consider the following instructions.

```java
b1.deposit(b2.getBalance());
b2.deposit(b1.getBalance());
```

Are the balances of `b1` and `b2` now identical? Explain.

13. What is the `this` reference?

Next come the answers to these problems.

Try to solve them on your own first, then read the answers.
Warmups Solutions (II)

Questions and answers:

1. Explain the difference between an object and a class.
   A class is like a factory, while an object is like the gadgets that the factory is producing. The difference between a class and an object is that between a factory that produces a certain kind of watches and a watch (of the one kind that the factory can produce) that is produced by that factory. Also, for a given class there could be 0 (zero), 1 (one), or more than one object(s) produced according to the blueprint that the class contains (or defines).

2. Give the Java code for an
   - object of class BankAccount
     ```java
     new BankAccount();
     ```
     and
   - for an object variable of class BankAccount.
     ```java
     BankAccount myChecking;
     ```

3. Explain the differences between an instance variable and a local (or method) variable.
   An instance variable is global to a method over time; an instance variable survives the successive invocations of a method, and exists for as long as the object to which it belongs exists. A local (method) variable on the other hand, is in effect only during the life time of a method invocation: while the method that contains it is in action the variable is used for whatever the method needs it for. When the method is finished the variable is also gone. Another difference is that local variables need to be initialized by the programmer, instance variables are initialized by default by Java.

4. Explain the difference between
   ```java
   new BankAccount(5000); // this is an actual object that is being returned
   ```
   and
   ```java
   BankAccount b; // this is only an object variable that is being declared
   ```
Overall this exercise is exercise 2 in reverse.

5. What are the construction parameters for a `BankAccount` object?
   There are two constructors: one takes no parameters, the other one takes an initial balance.

6. What is default construction?
   Java supplies a default constructor. It has no arguments and does nothing. Its action results in “default construction”: all instance variables remain set to the default values for their types.

7. Give Java code to construct the following objects:
   
   - A square with center (100, 100) and side length 25
     ```java
     new Rectangle(88, 88, 25, 25); // why 88?
     ```
   - A bank account with a balance of $5000.
     ```java
     new BankAccount(5000)
     ```
   - A console reader that reads from System.in
     ```java
     new ConsoleReader(System.in)
     ```

   Write just objects, not object variables.

8. Repeat the preceding exercise, but now define object variables that are initialized with the required objects.

   ```java
   Rectangle mySquare = new Rectangle(88, 88, 25, 25);
   BankAccount myChecking = new BankAccount(5000);
   ConsoleReader console = new ConsoleReader(System.in);
   ```

9. Find the errors in the following statements:

   ```java
   Rectangle r = [new Rectangle](5, 10, 15, 20);
   // this creates a new Rectangle
   double x = [new BankAccount(10000)].getBalance();
   // see how this works? first a new BankAccount object
   // is created, and as soon as it is available we call
   // getBalance on it. The net result is that x will be
   // set to 10000. Given that in this process the newly
   // created BankAccount object is lost, all we can say
   // is that this is a contrived exercise with a certain
   // educational purpose, as there are definitely better
   // ways to set a double variable x to 10000.
   
   BankAccount b;
   b = new BankAccount();
   b.deposit(10000);
   // if you don’t initialize b you can’t use it to deposit
   b = new BankAccount(10000);
   // contrived but legal: b discards previous account
   // and points now to a brand new account of 10000.
   ```
b.addCoins(new Coin(0.25, "quarters"));
// as b is not pointing to a Purse this won't work!

Purse p = null; // fine as long as you don't need p just yet.
[/b]

p = new Purse(); // we create a new Purse and call it p
p.addCoins(6, new Coin(0.25, "quarters"));
// Purse objects know how to add Coins, and we took care
// in creating a Purse so what is wrong? addCoins needs to
// be called with the right number of arguments and in the
// right order. More precisely: an int and a Coin. The int
// parameter was missing here, so we added one.

Purse p = new Purse();
p.addCoins(4, new Coin(0.10, "dimes"));
// Note that Coin does not have a no-args constructor any
// longer!! Also addCoins needs an int parameter in addition,
// and preceding the Coin parameter, by its definition.

10. Describe all constructors of the **BankAccount** class. List all methods that can be used to change a
**BankAccount** object. List all methods that don't change the **BankAccount** object.

```java
public class BankAccount {
    public BankAccount() { // no-arg constructor
        balance = 0;
    }
    public BankAccount(double initialBalance) { // one-arg constructor
        balance = initialBalance;
    }
    public void deposit(double amount) { // mutator method
        balance += amount;
    }
    public void withdraw(double amount) { // mutator method
        balance -= amount;
    }
    public double getBalance() { // accessor method
        return balance;
    }
    private double balance; // private instance variable
}
```

You should, of course, remember (?) that a mutator method is one that changes the values of instance
variables, mutating the state of the object; first two methods do that. The third one is an accessor method,
since it only reports the value of the instance variable, without touching (or changing) it.

11. What is the value of b after the following operations?

```java
BankAccount b = new BankAccount(10);
b.deposit(5000);
b.withdraw(b.getBalance() / 2);
```
12. If b1 and b2 store objects of class BankAccount, consider the following instructions.

```java
b1.deposit(b2.getBalance());
b2.deposit(b1.getBalance());
```

Are the balances of b1 and b2 now identical?
Only if the account pointed to by b2 was empty initially.

Explain.
Let’s say the first account has a balance of $x$ and the second one a balance of $y$. The first statement adds $y$ to $x$ to make the first account’s balance $(x + y)$; the second statement adds this value, the current balance in the first account, to the balance stored by the second account, so the second account becomes $(y + (x + y))$. These two formulas can be equal only when $y$ is 0 (zero) that is, if the second account starts up by being empty.

13. What is the this reference? A reference that points to the object that contains the instance method that uses the reference. Or, as in lab notes 4:

An object of type A

Hope you found this useful.
Problems (II)

Lab Four: This is your Problem Set Two.

Try to solve these problems to practice some of the things you learned this week. In the examples that follow, your program’s answers are always in [blue], to distinguish them from what you would type as a user. Remember: the resulting programs are elementary, and the problems are interesting.

1. Write a program that asks for an initial balance amount. Create a BankAccount object with that amount. Then ask for a deposit amount and a withdrawal amount. Carry out the deposit and withdrawal, then print the remaining balance. Use ConsoleReader from Lab Notes Two, and please place your main method in the class BankAccount.

Here’s a sample run of such a program:

    frilled.cs.indiana.edu%javac BankAccount.java
    frilled.cs.indiana.edu%java BankAccount
    Hello, and welcome to JavaOne Bank.
    An account will be created for you.
    What will the initial balance be?
    Type it now: 
    -40.2
    The current balance in your account is: -40.2
    You now want to make a deposit. How much?
    Type the amount here: 
    120.3
    The current balance in your account is: 80.1
    You now want to make a withdrawal. How much?
    Type it now: 
    34
    The current balance in your account is: 46.09999999999994
    Thanks for using class BankAccount. Good-bye!

    frilled.cs.indiana.edu%

2. Implement a class Employee. An employee has a name (a String) and a salary (a double). Write a default constructor, a constructor with two parameters (name and salary), and methods to return the name and salary. Write a small program to test your class.
Here's a sample run of such a program:

```
frilled.cs.indiana.edu% java Employee
Creating a new employee.
Please type the name:
Larry Bird
Please specify the salary:
200000
New employee has been created.
Name of employee: Larry Bird
Salary: 200000.0
Thank you for testing class Employee.
frilled.cs.indiana.edu%
```

3. Implement a class `Employee`. An employee has a name (a `String`) and a salary (a `double`). Write a default constructor, a constructor with two parameters (name and salary), and methods to return the name and salary. Test your program, then enhance the class by adding a method `raiseSalary(double byPercent)` that raises the employee's salary by a certain percentage.

Here's a sample run of such a program:

```
frilled.cs.indiana.edu% java Employee
Creating a new employee.
Please type the name:
Michael Jordan
Please specify the salary:
300000
New employee has been created.
Name of employee: Michael Jordan
Salary: 300000.0
Raising the salary of Michael Jordan
By what percentage (e.g., 10, 20, etc.)?
10.5
Name of employee: Michael Jordan
Current salary: 331500.0
Thank you for testing class Employee.
frilled.cs.indiana.edu%
```

4. Implement a class `Car` with the following properties. A car has a certain fuel efficiency (measured in miles per gallon or liters per km – pick one) and a certain amount of fuel in the gas tank. The efficiency is specified in the constructor, and the initial fuel level is 0. Supply a method `drive` that simulates driving the car for a certain distance, reducing the fuel level in the gas tank, and methods `getFuelLevel`, returning the current fuel level, and `tank`, to tank up.

Sample usage of the class:
public static void main(String[] args) {
    Car myBeemer = new Car(29);
    System.out.println(myBeemer.getFuelLevel());
    myBeemer.tank(20);
    System.out.println(myBeemer.getFuelLevel());
    myBeemer.drive(100);
    System.out.println(myBeemer.getFuelLevel());
}

Should produce:

    frilled.cs.indiana.edu%java Car
    0.0
    20.0
    16.551724137931036

5. Change the purse program Coins6 (see below) to ask the user to supply coins in a different currency. For example, you can use the following collection of German coins:

    new Coin(0.01, "Pfennig");
    new Coin(0.1, "Groschen");
    new Coin(1.0, "Mark");

What changes did you have to make? What changes would you have to make to the Coins4 (also below) program to change the currency? Which is easier? (Answer these questions in a comment that you can include either at the top or at the bottom of the program you submit).

Here's a sample run of such a program:

    frilled.cs.indiana.edu%java Coins6
    Hello and welcome to Coins6 program.
    Please specify four types of coins. Anything goes.
    Name of coin1 (e.g. groschen, mark, pfennig etc.):
        pfennig
    Value of pfennig (i.e., 0.01, 0.10, 0.25, etc.):
        0.01
    Coin pfennig (0.01) created.
    -------------------------------------------
    Name of coin2:
        groschen
    Value of groschen:
        0.1
    Coin groschen (0.1) created.
    -------------------------------------------
    Name of coin3:
        mark
    Value of mark:
        1
    Coin mark (1.0) created.
6. Add a method `askForCoins(Coin coinType)` to the `Purse` class (see below) that asks the user how many coins of that type to add to the purse and that updates the coin count. Then change `Coins6` (the same one that was mentioned above) and test your new method.

Here's a sample run of such a program:

```
frilled.cs.indiana.edu%java Coins6
Hello and welcome to Coins6 program.
How many pennies can you donate? 3
How many dimes can you donate? 4
How many quarters can you donate? 5
How many dollars can you donate? 6
The total value is: 7.68
frilled.cs.indiana.edu%
```

7. Implement a class `Student`. For the purpose of this exercise, a student has

- a name and
- a total quiz score

Supply an appropriate constructor and methods

- `getName()`
- `addQuiz(int score)`
- `getTotalScore()`, and
- `getAverageScore()`
To compute the latter, you also need to store the number of quizzes that the student took.

Here's a sample run of such a program:

```java
public static void main(String[] args) {
    Student a = new Student("Larry");
    a.addQuiz(10);
    a.addQuiz(9);
    a.addQuiz(8);
    System.out.println("Grade report for: " + a.getName());
    System.out.println("Total score: " + a.getTotalScore());
    System.out.println("Average score: " + a.getAverageScore());
}
```

Should produce:

```
frilled.cs.indiana.edu\%java Student
Grade report for: Larry
<table>
<thead>
<tr>
<th>Total score: 27.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average score: 9.0</td>
</tr>
</tbody>
</table>
```

8. Implement a class Product. A product has

- a name and
- a price

For example

```java
new Product("Toaster", 29.95)
```

Supply methods

- `printProduct()`
- `getPrice()`, and
- `setPrice()`

Write a program that

- makes two products,
- prints them,
- reduces their prices by $5.00, and then
- prints them again

Here's a sample run of such a program:
public static void main(String[] args) {
    Product a = new Product("Toaster", 29.95);
    Product b = new Product("Phone", 35.55);
    a.printProduct();
    a.setPrice(a.getPrice() - 5.00);
    a.printProduct();
    b.printProduct();
    b.setPrice(b.getPrice() - 5.00);
    b.printProduct();
}

Should produce:

frilled.cs.indiana.edu%java Product
Product name: Toaster
Product price: 29.95

Product name: Toaster
Product price: 24.95

Product name: Phone
Product price: 35.55

Product name: Phone
Product price: 30.54999999999997

frilled.cs.indiana.edu%

9. Implement a class Circle that has methods

- getArea()
- getDiameter()

In the constructor, supply the radius of the circle.

Here's a sample run of such a program:

```java
public static void main(String[] args) {
    ConsoleReader c = new ConsoleReader(System.in);
    System.out.println("Please specify the radius of your circle: ");
    Circle a = new Circle(c.readDouble());
    System.out.println("Circle created. ");
    System.out.println("Area: "+a.getArea());
    System.out.println("Circumference: "+a.getCircumference());
    System.out.println("Goodbye!");
}
```

Should produce:
frilled.cs.indiana.edu%java Circle
Please specify the radius of your circle: 1.0
Circle created.
Area: 3.141592653589793
Circumference: 6.283185307179586
Good-bye!

frilled.cs.indiana.edu%java Circle
Please specify the radius of your circle: 3.14
Circle created.
Area: 30.974846927333928
Circumference: 19.729201864543903
Good-bye!

frilled.cs.indiana.edu%

10. Implement a class BeerCan with methods
   
   - getSurfaceArea()
   - getVolume()

   In the constructor, supply the height and radius of the can.

   Here's a sample run of such a program:

   ```java
   public static void main(String[] args) {
       ConsoleReader c = new ConsoleReader(System.in);
       System.out.println("Please specify the height of the BeerCan.");
       double height = c.readDouble();
       System.out.println("Please specify the radius of the BeerCan.");
       double radius = c.readDouble();
       BeerCan b = new BeerCan(height, radius);
       System.out.println("The BeerCan has been created.");
       System.out.println("Its surface area is: " + b.getSurfaceArea());
       System.out.println("Its volume is: " + b.getVolume());
   }
   
   Could produce:

   frilled.cs.indiana.edu%java BeerCan
   Please specify the height of the BeerCan.
   2.5
   Please specify the radius of the BeerCan.
   1.0
   The BeerCan has been created.
   Its surface area is: 15.70963267948966
   Its volume is: 7.85398163397483
   frilled.cs.indiana.edu%java BeerCan
   Please specify the height of the BeerCan.
   ```
Please specify the radius of the BeerCan.

The BeerCan has been created.
Its surface area is: 12.566370614359172
Its volume is: 6.283185307179586
frilled.cs.indiana.edu%

Good luck and don't forget: the harder they seem the more you can learn from them.

But these problems are not difficult. They're just that: good practice.
Review (Part I)

Do you know what a Kanamit is?\(^{20}\)

Respectfully submitted for your perusal - a Kanamit. Height: a little over nine feet. Weight: in the neighborhood of three hundred and fifty pounds. Origin: unknown. Motives? Therein hangs the tale, for in just a moment we’re going to ask you to shake hands, figuratively, with a Christopher Columbus from another galaxy and another time. This is the Twilight Zone.

<table>
<thead>
<tr>
<th>Twenty snapshots.</th>
<th>Twenty diagrams.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start here.</td>
<td>A journey into the fifth dimension.</td>
</tr>
<tr>
<td>A journey into a dimension as vast as space,</td>
<td>...and as timeless as infinity.</td>
</tr>
<tr>
<td>You’ll be back.</td>
<td>Ready to start it all over.</td>
</tr>
<tr>
<td>Only a lot wiser,</td>
<td>…a lot wiser.</td>
</tr>
<tr>
<td>Yeah, wiser...</td>
<td>Well, you already said that.</td>
</tr>
</tbody>
</table>

\(^{20}\)“To Serve Man” (March 2, 1962. Writer: Rod Serling. Director: Richard Bare)
\( \text{translate}(dx, dy) \)

\[
x = x + dx; \\
y = y + dy;
\]
balance

deposit(amount)

balance = balance + amount;

balance

deposit(amount)

balance = balance + amount;
balance

getBalance()

return balance;

balance

getBalance()

return balance;
balance

BankAccount ( )

amount

balance = amount;

height

radius

BeerCan ( , )

getSurfaceArea ( )

getVolume ( )
Solutions (II)

Here are examples of solutions to the problems in the second set:

1. /* This is the file One.java -- which contains the solution to
   the first problem in the second problem set. We just added a main
   method to the BankAccount class which goes through the sequence
   of steps that the problem is asking for. ConsoleReader class is
   also included. Notice that for both classes to be allowed to make
   up this file, with a name of One.java, we need to make sure they
   are not declared public. To run this program compile One.java
   and then run java on BankAccount */

import java.io.*; // needed for ConsoleReader below

class BankAccount {
    double balance;

    BankAccount() {
    }

    BankAccount(double initialBalance) {
        balance = initialBalance;
    }

    void withdraw(double amount) {
        balance = balance - amount;
    }

    void deposit(double amount) {
        balance = balance + amount;
    }

    double getBalance() {
}
return balance;
}

public static void main(String[] args) {

    consoleReader c = new consoleReader(System.in);

    System.out.println("Hello, and welcome to JavaOne Bank.");
    System.out.println("An account will be created for you.");
    System.out.println("What will the initial balance be?"");
    System.out.println("Type it now: ");

    BankAccount b = new BankAccount(c.readDouble());

    System.out.println("The current balance in your account is: 
                        "+ b.getBalance());

    System.out.println("You now want to make a deposit. How much?\n\n"+ "Type the amount here: ");

    b.deposit(c.readDouble());

    System.out.println("The current balance in your account is: 
                        "+ b.getBalance());

    System.out.println("You now want to make a withdrawal. How much?\n\n"+ "Type it now: ");

    b.withdraw(c.readDouble());

    System.out.println("The current balance in your account is: 
                        "+ b.getBalance());

    System.out.println("Thanks for using class BankAccount. Good-bye!");
}

class ConsoleReader {
    public ConsoleReader(InputStream inStream) {
        reader = new BufferedReader(
            new InputStreamReader(
                inStream));
    }

    public String readLine() {
        String inputLine = "";
        try {
            inputLine = reader.readLine();
        } catch (IOException e) {
            System.out.println(e);
            System.exit(1);
    }
}  

return inputLine;
}

public int readInt() {
    String inputString = readLine();
    int n = Integer.parseInt(inputString);
    return n;
}

public double readDouble() {
    String inputString = readLine();
    double x = Double.parseDouble(inputString);
    return x;
}

private BufferedReader reader;

2. /* Second problem on the second problem set. Copy this in a file
   called Two.java then compile the file, and run Employee */

import java.io.*;

class Employee {

    String name;

    double salary;

    Employee() {
        this.name = "Volunteer";
        this.salary = 0.0;
    }

    Employee(String givenName, double initialSalary) {
        name = givenName;
        salary = initialSalary;
    }

    String getName() {
        return name;
    }

    double getSalary() {
        return salary;
    }

    public static void main(String[] args) {

        ConsoleReader c = new ConsoleReader(System.in);

        System.out.println("Creating a new employee.");
        System.out.println("Please type the name:");
String name = c.readLine();
System.out.println("Please specify the salary:");
double salary = c.readDouble();
Employee e = new Employee(name, salary);
System.out.println("New employee has been created.");
System.out.println("Name of employee: " + e.getName());
System.out.println("Salary: " + e.getSalary());
System.out.println("Thank you for testing class Employee.");
}
}

3. /* Third problem on the second problem set. Copy this in a file
called Three.java then compile the file, and run Employee */

import java.io.*;

class Employee {

    String name;
    double salary;

    Employee() {
        name = "Volunteer";
        salary = 0.0;
    }

    Employee(String givenName, double initialSalary) {
        name = givenName;
        salary = initialSalary;
    }

    String getName() {
        return name;
    }

    double getSalary() {
        return salary;
    }

    void raiseSalary(double percentage) {
        salary += 0.01 * percentage * salary;
    }
}
public static void main(String[] args) {
    ConsoleReader c = new ConsoleReader(System.in);

    System.out.println("Creating a new employee.");
    System.out.println("Please type the name: ");
    String name = c.readLine();
    System.out.println("Please specify the salary: ");
    double salary = c.readDouble();
    Employee e = new Employee(name, salary);

    System.out.println("New employee has been created.");
    System.out.println("Name of employee: " + e.getName());
    System.out.println("Salary: " + e.getSalary());
    System.out.println("Raising the salary of " + e.getName());
    System.out.println("By what percentage (e.g., 10, 20, etc.)? ");
    double percentage = c.readDouble();
    e.raiseSalary(percentage);

    System.out.println("Name of employee: " + e.getName());
    System.out.println("Current salary: " + e.getSalary());
    System.out.println("Thank you for testing class Employee.");
}
}

4. /* Fourth problem on the second problem set. Copy this to a file called Four.java then compile the file, and run java Car -- the examples above all use ConsoleReader feel free to use it that way, the test program below is minimal as in the text of the problem. */

class Car {
    double fuelEfficiency;

    double fuel;

    Car(double efficiency) {
        fuelEfficiency = efficiency;
    }

    void drive(double distance) {
        fuel -= distance / fuelEfficiency;
    }
}
double getFuelLevel() {
    return fuel;
}

void tank(double extraFuel) {
    fuel += extraFuel;
}

public static void main(String[] args) {
    Car myBeemer = new Car(29);
    System.out.println(myBeemer.getFuelLevel());
    myBeemer.tank(20);
    System.out.println(myBeemer.getFuelLevel());
    myBeemer.drive(100);
    System.out.println(myBeemer.getFuelLevel());
}

5. /* This is the solution to the fifth problem on the second
   problem set. Comments that answer the question in the text
   of the problem are included. The file name is Five.java and
   to run this you need to compile Five.java and then run java
   on Coins6 which has the main method. */

import java.io.*;

class Coin {
    public Coin(double aValue, String aName) {
        value = aValue;
        name = aName;
    }

    public double getValue() {
        return value;
    }

    public String getName() {
        return name;
    }

    private double value;

    private String name;
class Purse {
    public Purse() {
        total = 0;
    }
    public void addCoins(int coinCount, Coin coinType) {
        double value = coinCount * coinType.getValue();
        total += value;
    }
    public double getTotal() {
        return Math.round(total * 100) / 100.0;
    }
    private double total;
}

class Coins6 {
    public static void main(String[] args) {
        Purse thePurse = new Purse();
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Hello and welcome to Coins6 program.");
        System.out.println("Please specify four types of coins. Anything goes.");
        System.out.println("Name of coin1 (e.g. groschen, mark, pfennig etc.): ");
        String name = console.readLine();
        System.out.println("Value of " + name + " (i.e., 0.01, 0.10, 0.25, etc.): ");
        double value = console.readDouble();
        Coin coin1 = new Coin(value, name); /*
        This is the only thing we need to do to change the types
        of coins that we create. Compared with program Coins4 it
        results in big savings of time and increased reliability
        as in Coins4 we need to update all but three lines in the
program, messages and named constants everywhere.
*/

System.out.println("Coin " + coin1.getName() +
" (" + coin1.getValue() + ") created.");
System.out.println("---------------------------------------------");
System.out.println("Name of coin2:");
name = console.readLine();
System.out.println("Value of " + name + ":");
value = console.readDouble();
Coin coin2 = new Coin(value, name);
System.out.println("Coin " + coin2.getName() +
" (" + coin2.getValue() + ") created.");
System.out.println("---------------------------------------------");
System.out.println("Name of coin3:");
name = console.readLine();
System.out.println("Value of " + name + ":");
value = console.readDouble();
Coin coin3 = new Coin(value, name);
System.out.println("Coin " + coin3.getName() +
" (" + coin3.getValue() + ") created.");
System.out.println("---------------------------------------------");
System.out.println("Name of coin4:");
name = console.readLine();
System.out.println("Value of " + name + ":");
value = console.readDouble();
Coin coin4 = new Coin(value, name);
System.out.println("Coin " + coin4.getName() +
" (" + coin4.getValue() + ") created.");
System.out.println("---------------------------------------------");
System.out.println("How many " + coin1.getName() +
" do you have? ");
int coin1Count = console.readInt();
System.out.println("How many " + coin2.getName() +
" do you have? ");
int coin2Count = console.readInt();
System.out.println("How many " + coin3.getName() +
" do you have? ");
int coin3Count = console.readInt();
System.out.println(" How many " + coin4.getName() + " do you have?");
int coin4Count = console.readInt();

thePurse.addCoins(coin1Count, coin1);
thePurse.addCoins(coin2Count, coin2);
thePurse.addCoins(coin3Count, coin3);
thePurse.addCoins(coin4Count, coin4);

System.out.println("The total value is: " +
thePurse.getTotal());
}

6. /* This is the solution to the sixth problem on the second problem set. Please compare this to the program above. Adding one method to Purse simplifies the collection mechanism as we can delegate more of the user interface to objects of class Purse. */

import java.io.*;

class Coin {
    public Coin(double aValue, String aName) {
        value = aValue;
        name = aName;
    }

    public double getValue() {
        return value;
    }

    public String getName() {
        return name;
    }

    private double value;

    private String name;
}

class Purse {
    public Purse() {
        total = 0;
    }

    public void addCoins(int coinCount, Coin coinType) {
    double value = coinCount * coinType.getValue();
    total += value;
}

public double getTotal() {
    return Math.round(total * 100) / 100.0;
}

private double total;

void askForCoins(Coin coinType) {
    ConsoleReader console = new ConsoleReader(System.in);
    System.out.println("How many " + coinType.getName() + " can you donate?");
    int coinCount = console.readInt();
    addCoins(coinCount, coinType);
}
}

class Coins6 {
    public static void main(String[] args) {
        Purse thePurse = new Purse();
        System.out.println("Hello and welcome to Coins6 program.");

        thePurse.askForCoins(new Coin(0.01, "pennies"));
        thePurse.askForCoins(new Coin(0.1, "dimes"));
        thePurse.askForCoins(new Coin(0.25, "quarters"));
        thePurse.askForCoins(new Coin(1.00, "dollars"));

        System.out.println("The total value is: " + thePurse.getTotal());
    }
}

7. /* This is file Seven.java containing the solution to problem seven on the second set of problems. To run this compile the file Seven.java and then run java Student. Notice the instance variable quizzes which is only hinted at in the text of the problem. */

    class Student {
        String name;
        double totalQuizScore;
        Student(String givenName) {
name = givenName;
}

String getName() {
    return name;
}

void addQuiz(int score) {
    quizzes += 1;
    totalQuizScore += score;
}

int quizzes;

double getTotalScore() {
    return totalQuizScore;
}

double getAverageScore() {
    return totalQuizScore / quizzes;
}

public static void main(String[] args) {
    Student a = new Student("Larry");
    a.addQuiz(10);
    a.addQuiz(9);
    a.addQuiz(8);

    System.out.println("Grade report for: " + a.getName());
    System.out.println("Total score: " + a.getTotalScore());
    System.out.println("Average score: " + a.getAverageScore());
}

8. /* This is the solution to problem eight on the second set
of programs. The name of the file is Eight.java -- compile
it, then run java on Product (which has the main method). */

class Product {
    String name;
    double price;

    Product(String givenName, double givenPrice) {
        name = givenName;
        price = givenPrice;
    }
}
void printProduct() {
    System.out.println("Product name: ", name);
    System.out.println("Product price: ", price);
    System.out.println("--------------------------------");
}

double getPrice() {
    return price;
}

void setPrice(double newPrice) {
    price = newPrice;
}

public static void main(String[] args) {
    Product a = new Product("Toaster", 29.95);
    Product b = new Product("Phone", 35.55);
    a.printProduct();
    a.setPrice(a.getPrice() - 5.00);
    a.printProduct();
    b.printProduct();
    b.setPrice(b.getPrice() - 5.00);
    b.printProduct();
}

9. /* This is the solution to problem nine on the second set of problems. Copy this in a file called Nine.java and then compile Nine.java -- then run Circle. ConsoleReader has been included for the user of your program to specify the radius of the circle that (s)he is going to create interactively. */

import java.io.*;

class Circle {
    private double radius;

    Circle (double givenRadius) {
        radius = givenRadius;
    }

    double getArea() {
        return Math.PI * radius * radius;
    }
}
double getCircumference() {
    return Math.PI * radius * 2;
}

public static void main(String[] args) {
    ConsoleReader c = new ConsoleReader(System.in);
    System.out.println("Please specify the radius of your circle:");
    Circle a = new Circle(c.readDouble());
    System.out.println("Circle created. ");
    System.out.println("Area: ", + a.getArea());
    System.out.println("Circumference: ", + a.getCircumference());
    System.out.println("Good-bye!");
}

10. /* This is the solution to problem ten on the second problem
    set. Put the code in a file called Ten.java and compile the
    file. Then run java BeerCan as the BeerCan class defines the
    main method. ConsoleReader class included for convenience. */

import java.io.*;

class BeerCan {
    double height, radius;

    BeerCan(double givenHeight, double givenRadius) {
        height = givenHeight;
        radius = givenRadius;
    }

    double getSurfaceArea() {
        return Math.PI * 2 * radius * height;
    }

    double getVolume() {
        return height * Math.PI * radius * radius;
    }

    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.println("Please specify the height of the BeerCan.");
    }
}
double height = c.readDouble();
System.out.println("Please specify the radius of the BeerCan.");
double radius = c.readDouble();
BeerCan b = new BeerCan(height, radius);
System.out.println("The BeerCan has been created.");
System.out.println("Its surface area is: " + b.getSurfaceArea());
System.out.println("Its volume is: " + b.getVolume());

So, as always, I hope you found this useful.
Homework Two

A Simple Robot.

You have to define a class Robot that describes a robot very similar to the one you worked with in the first lab. To clarify how easy this problem is we need to say that metaphorically your robot is:

- a Penguin
- a Tigger
- a BankAccount
- a Rectangle
- a ConsoleReader
- a String

all at the same time. The Robot needs to be able to:

- turnLeft (90 degrees)
- moveForward (one step)

The Robot also needs to remember its location.

Two methods:
- getx(), and
- gety()

can be used to report the current values of the robot’s location.

The location of the robot is a pair \((x, y)\) of integers, that describes the position of the robot.

When you create a Robot one need specify:

- the location \((x, y)\)
- a name for the robot
- the direction the robot is facing

The robot also needs to be able to:
• report the direction() it's facing (N, S, W, E)

• produce a complete report() (x, y, name, direction)

Here's a sample program with two robots:

class Walk {
    public static void main(String[] args) {
        Robot a = new Robot("Alice", 2, 3, "North");
        a.report();
        Robot q = new Robot("Queen", -4, -1, "West");
        q.report();
        a.turnLeft();
        a.report();
        a.moveForward();
        a.report();
        a.turnLeft();
        a.report();
        a.moveForward();
        a.report();
        a.moveForward();
        a.report();
        q.moveForward();
        q.report();
        q.turnLeft();
        q.report();
    }
}

This would produce the following output:

    frilled.cs.indiana.edu%java Walk
    Robot Alice located at (2, 3) facing North
    Robot Queen located at (-4, -1) facing West
    Robot Alice now turns left.
    Robot Alice located at (2, 3) facing West
    Robot Alice now moves forward.
    Robot Alice located at (1, 3) facing West
    Robot Alice now turns left.
    Robot Alice located at (1, 3) facing South
    Robot Alice now moves forward.
    Robot Alice located at (1, 4) facing South
    Robot Alice now moves forward.
    Robot Alice located at (1, 5) facing South
    Robot Alice now moves forward.
    Robot Alice located at (1, 6) facing South
    Robot Queen now moves forward.
    Robot Queen located at (-5, -1) facing West
    Robot Queen now turns left.
Robot Queen located at (-5, -1) facing South
frilled.cs.indiana.edu%

Your task is to write Robot.java that describes the robots.

Please take a moment to think about the referential we use in this problem which is the same one that we use in computer graphics (see Lecture Notes Three if you will) which is also the reason for which Tigger is (let’s say) upside down on the main page of this web site (first page of Prelude anyway). As always, please let us know if you have any questions, of course.

Grading scale.

- 10 points for providing the constructor(s) needed
- 25 points for your Robot being able to store the location, name, and direction facing
- 15 points for reporting the location \((x, y)\), the direction, and the name individually
- 5 more points for putting all of that together in a method report(), and
- 40 points for your Robot’s ability to turnLeft() and moveForward() as shown

That’s a total of 95 (which is the highest A). For more come up with something worth it.

Let us know if you have any questions or if you need any help.
# Review (Part II)

*Classes, objects, constructors.*

<table>
<thead>
<tr>
<th>There is only one remaining issue with the <code>BankAccount</code>.</th>
<th>We need to define the default constructor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why do we need to talk about the default constructor?</td>
<td>Because that’s the one we’re using now, not having defined any constructor whatsoever yet.</td>
</tr>
<tr>
<td>Constructors are not methods, but they are used to create instances of the class (objects).</td>
<td>Many classes have more than one constructor.</td>
</tr>
<tr>
<td>The purpose of a constructor is to initialize the instance variables of the object.</td>
<td>The code for a constructor sets the initial state of the object.</td>
</tr>
<tr>
<td>When a <code>BankAccount</code> object comes into existence it will have an initial state with the current balance being 0 (zero).</td>
<td>So there’s just one constructor for the class.</td>
</tr>
<tr>
<td>Since it does not take any arguments it is called a no-arg constructor.</td>
<td>Its purpose is to initialize the instance variables of a bank account object when the object is created.</td>
</tr>
<tr>
<td>Objects of type <code>BankAccount</code> only have one instance variable: their <code>balance</code>.</td>
<td>If you do not initialize an instance variable that is a number it will automatically be initialized to 0 (zero) by Java, before the constructor even comes into play.</td>
</tr>
<tr>
<td>In this regard, instance variables act differently than local variables!</td>
<td>By local variables you mean &quot;variables declared in methods, such as in main&quot;, right?</td>
</tr>
<tr>
<td>Yes. Those have to be initialized by the programmer before they are used.</td>
<td>It’s not the same with instance variables.</td>
</tr>
<tr>
<td>Instance variables are set by Java to a default value.</td>
<td>Local variables will not.</td>
</tr>
<tr>
<td>No they won’t be initialized by Java. Instance variables, however, will be initialized by Java if you don’t initialize them (as a programmer).</td>
<td>OK. If the <code>balance</code> of a new account will be set to 0 (zero) even before the constructor starts working, then the constructor need not do anything.</td>
</tr>
</tbody>
</table>
Yes. And Java will always provide (by default) a no-arg constructor that doesn’t do anything, for every class that you define.

So if you don’t define any constructor you will be given one, by default.

Yes, and if you define at least one, those that you define are your constructors.

How do we write a constructor?

They are essentially initialization procedures so they look very similar to methods. They have a header, and a method body.

Their header contains an access specifier, but not a return type.

Their name is always that of the class.

Like methods they have a list of parameters: named locations of a certain type, in which their initial information is placed.

That is, the arguments.

Indeed. Can I see one?

Here’s the one that you get by default, if you don’t specify any constructor.

It’s the one in blue. Note that the body of the constructor is empty.

```java
public class BankAccount {
    public BankAccount() { // nothing
    }
    public void deposit(double amount) {
        balance = balance + amount;
    }
    public void withdraw(double amount) {
        balance = balance - amount;
    }
    public double getBalance() {
        return balance;
    }
    private double balance;
}
```

Since this is the default constructor that means I get it for free.

Indeed, but you might actually write it anyway to not forget that you can use it.

Can I define a second constructor?

How would you want to use it?

I’d like to create an account with an initial sum of money in it, like this:

Yes you can. You will only need to set balance to the initial value inside the constructor.

```java
BankAccount m = new BankAccount(300.00);
```
Is that it?

```java
public BankAccount(double initial) {
    balance = initial;
}
```

Yes, can you describe it a little?

It is used to create a bank account with an initial balance.

When you call it you need to specify that amount, like you did when you showed me the way you intended to use it.

A constructor looks like a method, only the header does not have a return type, and the name of a constructor is the name of the class.

The rest of it is just like a method.

Yes, so I defined a formal parameter initial which must be of floating-point type (that is, with a fractional part). I chose `double` for the type of the formal parameter.

So in your previous example this constructor gets called to create a new object, and the initialization steps start by storing `300.00` in a location of type `double` by the name `initial`.

Yes, and in its body I use `initial` to copy its value in `balance`.

Very well. You could have used it in a more involved way, but there was no need for you to do that.

There's a tricky rule in Java about the default no-arg constructor.

We mentioned it above, but in an implicit way.

We can avoid mentioning it here by stating another rule, that is easier to state (and remember).

Always declare all the constructors that you need.

And how's the actual rule?

The default constructor is provided by default when there are no constructors specified. If you specify at least one constructor, the default constructor no longer is provided and if you need it you need to write it explicitly in the class.

I see, so this class definition won't let me create bank accounts with an initial value of 0 (zero)?

Not directly.

```java
public class BankAccount {
    double balance;
    public BankAccount(double initial) {
        balance = initial;
    }
}
```

So I can't say

```java
BankAccount m = new BankAccount();
```

No, but you can create it this way:

```java
BankAccount m = new BankAccount(0.0);
```
Well, can we put the class to work? Sure, we did that last time, we can do it again. Here’s a different test program though:

```java
public class BankAccountTest {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(10000);
        final double INTEREST_RATE = 5;
        double interest;
        // compute and add interest for one period
        interest = account.getBalance() * INTEREST_RATE / 100;
        account.deposit(interest);
        System.out.println("Balance after the first year is "+ account.getBalance());
        // add interest again
        interest = account.getBalance() * INTEREST_RATE / 100;
        account.deposit(interest);
        System.out.println("Balance after the second year is "+ account.getBalance());
    }
}
```

And the class is still the same as last time, with the two constructors added, ...the no-arg empty constructor and the one that initializes the balance to a certain initial value, that is specified when you call the constructor.

Yes, here it is:

```java
public class BankAccount {
    private double balance; // instance variable, the account balance
    public BankAccount() { // the no-arg empty constructor
    }
    public BankAccount(double initial) { // another constructor
        balance = initial;
    }
    public void deposit(double amount) { // instance method deposit
        balance = balance + amount;
    }
    public void withdraw(double amount) { // instance method withdraw
        balance = balance - amount;
    }
    public double getBalance() { // instance method getBalance
        return balance;
    }
}
```

Once again to see this in action I need to copy the code in two files, one for the bank account class and the other one for the bank account test class (that has the main method). Then you compile them and run the test class.
Can we summarize now?  
We sure can.

I have a summary with two short examples.  
Let’s see them.

Objects are entities that can have memory and specific behaviour. Their memory is represented by variables that they have inside and their behaviour is defined by actions that they know how to perform

...that is, the methods that are associated with those objects.

All objects of the same kind, that have the exact same structure, make up a class.

In fact, in programming it’s always the other way around: one first defines a class,

which describes how that particular class of objects will look and behave (what methods they have),

...then one creates as many objects (of that kind) as needed

...and lets them loose,

...thus running the program.

To better clarify instance variables and instance methods let’s look at two examples.

Each one will resemble a short play (as in a stage representation of an action or a story).

Our dramatic compositions will be simple, since we will abstract away all the unwanted details.

The titles of the plays will be:

• Sports, and

• Babies.

Better than "You are old Father William" already.  
OK, let’s look at the first one.

A Hoosier basketball fan’s simple to describe:

...she cheers, by shouting ’Go Hoosiers!’ when she feels like cheering for the former team of Bob Knight, and that’s the end of it.

Write a short program (a play) that presents three Hoosier fans cheering for the IU Hoosiers,

.. each fan cheering once, and in no particular order.

Here’s how the program should behave:  
The output of the program is in blue:

```
tucotuco.cs.indiana.edu% javac Sports.java
tucotuco.cs.indiana.edu% java Sports
Go Hoosiers!
Go Hoosiers!
Go Hoosiers!
tucotuco.cs.indiana.edu%
```

At a basketball game the noise is so loud that you don’t know who is cheering and when.

The crowd is anonymous, more or less. Here’s the object oriented implementation of this play:

```
public class Sports {
    public static void main(String[] args) {
```
Hoosier a = new Hoosier();
Hoosier b = new Hoosier();
Hoosier c = new Hoosier();
a.cheer();
b.cheer();
c.cheer();
}
}

class Hoosier {
    void cheer() {
        System.out.println("Go Hoosiers!");
    }
}

Just a quick question: for all practical purposes cheering here essentially means printing, right?

...which reads (in this case): "Go Hoosiers!".

...and in only one way.

So we see that a Hoosier is an object that knows only one thing: to cheer,
The objects’ behaviour is defined by their methods, and since each object is
...an instance of a class the methods themselves are called instance methods.

OK. Here's the second play, Babies. We won't have much time for that.

Here's the play:

tucotuco.cs.indiana.edu% javac Babies.java
tucotuco.cs.indiana.edu% java Babies
Alice: Hello, my name is Alice
Susan: Hello, my name is Susan
Jimmy: Hello, my name is Jimmy
tucotuco.cs.indiana.edu%

Looks good to me.

Here's the screenplay and the cast.

public class Babies {
    public static void main(String[] args) {
        Baby a = new Baby("Alice");
        Baby b = new Baby("Susan");
        Baby c = new Baby("Jimmy");
a.talk();
b.talk();
c.talk();

}

}

class Baby {
    String name; // instance variable

    Baby(String givenName) { // constructor
        name = givenName;
    }

    void talk() { // instance method
        System.out.println(name + " : Hello, my name is " + name);
    }
}

Oh, but I think I understand instance variables now. Oh, but I am sure you do.
Methods

Wrap-up of Classes and Objects material.

Let’s go through a set of examples to clarify classes and objects even further.

1. What’s this program doing?
   Can you draw a diagram to illustrate what happens when you run it?

   ```java
   public class One {
       public static void main(String[] args) {
           Potato p = new Potato();
           Potato q;
           q = new Potato();
           p = q;
       }
   }
   class Potato {
   }
   How do the Potatoes get created?
   ```

2. What is this next program doing? Can you diagram it? What’s new?

   ```java
   public class Two {
       public static void main(String[] args) {
           Pair u = new Pair();
           Pair v;
           v = new Pair();
           u.a = 1;
           u.b = 2;
           u.a = u.a + u.b;
           u.b = 1 - u.a;
       }
   }
   class Pair {
       int a;
       int b;
   }
   ```
3. Why does the next program not compile?
   What’s wrong with it?

   public class Three {
       public static void main(String[] args) {
           Pair u = new Pair();
           Pair v = new Pair(1, 2);
       }
   }

   class Pair {
       int a;
       int b;
   }

   Can you fix it?

4. What does the next program print and why (or how).

   public class Four {
       public static void main(String[] args) {
           Calculator m = new Calculator();
           int value = m.fun(3);
           System.out.println(value);
       }
   }

   class Calculator {
       int fun(int x) {
           int result;
           result = 3 * x + 1;
           return result;
       }
   }

   Same question if we change the Calculator as follows:

   class Calculator {
       int fun(int x) {
           int result;
           result = (x) + 1;
           return result;
       }
       int g(int x) {
           int result;
           result = 3 * x;
           return result;
       }
   }

5. What does the following program print and why (or how)?
public class Five {
    public static void main(String[] args) {
        Calculator c = new Calculator();
        int value = c.fun(1) + c.fun(c.fun(2));
        System.out.println(value);
    }
}

class Calculator {
    int fun(int x) {
        int result;
        result = 3 * x + 1;
        return result;
    }
}

6. Same question about this one:

public class Six {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        int value = calc.fun(
            calc.fun(
                calc.fun(
                    calc.fun(calc.fun(5)))));
        System.out.println(value);
    }
}

class Calculator {
    int fun(int x) {
        int result;
        if (x % 2 == 0)
            result = x / 2;
        else
            result = 3 * x + 1;
        return result;
    }
}

7. What’s the output of the following program and why?

public class Seven {
    public static void main(String[] args) {
        Oracle a = new Oracle();
        System.out.println(a.odd(5));
        System.out.println(a.odd(6));
        System.out.println(a.odd(7));
        System.out.println(a.odd(8));
        System.out.println(a.odd(9));
    }
}
class Oracle {
    boolean odd(int n) {
        boolean result;
        if (n % 2 == 0) {
            result = false;
        } else {
            result = true;
        }
        return result;
    }
}

8. Let’s now review a previous step once again.
What’s the output of this program and why (or how)?

public class Eight {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        int value = calc.fun(
            calc.fun(
                calc.fun(
                    calc.fun(27))));
        System.out.println(value);
    }
}

class Calculator {
    int fun(int x) {
        int result;
        if (x % 2 == 0)
            result = x / 2;
        else
            result = 3 * x + 1;
        return result;
    }
}

What’s the output of this program and why (or how)?

public class Nine {
    public static void main(String[] args) {
        Alien x = new Alien();
        int value = x.what(4);
        System.out.println(value);
    }
}
class Alien {
    int what(int x) {
        int result;
        if (x == 1) {
            result = 1;
        } else {
            result = x + what(x - 1);
        }
        return result;
    }
}

10. What’s a good name for the method what?

11. What’s the output of this program?

    public class Eleven {
        public static void main(String[] args) {
            Alien x = new Alien();
            int value = x.what(10);
            System.out.println(value);
        }
    }
    class Alien {
        int what(int x) {
            int result;
            if (x == 1) {
                result = 1;
            } else {
                result = x + what(x - 1);
            }
            return result;
        }
    }

12. What’s the output of this program and why (or how)?

    public class Twelve {
        public static void main(String[] args) {
            Alien x = new Alien();
            int value = x.what(10);
            System.out.println(value);
        }
    }
    class Alien {
        int what(int x) {
            int result;
            if (x == 1) {
                result = 1;
            } else {

13. What's the output of this program and why (or how)?

```java
public class Thirteen
    public static void main(String[] args)
    Alien x = new Alien();
    int value = x.what(10);
    System.out.println(value);

class Alien {
    int what(int x) {
        int result;
        if (x == 1) {
            result = 1;
        } else {
            result = x + what(x - 1);
            System.out.println(x);
        }
        return result;
    }
}
```

14. What's the output of this program and why?

```java
public class Fourteen {
    public static void main(String[] args) {
        A a = new A();
        a.fun();
        a.fun();
        a.fun();
        System.out.println(a.n);
    }
}
class A {
    int n;
    void fun() {
        n += 1;
    }
}
```

15. What's the output of this program and why?
public class Fifteen {
    public static void main(String[] args) {
        Vegetable tomato = new Vegetable();
        tomato.f();
        tomato.g();
        System.out.println(tomato.n);
        tomato.g();
        tomato.g();
        System.out.println(tomato.n);
    }
}

class Vegetable {
    int n;
    void f() {
        n = n + 1;
    }
    void g() {
        n = n + 1;
    }
}

16. What is the output of this program and why?

public class Sixteen {
    public static void main(String[] args) {
        Vegetable tomato = new Vegetable();
        Vegetable potato = new Vegetable();
        tomato.fun();
        tomato.fun();
        potato.fun();
        potato.fun();
        potato.fun();
        potato.fun();
    }
}

class Vegetable {
    int n;
    int m;
    void fun() {
        n = n + 1;
        m = m + 1;
        System.out.println("n = " + n + ", m = " + m);
    }
}

17. What is the output of this program and why?
public class Seventeen {
    public static void main(String[] args) {
        Vegetable tomato = new Vegetable();
        Vegetable potato = new Vegetable();
        tomato.fun();
        potato.fun();
        tomato.fun();
        potato.fun();
        potato.fun();
        potato.fun();
        tomato.fun();
    }
}

class Vegetable {
    int n;
    static int m;
    void fun() {
        n = n + 1;
        m = m + 1;
        System.out.println("n = " + n + ", m = " + m);
    }
}

18. Please describe briefly what this picture represents:

```
instance variable

class A {
    int x;
    void fun(int y) {
        int z = 3;
        x = z + y;
    }
}

formal parameter

better write it as this.x

local variable
```

Define all terms involved, as briefly and completely as you can.

19. Take a look at the diagram below:
An object of type \( A \)

Then please answer the following question: What’s this?

20. More generally the picture looks like this: Picture "http://www.cs.indiana.edu/classes/a201-dger/sum2000/lectures/elev...\]

Can you summarize the diagram briefly?

What comes next is your: A201/A597 LAB ASSIGNMENT FIVE

What you need to do:

- answer all 20 questions
- on a piece of paper (one sheet front and back)
- as succinctly yet as completely and legibly as possible,
- then turn it in lecture to me instead of the minute paper

Please do not write more than one sheet of paper front and back.

Please write legibly so I can understand, this will be 40\% of your Lab Five grade!

This will also determine what the AIs will ask you during the lab, so the more answers you get right the less questions you will be asked during the lab (but you will still be asked something, surely). In your papers try to give honest, direct answers.

After I collect your papers

- I will grade them
- then give them to your AIs
- and will ask them to return them to you during the lab
I will ask the AIs to emphasize those questions that gave you most trouble.
(We will identify them on an individual basis).

In addition, please explain why this doesn’t work. Then fix it in two different ways.

```java
public class Math {
    public static void main(String[] args) {
        int a = 5;
        int b = 7;
        System.out.println(Math.max(a, b));
    }
}
```

If you have any questions or need any help please let us know!
Decisions

Branches and paths. If statement exercises. (L)oops.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s the purpose of if statements?</td>
<td>It allows us to include decisions in our programs.</td>
</tr>
<tr>
<td>What do we do with their results?</td>
<td>We can branch our course of action.</td>
</tr>
<tr>
<td>How do you code this branching situation?</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of branching structure]

Use the space above.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Did you remember to use curly braces?</td>
<td>I try to do this as often as possible, it helps me keep all my branches distinct.</td>
</tr>
<tr>
<td>That’s good practice.</td>
<td>Indeed, even though sometimes I don’t really need the brackets.</td>
</tr>
</tbody>
</table>
How would you code this slightly modified situation?

I see there's one *minor* change.

It may look minor on the diagram. I see... Now I *have to* use brackets.

Yes, there's no way around it. Otherwise the branches tangle.

In general, in our programs, we have the flowchart clear in mind and we just need to translate it into Java. But we need to do that with care, as illustrated above, and curly braces are more than just syntactic sugar in Java.

Occasionally we will have to do the reverse, ... that is, build the diagram out of Java text,

... when we read someone else's code. Let's see some examples.

OK, here's a bigger one. I can hardly wait.

**Bigger Example One**

Assume that *option* is an int variable that can only be: 1, 2 or 3 before each of the following code fragments start executing.

So this is *given.* Yes. Now the question.

Which of the fragments below set variable *i* to the same value that *option* has? We will look at this kind of problems with the help of diagrams, as announced before.
The (given) code is on the left, ...the diagram is on the right, in all examples below.

**Code**

```java
if (option == 1)
    i = 1;
else if (option == 2)
    i = 2;
else
    i = 3;
```

**Diagram**

![Diagram](image)

In this particular case it’s easy to see that the code sets i to the same value as `option` ...given the assumption about the possible values that `option` can have. We just explore all paths.

For the remaining of these notes we only sketch the diagrams here, ...and fill them with text in class.

**Code**

```java
i = 1;
if (option >= 2)
    i = i + 1;
if (option == 3)
    i = i + 1;
```

**Diagram**

![Diagram](image)
<table>
<thead>
<tr>
<th>Code</th>
<th>Diagram</th>
</tr>
</thead>
</table>
| if (option == 1)  
  i = 1;  
if (option == 2)  
  i = 2;  
else  
  i = 3; | ![Diagram 1](image1) |

Don’t forget we still need to provide an answer with each one of the diagrams.

<table>
<thead>
<tr>
<th>Code</th>
<th>Diagram</th>
</tr>
</thead>
</table>
| i = 1;  
if (option > 1)  
  if (option > 2)  
    i = 3;  
else  
    i = 2; | ![Diagram 2](image2) |

Indeed we need to say whether the code fragments presented are (or not) equivalent to

\[
i = \text{option};
\]

...which is the equivalent code.

If the equivalent code is *that* simple, why do we go through all this trouble, to rephrase a simple assignment statement?

We chose a simple situation to practice if statements on it. Don’t worry, I have a list of suggested exercises indexed at the end of these notes, below. Those are *real iffy* situations.

Very good. So we keep things simple while we learn the concept, and we use diagrams to reason about these simple, illustrative programs.

Indeed, we rely on diagrams for the time being. With time we won’t need them, as you’ll be able to see them without drawing them, but for beginners they seem to be more tangible.
Here's the last case:

**Code**

```plaintext
i = 1 if (option == 1);
i = 2 if (option == 2);
i = 3 if (option == 3);
```

**Diagram**

Incorrect syntax, no diagram.

But, you get a *Picasso*.

Time to move on. ✔

**Bigger Example Two**

Consider this code fragment

```plaintext
if (x > y)  
z = z + 1;
else  
z = z + 2;
```

Which one of the following (code fragments) are equivalent to it? Two code fragments are be considered *equivalent* when they behave in the same way.

Two code fragments are to be considered *equivalent* ...when they behave in the exact same way.

That is they have

- the same output, and
- same sequence of internal states

...over their all possible inputs.

Note that the structure of the diagram is as important as what gets written inside the boxes.

We sketch the diagrams below.

And leave the reasoning to you.
Code

```java
if (x > y)
    z = z + 1;
if (x <= y)
    z = z + 2;
```

Diagram

So, what’s the answer? Those paths are not independent, I don’t think.

How about this next one, then?

Code

```java
if (! (x > y))
    z = z + 2;
else
    z = z + 1;
```

Diagram

This one is much easier to think about, no doubt about it. But careful thinking is required when the problem is posed, as it is, with its upside down.

How about this next one, then? Not harder, but somewhat unfamiliar…

Code

```java
z = z + 1;
if (x <= y)
    z = z + 1;
```

Diagram

Code has been factored out.
How about this next one, then?

**Code**

```java
if (x > y)
    z = z + 1;
else if (x <= y)
    z = z + 2;
```

**Diagram**

If the last one was *efficient* this one’s *redundant.*

How about this next one, then?

**Code**

```java
z = z + 2;
if (!((x > y))
    z = z - 1;
```

**Diagram**

The approach looks a bit *contortive* this time.

Like this?

No, that’s *circular,* not even *recursive.*

I think it’s more like this:

That, my dear friend, is Paul Klee. With your permission.
**Bigger Example Three**

Consider the following two program fragments:

**Fragment 1**

```java
if (x == 5)
    x = x + 1;
else
    x = 8;
```

**Fragment 2**

```java
if (x == 5)
    x = x + 1;
else if (x != 5)
    x = 8;
```

Check all that apply:

- The two fragments are logically equivalent
- Fragment two contains a syntax error.
- If \( x \) is 6 initially then
  - the value in \( x \) is 8 after executing fragment one,
  - the value in \( x \) is 6 after executing fragment two.
- \( x \) always has the value 8 after executing fragment two.
- \( x \) has either the value 5 or the value 8 after executing fragment one.

I think this is a bit involved.

Very well, then, here’s a hint.

There's more to it than meets the eye.

That much is clear.

**Bigger Example Four**

Assume that \( x \) and \( y \) are integer variables.

Then consider the following nested if statement.

```java
if (x > 3)
    if (x <= 5)
        y = 1;
    else if (x != 6)
        y = 2;
    else y = 3;
else y = 4;
```

If \( y \) has the value 2 after executing the above program

...then what do you know about \( x \)?

Did you have to draw a diagram?
Bigger Example Five

Assume that $x$ and $y$ are integer variables, and consider the code fragment shown below:

```java
if (x > 3) {
    if (x <= 5)
        y = 1;
    else if (x != 6)
        y = 2;
} else
    y = 3;
```

I hope you noticed the difference between it and the previous one (Bigger Example Four).

You [bet].

Now the questions.

Question 1. If $x$ is 1 before the fragment gets executed what’s the value of $y$ after the fragment is executed? (Is it possible to give an answer to this question?)

Question 2. Now erase the curly braces. What value must $x$ have before the fragment gets executed, for $y$ to be 3 at the end of the fragment?

What do you think about these problems? They make for good practice.

Would you happen to have more? More? You mean, real problems, programs?

Yes, to practice even further. I actually do.

Very well, where are they? Here’s a set of warmup problems of the kind you seem to be looking for.

http://www.cs.indiana.edu/classes/a201/sum2002/notes/ifs.html
Here are my solutions. You’re pretty quick, aren’t you?


Yes, and ready for more. Then here are some programming problems of the kind you seem to be looking for.

http://www.cs.indiana.edu/classes/a201/sum2002/notes/pIfs.html

And I think you should work them all out.

http://www.cs.indiana.edu/classes/a201/sum2002/notes/pIfsSol.html

Weren’t we supposed to start loops today.

Yes, and we can still do it. Then let’s go for it!

What’s the motivation? Remember the investment problem from the first week? It was presented in lecture notes two.

Here’s the code in Java, most of it. What is in [red] and what is in blue?

do\text{uble } \text{balance} = 10000; 
\text{int } \text{year} = 0; 
\text{year} = \text{year} + 1; 
\text{balance} = \text{balance} + \text{balance} \times 0.05; 
\text{System.out.println("Year: "+year);}

The code in blue should be executed only once. The part in red looks like what one would want to have done repeatedly until the balance becomes big enough, or doubles (or reaches 20,000).

In Java there is a while statement which is able to do the iteration for us. Like if is a composite statement. With it one only need specify the red part, with an indication of when to stop. Or, rather, for how long it should go on with the computing of it.

Here’s the code:

do\text{uble } \text{balance} = 10000; 
\text{int } \text{year} = 0; 
\text{while (balance < 20000) \{ } 
\text{year} = \text{year} + 1; 
\text{balance} = \text{balance} + \text{balance} \times 0.05; 
\text{\}} 
\text{System.out.println("Year: "+year);}
Can I draw a flowchart for this?

You sure may.

Here it is:

```
START

year = 0

while { balance < 20000 }

    add interest
    increment year

print year

STOP
```

It takes a while to understand all this, ...but once you do it you’re home free.
Warmups (III)

Decisions practice problems.

Questions:

1. Find the errors in the following if statements.

```java
if quarters > 0 then System.out.println(quarters + "quarters");
if (1 + x > Math.pow(x, Math.sqrt(2))) y = y + x;
if (x = 1) y++; else if (x = 2) y = y + 2;
if (x && y == 0) p = new Point2D.Double(x, y);
if (1 <= x <= 10)
   { System.out.println("Enter y:");
     y = console.readDouble();
   }
if (s != "nickels" || s != "pennies"
    || s != "dimes" || s != "quarters")
   System.out.print("Input error!");
if (input.equalsIgnoreCase("N") || "NO")
   return;
int x = console.readDouble();
if (x != null) y = y + x;
language = "English";
if (country.equalsIgnoreCase("USA"))
   if (state.equalsIgnoreCase("PR")) language = "Spanish";
else if (country.equalsIgnoreCase("China"))
   language = "Chinese";
```

2. Explain the following terms and give an example for each construct:
• Expression
• Condition
• Statement
• Simple statement
• Compound statement
• Block

3. Explain the difference between an if/else /else statement and nested if statements. Give an example for each.

4. Give an example for an if/else /else statement where the order of the tests does not matter. Give an example where the order of the tests matter.

5. Of the following pairs of strings, which comes first in lexicographic order?

"Tom", "Dick"
"Tom", "Tomato"
"church", "Churchill"
"car manufacturer", "carburetor"
"Harry" hairy"
"C++", "Car"
"Tom", "Tom"
"Car", "Carl"
"car", "bar"

6. Complete the following truth table by finding the truth values of the Boolean expressions for all combinations of the Boolean inputs p, q, and r.

| p   | q   | r   | (p && q) || !r | !(p && (q || !r)) |
|-----|-----|-----|--------|----|-----------------|
| false| false| false| false  |    |                 |
| false| false| true |       |    |                 |
| false| true |     |       |    |                 |

7. True or false: A && B is the same as B && A for any Boolean conditions A and B?

8. Explain the difference between
s = 0;
if (x > 0) s++;
if (y > 0) s++;

and

s = 0;
if (x > 0) s++;
else if (y > 0) s++;

9. Use De Morgan’s law to simplify the following Boolean expressions.

!(x > 0 && y > 0)
!(x != 0 || y != 0)
!(country.equals("USA") && !state.equals("HI") && !state.equals("AK"))
!(x % 4 != 0 || !(x % 100 == 0 && x % 400 == 0))

10. Make up another Java code example that shows the dangling-else problem, using the following statement. A student with a GPA of at least 1.5, but less than 2, is on probation. With less than 1.5, the student is failing.

11. Explain the difference between the == operator and the equals method when comparing strings.

12. Explain the difference between the tests

r = s

and

r.equals(s)

where both r and s are of type Rectangle.

13. What is wrong with this test to see whether r is null? What happens when this code runs?

Rectangle r;
...
if (r.equals(null))
    r = new Rectangle(5, 10, 20, 30);

14. Write Java code to test whether two objects of type Line2D.Double represent the same line when displayed on the graphics screen. Do not use a.equals(b).

Line2D.Double a;
Line2D.Double b;
if (your condition goes here)
    g2.drawString("They look the same!", x, y);
**Hint:** If \( p \) and \( q \) are points, then `Line2D.Double(p, q)` and `Line2D.Double(q, p)` look the same.

15. Explain why it is more difficult to compare floating-point numbers than integers. Write Java code to test whether an integer \( n \) equals 10 and whether a floating-point number \( x \) equals 10.

16. Give an example for two floating-point numbers \( x \) and \( y \) such that `Math.abs(x - y)` is larger than 1000, but \( x \) and \( y \) are still identical except for a roundoff error.

17. Consider the following test to see whether a point falls inside a rectangle.

   ```java
   Point2D.Double p = ...
   boolean xInside = false;
   if (x1 <= p.getX() && p.getY() <= x2)
       xInside = true;
   boolean yInside = false;
   if (y1 <= p.getY() && p.getX() <= y2)
       yInside = true;
   if (xInside && yInside)
       g2.drawString("p is inside the rectangle.", x1, y1);
   ```

   Rewrite this code to eliminate the explicit `true` and `false` values, by setting `xInside` and `yInside` to the values of Boolean expressions.
Warmups Solutions (III)

Solutions to decisions practice problems.

Questions:

1. Find the errors in the following if statements.

   After each statement we list the correct version annotated.
   What needs to be added is underlined. What has to be removed is boxed.

   if quarters > 0 then System.out.println(quarters + "quarters");
   if _quarters > 0_ System.out.println(quarters + "quarters");

   **Condition must be enclosed in parens, and there is no then keyword in Java.**

   if (1 + x > Math.pow(x, Math.sqrt(2))) y = y + x;
   if (1 + x > Math.pow(x, Math.sqrt(2))) y = y + x;

   **Condition must be enclosed in parens, and one parenthesis was missing.**

   if (x = 1) y++; else if (x = 2) y = y + 2;
   if (x == 1) y++; else if (x == 2) y = y + 2;

   **We should always remember to use == to test for equality as = is used exclusively for assignment.**

   if (x && y == 0) p = new Point2D.Double(x, y);
   if (x == 0 && y == 0) p = new Point2D.Double(x, y);

   Presumably we want the condition to mean "both x and y are zero". But in that case we need to write a syntactically correct expression in which we refer to each variable in part: "(x is zero) and (y is zero)". The fact that later x is used as a coordinate eliminates all doubts that x might actually be a boolean variable (in which case the condition, as written originally, was correct).
if (1 <= x <= 10)
{ System.out.println("Enter y:");
  y = console.readDouble();
}

if (1 <= x && x <= 10)
{ System.out.println("Enter y:");
  y = console.readDouble();
}

Just like in the previous exercise we need to properly assemble a boolean expression by putting together something that reads as follows: 

"(x is greater than or equal to 1) and (x is smaller than or equal to 10)"

if (s != "nickels" || s != "pennies"
    || s != "dimes" || s != "quarters")
  System.out.print("Input error!");

if (   !s.equals("nickels") &&
     !s.equals("pennies") &&
     !s.equals("dimes") &&
     !s.equals("quarters")
)
  System.out.print("Input error!");

Remember to use equals when comparing Strings, (!= is the negation of == ).

Also, overall the code was supposed to implement an input error detection mechanism. If you think carefully about it, the error message should only appear if s is not "nickels" and s is not "pennies" and s is not "dimes" and s is not "quarters". So that was another error in the code, that has been fixed in the solution posted above; and you can simplify this even further with DeMorgan, if you want to.

if (input.equalsIgnoreCase("N") || "NO")
  return;

if (input.equalsIgnoreCase("N") ||
     input.equalsIgnoreCase("NO")
)  return;

The syntax of an OR operator requires a boolean expression on its left and another boolean expression on its right. There's no factoring out any common expression or part thereof.

int x = console.readDouble();
if (x != null) y = y + x;

double x = console.readDouble();
y = y + x;

First off readDouble returns a floating point number so the receiving variable should be able to hold the fractional part. Then, a number cannot be compared with null because their types are different: primitive vs. reference type.

Finally, to fix the code you have two options: either drop the if statement (which is what we did above) or turn the code around by using try and catch instead (as explained in chapter 2), to specify what needs to be done if the user does not enter a number at all, and just presses Return (which presumably the original code was trying to do too).
language = "English";
if (country.equals("USA"))
    if (state.equals("PR")) language = "Spanish";
else if (country.equals("China"))
    language = "Chinese";

language = "English";
if (country.equals("USA")) {
    if (state.equals("PR")) language = "Spanish";
} else if (country.equals("China"))
    language = "Chinese";

The original code was missing a pair of braces. Without them the sequence of decisions will never reach a situation where the country variable will be checked for being "China" as the else statement is actually attached to the second if (the one that checks if state is "PR") which is reached when the String variable by the name of country is "USA" already).

2. Explain the following terms and give an example for each construct:

• Expression
  A syntactical construct that is made up of constants, variables, method calls, and operators combining them:(a.length() >= 6)

• Condition
  An expression whose type (or final value) is boolean (and we have already given an example of that above).

• Statement
  A syntactical unit in a program. In Java a statement is either a simple statement, a compound statement, or a block.
  \[ x = x + 1; \]

• Simple statement
  Simple statements are atomic (more or less).
  System.out.println("Atoms are complicated.");

• Compound statement
  Compound statements contain (are composed of) simple statements but not the other way around. Compound statements can contain other compound statements as well.
  if (x != 0) x = 0;

• Block
  Compound statement that contains one or more statements inside it (simple or compound) and groups them together with the help of curly braces.
  \[
  \{ x = 10;
  \quad x += 1;
  \quad x += 1;
  \}
  \]

  We'll have more to say about blocks in future chapters. Meanwhile we have seen blocks being used here in if statements to avoid the dangling else problem.
3. Explain the difference between an if/else/else statement and nested if statements. Give an example for each.

The first is a particular case of the second. The particular aspect of the first one is its \(u \geq \) linear/\(u \geq \) structure: the structure is in fact identical to the one that the switch statement has, except the switch is not nearly as powerful. (Why?)

As for examples, ask me for some.

4. Give an example for an if/else/else statement where the order of the tests does not matter. Give an example where the order of the tests matter.

```java
if (score >= 96) grade = "A+";
else if (score >= 90 && score < 96) grade = "A";
else if (score >= 88 && score < 90) grade = "A-";
else grade = "B";

if (score >= 96) grade = "A+";
else if (score >= 90) grade = "A";
else if (score >= 88) grade = "A-";
else grade = "B";
```

The problem asks for one and the same structure so in our examples the two structures are identical. But if we move the conditions (and their corresponding alternatives, of course) around (for example by first checking for an A-) then the first of the two examples will still produce the right result, whereas the second one will produce only A-'s and B's.

The difference between the two implementations, of course, is that in the first one the conditions are distinct, disjoint, non-overlapping, whereas in the second one they're not.

5. Of the following pairs of strings, which comes first in lexicographic order?

```java
"Tom", "Dick"
"Tom", "Tomato"
"church", "Churchill"
"car_manufacturer", "carburetor"
"Harry", "hairy"
"C++", "Car"
"Tom", "Tom"
"Car", "Carl"
"car", "bar"
```

The question is, of course, why—because the actual answers can be easily checked with the help of a computer:

```java
frilled.cs.indiana.edu% cat Check.java
public class Check {
    public static void main(String[] args) {
        System.out.println(
            "Tom vs. Dick " +
            "Tom".compareTo("Dick"));
        System.out.println(
            "Tom vs. Tomato " +
            "Tom".compareTo("Tomato"));
        System.out.println(
            "church vs. Churchill " +
```
"church".compareTo("Churchill");
System.out.println(
   "\"car manufacturer\" vs. carburetor " +
   "car manufacturer".compareTo("carburetor");
System.out.println(
   "Harry vs. hairy " +
   "Harry".compareTo("hairy");
System.out.println(
   "C++ vs. Car " +
   "C++".compareTo("Car");
System.out.println(
   "Tom vs. Tom " +
   "Tom".compareTo("Tom");
System.out.println(
   "Car vs. Carl " +
   "Car".compareTo("Carl");
System.out.println(
   "car vs. bar " +
   "car".compareTo("bar");
}
frilled.cs.indiana.edu%javac Check.java
frilled.cs.indiana.edu%java Check
Tom vs. Dick 16
Tom vs. Tomato -3
church vs. Churchill 32
"car manufacturer" vs. carburetor -66
Harry vs. hairy -32
C++ vs. Car -54
Tom vs. Tom 0
Car vs. Carl -1
car vs. bar 1
frilled.cs.indiana.edu%

For the why of it—just think a bit (or ask us).

6. Complete the following truth table by finding the truth values of the Boolean expressions for all combinations of the Boolean inputs p, q, and r.

| p    | q    | r    | (p & q) | !r | !(p & (q || !r)) |
|------|------|------|---------|----|-----------------|
| false| false| false| true    | false| false          |
| false| false| true | false   | true | true           |
| false| true | false| true    | false| true           |
| false| true | true | false   | true | false          |
| true | false| true | true    | true | false          |
| true | false| false| true   | true | false          |
| true | true | false| false  | false| true           |
| true | true | true | true   | false| false          |

How do we check if we are right or wrong? We ask the computer:
frilled.cs.indiana.edu%cat Boole.java

public class Boole {
    public static void main(String[] args) {
        boolean p, q, r;
        p = false; q = false; r = false;
        System.out.println( ((p && q) || !r) + " " + !(p && (q || !r)) ) ;
        p = false; q = false; r = true;
        System.out.println( ((p && q) || !r) + " " + !(p && (q || !r)) ) ;
        p = false; q = true; r = false;
        System.out.println( ((p && q) || !r) + " " + !(p && (q || !r)) ) ;
        p = false; q = true; r = true;
        System.out.println( ((p && q) || !r) + " " + !(p && (q || !r)) ) ;
        p = true; q = false; r = false;
        System.out.println( ((p && q) || !r) + " " + !(p && (q || !r)) ) ;
        p = true; q = false; r = true;
        System.out.println( ((p && q) || !r) + " " + !(p && (q || !r)) ) ;
        p = true; q = true; r = false;
        System.out.println( ((p && q) || !r) + " " + !(p && (q || !r)) ) ;
        p = true; q = true; r = true;
        System.out.println( ((p && q) || !r) + " " + !(p && (q || !r)) ) ;
    }
}

frilled.cs.indiana.edu%javac Boole.java
frilled.cs.indiana.edu%java Boole
true true
false true
true true
false true
true false
false true
true false
false true
true false

frilled.cs.indiana.edu%

7. True or false: A && B is the same as B && A for any Boolean conditions A and B?

For all practical purposes in this class: yes, but what is "lazy evaluation"? We will revisit this question
later, when we discuss methods and ways of passing parameters to them.

8. Explain the difference between

    s = 0;
    if (x > 0) s++;
    if (y > 0) s++;

and

    s = 0;
    if (x > 0) s++; 
    else if (y > 0) s++;
The first of the two contains three statements one after another. The second one only two. For \( x > 0 \) and \( y > 0 \) the first code sets \( s \) to 2, while the second one to 1, incrementing only once. Draw the diagrams to see the effect of the else alternative.

9. Use De Morgan's law to simplify the following Boolean expressions.

\[
\begin{align*}
!(x > 0 \&\& y > 0) \\
x &< 0 \mid \mid y < 0
\end{align*}
\]

//-------------------------------------------------------------------------------------------------

\[
\begin{align*}
!(x != 0 \mid \mid y != 0) \\
x & = 0 \&\& y == 0
\end{align*}
\]

//-------------------------------------------------------------------------------------------------

\[
!(\text{country.equals("USA")} \&\& !\text{state.equals("HI")} \&\& !\text{state.equals("AK")})
\]

\[
!\text{country.equals("USA")} \mid \mid \text{state.equals("HI")} \mid \mid \text{state.equals("AK")}
\]

//-------------------------------------------------------------------------------------------------

\[
!(x \% 4 != 0 \mid \mid !(x \% 100 == 0 \&\& x \% 400 == 0))
\]

\[
x \% 4 == 0 \&\& (x \% 100 == 0 \&\& x \% 400 == 0)
\]

This can be further simplified to

\[
x \% 400 == 0
\]

10. Make up another (?!?) Java code example that shows the dangling-else problem, using the following statement. A student with a GPA of at least 1.5, but less than 2, is on probation. With less than 1.5, the student is failing.

\[
\begin{align*}
\text{if (gpa > 1.5) } \\
\text{if (gpa < 2)
}
\end{align*}
\]

System.out.println("Probation.");

else System.out.println("Failing.");

Without the curly braces the else would migrate (change position).

11. Explain the difference between the == operator and the equals method when comparing strings.

The first operator checks if the two Strings are stored in the same place (which means that they are in fact one and the same object). The other one checks to see if they would look the same if printed (case sensitive). You should always use the second one over the first.

12. Explain the difference between the tests

- \( r == s \) and
- \( r.equals(s) \)

where both \( r \) and \( s \) are of type Rectangle.

The first operator checks if the two Rectangles are stored in the same place (which means that they are one and the same object). The other one checks to see if you knew you drew more than one if you were to draw them (if one equals the other they'd be drawn on top of each other and as such you wouldn't be able to tell two have been drawn instead of one).
13. What is wrong with this test to see whether \( r \) is null? What happens when this code runs?

```java
Rectangle r;
...
if (r.equals(null))
  r = new Rectangle(5, 10, 20, 30);
```

*If the variable does not get initialized before the test we would get an error message at compile time. Otherwise it depends how the equals method is implemented in class Rectangle and you would find that there\(^\text{30}\). (If the variable \( r \) points to a real Rectangle the result can only be false). To check if \( r \) is null or not use the == operator.*

14. Note: For this exercise some facts from later one are needed (so we’d have to go back to the future).

Write Java code to test whether two objects of type `Line2D.Double` represent the same line when displayed on the graphics screen. *Do not use `a.equals(b).*

```java
Line2D.Double a;
Line2D.Double b;
if (a.getP1().equals(b.getP1()) && a.getP2().equals(b.getP2()) ||
    a.getP1().equals(b.getP2()) && a.getP2().equals(b.getP1()) )
g2.drawString("They look the same!", x, y);
```

*Hint: If \( p \) and \( q \) are points, then `Line2D.Double(p, q)` and `Line2D.Double(q, p)` look the same. Check the API for accessor used in the condition above (which is implementing the hint).*

15. Explain why it is more difficult to compare floating-point numbers than integers. Write Java code to test whether an integer \( n \) equals 10 and whether a floating-point number \( x \) equals 10.

*Inevitable lack of precision, with floating-point numbers.*

*For an integer \( n \) this would be a good test: \( n == 10 \).*

*For a floating-point number \( x \) this would be a good test:*

```java
Math.abs(x - 10) <= 1E-14 * Math.max(Math.abs(x), Math.abs(y))
```

*Also see the lecture notes and the following (next) exercise in this quiz.*

16. Give an example for two floating-point numbers \( x \) and \( y \) such that `Math.abs(x - y)` is larger than 1000, but \( x \) and \( y \) are still identical except for a roundoff error.

For example 1E18 vs. (1E18 + 1001)

*To see how this could happen consider the following:*

```java
frilled.cs.indiana.edu\%cat Test.java

class Test {
    public static void main(String[] args) {
        double x = 10.0;       // original value
        double y = Math.sqrt(x);        // compute square root
```
double z = y * y; // recover the original number (hopefully)
System.out.println("x = " + x + " y = " + y + " z = " + z);
// so z and x are "the same" now -- hopefully we agree on that
double result = (x - z);
// but result is > 0 (though small enough)
// so yes, x and z are one and the same (really)
System.out.println("Their difference: " + (x - z));
// so look at their difference now, equal as x and z are
// difference is now 2,048 but absolute value means nothing
// it's the ratio of this error to the magnitude of the two
// numbers that really counts 2,048/1e18 is still 1e-14...
}

frilled.cs.indiana.edu%javac Test.java
frilled.cs.indiana.edu%java Test
x = 10.0 y = 3.162276601683795 z = 10.000000000000002
10.0 - 10.000000000000002 = -1.7763568394002606E-15
1.0E19 is a roundoff error of
1.00000000000002E19
Their difference: -2048.0
frilled.cs.indiana.edu%

17. Note: For this exercise some facts from the future are needed.
Consider the following test to see whether a point falls inside a rectangle.

Point2D.Double p = ...

boolean xInside = false;
if (x1 <= p.getX() && p.getX() <= x2) xInside = true;

boolean yInside = false;
if (y1 <= p.getY() && p.getY() <= y2) yInside = true;

if (xInside && yInside) g2.drawString("p is inside the rectangle.", x1, y1);

Rewrite this code to eliminate the explicit true and false values, by setting xInside and yInside to the values of Boolean expressions.

Point2D.Double p = ...

boolean xInside = (x1 <= p.getX() && p.getX() <= x2),
    yInside = (y1 <= p.getY() && p.getY() <= y2);

if (xInside && yInside) g2.drawString("p is inside the rectangle.", x1, y1);
We mentioned in class that

```java
boolean a;
if (<condition>)
a = true;
else
a = false;
```

is equivalent to

```java
boolean a = <condition>;
```

always. So we can often simplify, as needed.
Problems (III)

Decisions programming problems.
These should all be straightforward (as opposed to iff) problems.

1. Write a program (called One) that prints all real solutions to the quadratic equation $ax^2 + bx + c = 0$. Read in $a$, $b$, $c$ and use the quadratic formula. If the discriminant $b^2 - 4ac$ is negative, display a message stating that there are no real solutions. Here's how your program should behave:

```java
frilled.cs.indiana.edu%java One
Please enter the value of a then press Enter : 1.0
Please enter the value of b then press Enter : -2.0
Please enter the value of c then press Enter : 1.0
2.0 2.0
frilled.cs.indiana.edu%java One
Please enter the value of a then press Enter : 1
Please enter the value of b then press Enter : 0
Please enter the value of c then press Enter : 1
There are no solutions.
frilled.cs.indiana.edu%java One
Please enter the value of a then press Enter : 1
Please enter the value of b then press Enter : 2
Please enter the value of c then press Enter : 3
There are no solutions.
frilled.cs.indiana.edu%java One
Please enter the value of a then press Enter : 1
Please enter the value of b then press Enter : -3
Please enter the value of c then press Enter : 1
1.881966011250105 4.11803388749895
frilled.cs.indiana.edu%java One
Please enter the value of a then press Enter : 0
Please enter the value of b then press Enter : 0
Please enter the value of c then press Enter : 3
3.0 is not zero
frilled.cs.indiana.edu%java One
Please enter the value of a then press Enter : 1
Please enter the value of b then press Enter : 0
Please enter the value of c then press Enter : 0
```
-0.0 0.0
frilled.cs.indiana.edu%java One
Please enter the value of a then press Enter : 0
Please enter the value of b then press Enter : 0
Please enter the value of c then press Enter : 0
Identity: zero == zero.
frilled.cs.indiana.edu%

2. Write a program (called Two) that takes user input describing a playing card in the shorthand notation described below, and then prints the full description of the card.

<table>
<thead>
<tr>
<th>A</th>
<th>Ace</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-10</td>
<td>Card values</td>
</tr>
<tr>
<td>J</td>
<td>Jack</td>
</tr>
<tr>
<td>Q</td>
<td>Queen</td>
</tr>
<tr>
<td>K</td>
<td>King</td>
</tr>
<tr>
<td>D</td>
<td>Diamonds</td>
</tr>
<tr>
<td>H</td>
<td>Hearts</td>
</tr>
<tr>
<td>S</td>
<td>Spades</td>
</tr>
<tr>
<td>C</td>
<td>Clubs</td>
</tr>
</tbody>
</table>

Here's a sample run of such a program:

frilled.cs.indiana.edu%java Two
Enter the card notation: AS
Ace of spades.
frilled.cs.indiana.edu%java Two
Enter the card notation: 2H
Two of hearts.
frilled.cs.indiana.edu%java Two
Enter the card notation: CJ
Unknown denomination of unknown colour.
frilled.cs.indiana.edu%java Two
Enter the card notation: JC
Jack of clubs.
frilled.cs.indiana.edu%java Two
Enter the card notation: QK
Queen of unknown colour.
frilled.cs.indiana.edu%java Two
Enter the card notation: QH
Queen of hearts.
frilled.cs.indiana.edu%java Two
Enter the card notation: A D
Unknown denomination of unknown colour.
frilled.cs.indiana.edu%java Two
Enter the card notation: A D
Ace of unknown colour.
frilled.cs.indiana.edu%java Two
Enter the card notation: AD
Ace of diamonds.
frilled.cs.indiana.edu%

3. Write a program (called Three) that
   
   • reads in three floating-point numbers and
   • prints the largest of the three inputs

Here are (essentially) all possible runs of your program:

    frilled.cs.indiana.edu%java Three
    Please enter the value of a then press Enter : 1
    Please enter the value of b then press Enter : 2
    Please enter the value of c then press Enter : 3
    The largest number is: 3.0
    frilled.cs.indiana.edu%java Three
    Please enter the value of a then press Enter : 1
    Please enter the value of b then press Enter : 3
    Please enter the value of c then press Enter : 2
    The largest number is: 3.0
    frilled.cs.indiana.edu%java Three
    Please enter the value of a then press Enter : 2
    Please enter the value of b then press Enter : 1
    Please enter the value of c then press Enter : 3
    The largest number is: 3.0
    frilled.cs.indiana.edu%java Three
    Please enter the value of a then press Enter : 2
    Please enter the value of b then press Enter : 3
    Please enter the value of c then press Enter : 1
    The largest number is: 3.0
    frilled.cs.indiana.edu%java Three
    Please enter the value of a then press Enter : 3.0
    Please enter the value of b then press Enter : 1.0
    Please enter the value of c then press Enter : 2.0
    The largest number is: 3.0
    frilled.cs.indiana.edu%java Three
    Please enter the value of a then press Enter : 3.0
    Please enter the value of b then press Enter : 2.0
    Please enter the value of c then press Enter : 1.0
    The largest number is: 3.0
    frilled.cs.indiana.edu%

4. Write a program (called Four) that creates a circle with radius 100 and center (110, 120).
   Ask the user to specify the x and y coordinates of a point.
   If the point lies inside the circle, then show a message “Congratulations!”
   Otherwise, show a message “You missed.”
   Feel free to use, and enhance with a method

        public boolean contains(double xPoint, double yPoint) { ... }
the Circle class that we developed in the second set of problems.

Here's a sample run of such a program:

```
frilled.cs.indiana.edu%java Four
Welcome. Circle created.
Center is at: (110.0, 120.0)
Radius is: 100.0
You will be asked to specify a point.
First enter x, then enter y.
Please enter x now: 110
Please enter y now: 120
Congratulations.
frilled.cs.indiana.edu%java Four
Welcome. Circle created.
Center is at: (110.0, 120.0)
Radius is: 100.0
You will be asked to specify a point.
First enter x, then enter y.
Please enter x now: -30
Please enter y now: 120
You missed.
frilled.cs.indiana.edu%java Four
Welcome. Circle created.
Center is at: (110.0, 120.0)
Radius is: 100.0
You will be asked to specify a point.
First enter x, then enter y.
Please enter x now: 170.98
Please enter y now: 190.05
Congratulations.
frilled.cs.indiana.edu%
```

5. Write a program (called Five) that asks the user to specify the radii of two circles. The first circle has center (100, 200) and the second circle has center (200, 100). Check to see if the circles intersect.

If they intersect, then display a message “Circles intersect.”
Otherwise, display “Circles don’t intersect.”

*Hint:* Compute the distance between the centers and compare it to the radii.

Here's a sample run of such a program:

```
frilled.cs.indiana.edu%java Five
Welcome. Please specify the radius of the first circle: 10
Great. Now please specify the radius of the second circle: 10
Circles don’t intersect.
frilled.cs.indiana.edu%java Five
Welcome. Please specify the radius of the first circle: 100
Great. Now please specify the radius of the second circle: 100
```
Circles intersect.
frilled.cs.indiana.edu%java Five
Welcome. Please specify the radius of the first circle: 70
Great. Now please specify the radius of the second circle: 71.42
Circles don’t intersect.
frilled.cs.indiana.edu%java Five
Welcome. Please specify the radius of the first circle: 70
Great. Now please specify the radius of the second circle: 71.43
Circles intersect.
frilled.cs.indiana.edu%java Five
Welcome. Please specify the radius of the first circle: 140
Great. Now please specify the radius of the second circle: 1.414299
Circles don’t intersect.
frilled.cs.indiana.edu%java Five
Welcome. Please specify the radius of the first circle: 140
Great. Now please specify the radius of the second circle: 1.4299
Circles intersect.
frilled.cs.indiana.edu%

6. Write a program (called Six) that prints the question

   Do you want to continue?

and reads the user input.
If the user input is any of the following:

- Y
- Yes
- OK
- Sure
- Why not?

print out
- OK

If the user input is

- N
- No

then print out
- Terminating
Otherwise, print
- Bad input

The case of the user input should not matter.
For example,
- y
• yes

are also valid inputs.

Hint: Convert the user input to lowercase and then compare.

Here's a sample run of such a program:

```
frilled.cs.indiana.edu%java Six
Do you want to continue? Y
OK
frilled.cs.indiana.edu%java Six
Do you want to continue? Yes
OK
frilled.cs.indiana.edu%java Six
Do you want to continue? OK
OK
frilled.cs.indiana.edu%java Six
Do you want to continue? suRe
OK
frilled.cs.indiana.edu%java Six
Do you want to continue? Sure
OK
frilled.cs.indiana.edu%java Six
Do you want to continue? whY n0t?
OK
frilled.cs.indiana.edu%java Six
Do you want to continue? why not?
Bad input
frilled.cs.indiana.edu%java Six
Do you want to continue? N
Terminating
frilled.cs.indiana.edu%java Six
Do you want to continue? NU
Terminating
frilled.cs.indiana.edu%java Six
Do you want to continue? n0
Terminating
frilled.cs.indiana.edu%java Six
Do you want to continue? No!
Bad input
frilled.cs.indiana.edu%
```

7. Write a program (called Seven) that translates a letter grade into a number grade. Letter grades are A, B, C, D, F possibly followed by + or -. Their numeric values are 4, 3, 2, 1, and 0. There is no F+ or F-. A + increases the numeric value by 0.3, a - decreases it by 0.3. However an A+ has a value of 4.0.

Here's a sample run of such a program:

```
frilled.cs.indiana.edu%java Seven
Enter a letter grade: A+
The numeric value is: 4.0
```
frilled.cs.indiana.edu%java Seven
Enter a letter grade: A +
Bad input.
frilled.cs.indiana.edu%java Seven
Enter a letter grade: U-
Bad input.
frilled.cs.indiana.edu%java Seven
Enter a letter grade: B-
The numeric value is: 2.7
frilled.cs.indiana.edu%java Seven
Enter a letter grade: C
The numeric value is: 2.0
frilled.cs.indiana.edu%java Seven
Enter a letter grade: F
frilled.cs.indiana.edu%java Seven
Enter a letter grade: D-
The numeric value is: 0.7
frilled.cs.indiana.edu%java Seven
Enter a letter grade: --
Bad input.
frilled.cs.indiana.edu%

8. Write a program (called Eight) that translates a number between 0 and 4 into the closest letter grade. For example, the number 2.8 (which might have been the average of several grades) would be converted to B-. Break ties in favor of the better grade; for example 2.85 should be a B.

Here's a sample run of such a program:

frilled.cs.indiana.edu%java Eight
Enter numeric score then press Enter : 3.85
A
frilled.cs.indiana.edu%java Eight
Enter numeric score then press Enter : 3.84
A-
frilled.cs.indiana.edu%java Eight
Enter numeric score then press Enter : 3.5
A-
frilled.cs.indiana.edu%java Eight
Enter numeric score then press Enter : 3.49
B+
frilled.cs.indiana.edu%java Eight
Enter numeric score then press Enter : 3.1
B
frilled.cs.indiana.edu%java Eight
Enter numeric score then press Enter : 10
A+
frilled.cs.indiana.edu%java Eight
Enter numeric score then press Enter : -2
frilled.cs.indiana.edu%java Eight
Enter numeric score then press Enter : -0.5
frilled.cs.indiana.edu%

9. Write a program (called Nine) that reads in three strings and sorts them lexicographically.

Here's a sample run of such a program:

frilled.cs.indiana.edu%java Nine
Enter three strings:
Alpha
Beta
Gamma
Alpha
Beta
Gamma
frilled.cs.indiana.edu%java Nine
Enter three strings:
Alpha
Gamma
Beta
Alpha
Beta
Gamma
frilled.cs.indiana.edu%java Nine
Enter three strings:
Username
User
Use
Use
User
Username
frilled.cs.indiana.edu%java Nine
Enter three strings:
Alpha
Alpha
Alpha
Alpha
Alpha
Alpha
frilled.cs.indiana.edu%java Nine
Enter three strings:
Gamma
gamma
gama
Gamma
gama
gama
gama
frilled.cs.indiana.edu%java Nine
Enter three strings:
10
1
2
1
10
2
frilled.cs.indiana.edu%

10. A year with 366 days is called a leap year. A year is a leap year if it is divisible by 4 (for example, the year 1980), except it is not a leap year if it is divisible by 100 (for example, the year 1900); however, it is a leap year if it is divisible by 400 (for example, the year 2000). There were no exceptions before the introduction of the Gregorian calendar on October 15, 1582 (for example, the year 1500 was a leap year). Write a program (called Ten) that asks the user for a year and computes whether that year is a leap year or not. Here's a sample run of such a program:

```
frilled.cs.indiana.edu%java Ten
Please enter the year then press Enter : 1500
Leap year: 1500
frilled.cs.indiana.edu%java Ten
Please enter the year then press Enter : 1900
1900 not a leap year!
frilled.cs.indiana.edu%java Ten
Please enter the year then press Enter : 1996
Leap year: 1996
frilled.cs.indiana.edu%java Ten
Please enter the year then press Enter : 1997
1997 not a leap year!
frilled.cs.indiana.edu%java Ten
Please enter the year then press Enter : 2000
Leap year: 2000
frilled.cs.indiana.edu%
```

11. Write a program (called Eleven) that asks the user to enter a month

- 1 = January
- 2 = February
- ... and so on

and then prints the number of days of the month.

For February, print 28 or 29 days

Here's a sample run of such a program:

```
frilled.cs.indiana.edu%java Eleven
Enter a month : 1
31 days
frilled.cs.indiana.edu%java Eleven
Enter a month : 2
28 or 29 days
frilled.cs.indiana.edu%java Eleven
Enter a month : 3
31 days
frilled.cs.indiana.edu%java Eleven
```
Enter a month : 4
30 days
frilled.cs.indiana.edu%java Eleven
Enter a month : 5
31 days
frilled.cs.indiana.edu%java Eleven
Enter a month : 12
31 days
frilled.cs.indiana.edu%java Eleven
Enter a month : 13
Bad input
frilled.cs.indiana.edu%java Eleven
Enter a month : 0
Bad input
frilled.cs.indiana.edu%java Eleven
Enter a month : 1.2
Exception in thread "main" java.lang.NumberFormatException: 1.2
at java.lang.Integer.parseInt(Integer.java:418)
at java.lang.Integer.parseInt(Integer.java:458)
at ConsoleReader.readInt(Eleven.java:58)
at Eleven.main(Eleven.java:7)
frilled.cs.indiana.edu%

12. Write a program (called Twelve) that reads in two floating-point numbers and tests

- BOTH whether they are the same up to two decimal places, AND
- whether they are within 0.01 of each other

Here's a sample run of such a program:

frilled.cs.indiana.edu%java Twelve
Comparing floating-point numbers.
Enter the first number : 1.3456
Enter the second number: 1.3402
1.3456 and 1.3402 are the same up to two decimal places.
1.3456 and 1.3402 are within 0.01 of one another.
Thanks for asking.
frilled.cs.indiana.edu%java Twelve
Comparing floating-point numbers.
Enter the first number : 2.003
Enter the second number: 1.998
2.003 and 1.998 are NOT the same up to two decimal places.
2.003 and 1.998 are within 0.01 of one another.
Thanks for asking.
frilled.cs.indiana.edu%java Twelve
Comparing floating-point numbers.
Enter the first number : 1.998
Enter the second number: 1.990
1.998 and 1.99 are the same up to two decimal places.
1.998 and 1.99 are within 0.01 of one another.
Thanks for asking.
frilled.cs.indiana.edu%java Twelve
Comparing floating-point numbers.
Enter the first number : 1.0000001
Enter the second number: 0.9999999
1.0000001 and 0.9999999 are NOT the same up to two decimal places.
1.0000001 and 0.9999999 are within 0.01 of one another.
Thanks for asking.
frilled.cs.indiana.edu%

13. Enhance the BankAccount class by

   (a) rejecting negative amounts in the deposit and withdraw methods
   (b) rejecting withdrawals that would result in a negative balance

Write a program (called Thirteem) that illustrates the behaviour of your new BankAccount.

Here's a sample run of such a program:

frilled.cs.indiana.edu%java Thirteem
Hello, and welcome to JavaOne Bank.
An account will be created for you.
What will the initial balance be?
Type it now:
20
The current balance in your account is: 20.0
You now want to make a deposit. How much?
Type the amount here:
30
The current balance in your account is: 50.0
You now want to make a withdrawal. How much?
Type it now:
100
Sorry, you cannot do that.
The current balance in your account is: 50.0
Thanks for using class BankAccount. Good-bye!
frilled.cs.indiana.edu%java Thirteem
Hello, and welcome to JavaOne Bank.
An account will be created for you.
What will the initial balance be?
Type it now:
-20
The current balance in your account is: -20.0
You now want to make a deposit. How much?
Type the amount here:
10
The current balance in your account is: -10.0
You now want to make a withdrawal. How much?
Type it now:
10
Sorry, you cannot do that.
The current balance in your account is: -10.0
Thanks for using class BankAccount. Good-bye!
frilled.cs.indiana.edu%java Thirteen
Hello, and welcome to JavaOne Bank.
An account will be created for you.
What will the initial balance be?
Type it now:
-30
The current balance in your account is: -30.0
You now want to make a deposit. How much?
Type the amount here:
-20
Sorry, you cannot do that.
The current balance in your account is: -30.0
You now want to make a withdrawal. How much?
Type it now:
-40
The current balance in your account is: 10.0
Thanks for using class BankAccount. Good-bye!
frilled.cs.indiana.edu%

Notice that you are free to allow negative initial balances (resembling a loan) but the two methods that implement deposit and withdraw should behave as stated in the text of the problem (rejecting negative arguments and rejecting operations that result in a negative balance other than the initial creation of the account). If you want you can disallow initial negative balances as well.

14. Write a program that reads in the name and hourly wage of an employee. Then ask how many hours the employee worked in the past week. Be sure to accept fractional hours. Compute the pay. Any overtime work (over 40 hours per week) is paid at 150 percent of the regular wage. Print a paycheck for the employee.

Here's a sample run of such a program:

frilled.cs.indiana.edu%java Fourteen
Please enter employee's name then press Enter : Larry Bird
Please enter hourly wage then press Enter : 12.50
Please enter hours worked then press Enter: 10
   Paycheck for employee Larry Bird
   Hours worked: 10.0
   Hourly wage: 12.5
   Total payment: 125.0
frilled.cs.indiana.edu%java Fourteen
Please enter employee's name then press Enter : Michael Jordan
Please enter hourly wage then press Enter : 10
Please enter hours worked then press Enter: 50
   Paycheck for employee Michael Jordan
   Hours worked: 50.0
   Hourly wage: 10.0
   Overtime hours: 10.0
   Overtime hourly wage: 15.0
   Total payment: 550.0
frilled.cs.indiana.edu%java Fourteen
Please enter employee’s name then press Enter: Dennis Rodman
Please enter hourly wage then press Enter: 2
Please enter hours worked then press Enter: -4
Paycheck for employee Dennis Rodman
Hours worked: -4.0
Hourly wage: 2.0
Total payment: -8.0
frilled.cs.indiana.edu

Notice it's up to you what you do when the input is out of range. Also, you don’t need to use the class Employee developed in the previous chapter, the emphasis is here on decisions.
Solutions (III)

The solutions presented below are in plain format.

```
:::::::::::::::
One.java
:::::::::::::::
import java.io.*;
public class One {
    public static void main(String[] args) {
        double a, b, c;
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the value of a then press Enter : ");
        a = console.readDouble();
        System.out.print("Please enter the value of b then press Enter : ");
        b = console.readDouble();
        System.out.print("Please enter the value of c then press Enter : ");
        c = console.readDouble();
        double x1, x2;
        if (b * b - 4 * a * c < 0) {
            System.out.println("There are no solutions.");
        } else {
            if (a == 0) {
                if (b == 0) {
                    if (c == 0) {
                        System.out.println("Identity: zero == zero.");
                    } else {
                        System.out.println(c + " is not zero");
                    }
                } else {
                    System.out.println(-c/b);
                }
            } else {
                // Notice a big mistake here:
                x1 = -b - Math.sqrt(b * b - 4 * a * c) / (2 * a);
            }
        }
    }

```

Decisions programming problems. Solutions.
Were any of these problems iffy?
// should be:  
// x1 = (- b - Math.sqrt(b * b - 4 * a * c)) / (2 * a);
// x2 = - b + Math.sqrt(b * b - 4 * a * c) / (2 * a);
// should be:  
// x2 = (- b + Math.sqrt(b * b - 4 * a * c)) / (2 * a);
// System.out.println(x1 + " " + x2);
// Thanks to nminibay for catching the error (also see previous set of warmups)
// Apologies for the unexplainable lack of attention. Darn!
}  
}  
}  

:.........:
Two.java  
:.........:
import java.io.*;
public class Two {  
  public static void main(String[] args) {  
    ConsoleReader console = new ConsoleReader(System.in);
    System.out.println("Enter the card notation: ");
    String card = console.readLine();
    String one = card.substring(0, 1);
    String two = card.substring(1, 2);
    if (one.equalsIgnoreCase("A")) System.out.println("Ace of ");
    else if (one.equalsIgnoreCase("2")) System.out.println("Two of ");
    else if (one.equalsIgnoreCase("3")) System.out.println("Three of ");
    else if (one.equalsIgnoreCase("4")) System.out.println("Four of ");
    else if (one.equalsIgnoreCase("5")) System.out.println("Five of ");
    else if (one.equalsIgnoreCase("6")) System.out.println("Six of ");
    else if (one.equalsIgnoreCase("7")) System.out.println("Seven of ");
    else if (one.equalsIgnoreCase("8")) System.out.println("Eight of ");
    else if (one.equalsIgnoreCase("9")) System.out.println("Nine of ");
    else if (one.equalsIgnoreCase("1")) {
      if (two.equalsIgnoreCase("0")) {
        System.out.println("Ten of ");
        two = card.substring(2, 3);
      }
    }
    else if (one.equalsIgnoreCase("J")) System.out.println("Jack of ");
    else if (one.equalsIgnoreCase("Q")) System.out.println("Queen of ");
    else if (one.equalsIgnoreCase("K")) System.out.println("King of ");
    else System.out.println("Unknown denomination of ");
    if (two.equalsIgnoreCase("D")) System.out.println("diamonds.");
    else if (two.equalsIgnoreCase("H")) System.out.println("hearts.");
    else if (two.equalsIgnoreCase("S")) System.out.println("spades.");
    else if (two.equalsIgnoreCase("C")) System.out.println("clubs.");
    else System.out.println("unknown colour.");
  }  
}
Three.java
import java.io.*;
public class Three {
    public static void main(String[] args) {
        double a, b, c;
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Please enter the value of a then press Enter : ");
        a = console.readDouble();
        System.out.println("Please enter the value of b then press Enter : ");
        b = console.readDouble();
        System.out.println("Please enter the value of c then press Enter : ");
        c = console.readDouble();
        if (a >= c) {
            if (a >= b) {
                System.out.println("The largest number is: "+a);
            } else {
                System.out.println("The largest number is: "+b);
            }
        } else {
            if (c >= b) {
                System.out.println("The largest number is: "+c);
            } else {
                System.out.println("The largest number is: "+b);
            }
        }
    }
}

Four.java
import java.io.*;
public class Four {
    public static void main(String[] args) {
        Circle circle;
        ConsoleReader console;
        circle = new Circle(110, 120, 100);
        System.out.println("Welcome. Circle created. ");
        System.out.println("Center is at: "+circle.getCenter());
        System.out.println("Radius is: "+circle.getRadius());
        System.out.println("You will be asked to specify a point.");
        System.out.println("First enter x, then enter y.");
        System.out.println("Please enter x now: ");
        console = new ConsoleReader(System.in);
        double x = console.readDouble();
        System.out.println("Please enter y now: ");
        double y = console.readDouble();
        if (circle.contains(x, y)) {
            System.out.println("Congratulations.");
        }
    }
}
} else {
    System.out.println("You missed.");
}

}  

class Circle {
    private double xCenter, yCenter, radius;
    Circle(double x, double y, double r) {
        xCenter = x; yCenter = y; radius = r;
    }
    public boolean contains(double xPoint, double yPoint) {
        double distance = Math.pow(xCenter - xPoint, 2) +
                        Math.pow(yCenter - yPoint, 2);
        return distance <= radius * radius;
    }
    public String getCenter() {
        return "(" + xCenter + ", " + yCenter + ")";
    }
    public double getRadius() {
        return radius;
    }
}

:::::::::::::
Five.java
:::::::::::::
public class Five {
    public static void main(String[] args) {
        ConsoleReader console;
        console = new ConsoleReader(System.in);
        System.out.println("Welcome. Please specify the " +
                          "radius of the first circle: ");
        double r1 = console.readDouble();
        System.out.println("Great. Now please specify the " +
                          "radius of the second circle: ");
        double r2 = console.readDouble();
        Circle combined = new Circle(100, 200, r1 + r2);
        if (combined.contains(200, 100)) {
            System.out.println("Circles intersect.");
        } else {
            System.out.println("Circles don't intersect.");
        }
    }
    
    class Circle {
        private double xCenter, yCenter, radius;
        Circle(double x, double y, double r) {
            xCenter = x; yCenter = y; radius = r;
        }
        public boolean contains(double xPoint, double yPoint) {
double distance = Math.pow(xCenter - xPoint, 2) +
    Math.pow(yCenter - yPoint, 2);
    return distance <= radius * radius;
}
public String getCenter() {
    return "(" + xCenter + ",, " + yCenter + ")";
}
public double getRadius() {
    return radius;
}

------------
Six.java
------------
import java.io.*;
public class Six {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Do you want to continue? ");
        String answer = console.readLine();
        if (answer.equalsIgnoreCase("y") ||
            answer.equalsIgnoreCase("Yes") ||
            answer.equalsIgnoreCase("OK") ||
            answer.equalsIgnoreCase("Sure") ||
            answer.equalsIgnoreCase("Why not?")) {
            System.out.println("OK");
        } else if (answer.equalsIgnoreCase("N") ||
            answer.equalsIgnoreCase("No")) {
            System.out.println("Terminating");
        } else {
            System.out.println("Bad input");
        }
    }
}

------------
Seven.java
------------
import java.io.*;
public class Seven {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter a letter grade: ");
        String grade = console.readLine();
        String letter = grade.substring(0, 1);
double value = 0;
if (letter.equals("A")) {
    value = 4;
} else if (letter.equals("B")) {
    value = 3;
} else if (letter.equals("C")) {
    value = 2;
} else if (letter.equals("D")) {
    value = 1;
} else if (letter.equals("F")) {
    value = 0;
} else {
    System.out.println("Bad input.");
    System.exit(0);
}
if (grade.length() > 1) {
    String sign = grade.substring(1, 2);
    double extra = 0;
    if (sign.equals("+")) {
        extra = 0.3;
    } else if (sign.equals("-")) {
        extra = -0.3;
    } else {
        System.out.println("Bad input.");
        System.exit(0);
    }
    if (value > 0 && (value < 4.0 && sign.equals("-"))) {
        value += extra;
    }
}
System.out.println("The numeric value is: \" + value);
}

-------------
Eight.java
-------------
import java.io.*;
public class Eight {
    public static void main(String[] args) {
        double score;
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter numeric score then press Enter : ");
        score = console.readDouble();
        if (score >= 4.0) {
            System.out.println("A+");
        } else if (score >= 3.85) {
            System.out.println("A");
        } else if (score >= 3.5) {
            System.out.println("A-");
        }
else if (score >= 3.15) {
    System.out.println("B+");
} else if (score >= 2.86) {
    System.out.println("B");
} else if (score >= 2.5) {
    System.out.println("B-");
} else if (score >= 2.15) {
    System.out.println("C+");
} else if (score >= 1.86) {
    System.out.println("C");
} else if (score >= 1.5) {
    System.out.println("C-");
} else if (score >= 1.15) {
    System.out.println("D+");
} else if (score >= 0.86) {
    System.out.println("D");
} else if (score >= 0.36) {
    System.out.println("D-");  
} else {
    System.out.println(""");
}

public class Nine {
    public static void main(String[] args) {
        String a, b, c;
        java.io.BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter three strings: ");
        a = reader.readLine();
        b = reader.readLine();
        c = reader.readLine();
        if (a.compareTo(b) < 0) {
            if (a.compareTo(c) < 0) {
                if (b.compareTo(c) < 0) {
                    System.out.println(a + "\n" + b + "\n" + c);
                } else {
                    System.out.println(a + "\n" + c + "\n" + b);
                }
            } else {
                System.out.println(c + "\n" + a + "\n" + b);
            }
        } else {
            if (a.compareTo(c) < 0) {
                if (b.compareTo(c) < 0) {
                    System.out.println(b + "\n" + a + "\n" + c);
                } else {
                    System.out.println(b + "\n" + c + "\n" + a);
                }
            } else {
                System.out.println(c + "\n" + b + "\n" + a);
            }
        }
    }
}
if (b.compareTo(c) < 0) { // b, c, a
    System.out.println(b + "\n" + c + "\n" + a);
} else { // c, b, a
    System.out.println(c + "\n" + b + "\n" + a);
}

-----------------------------
Ten.java
-----------------------------
import java.io.*;
public class Ten {
    public static void main(String[] args) {
        int year;
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the year then press Enter : ");
        year = console.readInt();
        if ( (((year % 4) == 0) && ((year % 100) != 0) || (year < 1582))
             ||
            ( year % 400 == 0) )
        {
            System.out.println("Leap year: " + year);
        } else {
            System.out.println(year + " not a leap year!");
        }
    }
}

-----------------------------
Eleven.java
-----------------------------
import java.io.*;
public class Eleven {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter a month : ");
        int m = console.readInt();
        if (m == 1) {
            System.out.println("31 days");
        } else if (m == 2) {
            System.out.println("28 or 29 days");
        } else if (m == 3) {
            System.out.println("31 days");
        } else if (m == 4) {
            System.out.println("30 days");
        } else if (m == 5) {
            System.out.println("31 days");
        } else if (m == 6) {
            System.out.println("30 days");
        } else if (m == 7) {
            System.out.println("31 days");
        } else if (m == 8) {
            System.out.println("31 days");
        } else if (m == 9) {
            System.out.println("30 days");
        } else if (m == 10) {
            System.out.println("31 days");
        } else if (m == 11) {
            System.out.println("30 days");
        } else if (m == 12) {
            System.out.println("31 days");
        } else {
            System.out.println("Invalid month!");
        }
    }
}
null

::: Eleven.java

```
import java.io.*;
public class Eleven { 
  public static void main(String[] args) { 
    double a, b;
    ConsoleReader console = new ConsoleReader(System.in);
    System.out.println("Comparing floating-point numbers.");
    System.out.print("Enter the first number: ");
    a = console.readDouble();
    System.out.print("Enter the second number: ");
    b = console.readDouble();
    int n, m;
    n = (int)(100 * a);
    m = (int)(100 * b);
    if (n == m) { 
      System.out.println(a + " and " + b + 
                        " are the same up to two decimal places.");
    } else { 
      System.out.println(a + " and " + b + 
                        " are NOT the same up to two decimal places.");
    }
    if (Math.abs(a - b) < 0.01) {
      System.out.println(a + " and " + b + 
                         " are within 0.01 of one another.");
    } else 
  }
```
{  
    System.out.println(a + " and " + b + 
    " are NOT within 0.01 of one another.");
}  
System.out.println("Thanks for asking.");
}

Thirteen.java

import java.io.*;
class BankAccount {
    double balance;
    BankAccount() {
    }
    BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }
    void withdraw(double amount) {
        if (this.balance > amount) {
            this.balance = this.balance - amount;
        } else {
            System.out.println("Sorry, you cannot do that.");
        }
    }
    void deposit(double amount) {
        if (amount > 0) {
            this.balance = this.balance + amount;
        } else {
            System.out.println("Sorry, you cannot do that.");
        }
    }
    double getBalance() {
        return balance;
    }
}
class Thirteen {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.println("Hello, and welcome to JavaOne Bank.");
        System.out.println("An account will be created for you.");
        System.out.println("What will the initial balance be?");
        System.out.println("Type it now: ");
        BankAccount b = new BankAccount(c.readDouble());
        System.out.println("The current balance in your account is: "
            + b.getBalance());
        System.out.println("You now want to make a deposit. How much?");
        System.out.println("Type the amount here: ");
        b.deposit(c.readDouble());
    }
}
System.out.println("The current balance in your account is: "+ b.getBalance());
System.out.println("You now want to make a withdrawal. How much?");
System.out.println("Type it now: ");
b.withdraw(c.readDouble());
System.out.println("The current balance in your account is: "+ b.getBalance());
System.out.println("Thanks for using class BankAccount. Good-bye!");
}
}

Fourteen.java

import java.io.*;
public class Fourteen {
    public static void main(String[] args) {
        String name;
double wage;
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter employee's name then press Enter: ");
        name = console.readLine();
        System.out.print("Please enter hourly wage then press Enter: ");
        wage = console.readDouble();
        System.out.print("Please enter hours worked then press Enter: ");
        double hours = console.readDouble();
        double overtime = (hours - 40);
        System.out.println("Paycheck for employee " + name + "\n" +
            "Hours worked: " + hours + "\n" +
            "Hourly wage: " + wage + "\n" );
        double pay;
        if (overtime > 0) {
            pay = 40 * wage + overtime * (1.5 * wage);
            System.out.println(
                "Overtime hours: " + overtime + "\n" +
                "Overtime hourly wage: " + (1.5 * wage) + "\n");
        } else {
            pay = hours * wage;
        }
        System.out.println("Total payment: " + pay);
    }
}

Remember that
1. there's always more than one way to achieve a result,
2. and if you have a different solution that's actually even better!
The First Dilbert Lecture

Java review by Dilbert. Sharpen your pencils everybody.
This is Review One and is entitled: Classes and Objects.

Introduction
Java programs are built from classes.

From a class definition you can create any number of objects that are known as instances of that class. (Think of a class as a factory with blueprints and instructions to build gadgets - objects are the gadgets the factory makes. Each class is a factory, and the factory can make 0, 1, or more gadgets.)

Here’s a declaration of a simple class that might represent a point on a two-dimensional plane:

```java
class Point {
    public double x, y;
}
```

Exercise: Compile and run, then explain the following program:

```java
class Play {
    public static void main(String[] args) {
        Point a, b;
        a = new Point();
        b = new Point();
    }
}
class Point {
    public double x, y;
}
```

Objects are created using an expression containing the new keyword.

```java
new Point();
```

Creating an object from a class definition is also known as instantiation; thus, objects are often called instances. All objects in Java are accessed via object references - any variable that appears to hold an object actually contains a reference to that object.
p = new Point();

Object references are null when they do not reference any object.

Objects in Java have a type; the type is the object’s class.

```java
Point p;
p = new Point();
```

A class can contain two kinds of members:

1. **fields**

   Fields are data belonging either to the class itself or to objects of the class. They make up the state of the object or class. Fields store results of computations performed by the methods.

2. **methods**

   Methods are collections of statements that operate on the fields to manipulate the state. Methods contain the executable code of a class. The ways in which methods are invoked, and the statements contained within these methods, is what ultimately directs program execution.

Java now supports inner classes but let’s ignore that for the time being.

The Point class

- has two fields representing the x and y coordinates of a point
- and has (as of yet) no methods.

But that will change soon.

Exercise: Compile, run, and explain the following program:

```java
class Play {
    public static void main(String[] args) {
        Point a, b;
        a = new Point();
        a.x = 1;
        a.y = 2;
        a.report();
    }
}
class Point {
    double x, y;
    void report() {
        System.out.println("Point at: (" + x + ", " + y +")");
    }
}
```

The fields in objects are known as instance variables, because there is a unique copy of the field in each object (instance) of the class. Take a look at the following example.
Point lowerLeft = new Point();
Point upperRight = new Point();
Point middlePoint = new Point();
lowerleft.x = 0.0;
lowerleft.y = 0.0;
upperRight.x = 1280.0;
upperRight.y = 1024.0;
middlePoint.x = 640.0;
middlePoint.y = 512.0;
double d = lowerLeft.distanceTo(upperRight); // [1]

Each Point object is unique and has its own copy of the x and y fields. Changing x in lowerLeft, for example, does not affect the value of x in the upperRight object.
Per-object fields are usually what you need. You usually want a field in one object to be distinct from the similarly named field in every other object instantiated from that class.
Notice that we introduced a new method, distanceTo (another Point).

Exercise: Compile, run, and explain the following program:

class Point {
    double x, y;
    Point() {
        x = 0;
        y = 0;
    }
    Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }
    void report() {
        System.out.println("Point at: (" + x + ", " + y + ")");
    }
}
double distanceTo(Point other) {
    double temp;
    temp = (x - other.x) * (x - other.x) +
           (y - other.y) * (y - other.y);
    return Math.sqrt(temp);
}
}
class Play {
    public static void main(String[] args) {
        Point a, b, c;
        a = new Point(34, -34);
        a.report();
        b = new Point();
        c = new Point();
        b.x = 1280.0;
        b.y = 1024.0;
        b.report();
        c.x = 640.0;
        c.y = 512.0;
        c.report();
        Point u, i;
        u = new Point();
        i = new Point(10, 10);
        System.out.println(i.distanceTo(u));
    }
}

Sometimes, though, you want fields that are shared among all objects of that class. The shared variables are
known as class variables - variables specific to the class as opposed to objects of the class. In Java, you obtain
class-specific fields by declaring them static, and they are therefore sometimes called static fields.

Keep in mind:

1. (instance) fields == instance variables == one per object (or gadget)

2. static fields == class variables == one per class (or factory)

   A static field is there no matter how many objects are created, even if none are created. It relates to the
class not to the instances (or objects) of the class.

Exercise: Compile, run, and explain the following program:

class One {
    int a = 0;
    static int b = 0;
    void fun() {
        b += 1;
        a += 1;
        System.out.print(a + " " + b + " ");
    }
}
public static void main(String[] args) {
    One alpha = new One();
}
One beta = new One();
alpha.fun();
beta.fun();
alpha.fun();

In addition to these kinds of variables, methods can use (as local workspace, or scratch paper) method variables and parameters.

They are gone when the method terminates and they are different from instance and class variables. (Not to mention that instance variables are initialized by default).

Wait, wait: let’s try to clarify this. Instance variables and local variables are different. They don’t behave in the same way.

Yes, instance variables are global to the instance methods, unlike local variables, which are local to the methods they are defined in.

Therefore instance variables are global to successive invocations of instance methods. (They stay with the object for as long as the object exists).

Exactly. But can you prove that to me?

Proof is the bottom line for everyone.

If you say so.

What do you think of this one:

public class Three {
    public static void main(String[] args) {
        Three a = new Three();
        a.fun();
        a.fun();
        a.fun();
    }
    void fun() {
        int m = 0;
        m = m + 1;
        this.n = this.n + 1;
        System.out.println("n is: " + this.n +
                           " and m is: " + m + " now.");
    }
    int n; // initialized to zero by default
}

Can I see an example with two functions?

How about this one:

public class Four {
    public static void main(String[] args) {
        Four a = new Four();
        a.fun1();
        a.fun2();
        a.fun1();
    }
a.fun2();
a.fun1();

}  
void fun1() {
    int m = 0;
    m = m + 1;
    this.n = this.n + 1;
    System.out.println("fun1 --> n is: " + this.n + " and m is: " + m + " now.");
}

void fun2() {
    int m = 0;
    m = m + 1;
    this.n = this.n + 1;
    System.out.println("fun2 --> n is: " + this.n + " and m is: " + m + " now.");
}

int n; // initialized to zero by default

OK, I am convinced now.

Note how m is so local you can have a variable with this name in every function you define. I can see that. I have a question though, perhaps a bit unrelated, but it keeps bothering me.

What’s that? What is a static variable, again?

Since you asked:

public class Five {
    int n;
    static int m;
    void fun() {
        this.n = this.n + 1;
        this.m = this.m + 1; // better write this as Five.m += 1;
        System.out.println("n = " + this.n + ", m = " + this.m);
        // Five.m, again...
    }
}

class Tester {
    public static void main(String[] args) {
        Five alpha = new Five();
        Five beta = new Five();
        alpha.fun();
        alpha.fun();
        beta.fun();
        alpha.fun();
        beta.fun();
        beta.fun();
    }
}
beta.fun();
    alpha.fun();
}
}

Oh, I see...

<table>
<thead>
<tr>
<th>It's a factory variable.</th>
<th>Global over all objects ever created by the factory. By the way, I suppose we could have kept everything in just one class, Five, no?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes, indeed. I used Tester to just make sure I don't clutter the main point of the example.</td>
<td>I see. Of course, having Tester, I need to compile the file that contains the source code first, ...</td>
</tr>
<tr>
<td>... and given the way I wrote the code (class Five is public) the file must be called Five.java ...</td>
<td>...I can see that, ...</td>
</tr>
<tr>
<td>... then run java on Tester, for the results.</td>
<td>Very good. We can move on now.</td>
</tr>
</tbody>
</table>

Members of a class can have various levels of visibility. The public declaration of x and y in the Point class means that any code with access to a Point can read or modify those values. (Other levels of visibility limit member access to code in the class itself, or to other related classes).

Objects of the Point class as defined above are exposed to manipulation by any code that has a reference to a Point object, because its fields are declared public. (See the code snippet above). The real benefits of object orientation, however, come from hiding the implementation of a class behind operations performed on its internal data. In Java, operations of a class are declared via its methods - instructions that operate on an object’s data to obtain results. Methods access implementation details that are otherwise hidden from other objects. Hiding data behind methods so that it is inaccessible to other objects is the fundamental basis of data encapsulation.

<table>
<thead>
<tr>
<th>Methods, I like methods...</th>
<th>Well, then, here’s some:</th>
</tr>
</thead>
</table>

Let's briefly consider methods.

If we have a mathematical function f from the set of positive integers \( \mathbb{N} \) to the set of positive integers \( \mathbb{N} \) and with the following definition:

\[
f(x) = 3x + 1
\]

then we can write it in Java as follows:

```java
int f(int n) {
    return 3 * n + 1;
}
```

The domain and codomain of the function as well as the number of arguments are specified in this definition. A return statement is used to pass back the result.

Functions are encapsulating computations. In Java functions are called methods. Let’s suppose we now want to implement a (still simple) but more complicated function, that computes the sum of the first \( n \) positive numbers.
We can do it in several ways:

a) iterative approach

```java
int sum (int n) {
    int sum = i = 0;
    while (i <= n) {
        sum = sum + i;
        i = i + 1;
    }
    return sum;
}
```

b) closed form formula approach

```java
int sum(int n) {
    return n * (n + 1) / 2;
}
```

c) recursive approach

```java
int sum(int n) {
    if (n == 1) return 1;
    else return n + sum(n - 1);
}
```

The iterative approach is very constructive. It uses a local variable, `sum`.

The closed form is OK as long as we can prove that the formula we use does indeed compute the sum of the first \( n \) positive integers. To prove that the formula is correct we can use **mathematical induction**. However, coming up with the right formula to prove, is a different story altogether.

Exercises

1. Write a function that computes the sum of the squares of the first \( n \) positive integers.

2. Write another that computes the sum of the cubes of the first \( n \) integer numbers.

The recursive approach is an elegant approach.

Functions can also be mutually recursive, for example consider this definition of when a given positive integer number \( n \) is odd or even:

```java
boolean odd(int n) {
    if (n == 0) return false;
    else return even(n - 1);
}

boolean even(int n) {
    if (n == 0) return true;
    else return odd(n - 1);
}
```
We notice that the codomain (or, return type, here) of the function is `boolean`, so they’re both `predicates`. (A method whose return type is `boolean` is called a `predicate`).

Now we know how to write functions, but where do we put them. Functions can’t exist on their own in Java. They must be located either inside objects or inside classes. We’ll first put them in an object. But for this we need to create an object, so we need to talk about objects and classes.

We create a type of object `Abacus` and put the methods in it.

```java
class Abacus {
    int sum(int n) {
        if (n == 1) return 1;
        else return n + sum(n - 1);
    }
    boolean odd(int n) {
        if (n == 0) return false;
        else return even(n - 1);
    }
    boolean even(int n) {
        if (n == 0) return true;
        else return odd(n - 1);
    }
}
```

Exercise: Compile, run, and explain the following program:

```java
class Abacus {
    int sum(int n) {
        if (n == 1) return 1;
        else return n + sum(n - 1);
    }
    boolean odd(int n) {
        if (n == 0) return false;
        else return even(n - 1);
    }
    boolean even(int n) {
        if (n == 0) return true;
        else return odd(n - 1);
    }
}
class Example {
    public static void main(String[] args) {
        Abacus a = new Abacus();
        System.out.println(a.sum(10));
        System.out.println(a.odd(4));
        System.out.println(a.odd(5));
        System.out.println(a.odd(a.sum(10)));
    }
}
```

Important things about methods:
• Methods have zero or more parameters. A method can return a value, or it can be declared `void` to indicate that it does not return any value.

• A method’s statements appear in a block of code between curly braces `{` and `}` that follow the method’s name and the declaration of its signature.

• The signature of a method is what it’s known by, and it’s comprised of the name of the method and the number, order and types of the method’s parameters.

We enhance the `Point` class with a simple `clear` method.

```java
class Point {
    double x, y;
    public void clear() {
        x = 0;
        y = 0;
    }
    public double distance (Point theOtherPoint) {
        double xDiff, yDiff;
        xDiff = x - theOtherPoint.x;
        yDiff = y - theOtherPoint.y;
        return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
    }
}
```

The `clear` method has no parameters, hence the empty (and ) after its name. In addition `clear` is declared `void` because it does not return any value. Inside a method, fields and other methods of the class can be named directly - we can simply say `x` and `y` without an explicit object reference.

Objects in general do not operate directly on the data of other objects although, as we saw in the `Point` class, a class can make its fields publicly accessible. (In general, though, well-designed classes hide their data so that it can be changed only by methods of that class.)

To invoke a method, you provide an object reference and the method name, separated by a dot. Same rule applies for instance or class variables. Parameters are passed to the method as a comma-separated list of values enclosed in parentheses. Methods that take no parameters still require the parentheses, with nothing between them. The object on which the method is invoked (the object receiving the object invocation) is often known as the receiving object (or the receiver).

A method can return a single value as a result. To return more than one value from a method, create an object whose sole purpose is to hold return values in a single unit, and return that object. We have also enhanced the `Point` class with a `distance` method. The `distance` method accepts another point object as a parameter, computes the Euclidean distance between itself and the other point, and returns a double precision floating-point result. Based on our `lowerLeft` and `upperRight` objects created earlier in this section one could invoke `distance` as indicated in line [1] in the first code snippet above. Also note that so far we have been talking only about `instance` (or per object) methods.

• Exercise: Revisit, when you have time, Lab Notes Five.

| Occasionally, the receiving object needs to know its own reference. | For example, the receiving object might want to add itself to a list of objects somewhere. |
An implicit reference called this is available to (instance) methods, ...and this is a reference to the current (receiving) object.

The following definition of clear is equivalent to the one just presented:

```java
public void clear() {
    this.x = 0;
    this.y = 0;
}
```

You usually use this as a parameter to other methods that need an object reference.

The this reference can also be used to explicitly name the members of the current object. Here's another one of Point's methods called move that sets the x and y fields to specified values.

```java
class Point {
    double x, y;
    public static Point origin = new Point(); // explain!
    public void clear() {
        this.x = 0;
        this.y = 0;
    }
    public double distance (Point theOtherPoint) {
        double xDiff, yDiff;
        xDiff = x - theOtherPoint.x;
        yDiff = y - theOtherPoint.y;
        return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
    }
    public void move(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

Hey, a Point is a Robot as seen from an airplane, isn't it?

This move method uses this to clarify which x and y are being referred to. Naming the parameters of move "x" and "y" is reasonable, because you pass x and y coordinates to the method.

But then those parameters have the same names as Point's fields, and therefore the parameter names are said to hide the field names. (If we simply write

```java
    x = x
```

we would assign the value of the x parameter to itself, not to the x field as required. The expression

```java
    this.x
```

refers to the object's x field, not the x parameter of move).

So we review and say once again that variables could belong to
• an instance
• a class
• a method

In addition you also have parameters, which are like local variables, except their values are provided by whoever calls the method (and supplies the values of the parameters, or arguments).

It is important to keep this distinction in mind.

Just as you can have per-class (static) fields, you can also have per-class (static) methods, often known as class methods:

• public static void main(String[] args) is such a method (at definition time).

And Math.sqrt(...) is also a class method (being used).

Class methods are usually intended to do class-like operations specific to the class itself, and usually on static fields, not on specific instances of that class. Class methods are declared using the static keyword, and are therefore also known as static methods.

A static method cannot directly access non-static members. (One normally fully understands this only at compile time, and not just by reading it here). When a static method is invoked, there’s no specific object reference for the method to operate on, (so using this in a static method is completely inappropriate and causes an error).

An explicit object reference can be passed to the static method as a parameter if we want to work on (or with) a certain specific instance, but in general static methods do class kind of operations and non-static methods do object kind of things.

This is the end of the introduction. But every end is a new beginning so let’s go over the same things again now, and in a somewhat more systematic manner.

All Over Again

The fundamental unit of programming in Java is the class. Classes contain methods—collections of executable code that are the focus of computation. Classes also provide the structure for objects, plus the mechanism to manufacture objects from the class definitions (the so-called constructors).

You can compute with only primitive types - integer, floating-point, and so on - but almost any interesting Java program will create and manipulate objects.

Each object is an instance of a class. A newly created object is given an initial state.

When a method is invoked on an object, the class is examined to find the code to be run.

The basics of a class are:

• its fields (data)
• and its methods (code to manipulate the data)

Here’s a simple class called Body that could be used to store data about celestial bodies such as comets, asteroids, planets, and stars.

Note: One should not argue for (or against) the usefulness of this example. We are not trying to be practical at all cost, we simply want to understand the nature of things. At the same time, one should always remember
that Java was, after all, invented from people from Sun. (As everybody knows the Sun is located behind a huge firewall and has lots of intranets inside, but that’s already known).

So here’s a simple class called Body that could be used to store data about celestial bodies such as comets, asteroids, planets, and stars.

```java
class Body {
    public long idNum;
    public String nameFor;
    public Body orbits;
    public static long nextID = 0;
}
```

First we declare the name of the class.

A class declaration creates a type name in Java, so that references to objects of that type can be declared with a simple

```java
Body mercury;
```

This declaration states that mercury is a reference to an object of class Body.

The declaration does not create an object - it declares only a reference to a Body object. The reference is initially null, and the object referenced by mercury does not actually exist until you create it explicitly. (In this respect Java is different from languages where objects are created when you declare variables - this just for the record.)

This first version of Body is poorly designed. This is intentional: we will demonstrate the value of certain language features as we improve the class below. Recall that a class’s variables are called fields; the Body class’s nameFor and orbits are examples. They are instance variables.

Every Body object has its own specific instances of these fields:

- a long that uniquely identifies the body from all others (there are so many, int’s too short)
- a String, that is its name
- and a reference to another Body around which it orbits

Giving each separate object a different instance of the fields means that each object has its own unique state. Changing the orbits field in one Body object does not affect the orbits field in any other Body object.

Sometimes, though, you want only one instance of a field shared by all objects of a class. You obtain such fields by declaring them static, so they are called static fields or class variables.

When you declare a static field in a class, all objects created from that class share a single copy of that field. In our case Body has one static field, nextID, which contains the next body identifier to use. The nextID field is initialized to zero when the class is initialized after it is loaded and linked. We will see below that each newly created Body object will have the current value of nextID as its identifier.

All fields and methods of a class are always available to (the) code (that is described) in the class itself. Members declared private are accessible only in the class itself. Members declared public are accessible anywhere the class is accessible, and they are inherited by subclasses. We declared the Body class’s fields public because programmers need access to them to do the work the class is designed for. In a later version of the Body class, we will see that such a design is not usually a good idea.
Body sun = new Body();
sun.idNum = Body.nextID++;
sun.nameFor = "Sol";
sun.orbits = null; // in solar system sun is in the middle
Body earth = new Body();
earth.idNum = Body.nextID++;
earth.nameFor = "Earth";
earth.orbits(sun);

In this first version of Body, objects that represent particular celestial bodies are created and initialized.

First we declare two references (sun and earth) to hold objects of type Body. As mentioned before, these declarations do not create objects; they only declare variables that reference objects. The references are initially null and the objects they may reference must be created explicitly.

We create the sun using the new operator. When you create an object with the new operator, you specify the type of object you want to create and any parameters to its constructor. The Java runtime system allocates enough space to store the fields of the object and initializes it in ways you will soon see. When initialization is complete, the runtime system returns a reference to the new object.

Having created a new Body object, we initialize its variables. Each Body object needs a unique identifier, which it gets from the static nextID field of Body. The code must increment nextID so that the next Body object created will get a unique identifier.

This example builds a solar system model. In this model, the Sun is in the center, and sun’s orbits field is null because it doesn’t orbit anything. When we create and initialize earth, we set its orbits field to sun. A Moon object that orbited the Earth would have its orbits field set to earth.

Constructors A newly created object is given an initial state. Fields can be initialized with a value when they are declared, which is sometimes sufficient to ensure a correct initial state (the rule is that if no value is assigned to a field it will be a zero, \u0000, false or null, depending on its type). But often you need more than simple data initialization to create the initial state; the creating code may need to supply initial data, or perform operations that cannot be expressed as simple assignment. For purposes other than simple initialization, classes can have constructors.

Constructors have the same name as the class they initialize. Like methods they take zero or more parameters, but constructors are not methods and thus have no return type. Parameters, if any, are provided between the parentheses that follow the type name when the object is created with new. Constructors are invoked after the instance variables of a newly created object of the class have been assigned their default initial values, and after their explicit initializers are executed.

This improved version of the Body class uses constructors to set up the initial state, partly by initialization and partly by the constructor:

class Body {
    public long idNum;
    public String name = "<unnamed>";
    public Body orbits = null;
    private static long nextID = 0;
    Body() {
        idNum = nextID++;
    }
}
The constructor for `Body` takes no arguments, but it performs an important function, namely, assigning a proper `idNum` to the newly created object. In the original code, a simple programmer error – forgetting to assign the `idNum`, or not incrementing `nextID` after use – could result in different `Body` objects with the same `idNum`, creating bugs in code relying on the part of the contract that says “all `idNum` values are different”.

By moving responsibility for `idNum` generation inside the `Body` class, we have prevented errors of this kind. The `Body` constructor is now the only entity that assigns `idNum`, and is therefore the only entity needing access to `nextID`. We can and should make `nextID` private, so that only the `Body` class can access it. By doing so, we remove a source of error for programmers using the `Body` class.

You can call this *encapsulation*, if you want. We are now free to change the way `idNums` are assigned to `Body` objects. A future implementation of this class might, for example, look up the name in a database of known astronomical entities and assign a new `idNum` only if an `idNum` had not already been assigned. This change would not affect any existing code, because that existing code wouldn’t have been involved at all in the mechanism for `idNum` allocation.

The data initializations for `name` and `orbits` set them to reasonable values. Therefore, when the constructor returns from the invocation shown below, all data fields in the new `Body` object have been set to some reasonable initial state. You can then set state in the object to the values you want:

```java
Body sun = new Body(); // idNum is 0
sun.name = "Sol";
Body earth = new Body(); // idNum is 1
earth.name = "Earth";
earth.orbits = sun;
```

The `Body` constructor is invoked while the `new` operator creates the object, but after `name` and `orbits` have been set to their initial values. Initializing `orbits` to `null` means that `sun.orbits` doesn’t need to be set in our code.

The case shown here, where you create a body knowing its name and what it orbits, is likely to be fairly common. You can provide another constructor that takes both the name and the orbited body:

```java
Body(String bodyName, Body orbitsAround) {
    this();
    name = bodyName;
    orbits = orbitsAround;
}
```

As shown here, one constructor can invoke another constructor from the same class using the `this()` invocation as its first executable statement. This is called an explicit constructor invocation. If the constructor you want to invoke has parameters, they can be passed to the constructor invocation. Here we use it to call the constructor that has no arguments in order to set up the `idNum`. Now the allocation code is much simpler:

```java
Body sun = new Body("Sol", null);
Body earth = new Body("Earth", sun);
```

You could, if you wanted, provide a one-argument constructor for those cases where you’re constructing a `Body` object that doesn’t orbit anything, rather than invoking the two-argument `Body` constructor with a second argument of `null`. Some classes always require that the creator supply certain kinds of data. For example, your application might require that all `Body` objects have a name. To ensure that all statements creating `Body` objects supply a name, you would define all `Body` constructors with a name parameter.

Here are some common reasons for providing specialized constructors:

- some classes have no reasonable initial state without parameters
• providing an initial state is convenient and reasonable when constructing some kinds of objects (the two-argument constructor of \texttt{Body} is an example)

• constructing an object can be a potentially expensive operation, so you want objects to have a correct initial state when they're created. For example, if each object of a class had a table, a constructor to specify the initial size would enable the object to create the table with the right size from the beginning

• a constructor that isn't \texttt{public} restricts who can create objects using it. (You could, for example, prevent programmers using your package from extending a class by making all its constructors accessible only inside the package. You can also mark as \texttt{protected} constructors that make sense only for subclasses. You may also ignore this comment for now.)

Constructors without arguments are so common that there is a term for them: they are called \texttt{no-arg} (for "no arguments") constructors. How many of them can you have?

If you don't provide any constructors of any kind in a class, the language provides a default no-arg constructor that does nothing. This constructor is provided automatically only if no other constructors exist because there are classes for which a no-arg constructor would be incorrect! Think, for example, of a \texttt{Fractions} class, with only two instance variables (a numerator and a denominator, both of type \texttt{int},) where zero divided by zero (which is what the default constructor would give you) would induce severe discomfort.

- Suggested Exercise

Define a \texttt{Fraction} class that would behave like this:

```java
class TestFrac {
    public static void main(String[] args) {
        Fraction x = new Fraction(1, 20);
        Fraction \( u \) = new Fraction(-1, 60);
        Fraction \( v \) = new Fraction(1, 30);
        Fraction \( y \);
        \( y = x.\text{plus}([u]).\text{minus}([v]); \quad \text{// in one step!} \)
        System.out.println(\( x \times " + " + \\)
            \( u \times " - " + \) \v
            \( v \times " = " + y); \)
    }
}
```

Notice how much they would look like \texttt{BigIntegers} (and Points, too).

If you want both a no-arg constructor and one or more constructors with arguments, you can explicitly provide a no-arg constructor. The automatically provided no-arg constructor for a class that has no superclass is equivalent to the following

```java
public class SimpleClass {
    public SimpleClass () {
        // same functionality as the default constructor
    }
} // a Potato, by any other name...
```

The default constructor is public if the class is, and not if the class isn’t.
Methods A class’s methods typically contain the code that understands and manipulates an object’s state. Some classes have public fields for programmers to manipulate directly, but in most cases this isn’t a very good idea. Many objects have tasks that cannot be represented as a simple value to be read or modified, but require computation. Methods are invoked as operations on objects via references using the . (dot) operator:

    objectReference.methodName( parameters)

Each method takes a specific number of parameters. Java does not include methods that can accept a variable number of parameters. Each parameter has a specified type, either a primitive type or a reference type. Methods also have a return type, which is declared before the method name. For example here is a method of the Body class to create a String that describes a particular Body object:

    public String toString() {
        String desc = idNum + " (" + name + ")";
        if (orbits != null)
            desc += " orbits " + orbits.toString();
        return desc;
    }

This method uses += and -= to concatenate String objects. It first builds a string that describes the identifier and name. If the body orbits another body, we append the string that describes that body by invoking its toString method. This recursion builds a string of bodies orbiting other bodies until the chain ends with some object that doesn’t orbit anything.

The toString method is special. If an object has a method named toString that takes no parameters and returns a String, it is invoked to get a String when that object is used in a string concatenation using the + operator. In these expressions:

    System.out.println("Body " + sun);
    System.out.println("Body " + earth);

the toString methods of sun and earth are invoked implicitly, and produce the following output:

    Body 0 (Sol)
    Body 1 (Earth) orbits 0 (Sol)

Methods can return more than one result in several ways: return references to objects that store results as fields, take one or more parameters that reference objects in which to store the results, or return an array containing the results.

If a method does not return any value, the place where a return type would go is filled with a void. In methods that return a value, every path through the method must return a value assignable to a variable of the declared return type.

Parameter Values to Methods All parameters to methods Java are “call by value”

That is, values of parameter variables in a method are copies of the values the invoker specified.

If you pass a boolean to a method, its parameter is a copy of whatever value was being passed, and the method can change it without affecting values in the code that invoked the method.

For example:
class PassByValue {
    public static void main(String[] args) {
        double one = 1.0;
        System.out.println("before: one = " + one);
        halfIt(one);
        System.out.println("after: one = " + one);
    }
    public static void halfIt(double arg) {
        arg /= 2.0; // divide the arg(ument) by two
        System.out.println("halved: arg = " + arg);
    }
}

The following output illustrates that the value of arg inside halfIt is divided by two without affecting the value of the variable one in main:

    before: one = 1
    halved: arg = 0.5
    after: one = 1

When the parameter is an object reference, however, the object reference is what is passed “by value,” not the object itself. Thus, you can change which object a parameter refers to inside the method without affecting the reference that was passed. But if you change any fields of the objects, or invoke methods that change the object’s state, the object is changed for every part of the program that holds a reference to it. Here’s an example to show this distinction:

class PassRef {
    public static void main(String[] args) {
        Body sirius = new Body("Sirius", null);
        System.out.println("before: " + sirius);
        /*PassRef.*\commonName\{sirius\}*/
        System.out.println("after: " + sirius);
    }
    public static void commonName (Body bodyRef) {
        bodyRef.name = "Dog Star";
        bodyRef = null;
    }
}

This produces the following output:

    before: 0 (Sirius)
    after: 0 (Dog Star)

Notice that the contents of the object have been modified with a name change, while the reference bodyRef still refers to the Body object, even though commonName changed the value of its bodyRef parameter to null. One could draw a picture to show the state of the references just as main invokes commonName. At that point, the two references sirius (in main) and bodyRef (in commonName) both refer to the same underlying object. When commonName changes the field bodyRef.name, the name is changed in the underlying object that the two references share. When commonName changes the value of bodyRef to null, only the value of the bodyRef reference is changed, while the value of sirius remains unchanged, since the parameter bodyRef is a pass by value copy of sirius. Inside the method commonName all you are changing is the value in the parameter variable bodyRef,
just as all you changed in halfIt was the value in the parameter variable, arg. If changing bodyRef affected
the value of sirius in main, the “after” line would say “null”. However, the variables bodyRef in commonName
and sirius in main both refer to the same underlying object so the change made inside commonName is reflected
in the object that sirius refers to.

**Using Methods to Control Access** The Body class with its various constructors is considerably easier to use
than its simple data-only form, and we have ensured that the idNum is set both automatically and correctly.
But a programmer could still mess up the object by setting its idNum field after construction, because the idNum
field is public and therefore exposed to change. The idNum should be read-only data. Read-only in objects is
common, but there is no keyword to apply to a field that allows read-only access outside the class.

To enforce read-only access, you must

1. first hide the field, then
2. provide regulated access to it

You do this by

1. making the idNum field private and
2. providing a new method say, id()

so that code outside the class can read its value using that method.

```java
class Body {
    private long idNum; // now "private"
    public String name = "<unnamed>";
    public Body() {
        idNum = nextID++;
    }
    public long id() {
        return idNum;
    }
    // ...
}
```

Now programmers who want to use the body’s identifier will invoke the id() method, which returns the value
(recall that a value can be used in an expression). There is no longer any way for the programmers to modify
the identifier directly at all: it has effectively become a read-only value outside the class. It can be modified only
by the internal methods of the Body class.

Methods that regulate access to internal data are sometimes called *accessor methods*. You could also use accessor
methods to protect the name and orbits fields, and you probably should.

Even if an application doesn’t require fields to be read-only, making fields private and adding methods to set
and fetch them enables you to add actions that may be needed in the future. If programmers can access a class’s
fields directly, you have no control over what values they will use or what happens when values are changed, and
that should make you very nervous.

To summarize:
• to set, setter, is a mutator
• to fetch, report, reporter is an accessor

Most methods are effectors even more so when they do something else besides setting or fetching.

The this (Host) Reference

We have already seen how you can use an explicit constructor invocation to invoke another one of your class’s constructors at the beginning of a constructor. You can also use the special object reference this inside a non-static method, where it refers to the current object on which the method was invoked. The this reference is most commonly used as a way to pass a reference to the current object as a parameter to other methods. Suppose a method requires adding the object to a list of objects awaiting some service.

It might look something like this:

    Service.add(this);

In this example the method’s name is add and by the conventions we always use, Service is a class, and therefore the method is a static member of that class. The statement above would occur in an instance method, somewhere (belonging to some object that says: “this object also needs service”).

Let’s look at another example. The assignment to str in this class:

    class Name {
       public String str;
       Name() {
          str = "<unnamed>";
       }
    }

is equivalent to the following:

    this.str = "<unnamed>"

Conventionally, you use this only when needed which is when the name of the field you need to access is hidden by a variable or parameter declaration. For example:

    class SwedishChef {
       String chefsName;
       SwedishChef(String chefsName) {
          this.chefsName = chefsName;
       }
    }

The chefsName field is hidden inside the constructor by the parameter of the same name. To ensure we access the chefsName field instead the chefsName parameter, we prefix it with this to specify that the field is the one belonging to this object, the one that is currently being created.

Deliberately hiding identifiers in this manner is considered good programming practice only in this idiomatic use in constructors and accessor methods. But when you start programming it’s better to always use an object reference to refer to instance variables of any object, and a class name to refer to class (static) variables of a class, for any class, thus leaving local variables and parameters easily detectable by their not having to use any dots in qualifying their names.
Overloading Methods. In Java, each method has a signature, which is its name together with the number and types of its parameters. Two methods can have the same name if their signatures have different numbers or types of parameters. This feature is called overloading, because the simple name of the method has overloaded (more than one) meaning. When a programmer invokes a method, the compiler compares the number and type of parameters to find the method that best matches the available signatures. (It works the same with constructors).

Here are some orbitsAround methods for our Body class.

```java
public Body orbitsAround() {
    return orbits;
}
public void orbitsAround(Body around) {
    orbits = around;
}
```

This example deserves careful attention. The methods are written using a programming style that uses overloading to differentiate between fetching a value (no parameters) and setting the value (a parameter with the new value). The number of parameters is different so the overload resolution is simple. If orbitsAround is invoked with no parameters, the method that returns the current value is used. If orbitsAround is invoked with one argument that is a Body, the method that sets the value is used. If the invocation matches neither of these signatures, it is invalid, and the code will not compile.

Static Members. A class has two kinds of members: fields and methods. Each member specifies how it may be accessed and how it may be inherited. Each member can also be made static if so desired.

A static member is a member that is only one per class, rather than one in every object created from that class. For static fields (class variables), there is exactly one variable, no matter how many objects (even zero) there are of the class. The nextID field in our Body class is an example. The static fields of a class are initialized before any static field in that class is used or any method of that class is run. A class can also have static initialization blocks to set up static fields or other necessary states. A static initializer is most useful when simple initialization clauses on the field declaration aren’t up to the task of initialization. For example creating a static array and initializing its members often must be done with executable statements. The order of static initialization within a class is left-to-right and top-to-bottom.

A static method is invoked on behalf of an entire class, not on a specific object instantiated from that class. Such methods are also known as class methods. A static method might perform a general task for all objects of the class—such as returning the next available serial number or something of that nature. (You know this has to be done by Toyota “The Factory” and not by individual retailers—perhaps).

A static method can access only static variables and static methods of the class. There is no this reference, because there is no specific object being operated upon.

Outside of a class, a static member is usually accessed by using the class name, rather than through an object reference.

The main Method of a Class. Details of invoking a Java application vary from system to system, but whatever the details, you must always provide the name of a Java class that drives the application. When you run a Java program, the system locates and runs the main method for that class. The main method must be public, static, and void (it returns nothing), and it must accept a single argument of type String[] args.

Here’s an example that prints its parameters:

```java
class Echo {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
```
System.out.print(args[i] + " ");
System.out.println();
}
}

The arguments in the string array are the "program arguments." These are usually typed by users when they run the program. For example, on a command-line system such as UNIX or a DOS shell, you might invoke the Echo application like this:

    java Echo in here

In this command, java is the Java bytecode interpreter, Echo is the name of the class, and the rest of the parameters are the program arguments. The java command finds the compiled bytecodes for the class Echo, loads them into a runtime in a virtual machine, and invokes Echo.main with the program arguments contained in strings in the String array. The result is the following output:

        in here

The name of the class is not included in the strings passed to main. You already know the name because it is the name of the enclosing class, where you wrote main.

An application can have any number of main methods, since each class can have one. Only one main is used for any given program. The main that’s actually used is specified when the program is run, as Echo was above. Being able to have multiple main methods has one salutary effect – each class can have a main that tests its own code, providing an excellent hook for unit-testing a class.

<table>
<thead>
<tr>
<th>Boy, this lecture was long.</th>
<th>Yes, thank Goodness it’s not double!</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s next?</td>
<td>For the short term: decisions, loops, and methods.</td>
</tr>
<tr>
<td>Which one will be the most important of those?</td>
<td>I’d say Loops, and by a long shot.</td>
</tr>
</tbody>
</table>
Loops

What is the purpose of a while loop? To execute a statement while a condition is true.

Let’s see some examples. Here’s one that prints the first n numbers, where the value of n comes from the user:

```java
ConsoleReader console = new ConsoleReader(System.in);
int n = console.readInt();
int i = 0;
while (i < n) {
    System.out.println(i);
    i += 1;
}
```

Very good. What do you think of this one?

```java
int year = 0;
while (year < 20) {
    double interest = balance * 0.06;
    balance = balance + interest;
}
```

Almost good, except it’s an infinite loop.

What did I forget? The year doesn’t change.

OK. What is the purpose of a for loop? To do what while does, except in a more systematic manner.

In what way? It clearly distinguishes an initialization step, the condition that needs to be true for the loop to keep going, and what we do from one step to the other.

Is this what you mean?
Yes, this is printing a line of 10 asterisks.

Could you do that with a `while` statement?

Yes, and here's how:

```java
i = 0;
while (i < 10) {
    System.out.print("*");
    i = i + 1;
}
```

The `for` and `while` statements are equivalent. Yes, and this makes for good exercises.

What's the purpose of a `do-while`?

It lets you do the body once, first.

Can we see an example?

First the syntax:

```java
do {
    statement
} while (condition);
```

Very good, now the example. OK. Here's a code fragment that adds all the numbers that the user types in and then reports the sum of all these numbers.

```java
int sum = 0;
do {
```
```java
int number = console.readInt();
sum = sum + number;
} while (number != 0);
System.out.println(sum);
```

Does this go on for ever? No. Our *ad-hoc* convention is that the program

```java
frilled.cs.indiana.edu%webster ad-hoc
lad hoc \('\)ad-\'ha":k, \-'ho"-k; ('a":d-"ho"-k\ adv
[L, for this]
(1659)
:for the particular end or case at hand without consideration of wider
application
frilled.cs.indiana.edu%
```

...ends when the user types a value of 0 (zero).

<table>
<thead>
<tr>
<th>So you use a <em>sentinel</em>.</th>
<th>Yes, the value of 0 (zero) acts as a sentinel in this case, guarding the end of our processing.</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is only a convention, right?</td>
<td>Yes, but it works for us.</td>
</tr>
<tr>
<td>Can you do this with a <em>while</em> or a <em>for</em> loop?</td>
<td>Yes, but you’d have to test <em>first</em>.</td>
</tr>
<tr>
<td>And <em>do-while just as well in this case.</em></td>
<td>Plus, <em>zero</em> is such a great sentinel for addition!</td>
</tr>
<tr>
<td>How do you break a loop?</td>
<td>Use the <em>break</em> statement.</td>
</tr>
<tr>
<td>What’s <em>continue</em> doing?</td>
<td>It just <em>resumes</em> the loop from that point.</td>
</tr>
<tr>
<td>How often are you likely to use these?</td>
<td>Not often (<em>break</em> and <em>continue</em>, that is) but in some situations they can come in <em>real</em> handy.</td>
</tr>
<tr>
<td>OK, let’s do an exercise.</td>
<td>Let’s see it.</td>
</tr>
<tr>
<td>What’s this code doing?</td>
<td></td>
</tr>
<tr>
<td>int i = 0;</td>
<td></td>
</tr>
<tr>
<td>while (i &lt; 10)</td>
<td></td>
</tr>
<tr>
<td>i += 1;</td>
<td></td>
</tr>
<tr>
<td>Just going through the first 10 integers I suppose.</td>
<td></td>
</tr>
<tr>
<td>Correct. What’s this one do?</td>
<td>&lt;snicker&gt;</td>
</tr>
<tr>
<td>int i = 0;</td>
<td></td>
</tr>
<tr>
<td>while (i &lt; 10) ;</td>
<td></td>
</tr>
<tr>
<td>i += 1;</td>
<td></td>
</tr>
</tbody>
</table>
Doesn’t it do the *same* thing?

Don’t you see a difference? Ah, the semicolon – that’s an infinite loop now.

Tricky. Indeed.

You have to be careful. How do you write 10 asterisks on a line.

Use a for loop. How do you write 10 such lines?

Use a for inside a for?

Yes, here’s a diagram:

```java
for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```
And the diagram again:

So these are *nested* loops. Yes, nested for loops.

Can you draw a square of asterisks of any size? Yes.

Just replace 10 in the code above by the size.

Can you make it hollow? With no stars inside?

Yes.

I'd have to distinguish between

- the border and
- the inside,

and print

- spaces inside and
- stars on the border.

Can you do that?

Take a look.
public class Square {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter the size: ");
        int size = console.readInt(), i, j;
        for (i = 0; i < size; i++) {
            for (j = 0; j < size; j++) {
                if (i == 0 || j == 0 ||
                    i == (size - 1) ||
                    j == (size - 1)) {
                    System.out.print("* ");
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}

How does it work?

Here you go:

frilled.cs.indiana.edu%java Square
Enter the size: 4
  * * *
  *   *
  * * *
  * * *
frilled.cs.indiana.edu%java Square
Enter the size: 8
  * * * * * *
  *   *   *
  *   *   *
  *   *   *
  *   *   *
  *   *   *
  *   *   *
  * * * * * *
frilled.cs.indiana.edu%java Square
Enter the size: 10
  * * * * * * * *
  *   *   *   *
  *   *   *   *
  *   *   *   *
  *   *   *   *
  *   *   *   *
  *   *   *   *
  *   *   *   *
  *   *   *   *
  * * * * * * * *
Looks good.  

Thanks. Lab notes contain all the details.

So here’s a more appropriate challenge for you.  

Oh, no...

Write a program that produces

- an "E" (uppercase)
- whose size is user-defined.

Can you draw that for me?

There you go:  

Three borders and half of a middle line...

And the user inputs the size?

Yes.  

Not bad, not bad at all.

```java
public class E {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter the size: ");
        int size = console.readInt(), i, j;
        for (i = 0; i < size; i++) {
            for (j = 0; j < size; j++) {
                if (i == 0 ||
                    i == (size - 1) ||
                    j == 0 ||
                    (i == (size - 1) / 2 && j < (size / 2))
                ) {
                    System.out.print("* ");
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
```

Could have been much harder.
I agree.                  I know you do.

<table>
<thead>
<tr>
<th>Yes: X, 4...</th>
<th>Well, let’s not even get into that now.</th>
</tr>
</thead>
</table>

Sure, let's wait for the lab.                  I can hardly wait.

    frilled.cs.indiana.edu%java E
    Enter the size: 10
    * * * * * * * * * *
    *
    *
    *
    * * * * *
    *
    *
    *

    frilled.cs.indiana.edu%java E
    Enter the size: 19
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    *
    *
    *
    *
    *
    *
    *
    *
    *
    *
    *
    *
    *
    *
    *
    *
    *

    frilled.cs.indiana.edu%
Patterns and Loops

*Nested loops, other loops, loops and a half, scalable letters.*

---

Let's practice a bit with for loops. Can you print the numbers from 0 to 9?

Easy:

```java
frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println(i);
        }
    }
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
0
1
2
3
4
5
6
7
8
9
frilled.cs.indiana.edu%
```

Well, what if you want to print the numbers on the same line?

Just use print in the loop,

```java
frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.print(i);
        }
    }
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
0123456789
frilled.cs.indiana.edu%
```
System.out.println(i);
}
System.out.println();
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
0123456789
frilled.cs.indiana.edu%

... and one (empty) println outside of it:

<table>
<thead>
<tr>
<th>Very good.</th>
<th>But don’t you want to space them out a bit?</th>
</tr>
</thead>
</table>

OK, I will print each number in parentheses.

frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.print(" " + i + ");
        }
        System.out.println();
    }
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
frilled.cs.indiana.edu%

Looks good. Can you write 10 such lines?

Let me first highlight the code that prints a line.

class One {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.print(" " + i + ");
        }
        System.out.println();
    }
}

That’s the part that you have in [blue]

Exactly. Now let’s do that 10 times. Use a for loop, with a different index, j.
Why j, when I can call it \texttt{line}? Calling it \texttt{line} would be just fine with me.

\begin{verbatim}
frilled.cs.indiana.edu\%cat One.java
class One {
    public static void main(String[] args) {
        for (int \texttt{line} = 0; \texttt{line} < 10; \texttt{line}++) {
            for (int i = 0; i < 10; i++) {
                System.out.print(" (" + \texttt{i} + ")");
            }
            System.out.println();
        }
    }
}
frilled.cs.indiana.edu\%javac One.java
frilled.cs.indiana.edu\%java One
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
frilled.cs.indiana.edu\%
\end{verbatim}

Although that doesn’t influence in the very least the way the program actually works.

\begin{tabular}{|c|c|}
\hline
Looks good, doesn’t it? & Relax. I marked two of the (4)’s in your output with \textcolor{red}{red} and \textcolor{blue}{blue}. Can you tell me what the difference is between them? \\
\hline
They appear on different lines. & Indeed, for the first one \texttt{line} is 0 (zero), while for the second one \texttt{line} has a value of 3 (three). \\
\hline
Let me change the output to include information about this second dimension. & Good idea.
\end{tabular}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
(0, 0) (0, 1) (0, 2) (0, 3) (0, 4) (0, 5) (0, 6) (0, 7) (0, 8) (0, 9)
(1, 0) (1, 1) (1, 2) (1, 3) (1, 4) (1, 5) (1, 6) (1, 7) (1, 8) (1, 9)
(2, 0) (2, 1) (2, 2) (2, 3) (2, 4) (2, 5) (2, 6) (2, 7) (2, 8) (2, 9)
(3, 0) (3, 1) (3, 2) (3, 3) (3, 4) (3, 5) (3, 6) (3, 7) (3, 8) (3, 9)
(4, 0) (4, 1) (4, 2) (4, 3) (4, 4) (4, 5) (4, 6) (4, 7) (4, 8) (4, 9)
(5, 0) (5, 1) (5, 2) (5, 3) (5, 4) (5, 5) (5, 6) (5, 7) (5, 8) (5, 9)
(6, 0) (6, 1) (6, 2) (6, 3) (6, 4) (6, 5) (6, 6) (6, 7) (6, 8) (6, 9)
(7, 0) (7, 1) (7, 2) (7, 3) (7, 4) (7, 5) (7, 6) (7, 7) (7, 8) (7, 9)
(8, 0) (8, 1) (8, 2) (8, 3) (8, 4) (8, 5) (8, 6) (8, 7) (8, 8) (8, 9)
(9, 0) (9, 1) (9, 2) (9, 3) (9, 4) (9, 5) (9, 6) (9, 7) (9, 8) (9, 9)
frilled.cs.indiana.edu%

It was an easy change. I see, the part in blue is new.

Now I have 100 cells in the output, and I have a name for each one of them.

Yes, line and i, as a pair of numbers. Might as well rename i as something more meaningful. Such as column.

And let's ask the user to specify the size of the square (number of lines and columns).

frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("What size? ");
        int size = c.readInt();
        for (int line = 0; line < size; line++) {
            for (int column = 0; column < size; column++) {
                System.out.print("(" + line + ", " + column + ")");
            }
        }
        System.out.println();
    }
}
(1, 0)  (1, 1)  (1, 2)  (1, 3)  (1, 4)  (1, 5)  
(2, 0)  (2, 1)  (2, 2)  (2, 3)  (2, 4)  (2, 5)  
(3, 0)  (3, 1)  (3, 2)  (3, 3)  (3, 4)  (3, 5)  
(4, 0)  (4, 1)  (4, 2)  (4, 3)  (4, 4)  (4, 5)  
(5, 0)  (5, 1)  (5, 2)  (5, 3)  (5, 4)  (5, 5)  

frilled.cs.indiana.edu%

<table>
<thead>
<tr>
<th>Looks good, doesn’t it?</th>
<th>Relax. What’s new is in red and blue, isn’t it?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes. It looks really good. I have 6 characters for each cell, and everything looks nice and tidy.</td>
<td>Can you highlight the second column?</td>
</tr>
<tr>
<td>You mean the set of cells for which column has a value of 1 (one)?</td>
<td>You got it.</td>
</tr>
</tbody>
</table>

```java
cat One.java

class One {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("What size? ");
        int size = c.readInt();
        for (int line = 0; line < size; line++) {
            for (int column = 0; column < size; column++) {
                if (column == 1) {
                    System.out.print(" ** ");
                } else {
                    System.out.print(" (" + line + ", " + column + ")");
                }
            }
        }
        System.out.println();
    }
}
```

frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One

What size? 6

(0, 0) ** (0, 1) (0, 2) (0, 3) (0, 4) (0, 5)  
(1, 0) ** (1, 1) (1, 2) (1, 3) (1, 4) (1, 5)  
(2, 0) ** (2, 1) (2, 2) (2, 3) (2, 4) (2, 5)  
(3, 0) ** (3, 1) (3, 2) (3, 3) (3, 4) (3, 5)  
(4, 0) ** (4, 1) (4, 2) (4, 3) (4, 4) (4, 5)  
(5, 0) ** (5, 1) (5, 2) (5, 3) (5, 4) (5, 5)  

frilled.cs.indiana.edu%

I can do the first diagonal now.  
I know, the change is minor.

```java
cat One.java

class One {
```
public static void main(String[] args) {
    ConsoleReader c = new ConsoleReader(System.in);
    System.out.print("What size? ");
    int size = c.readInt();
    for (int line = 0; line < size; line++) {
        for (int column = 0; column < size; column++) {
            if (column >= size) {
                System.out.print(" ** ");
            } else {
                System.out.print(" (" + line + ", " + column + ")");
            }
        }
        System.out.println();
    }
}

frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
What size? 6
    ** (0, 1) (0, 2) (0, 3) (0, 4) (0, 5)
    (1, 0) ** (1, 2) (1, 3) (1, 4) (1, 5)
    (2, 0) (2, 1) ** (2, 3) (2, 4) (2, 5)
    (3, 0) (3, 1) (3, 2) ** (3, 4) (3, 5)
    (4, 0) (4, 1) (4, 2) (4, 3) ** (4, 5)
    (5, 0) (5, 1) (5, 2) (5, 3) (5, 4) **

What if you want to see both the last column and first diagonal?
I don’t know, you tell me.
Well, here's what I think: I go through all the cells anyway.
I check their names. And if you can tell by their name
...that they belong to either the first diagonal or to the last column,
...you turn them on.
That's it. This would turn on all of the cells that appear on the first diagonal and all of those that appear on the last column.
It's just a union of sets.
Easy for you to say that, but here's the program:

frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("What size? ");
        int size = c.readInt();
        for (int line = 0; line < size; line++) {
            for (int column = 0; column < size; column++) {
                if (column >= size) {
                    System.out.print(" ** ");
                } else {
                    System.out.print(" (" + line + ", " + column + ")");
                }
            }
            System.out.println();
        }
    }
}
for (int column = 0; column < size; column++) {
    if (column == line || column == (size - 1)) {
        System.out.print(" ** ");
    } else {
        System.out.print(\" + line + ", \" + column + \\n\")
    }
}
System.out.println();
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
What size? 6
** (0, 1) (0, 2) (0, 3) (0, 4) **
(1, 0) ** (1, 2) (1, 3) (1, 4) **
(2, 0) (2, 1) ** (2, 3) (2, 4) **
(3, 0) (3, 1) (3, 2) ** (3, 4) **
(4, 0) (4, 1) (4, 2) (4, 3) ** **
(5, 0) (5, 1) (5, 2) (5, 3) (5, 4) **
frilled.cs.indiana.edu%

Looks good.

And I was the first to say that. How can we make this look more like a square?

Maybe change the output a bit. How about this:

frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("What size? ");
        int size = c.readInt();
        for (int line = 0; line < size; line++) {
            for (int column = 0; column < size; column++) {
                if (column == line || column == (size - 1)) {
                    System.out.print(" **");
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
What size? 6
* *
frilled.cs.indiana.edu%

Did you catch that?
You bet...
Now the names are only implicit.
Only in our program's mind.

We can draw patterns, scalable patterns. Here's code for a Z:

```java
frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("What size? ");
        int size = c.readInt();
        for (int line = 0; line < size; line++) {
            for (int column = 0; column < size; column++) {
                if (column == (size - 1 - line) ||
                    line == 0 ||
                    line == (size - 1)) {
                    System.out.print("*");
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
What size? 7
* * * * * * * 
* 
* 
* 
* 
* 
frilled.cs.indiana.edu%java One
What size? 4
Can you draw a circle?  

Sure, why not?  A circle?  

Of course:

```java
frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("What size? ");
        int size = c.readInt();
        for (int line = 0; line < size; line++) {
            for (int column = 0; column < size; column++) {
                if (Math.abs((line - size / 2) * (line - size / 2) +
                             (column - size / 2) * (column - size / 2) -
                             (size - 1) * (size - 1) / 4) <= (0.15 * size))
                    { System.out.print("* ");
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
What size? 20

*   *   *
   *   *
*   *
* * *
*   *
*   *

frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
What size? 20

*   *   *
   *   *
*   *
* * *
*   *
*   *
```
Hey, now wait a minute!

I hope you understand approximations.

And the equation of a circle.

If you don’t, don’t worry...

Don’t worry, be happy!

Well, then it’s time for...

```java
public class Patterns {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter the size: ");
        int size = console.readInt(), i, j;
        for (i = 0; i < size; i++) {
            for (j = 0; j < size; j++) {
                if (false) {
                    System.out.print("*");
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
```

Here’s the grid:

Now, let’s look at all the patterns.

**Pattern 1:**

(j = 0)

**Pattern 2:**

(i = 0)
Pattern 3:  
(j == (size - 1))

Pattern 4:  
(i == (size - 1))

Pattern 5:  
((j == 0) && (i <= size/2))

Pattern 6:  
((j == 0) && (i > size/2))

If you see that (in places) we’re off by 1 (one), ... then you’re on the right track.

But keep in mind we are interested here in the overall understanding of inequalities that determine (or define) the patterns, ... so we trade (where appropriate) absolute accuracy for a shorter (simpler, yet still reasonably exact) formula.

Pattern 7:  
((i == 0) && (j <= size/2))

Pattern 8:  
((i == 0) && (j > size/2))

Pattern 9:  
((j == (size - 1)) && (i <= size/2))
Pattern 10: \(((j == (size - 1)) && (i > size/2))\)

Pattern 11: \(((i == (size - 1)) && (j < size/2))\)

Pattern 12: \(((i == (size - 1)) && (j > size/2))\)

Pattern 13: \((j == size/2)\)

Pattern 14: \((i == size/2)\)

Pattern 15: \(((j == size/2) && (i <= size/2))\)

Pattern 16: \(((j == size/2) && (i > size/2))\)
<table>
<thead>
<tr>
<th>Pattern 17:</th>
<th>Pattern 18:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Pattern 17" /></td>
<td><img src="image2" alt="Pattern 18" /></td>
</tr>
</tbody>
</table>

17: \((i = \text{size}/2) \land (j \leq \text{size}/2)\)  
18: \((i = \text{size}/2) \land (j > \text{size}/2)\)

<table>
<thead>
<tr>
<th>Pattern 19:</th>
<th>Pattern 20:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Pattern 19" /></td>
<td><img src="image4" alt="Pattern 20" /></td>
</tr>
</tbody>
</table>

\((i = j)\)  
\((i + j = (\text{size} - 1))\)

<table>
<thead>
<tr>
<th>Pattern 21:</th>
<th>Pattern 22:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5" alt="Pattern 21" /></td>
<td><img src="image6" alt="Pattern 22" /></td>
</tr>
</tbody>
</table>

21: \((i = j) \land (i \leq \text{size}/2) \land (j \leq \text{size}/2)\)  
22: \((i = j) \land (i > \text{size}/2) \land (j > \text{size}/2)\)

Can either one of these conditions be simplified at all?

<table>
<thead>
<tr>
<th>Pattern 23:</th>
<th>Pattern 24:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image7" alt="Pattern 23" /></td>
<td><img src="image8" alt="Pattern 24" /></td>
</tr>
</tbody>
</table>

23: \((i + j = (\text{size} - 1)) \land (i \leq \text{size}/2) \land (j > \text{size}/2)\)  
24: \((i + j = (\text{size} - 1)) \land (i > \text{size}/2) \land (j \leq \text{size}/2)\)

Can either one of these be simplified?
If we have these "atomic" patterns (described above), how we can combine them to obtain more complicated patterns (such as the ones illustrated below):

**Uppercase T:**
A cell should be turned ON if it appears in the group described by pattern 13 OR if it appears in the group that is described by pattern 2, otherwise the cell is OFF (blank)

**Uppercase L:**
A cell should be turned ON if it appears in the group described by pattern 13 OR if it appears in the group described by pattern 12

**Uppercase E:**
If cell is in pattern 1 OR in pattern 2 OR in pattern 4 or in pattern 17 then the cell should be turned ON otherwise leave the cell blank (print a space)

**Uppercase W:**
Cell to be turned ON if pattern 1 matches OR if pattern 24 matches OR if pattern 22 matches OR if pattern 3 matches, otherwise leave cell blank

**Uppercase A:**
(20) OR (3) OR (18)
Uppercase R:

(22) || (1) || (14) || (2) || (9)

I am sure you can think of many other patterns: Y, Q (for example). A diamond.

Your task is to write a program that produces a scalable 4 (four):

This is your lab assignment 6 (six), due at the beginning of next lab.
Homework Three

In this assignment you will be practicing with \texttt{while} and \texttt{for} loops.

First, write an algorithm to settle the following question:

A bank account starts out with $10,000. Interest is compounded at the end of every month at 6 percent per year (0.5 percent per month). At the beginning of every month, $500 is withdrawn to meet college expenses after the interest has been credited. After how many years is the account depleted?

Now suppose the numbers ($10,000, 6 percent, $500) were user-selectable. Are there values for which the algorithm you developed would not terminate? If so, make sure it always terminates.

This will be your assignment.

Therefore you are to write a program that asks the user for three values:

- initial balance (a \texttt{double})
- interest per year (another \texttt{double}, a percent value)
- monthly expenses (another \texttt{double}, in dollars)

Then if the program can determine that the account will eventually be depleted, it will report in how many years this will happen, otherwise, if the program determines that for the values provided by the user the account will never get depleted, the program will report that and end.

Here's a sample session with your program:

\begin{verbatim}
frilled.cs.indiana.edu% java Interests
Welcome to the financial calculator.
What's your initial balance? 10000
What's the yearly interest? 6
How much do you plan to withdraw monthly? 500
The account will last 1 year(s) and 9 month(s).
Ending balance will be: 62.2 dollars.
frilled.cs.indiana.edu% java Interests
Welcome to the financial calculator.
What's your initial balance? 10000
\end{verbatim}
What's the yearly interest? 6
How much do you plan to withdraw monthly? 100
The account will last 11 year(s) and 6 month(s).
Ending balance will be: 97.09 dollars.
frilled.cs.indiana.edu%java Interests
Welcome to the financial calculator.
What's your initial balance? 10000
What's the yearly interest? 6
How much do you plan to withdraw monthly? 10
This will last forever.
frilled.cs.indiana.edu%java Interests
Welcome to the financial calculator.
What's your initial balance? 10000
What's the yearly interest? 6
How much do you plan to withdraw monthly? 50
The account will last 525 year(s) and 1 month(s).
Ending balance will be: 48.1 dollars.
frilled.cs.indiana.edu%

Please call your program (the class) Interests for faster grading.

Do you see anything fishy in the examples given? (I hope you do.) What is it?

frilled.cs.indiana.edu%webster fishy
fishy \'fish-e"-\ fish-i-er; -est
(1547)
1: of or resembling fish esp. in taste or odor
2: creating doubt or suspicion: QUESTIONABLE
frilled.cs.indiana.edu%

What do you plan to do about it in your program?

Now here's the code used to produce the output above:

class Interests {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.println("Welcome to the financial calculator.");
        System.out.print("What's your initial balance? ");
        double initialBalance = c.readDouble();
        System.out.print("What's the yearly interest? ");
        double yearlyInterest = c.readDouble() / 100;
        System.out.print("How much do you plan to withdraw monthly? ");
        double monthlyStipend = c.readDouble();
        double calculation =
            initialBalance * (1 + yearlyInterest / 12) - monthlyStipend;
        if (calculation >= initialBalance) {
            System.out.println("This will last forever.");
        } else {
            System.out.println("The account will last ");
            int years = (int) Math.ceil(calculation / initialBalance);
            System.out.println(years + " years and ");
            int months = (int) Math.ceil(calculation % initialBalance * 12);
            System.out.println(months + " months.");
        }
    }
}
System.out.println("This will last forever.");
} else {
    double balance = initialBalance;
    int months = 0;
    while (balance * (1 + yearlyInterest / 12) > monthlyStipend) {
        months += 1;
        balance = balance * (1 + yearlyInterest / 12) - monthlyStipend;
    }
    int years = months / 12;
    months = months % 12;
    System.out.println("The account will last " + years +
                        " year(s) and " + months + " month(s)." );
    System.out.println("Ending balance will be: " +
                        Math.round(balance * 100) / 100.00
                        + " dollars."
                        );
}
}

Why is this program not calculating the last case correctly?
Also, is this program printing the right (correct) output?
Why or why not?

frilled.cs.indiana.edu%cat Why.java
class Why {
    public static void main(String[] args) {
        double sum = 0;
        for (int i = 0; i < 10; i++)
        sum = sum + 0.1;
        System.out.println(sum);
    }
}
frilled.cs.indiana.edu%javac Why.java
frilled.cs.indiana.edu%java Why
0.9999999999999999
frilled.cs.indiana.edu%
More Loops

More practice with loops. Loops, tokenizers, and Monte Carlo problems.

Do you like loops? I don’t know; so far, so good.

Now we need to move on. First a few simple, basic exercises.

Can you explain this? I sure can.

```java
frilled.cs.indiana.edu% cat Two.java
class Two {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      System.out.print(i);
    }
    i = 10;
    System.out.println(i);
  }
}
frilled.cs.indiana.edu% javac Two.java
Two.java:6: Undefined variable: i
  i = 10;
       ~
Two.java:7: Undefined variable: i
  System.out.println(i);
       ~
2 errors
frilled.cs.indiana.edu%
```

Can you fix it? I sure can.

```java
frilled.cs.indiana.edu% cat Two.java
class Two {
  public static void main(String[] args) {
    int i;
    for (i = 0; i < 10; i++) {
      System.out.print(i);
    }
frilled.cs.indiana.edu%
```
```java
class Wow {
    public static void main(String[] args) {
        int i;
        i = 3;
        System.out.println(i);
    }
}

frilled.cs.indiana.edu%javac Two.java
frilled.cs.indiana.edu%java Two
012345678910
frilled.cs.indiana.edu%
```

How do you call that? I’d say: *scope* of a variable.

frilled.cs.indiana.edu%webster scope

scope \'sko\'-p\ n
[It scopo purpose, goal, fr. Gk skopos; akin to Gk skeptesthai to watch, look at -- more at SPY]

(1555)
1: space or opportunity for unhampered motion, activity, or thought
2: INTENTION, OBJECT
3: extent of treatment, activity, or influence
4: range of operation\f
syn see RANGE

frilled.cs.indiana.edu%

Sounds good. It’s the range of operation for that variable.

Take a look at this: It doesn’t compile!

```java
class Wow {
    public static void main(String[] args) {
        int i;
        i = 3;
        System.out.println(i);
    }
    System.out.println(i);
}
```

Why? The curly braces!

Indeed, they are defining the scope. And they do it in the same way for:

- *classes*
- *if statements*
- *for loops*
- *while loops*
As well as other kinds of loops. There's just one more, as we will see below.

In any event, having blocks of statements available as standalone entities that can be placed anywhere inside the program can sometimes be a source of confusion for the beginning Java programmer.

Let me see some examples.

Here's one: That is not a source of any confusion!

```java
frilled.cs.indiana.edu%cat Wow.java
class Wow {
    public static void main(String[] args) {
        int x = 1;
        if (x > 2) {
            System.out.println("Yes, " + x + " is greater than 2.");
        }
    }
}
frilled.cs.indiana.edu%javac Wow.java
frilled.cs.indiana.edu%java Wow
```

I know, but consider this: Ah, I see the semicolon!

```java
frilled.cs.indiana.edu%cat Wow.java
class Wow {
    public static void main(String[] args) {
        int x = 1;
        if (x > 2) { ;
            System.out.println("Yes, " + x + " is greater than 2.");
        }
    }
}
frilled.cs.indiana.edu%javac Wow.java
frilled.cs.indiana.edu%java Wow
```

I hope you do. That is an if with an empty body.

That's how we got an infinite loop last time. There's one thing to be learned from this.

Syntax rules. And people who use it properly, rock!

Let's look at other kind of loops. OK, here's a program that talks to the user.

```java
frilled.cs.indiana.edu%cat Three.java
class Three {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
```
String line;
    do {
        System.out.print("Type something: ");
        line = c.readLine();
        System.out.println("You typed: " + line);
    } while (! line.equals("bye"));
    System.out.println("Good bye!");
}

using while is just a bit longer.

frilled.cs.indiana.edu%javac Three.java
frilled.cs.indiana.edu%java Three
Type something: I am here
You typed: I am here
Type something: You are there
You typed: You are there
Type something: Your name is Echo
You typed: Your name is Echo
Type something: Bye
You typed: Bye
Type something: bye
You typed: bye
Good bye!
frilled.cs.indiana.edu%

Yes, since the test comes first.

frilled.cs.indiana.edu%cat Three.java
frilled.cs.indiana.edu%javac Three.java
frilled.cs.indiana.edu%java Three
Type something: Works the same, doesn’t it?
You typed: Works the same, doesn’t it?
Type something: It does seem so.
You typed: It does seem so.
Type something: I am happy.
You typed: I am happy.
Type something: Bye
You typed: BYe
Type something: byE
You typed: byE
Type something: bye
You typed: bye
Good bye!
frilled.cs.indiana.edu%

<table>
<thead>
<tr>
<th>In both cases we work with whole loops.</th>
<th>Yes, we know we’re done either at the beginning or at the end of the loop’s body of statements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sometimes we’d like to allow for being able to realize we’re done halfway through the loop.</td>
<td>And end your processing in mid-loop? Doesn’t this coding situation have a specific name?</td>
</tr>
<tr>
<td>Yes, it’s called the loop and a half problem.</td>
<td>Very good. Here’s an example: A program that reads lines from a file, then reverses them.</td>
</tr>
</tbody>
</table>

```java
frilled.cs.indiana.edu%cat Four.java
class Four {
    public static void main(String[] args) {
        BufferedReader c = new BufferedReader(System.in);
        boolean done = false;
        String line;
        while (!done) {
            System.out.print("Echo > ");
            line = c.readLine();
            if (line == null) { // EOF or ctrl-D for that
                done = true;
            } else {
                System.out.println(line);
            }
        }
    }
}
frilled.cs.indiana.edu%javac Four.java
frilled.cs.indiana.edu%java Four
Echo> Hello!
Hello!
Echo> I am here.
I am here.
Echo> How are you?
How are you?
Echo> I am fine, how about you?
I am fine, how about you?
Echo> You don’t say...
You don’t say...
Echo> I am going to type control-D now
I am going to type control-D now
Echo> Bye!
Bye!
Echo> frilled.cs.indiana.edu%
```
Where's the file? You can pipe one into your program!

(See your friendly TA or ask me for more details).

Alright. So we read lines, one by one. When the line we read is empty there's nothing to be reversed. So the program simply quits the loop.

Oh, I see: and we skip the lower half of the loop.

Indeed, at that point we simply quit it. How do you do that?

One can do that with break, or... using a boolean variable, and an if statement.

We choose the second approach, because one can argue it's a bit more structured.

But the first one is also often used, and it greatly simplifies code on occasion.

So now we have a basic echo program. Yes, but as of right now nothing is being reversed. Wasn't that what we set out to do?

OK, let's make this more exciting, as if through a looking glass.

Yes, s'tel esrever sretcarahc ni sdrow!

Well, yletanutrofn u ew t'nac od taht. ?neht ,elbissop si i iht spahrep tuB

That, we can do.

```java
frilled.cs.indiana.edu%cat Four.java
class Four {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        boolean done = false;
        String line;
        while (! done) {
            System.out.print("Echo>");
            line = c.readLine();
            if (line == null) { // EOF or ctrl-D for that
                done = true;
            } else {
                String rev = "";
                int i;
                for (i = line.length() - 1; i >= 0; i--) {
                    rev += line.charAt(i);
                }
                System.out.println(rev);
            }
        }
    }
}
frilled.cs.indiana.edu%javac Four.java
frilled.cs.indiana.edu%java Four
Echo> Hello!
```
I like this guy, Gotcha. Once upon a time there was a Polish carpenter, by
the name of Zbigniew Gotcha, who lived in Krakow. You know the story?

No. I don’t either, but I like the way it starts.

Time to wrap up. Let’s do random numbers.

Yes, we kept π for dessert. Do you like sherbet?

Let’s program the following simulation:

Darts are thrown at random points onto the square with corners (1,1) and (-1,-1). If the dart lands
inside the unit circle, that is, the circle with center (0,0) and radius 1 it is a hit. Otherwise it is
a miss. Run this simulation to determine an experimental value for the fraction of hits in the total
number of attempts, multiplied by 4.

Oh, this is so easy! Easy as π!

Fine, if it’s easy, why don’t you do it first? Relax, here it is:

frilled.cs.indiana.edu%cat Pi.java
import java.util.Random;
public class Pi {
    public static void main(String[] args) {
        Random r = new Random();
        double x, y, d;
        int i, count = 0;
        for (i=0; i < 100000 ; i++) {
            x = r.nextDouble() * 2 - 1;
            y = r.nextDouble() * 2 - 1;
            d = Math.sqrt(x * x + y * y);
if (d < 1) count++;
}
System.out.println("Pi is approximately " + 4.0 * count / i);
}
frilled.cs.indiana.edu%javac Pi.java
frilled.cs.indiana.edu%java Pi
Pi is approximately 3.13648
frilled.cs.indiana.edu%javac Pi
Pi is approximately 3.15064
frilled.cs.indiana.edu%javac Pi
Pi is approximately 3.14426
frilled.cs.indiana.edu%javac Pi
Pi is approximately 3.1422
frilled.cs.indiana.edu%javac Pi
Pi is approximately 3.1438
frilled.cs.indiana.edu%

<table>
<thead>
<tr>
<th>Can you explain it?</th>
<th>The probability of a hit is the fraction that the circle represents of the total area.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>It is also close to the ratio of the measured frequencies: hits divided by attempts.</th>
<th>So we write the formulas, and the radius simplifies, as it appears on both sides.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>But there is a factor of half ( \frac{1}{2} ) which...</th>
<th>...is squared, so it participates as a fourth ( \frac{1}{4} ) in the end, and there you have it.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Pretty good.</th>
<th>I want to do more problems.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>I was hoping you would.</th>
<th>I'm in great shape today.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Here's a program that helps with problem 6.1</th>
<th>Where does this number come from?</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Just ignore it for now...</th>
<th>What does the program do?</th>
</tr>
</thead>
</table>

```java
import java.util.*;
import java.io.*;

class Six {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Hello> ");
        String line = console.readLine();
        while (line != null) { // ^D would do it
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken();
                System.out.println(token.toUpperCase());
            }
        }
        System.out.print("Hello> ");
        line = console.readLine(); // what if we take this out?
    }
}
```java
} System.out.println("End of program.");
}

Reads lines, one by one. When does it end?

When you type Control-D. Which is EOF (end of file).

Yes, in that case the line is null. What does it do, line by line?

Looks at the tokens, and prints them back, but converted into uppercase. Not much different from the one about Zbigniew, except this one uses this StringTokenizer.

Exactly, that's the main difference. How does that work?

Take a closer look:

```java
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
    String token = tokenizer.nextToken();
    System.out.println(token.toUpperCase());
}
```

Better look this class up.

*http://java.sun.com/products/jdk/1.2/docs/api/java/util/StringTokenizer.html

It’s like a machine gun loaded with words. Or like a stapler, if you don’t mind.

A stapler would also be a good analogy. With staples of variable length.

Glued together by blank spaces. Staples are tokens.

frilled.cs.indiana.edu%webster token
to-ken \"to\"--ken n
[ME, fr. OE ta\"-cen, ta\"-cn sign, token; akin to OHG zeihhan sign, Gk deiknavai to show -- more at DICTION]
(bef. 12c)
1: an outward sign or expression <his tears were tokens of his grief>
   2a: SYMBOL, EMBLEM <a white flag is a token of surrender>;
   2b: an instance of a linguistic expression
   3: a distinguishing feature: CHARACTERISTIC
   [...]
frilled.cs.indiana.edu%
```

For both problem 6.2 and 6.3 the trick is to interpret I think I can handle that (ignore the numbers), (read and promptly evaluate) input.
You’re going to have to read

- a rate,
- then numbers,
- finished by zero...

...then numbers again, ending with EOF.

Or some such thing.

I could even end it with a keyword, such as “quit” or “bye”, or some other meaningful word.

Really, how?

Take a look.

```java
import java.util.*;
import java.io.*;

class TwoAndThree {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Rate");
        String line = console.readLine();
        StringTokenizer tokenizer = new StringTokenizer(line);
        double rate = Double.parseDouble(tokenizer.nextToken());
        double amount;
        do {
            System.out.print("Dollars");
            line = console.readLine();
            tokenizer = new StringTokenizer(line);
            amount = Double.parseDouble(tokenizer.nextToken());
        } while (amount > 0);
        do {
            System.out.print("Euros");
            line = console.readLine();
            if (line == null || line.equalsIgnoreCase("quit"))
                break;
            tokenizer = new StringTokenizer(line);
            amount = Double.parseDouble(tokenizer.nextToken());
        } while (true);
        System.out.println("Thank you for using this program.");
    }
}
```

This should get you started.

Nice, but I see that you’re assuming the user will never type more than one number on a line.

So I could get by *without* a tokenizer. I agree, but I used one for the sake of practice.
I have a larger set of problems for next time. Can’t wait. Let’s press on with the ones for today. Next one up: problem 6.4, page 263.

Simulated the wandering of an intoxicated person in a square street grid. For 100 times, have the simulated drunkard randomly pick a direction (east, west, north, south) and move one block in the chosen direction. After the iterations, display the distance that the drunkard has covered. (one might expect that on average the person might not get anywhere because the moves to different directions cancel another out in the long run, but in fact it can be shown that with probability 1 (certainty) the person eventually moves outside any finite region.

<table>
<thead>
<tr>
<th>There are many ways of solving problem 6.4.</th>
<th>Yes, but empathy would not be one of them.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Of course. What I meant was that you need to generate random directions.</td>
<td>And you have more than one way to produce random numbers.</td>
</tr>
<tr>
<td>Let’s use whatever the book uses.</td>
<td>It’s a good book.</td>
</tr>
<tr>
<td>I know.</td>
<td>What book?</td>
</tr>
<tr>
<td>In addition to that, I would like to use an object oriented approach.</td>
<td>And bring a drunkard into the picture.</td>
</tr>
<tr>
<td>Exactly.</td>
<td>A drunkard is like a BankAccount.</td>
</tr>
<tr>
<td>I was going to say.</td>
<td>Well, here’s how I’d get started.</td>
</tr>
</tbody>
</table>

```java
import java.util.*;
class Drunkard {
    int x, y;
    Drunkard(int x, int y) {
        this.x = x;
        this.y = y;
    }
    void moveNorth() {
        this.y -= 1;
    }
    void moveEast() {
        this.x += 1;
    }
    void report() {
        System.out.println("Hiccup: " + x + ", " + y);
    }
}
class Four {
    public static void main(String[] args) {
        Random generator = new Random();
    }
```
Drunkard drunkard = new Drunkard(100, 100);
int direction;
for (int i = 0; i < 100; i++) {
    direction = Math.abs(generator.nextInt()) % 4;
    if (direction == 0) { // N
        drunkard.moveNorth();
    } else if (direction == 1) { // E
        drunkard.moveEast();
    } else if (direction == 2) { // S
        System.out.println("Should move South.");
    } else if (direction == 3) { // W
        System.out.println("Should move West.");
    } else {
        System.out.println("Impossible!");
    }
    drunkard.report();
}

Not bad at all. Of course, one needs to finish it first.

Reminds me of homework assignment two. Somewhat. Now let’s look at 6.5 and 6.6.

Suppose a cannonball is propelled vertically into the air with a starting velocity $v_0$. Any calculus book will tell us that the position of the ball after $t$ seconds is

$$s(t) = -0.5gt^2 + v_0t$$

where $g = 9.81 \, \text{m/s}^2$ is the gravitational force of the earth. No calculus book ever mentions why someone would want to carry out such an obviously dangerous experiment, so we will do it in the safety of the computer. In fact, we will confirm the theorem from calculus by a simulation. In our simulation, we will consider how the ball moves in very short time intervals $\Delta t$. In a short time interval the velocity $v$ is nearly constant, and we can compute the distance the ball moves as $\Delta s = v\Delta t$. In our program, we will simply set

$$\text{double deltaT} = 0.01;$$

and update the position by

$$s = s + v \times \text{deltaT};$$

The velocity changes constantly—in fact it is reduced by the gravitational force of the earth. In a short time interval, $v = -g\Delta t$, and we must keep the velocity updated as

$$v = v - g \times \text{deltaT};$$
In the next iteration the new velocity is used to update the distance. Now run the simulation until the cannonball falls back onto earth. Get the initial velocity as an input (100 m/s is a good value). Update the position and velocity 100 times per second, but print out the position only every full second. Also print out the the values from the exact formula \( s(t) = -0.5gt^2 + v_0t \) for comparison.

What is the benefit of this kind of simulation when an exact formula is not available? Well, the formula from the calculus book is not exact. Actually the gravitational force diminishes the further the cannonball is away from the surface of the earth. This complicates the algebra sufficiently that it is not possible to give an exact formula for the actual motion, but the computer simulation can simply be extended to apply a variable gravitational force. For cannonballs, the calculus book formula is actually good enough, but computers are necessary to compute accurate trajectories for higher-flying objects such as ballistic missiles.

Now to complete the picture we need to say that most cannonballs are not shot upright but at an angle. If the starting velocity has magnitude \( v \) and the starting angle is \( \alpha \), then the velocity is actually a vector with components \( v_x = v \cos \alpha \) and \( v_y = v \sin \alpha \). In the \( x \)-direction the velocity does not change. In the \( y \)-direction the gravitational force takes its toll. Repeat the simulation from the previous exercise, but store the position of the cannonball as a Point2D variable. Update the \( x \) and \( y \) positions separately, and also update the \( x \) and \( y \) components of the velocity separately. Every full second, plot the location of the cannonball on the graphics display. Repeat until the cannonball has reached the earth again.

This kind of problem is of historical interest. The first computers were designed to carry out just such ballistic calculations, taking into account the diminishing gravity for high-flying projectiles and wind speeds.

<table>
<thead>
<tr>
<th>Isn’t a cannonball like a drunkard?</th>
<th>Yes, they’re both tiggers.</th>
</tr>
</thead>
</table>

They all deal directly with gravitational fields.

Cannonballs require more physics, though.

class Cannonball {
    double x;
    double y;
    double vx;
    double vy;
    final double g = 9.81;
    Cannonball(double speed, double angle) {
        x = 0;
        y = 0;
        vx = Math.cos(angle) * speed;
        vy = Math.sin(angle) * speed;
    }
    void move() {
        double deltaT = 0.01;
        x += vx * deltaT;
        y += vy * deltaT;
        vy -= g * deltaT;
    }
    void report() {
        System.out.println("Located at: (" + x + ", " + y + ")");
    }
}
class FiveAndSix {
  public static void main(String[] args) {
    Cannonball c = new Cannonball(10, Math.PI / 4);
    for (int i = 0; i < 10 * 100; i++) { // flying 10 seconds
      c.move();
      c.report();
      if (c.height() < 0) {
        System.out.println("SPL000F! The cannonball landed!");
        break;
      }
    }
  }
}

Homework assignment one! Almost, only in finer steps.

Could you also compute the highest point that the cannonball gets to?

That would almost be problem 16, wouldn’t it?

Problem 16. Write a program that reads a series of floating-point numbers and prints:

- the maximum value
- the minimum value
- the average value

It would. We’d have to update our notion of a current maximum every time we are looking at a height.

Come to think of it the cannonball could do it. Intelligent cannonball.

Intelligent, but misguided. Well, here it is anyway:

class Cannonball {
  double x;
  double y;
  double vx;
  double vy;
  final double g = 9.81;
  double max;
  Cannonball(double speed, double angle) {
    x = 0;
    y = 0;
    vx = Math.cos(angle) * speed;
    vy = Math.sin(angle) * speed;
  }
}
vy = Math.sin(angle) * speed;

max = 0;
}

void move() {
    double deltaT = 0.01;
    x += vx * deltaT;
    y += vy * deltaT;
    vy -= g * deltaT;
    if (y > max) { max = y; }
}

void report() {
    System.out.println("Located at: (" + x + ", " + y + ")");
    System.out.println(" max altitude so far: " + max);
}

double height() {
    return y;
}

}

class Max {
    public static void main(String[] args) {
        Cannonball c = new Cannonball(10, Math.PI / 4);
        for (int i = 0; i < 10 * 100; i++) {
            // flying 10 seconds
            c.move();
            c.report();
            if (c.height() < 0) {
                System.out.println("The cannonball landed!");
                break;
            }
        }
    }
}

You just added three lines? Four, and yes, that was all.

But how often do those get executed? They’re part of the cannonball’s movement.

Let’s move on to problem 6.7. This one is easy.

The Fibonacci sequence is defined by the following rule. The first two values in the sequence are 1 and 1. Every subsequent value is the sum of the two values preceding it. Write a program that prompts the user for n and prints the nth value in the Fibonacci sequence.

Hint: this problem is easy.

Yes, from two values we compute a third. And we keep doing this over and over again. Now only this new value and the most recent of the two it was computed from should be kept.

And these two values are then added to compute a new value. And the whole process is repeated, as follows:
for (int i = 3; i <= n; i++) {
    fNew = fOld1 + fOlder;
    fOlder = fOld1;
    fOld1 = fNew;
}

Yes, that’s it. This reminds me of another problem:

What problem is that? See if you can figure it for yourself.

import java.io.*;
class Mysterly {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Pass the salt please: ");
        double a = console.readDouble();
        System.out.print("And the butter: ");
        int n = console.readInt();
        double xold, xnew;
        xold = ... ;
        do {
            xnew = xold - (Math.pow(xold, n) - a) / (n * Math.pow(xold, n - 1));
            xold = xnew;
            System.out.println(" " + (Math.pow(xnew, n) - a));
        } while (Math.abs(Math.pow(xnew, n) - a) > 0.001);
        System.out.println(xnew);
        System.out.println(Math.pow(xnew, n) + " " + a);
        System.out.println("Thank you!");
    }
}

P6.14 perhaps? But what’s the ...? It doesn’t matter, if it has a value.

For this problem, at least. Problems 6.9, 6.10, and 6.15 are easy, although for 10 and 15 we need to wait until we look at applets to do any graphics.

Let’s do number 10. OK, but I won’t draw circles, I’ll just create them.

Drunkards, circles, cannonbals: they’re all the same.

Especially drunkards. Here’s the code:

import java.util.*;
class Ten {
    public static void main(String[] args) {
        System.out.print("Enter number of circles: ");
        ConsoleReader console = new ConsoleReader(System.in);
        int n = console.readInt();
        System.out.println("Generating " + n + " circles.");
Random generator = new Random();
for (int i = 0; i < n; i++) {
    int x = 100 + Math.abs(generator.nextInt()) % 200;
    int y = 100 + Math.abs(generator.nextInt()) % 200;
    int r = 10 + Math.abs(generator.nextInt()) % 40;
    Circle e = new Circle(x, y, r);
    System.out.println(e); // looks better when you draw it
}
System.exit(0);

class Circle {
    double xCenter, yCenter, radius;
    Circle (double x, double y, double r) {
        xCenter = x;
        yCenter = y;
        radius = r;
    }
    public String toString() {
        return "I am a circle at: (" +
            xCenter + ", ", + yCenter +
            ") with a radius of "+
            radius;
    }
}

| Why is 6.15 easy? | Because it's like 10, and in addition you have complete information about the position of the squares. You'll see, later. |
| Why is 6.15 hard? | Because you should come up with a formula for the position of a square given its row and column (or line and column). |
| This way you can use a for loop for the lines, | ...and another one for the columns. |
| One, inside each other. | Like for the patterns we developed last week. |
| And why is 6.9 easy? | Because you just have to compute two sums, and take a square root at the end. |
| Why is 6.9 hard? | 6.9 is not hard, but you have to read the problem carefully and use the right formula. |
| Which is the last one (second of two). | The last one for today anyway. |
| We'll do a lot more next time. | As always, I can hardly wait. |
Here's another approach to producing random numbers.

Say we need random numbers between \( a \), and \( b \), where \( a \) is less than \( b \). Let \( y \) be such a random number. Then we can calculate how far into the segment \( y \) is, as a percentage.

\[
\text{double } p = (y - a) / (b - a); // [1]
\]

Since \( y \) is random that means \( p \) can be anywhere between 0 and 1.

Well, such numbers can be generated with \texttt{Math.random()}.

So, let's turn [1] on its head, to obtain \( y \) as a function of \( p \).

Then, we have:

\[
y = p * (b - a) + a;
\]

Now replace \( p \) by \texttt{Math.random()}:

\[
y = (b - a) * \texttt{Math.random()} + a;
\]

Same as stretching followed by a translation. End of story.

**Problem 6.14.** Write a program that asks the user for an integer and then prints out all its factors. For example, when the user enters 150, the program should print: 2 3 5 5.

**Problem 6.9.** Write a program that reads a set of floating-point data values from the input. When the end of file is reached, print out the count of the values, the average, and the standard deviation. The average of a data set \( x_{i=1...n} \) is \( \mu = \sum x_i / n \). The standard deviation is

\[
S = \sqrt{\frac{\sum (x_i - \mu)^2}{n - 1}}
\]

However that formula is not suitable for our task. by the time you have computed the mean, the individual \( x_i \)'s are long gone. Until you know how to save these values, use the numerically less stable formula

\[
S = \sqrt{\frac{\sum x_i^2 - \frac{1}{n}(\sum x_i)^2}{n - 1}}
\]

You can compute this quantity by keeping track of the count, the sum, and the sum of squares as you process the input values.

**Problem 6.10.** Write a graphical applet (?)—see if you get around that) which prompts a user to enter a number \( n \) and then draws \( n \) circles with random center and random radius.

**Problem 6.15.** Write a program that prompts the user for an integer and then prints out all prime numbers up to that integer. For example, when the user enters 20, the program should print: 2 3 5 7 11 13 17 19. Recall that a number is a prime number if it is not divisible by any number except 1 and itself.
Computer Games

Computer Games. (In which Tigger has guests from another tale).

Have you ever played Nim?  No, But do you play croquet?

I’d love to but I don’t have the time.  Well, how do you play Nim?

Allow me to describe it.  Very well, but please be very clear, my dear.

Yes, try to be very clear.
Nim is a well-known game with a number of variants. We will consider the following variant, which has an interesting winning strategy.

Two players alternately take marbles from a pile. In each move, a player chooses how many marbles to take, then removes the marbles.

The player must take

- at least one but
- at most half

of the marbles.

You already said that.

I thought you were sleeping.

This is very provoking...

Sorry, dear. *Ahem.*

The player who takes the last marble loses.

You will write a program in which a computer plays against a human opponent.

Generate a random number between 10 and 100 to denote the initial size of the pile.

Generate a random integer between 0 and 1 to decide whether the computer or the human takes the first turn. Then start the game.

This variant of the game has an interesting winning strategy. Careful thinking will reveal that whoever moves first can win.

How?

Take off enough marbles to make the pile a power of two minus one, that is 1, 3, 7, 15, 31, or 63.

I see.

One could program the computer to always play in what could be called *smart* mode.

That would not be too much fun for the user.

One could also program the computer to always perform a random legal move.

That’s what we’ll do, as it seems a bit more fair.

OK, let’s get started.

First of all, what do we need?

No pile of marbles, no Nim.

Let’s bring one in.

How does it look?

What do you think of this?

```java
class PileOfMarbles {
    int height;
    PileOfMarbles (int height) {
        this.height = height;
    }
    int report() {
        return this.height;
    }
}
```
A pile of marbles is like a bank account. Whoever withdraws the last cent loses.

It has a balance (height). An a get balance (report) method.

We also need a withdraw, don’t we? Yes, let’s call it move.

What do you think of this? You have to be careful when withdrawing.

```java
void move(int number) {
    System.out.println("***Removing " + number + " marbles from the pile.");
    this.height -= number;
    System.out.println("Pile of marbles is now: " + this.report());
}
```

So how do you check that? How about this condition?

```java
if (number <= 0 || ((number > height / 2) && (number != 1))) {
    // then it’s a bad move
} else {
    // it is a good move
}
```

Can I look at this again? You mean the check for a bad move?

Yes. Here it is (a bad move).

```java
if (number <= 0 || ((number > height / 2) && (number != 1))) {
    System.out.println("***Bad move: you lose.");
    System.exit(0);
} else {
```

Or, just the condition, again:

```java
(number <= 0 || ((number > height / 2) && (number != 1)))
```

Either zero marbles or less, ... ...or more than half, and not the last.

I think this last part is rather tricky. Using de Morgan’s law we can reformulate this to represent a good move. Perhaps that would help.

Yes, let’s write down its negation. Here it is (a good move):

```java
(number > 0 && ((number <= height / 2) || (number == 1)))
```

To me this looks better, easier to understand. It does seem that way to me too.

Even if this is no easier to read than the first version you now have a choice, an option. I quite agree with that.
So we can finish report now. Yes. Let's make the pile of marble responsible for announcing the end of the game too.

Then it needs to know who moved. I think I can accommodate that.

```java
void move(int number, String user) {
    System.out.println("***Removing " + number + " marbles from the pile for: " + user);
    if (number <= 0 || ((number > height / 2) && (number != 1))) {
        System.out.println("***Bad move for " + user + ". " + user + " loses.");
        System.exit(0);
    } else {
        this.height -= number;
        if (this.height == 0) {
            System.out.println("***End of game. " + user + " loses.");
            System.exit(0);
        }
    }
    System.out.println("Pile of marbles is now: " + this.report());
}
```

That's pretty much it, isn't it? Yes. Now we need to set up the game.

We need a while loop. We need a loop, like we did in Echo, yes.

Here's my suggestion. Looks like you finished it altogether now.

```java
int height = (int)(Math.random() * 90 + 10);
PileOfMarbles pile = new PileOfMarbles(height);
System.out.println("Game starts with a pile of height: " + pile.report());

int number, currentHeight;
while (true) {
    System.out.println("*** Computer moves.");
    System.out.println("Pile of marbles of height: " + pile.report());
    currentHeight = pile.report();
    if (currentHeight == 1) {
        number = 1;
    } else {
        number = (int)(Math.random() * (currentHeight / 2)) + 1;
    }
    System.out.println("Computer chooses to remove: " + number + " marbles.");
    pile.move(number, "Computer");
    System.out.println("--------------------------");
    System.out.println("*** Now " + user + " has to move.");
    System.out.println("Pile of marbles of height: " + pile.report());
    System.out.print(user + ", please enter number of marbles you want to take: ");
    number = console.readInt();
```
pile.move(number, user);

System.out.println("---------------------");

}

Yes, here's the whole thing:

class Nim {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);

        System.out.println("Hello, and welcome to the game of Nim!");
        System.out.print("What is your name: ");
        String user = console.readLine();

        int height = (int)(Math.random() * 90 + 10);
        PileOfMarbles pile = new PileOfMarbles(height);

        System.out.println("Game starts with a pile of height: 
                           + pile.report());

        int number, currentHeight;

        while (true) {
            System.out.println("*** Computer moves.");

            System.out.println("Pile of marbles of height: 
                               + pile.report());

            currentHeight = pile.report();

            if (currentHeight == 1) {
                number = 1;
            } else {
                number = (int)(Math.random() * (currentHeight / 2)) + 1;
            }

            System.out.println("Computer chooses to remove: 
                                + number + " marbles.");

            pile.move(number, "Computer");

            System.out.println("---------------------");
            System.out.println("*** Now "+ user + " has to move.");
            System.out.println("Pile of marbles of height: 
                                + pile.report());
            System.out.print(user + 
                                ", please enter number of marbles you want to take: ");
number = console.readInt();
pile.move(number, user);
System.out.println("-----------------------");
}
}

class PileOfMarbles {
    int height;
PileOfMarbles (int height) {
        this.height = height;
    }
    int report() {
        return this.height;
    }
    void move(int number, String user) {
        System.out.println("***Removing " + number + " marbles from the pile for: " + user);
        if (number <= 0 || ((number > height / 2) && (number != 1))) {
            System.out.println("***Bad move for " + user + ". " + user + " loses.");
            System.exit(0);
        } else {
            this.height -= number;
            if (this.height == 0) {
                System.out.println("***End of game. " + user + " loses.");
                System.exit(0);
            }
        }
        System.out.println("Pile of marbles is now: " + this.report());
    }
}

Don’t forget your ConsoleReader!                          I’m ready!

Let’s play!                                             (This is Problem Six-Seventeen).
frilled.cs.indiana.edu%javac Nim.java
frilled.cs.indiana.edu%java Nim
Hello, and welcome to the game of Nim!
What is your name: [Mary-Ann]
Game starts with a pile of height: 86
*** Computer moves.
Pile of marbles of height: 86
Computer chooses to remove: 23 marbles.
***Removing 23 marbles from the pile for: Computer
Pile of marbles is now: 63
-----------------------------------
*** Now Mary-Ann has to move.
Pile of marbles of height: 63
Mary-Ann, please enter number of marbles you want to take: 30
***Removing 30 marbles from the pile for: Mary-Ann
Pile of marbles is now: 33
-----------------------------------
*** Computer moves.
Pile of marbles of height: 33
Computer chooses to remove: 1 marbles.
***Removing 1 marbles from the pile for: Computer
Pile of marbles is now: 32
-----------------------------------
*** Now Mary-Ann has to move.
Pile of marbles of height: 32
Mary-Ann, please enter number of marbles you want to take: 16
***Removing 16 marbles from the pile for: Mary-Ann
Pile of marbles is now: 16
-----------------------------------
*** Computer moves.
Pile of marbles of height: 16
Computer chooses to remove: 7 marbles.
***Removing 7 marbles from the pile for: Computer
Pile of marbles is now: 9
-----------------------------------
*** Now Mary-Ann has to move.
Pile of marbles of height: 9
Mary-Ann, please enter number of marbles you want to take: 2
***Removing 2 marbles from the pile for: Mary-Ann
Pile of marbles is now: 7
-----------------------------------
*** Computer moves.
Pile of marbles of height: 7
Computer chooses to remove: 1 marbles.
***Removing 1 marbles from the pile for: Computer
Pile of marbles is now: 6
-----------------------------------
*** Now Mary-Ann has to move.
Pile of marbles of height: 6
Mary-Ann, please enter number of marbles you want to take: 3
***Removing 3 marbles from the pile for: Mary-Ann
Pile of marbles is now: 3
-------------------------------
*** Computer moves.
Pile of marbles of height: 3
Computer chooses to remove: 1 marbles.
***Removing 1 marbles from the pile for: Computer
Pile of marbles is now: 2
-------------------------------
*** Now Mary-Ann has to move.
Pile of marbles of height: 2
Mary-Ann, please enter number of marbles you want to take: [1]
***Removing 1 marbles from the pile for: Mary-Ann
Pile of marbles is now: 1
-------------------------------
*** Computer moves.
Pile of marbles of height: 1
Computer chooses to remove: 1 marbles.
***Removing 1 marbles from the pile for: Computer
***End of game. Computer loses.
frilled.cs.indiana.edu%java Nim
Hello, and welcome to the game of Nim!
What is your name: Queen
Game starts with a pile of height: 77
*** Computer moves.
Pile of marbles of height: 77
Computer chooses to remove: 8 marbles.
***Removing 8 marbles from the pile for: Computer
Pile of marbles is now: 69
-------------------------------
*** Now Queen has to move.
Pile of marbles of height: 69
Queen, please enter number of marbles you want to take: [33]
***Removing 33 marbles from the pile for: Queen
Pile of marbles is now: 36
-------------------------------
*** Computer moves.
Pile of marbles of height: 36
Computer chooses to remove: 11 marbles.
***Removing 11 marbles from the pile for: Computer
Pile of marbles is now: 25
-------------------------------
*** Now Queen has to move.
Pile of marbles of height: 25
Queen, please enter number of marbles you want to take: [12]
***Removing 12 marbles from the pile for: Queen
Pile of marbles is now: 13
-------------------------------
*** Computer moves.
Pile of marbles of height: 13
Computer chooses to remove: 5 marbles.
***Removing 5 marbles from the pile for: Computer
Pile of marbles is now: 8

*** Now Queen has to move.
Pile of marbles of height: 8
Queen, please enter number of marbles you want to take: [1]
***Removing 1 marbles from the pile for: Queen
Pile of marbles is now: 7

*** Computer moves.
Pile of marbles of height: 7
Computer chooses to remove: 2 marbles.
***Removing 2 marbles from the pile for: Computer
Pile of marbles is now: 5

*** Now Queen has to move.
Pile of marbles of height: 5
Queen, please enter number of marbles you want to take: [2]
***Removing 2 marbles from the pile for: Queen
Pile of marbles is now: 3

*** Computer moves.
Pile of marbles of height: 3
Computer chooses to remove: 1 marbles.
***Removing 1 marbles from the pile for: Computer
Pile of marbles is now: 2

*** Now Queen has to move.
Pile of marbles of height: 2
Queen, please enter number of marbles you want to take: [1]
***Removing 1 marbles from the pile for: Queen
Pile of marbles is now: 1

*** Computer moves.
Pile of marbles of height: 1
Computer chooses to remove: 1 marbles.
***Removing 1 marbles from the pile for: Computer
***End of game. Computer loses.

Oh, Nim is so easy. I like Nim.
Hello, and welcome to the game of Nim!
What is your name: Mary-Ann
Game starts with a pile of height: 11
*** Computer moves.
Pile of marbles of height: 11
Computer chooses to remove: 4 marbles.
***Removing 4 marbles from the pile for: Computer
Pile of marbles is now: 7

-------------
*** Now Mary-Ann has to move.
Pile of marbles of height: 7
Mary-Ann, please enter number of marbles you want to take: 6
***Removing 6 marbles from the pile for: Mary-Ann
frilled.cs.indiana.edu%

“Why, Mary Ann, what are you doing out here?”

“Run home this moment, and fetch me a pair of gloves and a fan! Quick, now!”
### Designing Fractions

You mentioned De Morgan’s name yesterday. 

Augustus De Morgan (1806-1871), indeed.

http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/De_Morgan.html

<table>
<thead>
<tr>
<th>You mentioned De Morgan’s name yesterday.</th>
<th>Augustus De Morgan (1806-1871), indeed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>De Morgan was the one that, in 1838, defined and introduced the term <em>mathematical induction</em>, thus putting a process that had been used without clarity on a rigorous basis.</td>
<td>I don’t know if he ever played <em>Nim</em>, but here’s how he was described by some of his colleagues: &quot;*A dry dogmatic pedant I fear is Mr De Morgan, notwithstanding his unquestioned ability.&quot;</td>
</tr>
<tr>
<td>De Morgan was never a Fellow of the Royal Society as he refused to let his name be put forward. He also refused an honorary degree from the University of Edinburgh.</td>
<td>He recognised the purely symbolic nature of algebra and he was aware of the existence of algebras other than ordinary algebra.</td>
</tr>
<tr>
<td>He introduced De Morgan’s laws and his greatest contribution is as a reformer of mathematical logic.</td>
<td>He introduced De Morgan’s laws and his greatest contribution is as a reformer of mathematical logic.</td>
</tr>
<tr>
<td>Very interesting. What other mathematicians were involved in the lecture notes of yesterday?</td>
<td>Charles Lutwidge Dodgson (1832-1898).</td>
</tr>
<tr>
<td>Charles Dodgson is known especially for <em>Alice’s adventures in wonderland</em> (1865) and <em>Through the looking glass</em> (1872), children’s books that are also distinguished as satire and as examples of verbal wit.</td>
<td>He invented his pen name of Lewis Carroll by anglicizing the translation of his first two names into the Latin ‘<em>Carus Ludovicus</em>’.</td>
</tr>
<tr>
<td>As a mathematician, Dodgson was conservative.</td>
<td>He was the author of a fair number of mathematics books, for instance <em>A syllabus of plane algebraical geometry</em> (1860).</td>
</tr>
</tbody>
</table>

http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/Dodgson.html

375
None of his math books have proved of enduring importance except for *Euclid and his modern rivals* (1879) which is of historical interest.

You bet he did.

As a logician, he was more interested in logic as a game than as an instrument for testing reason.

“I know what you’re thinking about,” said Tweedle-dum: “but it isn’t so, nohow.”

“Contrariwise,” continued Tweedledee, “if it was so, it might be; and if it were so, it would be: but as it isn’t, it ain’t. That’s logic.”

Here’s a picture of him.

Yes, he’s the one in the middle.

He contributed in *Jabberwocky*, the word *chortle* (a word that combines *snort* and *chuckle*) to the English language.

```
frilled.cs.indiana.edu%webster chortle
chor-tle vb  chor-tled; chor-tling
[bmeld of chuckle and snort]
vi
(1872)
1: to sing or chant exultantly <he chortled in his joy --Lewis Carroll>
2: to laugh or chuckle esp. in satisfaction or exultation
~ vt :to say or sing with a chortling intonation
-- chortle n
-- chor-tler n
```

Yet *nomeraths* and *brilkg* didn’t quite make it.
Today we’re going to implement Fractions. And in the process mention Euclid (325-265).

http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/Euclid.html

<table>
<thead>
<tr>
<th>Have you noticed the numbers?</th>
<th>Yes, that was a long time ago!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclid’s most famous work is his treatise on mathematics <em>The Elements</em>.</td>
<td>The book was a compilation of knowledge that became the centre of mathematical teaching for 2000 years.</td>
</tr>
<tr>
<td>The <em>Elements</em> is divided into 13 books.</td>
<td>Books one to six deal with plane geometry.</td>
</tr>
<tr>
<td>Books seven to nine deal with number theory.</td>
<td>In particular book seven is a self-contained introduction to number theory and contains the Euclidean algorithm for finding the greatest common divisor of two numbers.</td>
</tr>
<tr>
<td>Which we will use today.</td>
<td>Very good.</td>
</tr>
<tr>
<td>A fraction is of course the ratio of two integers:</td>
<td>We will define a class Fraction, which will supply the necessary operations on fractions.</td>
</tr>
<tr>
<td>• a numerator and</td>
<td></td>
</tr>
<tr>
<td>• a denominator</td>
<td></td>
</tr>
<tr>
<td>There are many ways in which we could do this.</td>
<td>Here’s a summary, to be augmented with a more detailed explanation in class.</td>
</tr>
<tr>
<td>Let’s look first at the new part.</td>
<td>Of which there are two parts, as well.</td>
</tr>
</tbody>
</table>
| ```java
class Euclid {
    static int gcd(int a, int b) {
        a = Math.abs(a);
        b = Math.abs(b);
        if (a == 0) return b; // 0 is error value
        if (b == 0) return a;
        int temp;
        while (b > 0) {
            temp = a % b;
            a = b;
            b = temp;
        }
        return a;
    } // there are other ways too...
}
``` |
| We need to get the hang of it, first. |
| Then, when we become comfortable using it, we need to become sure it always works right. | Proof is the bottom line for everyone. |
That’s from Paul Simon, isn’t it? Yes, but it applies here.

Both goals may take a long time. But when you’re done you can write Fraction.

Like this. That’s a lot.

```java
class Fraction {
    private int numerator;
    private int denominator;
    public Fraction(int num, int den) {
        int divisor;
        if (den == 0) {
            System.out.println("Fraction with denominator zero!");
            System.exit(1);
        }
        if (num == 0) { numerator = 0; denominator = 1; }
    else {
        if (den < 0) {
            num *= -1;
            den *= -1;
        }
        if ((divisor = Euclid.gcd(num, den)) != 1) {
            num /= divisor;
            den /= divisor;
        }
        numerator = num;
        denominator = den;
    }
    }
    public String toString() {
        String fraction;
        if (denominator == 1) { fraction = numerator + ";"; }
    else { fraction = numerator + "/" + denominator; }
        if (denominator * numerator < 0) {
            return "(" + fraction + ");";
    else {
            return fraction;
        }
    }
    public boolean isZero() {
        return (denominator == 1 && numerator == 0);
    }
    public boolean isInt() {
        return (denominator == 1);
    }
    public boolean equals(Fraction other) {
        return (numerator == other.numerator && denominator == other.denominator);
    }
    public boolean greaterThan(Fraction other) {
        return (numerator * other.denominator >
```

denominator * other.numerator);
}
public Fraction minus(Fraction other) {
    return new Fraction(
        numerator * other.denominator - other.numerator * denominator,
        denominator * other.denominator
    );
}
public Fraction plus(Fraction other) {
    return new Fraction(
        numerator * other.denominator + other.numerator * denominator,
        denominator * other.denominator
    );
}
public Fraction times(Fraction other) {
    return new Fraction(numerator * other.numerator, denominator * other.denominator);
}
public Fraction divideBy(Fraction other) {
    return new Fraction(numerator * other.denominator, denominator * other.numerator);
}
public static void main(String[] args) {
    Fraction f = new Fraction(6, 9);
    Fraction g = new Fraction(-4, 6);
    System.out.println("Test of operations: ");
    System.out.println(" Add: " + f + " + " + g + " = " + f.plus(g));
    System.out.println(" Sub: " + f + " - " + g + " = " + f.minus(g));
    System.out.println(" Mul: " + f + " * " + g + " = " + f.times(g));
    System.out.println(" Div: " + f + " / " + g + " = " + f.divideBy(g));
    System.out.println("Test of predicates: ");
    System.out.print(" 1. Does " + f + " equal " + g + "? ");
    System.out.println(" The answer is: " + f.equals(g));
    Fraction h = new Fraction(8, -2);
    System.out.print(" 2. Is " + h + " an integer? ");
    System.out.println("The answer is: " + h.isInt());
    Fraction i, j;
    i = (f.minus(g)).times(f.plus(g));
    j = f.times(f).minus(g.times(g));
    System.out.print(" 3. Does " + i + " equal " + j + "? ");
    System.out.println("The answer is: " + i.equals(j));
    System.out.print(" 4. Is 5/8 greater than 2/3? The answer is: ");
    System.out.println((new Fraction(5, 8)).greaterThan(new Fraction(2, 3)));
}

There are two parts to it.
First, the blueprint.

Then, the main.
Both important.

When you're done you can improve it.
boolean equals(Fraction other) {
    return (this.minus(other)).isZero();
}

That’s a different equals.

Here’s a different greaterThan.

boolean greaterThan(Fraction other) {
    return (this.minus(other)).isPositive();
}

That wouldn’t work just yet.

I know, you need another predicate. Can I write it?

Sure, what’s its signature? boolean isPositive() is it’s signature.

It’s a simple one. I agree.

boolean isPositive() {
    return numerator * denominator > 0;
}

This was a long example. Long, but useful.

And interesting. If you say so...

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.
Tokens and Tokenizers

The practical exam is coming, let’s work out some problems.

**Problem 1** Write a program *stenographer* that accepts lines of text from the user and translates them (one by one) in shorthand. For the purposes of this exercise let’s define *shorthand notation* as the one in which all vowels are removed from all the words everywhere.

Thus shorthand version of

**Good evening, and welcome to Minneapolis!**

would be

**Gd vnng, nd wlc m t mnnpls!**

🔍 Solution

Write a program *stenographer*

```java
class Stenographer {
    public static void main(String[] args) {
        //
    }
}
```

that accepts lines of text from the user

```java
class Stenographer {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("Please type a line of text: ");
        String line = c.readLine();
    }
}
```

and translates them (character by character)
class Stenographer {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("Please type a line of text: ");
        String line = c.readLine();
        for (int i = 0; i < line.length(); i++) {
            String temp = line.substring(i, i+1);
            if (temp.equalsIgnoreCase("A") ||
                    temp.equalsIgnoreCase("E") ||
                    temp.equalsIgnoreCase("I") ||
                    temp.equalsIgnoreCase("O") ||
                    temp.equalsIgnoreCase("U")) {
                // skip that letter
            } else {
                System.out.print(temp);
            }
        }
        System.out.println();
    }
}

in shorthand. For our purposes shorthand means removing the vowels everywhere.

class Stenographer {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("Please type a line of text: ");
        String line = c.readLine();
        for (int i = 0; i < line.length(); i++) {
            String temp = line.substring(i, i+1);
            if (temp.equalsIgnoreCase("A") ||
                    temp.equalsIgnoreCase("E") ||
                    temp.equalsIgnoreCase("I") ||
                    temp.equalsIgnoreCase("O") ||
                    temp.equalsIgnoreCase("U")) {
                // skip that letter
            } else {
                System.out.print(temp);
            }
        }
        System.out.println();
    }
}

Here's running it twice:

    school.cs.indiana.edu%java Stenographer
    Please type a line of text: Good evening, and welcome to Minneapolis!
    Gd vnng, nd wlc m Mnnpls!
    school.cs.indiana.edu%java Stenographer
    Please type a line of text: The Stenographer program works pretty well!
    Th Stngrphr prgrm wrks prtty wll!
    school.cs.indiana.edu%

Of course, you could change or enhance it in many ways, so go ahead and experiment with it!

**Problem 2** Take a look at this problem, and become clear how it works, first.
Your lab assignment seven (below) is based on this and the previous question.
Your task is to write a program that simulates a vending machine that sells items which are worth 65 cents.
Your program (called Vending) starts by greeting the user, then accepting coins.
The machine accepts one coin at a time and understands the following coins:

- nickel
- dime and
- quarter

After each coin is entered (into the vending machine, that is, the program) the program prints back the amount that is still needed to purchase the item. When the amount exceeds 65 cents the program will print the change (if any) and thanks the user for making use of it. See below.

Example run:

```
prompt> java Vending
Welcome. Please enter coins:
coin> nickel
60 cents remaining
coin> quarter
35 cents remaining
coin> quarter
10 cents remaining
coin> quarter
Thank you. Your change is: 15 cents.
Thanks for using this program.
```

Here's a possible solution:

```java
class Vending {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Welcome. Please enter coins:");
        int amount = 0;
        final int PRICE = 65;
        while (amount < PRICE) {
            String coin;
            System.out.print("coin> ");
            coin = console.readLine();
            if (coin.equals("nickel")) {
                amount += 5;
            } else if (coin.equals("dime")) {
                amount += 10;
            } else if (coin.equals("quarter")) {
                amount += 25;
            }
            if (amount < 65) {
                System.out.println((PRICE - amount) + " cents remaining.");
            }
        }
    }
}
```
Now we can state the lab assignment.

A201/A597 LAB ASSIGNMENT SEVEN

Write a program that simulates a vendor that sells books of stamps which are worth 3 dollars and 40 cents each. (The vendor only sells whole books, each book is $3.40). Your program (called Vendor) starts by greeting the user, then accepting money.

Your vendor accepts the following monies only:

- cent
- nickel
- dime
- quarter and
- dollar

Your vendor accepts money one line at a time, and should allow more than one coin to be entered on a line. After that, the program calculates and prints the amount that is still needed to purchase the item. When the amount finally exceeds 340 cents the program prints the change (if any) and thanks the user for using the program. In the example below what the customer enters is rendered in boxes.

Example run:

```
school.cs.indiana.edu%java Vendor
Welcome. We sell stamps ($3.40) Please enter money:
enter>  nickel nickel dollar dollar quarter
Thanks. Your credit is $2.35 I need $1.05 more.
enter>  cent cent
Thanks. Your credit is $2.37 I need $1.03 more.
enter>  dollar quarter
Thanks. Your credit is $3.62 The stamps are yours.
Your change is: $0.22
Thanks for using this program.
```

Hints and Advice.

Your strategy should be simple:

- Keep reading lines until you have received enough money.
• Initially you start from having received nothing (zero).
• With each line you need to use a **StringTokenizer**.
• That’s because more than one coin might be specified per line.
• So you need a loop for each line, to look at the words.
• Every word looked at translates into a coin value.
• After every line you check to see if you have enough money.
• If not, you keep going, asking for more (and indicating how much).
• So the loop for each line is inside a bigger loop for the conversation.
• But you knew that, already!

Good luck and let us know if we can be of more help.
Milestones

More about methods.

<table>
<thead>
<tr>
<th>We’ve reached an important milestone now.</th>
<th>Although you may not realize it now,</th>
</tr>
</thead>
<tbody>
<tr>
<td>...it will become apparent in about a week.</td>
<td>Meanwhile I think it’s reasonably safe to say that we’re halfway through now.</td>
</tr>
<tr>
<td>I think so. Diagrams like the ones presented in the previous sets of lecture notes are important.</td>
<td>They can help you understand what’s happening inside a program when it’s compiled and run.</td>
</tr>
<tr>
<td>You should not expect to use diagrams all the time, always, for each and every program.</td>
<td>Their purpose is mostly to help you understand the concepts, the basics, by providing a very detailed picture, as if under a microscope.</td>
</tr>
<tr>
<td>Once you understand those, you’re all set.</td>
<td>And you can think Java without drawing diagrams. You’d be manipulating them in your mind, almost without knowing it.</td>
</tr>
<tr>
<td>Today in class we’ll touch on lecture notes of Wednesday and then start chapter 7 (methods).</td>
<td>We’ll finish chapter 7 quickly, although there will be one important new thing we will touch on.</td>
</tr>
<tr>
<td>Recursion.</td>
<td>The saying goes that “to understand recursion you first need to understand recursion”.</td>
</tr>
<tr>
<td>That’s just a humorous saying.</td>
<td>Recursion, in fact, is easy, and thoroughly useful.</td>
</tr>
<tr>
<td>Once you have a fixed point.</td>
<td>(But we’ll get to that shortly.)</td>
</tr>
<tr>
<td>Not to mention that we have already seen it.</td>
<td>Yes, it was that method in lab five.</td>
</tr>
<tr>
<td>That method, what’s its name?</td>
<td>Well, what was its name, but we changed it.</td>
</tr>
<tr>
<td>That we did.</td>
<td>Now let’s start chapter 7.</td>
</tr>
<tr>
<td>Methods.</td>
<td>What about them?</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>All about them.</td>
<td>You have already implemented several simple methods and are familiar with the basic concepts.</td>
</tr>
<tr>
<td>Let’s go over parameters, return values, and variable scope in a more systematic fashion.</td>
<td>We will also review some of the more technical issues, such as static methods and variables.</td>
</tr>
</tbody>
</table>

When we implement a method we define the parameters of the method.

Here’s an example:

```java
public class BankAccount {
    ...
    public void deposit(double amount) {
        ...
        ...
    }
}
```

The **deposit** method has two parameters: one is explicit, called **amount**, and has type **double**. We expect to receive a value in it to be deposited in the account.

The other one is implicit, and can be referred to by the keyword **this**. What we mean by that is that inside an instance method (like **deposit**) the keyword **this** will always refer to the host object.

So if it’s always available and always in the same way, the **this** reference is not even mentioned in the list of parameters. But from any instance method we can use the keyword **this** to refer to the object that contains the method, always.

Therefore **amount** is called a **formal** parameter for the method **deposit**. When we want to deposit some money we need to know two things:

- a) the account’s name, and

- b) the amount of money

...and we need to actually invoke the method, ...in order to deposit the money.

```java
    // somewhere in main (or another method)
    myChecking.deposit(allowance - 200);
```

In this example **myChecking** is an object of type **BankAccount**... ...and **allowance** is probably a **double**.
Both (are names, and the names) should be declared and initialized before we use them.

When `deposit` starts its work, the value of the allowance - 200 expression becomes the

- *actual parameter*
- *argument*

to the method...

...and will be known by the name of `amount`... while `deposit` is running (working on it).

When the method `returns` (or ends) the formal parameter variables are abandoned (they're disposable) and their values are lost.

The entire process is like a phone call: you call `myChecking.getDeposit` method and you give it some input: the `amount` that you want to deposit.

Once it has that it immediately starts working for you and you stay on the line, waiting for it to tell you: I am done, and your transaction is completed.

You can't write your checks before that.

When it's done it says so, before you hang up.

Sometimes a method also `returns` a value before the end of conversation, but not `deposit`.

deposit only says when it's done, without returning anything. It is declared as `void`.

`void` is its return `type`.

Yes: it does not return *anything* to its caller.

Hence: `void`.

It only does what it is supposed to do, and then it says: "I'm done." And `balance` has changed.

One could also call this `returning` except it’s not as in "returning a value", but rather...

...more like in "returning from a trip".

A trip to the bank.

Explicit parameter variables (the formals) are no different from other variables.

You can modify them during the execution of a method: but unless you have a good reason for that, that is considered bad style.

Let's now consider a more complicated example:

```java
public class BankAccount {
    ...
    public void transfer(BankAccount other, double amount) {
        withdraw(amount);
        other.deposit(amount);
    }
    ...
}
```

This method can be used to transfer money from one account to another.
Here’s how we can use it:

```java
momsSavings.transfer(myChecking, allowance);
```

I have one question before we go further though...

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes, what is it?</td>
<td>Weren’t we supposed to write this.withdraw(amount) in the definition of the transfer method?</td>
</tr>
<tr>
<td>Yes, since we decided to always put an object reference in front of any instance variable name, so please make the correction in your notes.</td>
<td>OK, I’ve updated mine.</td>
</tr>
<tr>
<td>But why does it work though?</td>
<td>Because it defaults to it, anyway.</td>
</tr>
<tr>
<td>But I think that using this makes the code more uniform and explicit.</td>
<td>And I think so too.</td>
</tr>
<tr>
<td>How many formal parameters does this new function (method) have?</td>
<td>Two of them are explicit: other and amount.</td>
</tr>
<tr>
<td>The first one is a BankAccount.</td>
<td>The second one is a double.</td>
</tr>
<tr>
<td>And in addition to that the method will be able to access the object to which it belongs using this.</td>
<td>Yes, this is always available in an instance method and means: &quot;the object that contains this method&quot;, or &quot;this method’s host&quot;.</td>
</tr>
<tr>
<td>What happens when the method is invoked?</td>
<td>When the method is invoked the reference to myChecking is copied into the method’s other formal parameter...</td>
</tr>
<tr>
<td>...and allowance will be copied into amount.</td>
<td>Note that both the object references and the numbers are copied into the method.</td>
</tr>
<tr>
<td>After the method exits, the two bank account balances have changed.</td>
<td>The method was able to change the accounts because it received copies of the object references.</td>
</tr>
<tr>
<td>Of course, the contents of the allowance variable was not changed.</td>
<td>In Java no method can modify the contents of a number variable that is passed as a parameter.</td>
</tr>
<tr>
<td>Or the contents of any other variable of primitive type, for that matter.</td>
<td>Yes: parameters are always passed by value.</td>
</tr>
<tr>
<td>What names should we give to the parameters?</td>
<td>You can give them any names you want.</td>
</tr>
<tr>
<td>As a rule, choose explicit names for parameters that have specific roles. Choose simple names for those that are completely generic.</td>
<td>Your goal is to make the reader understand the purpose of the parameter without having to read the method’s description.</td>
</tr>
</tbody>
</table>
The compiler takes the types of the method parameters and return values very seriously.  

It is an error to call a method by passing it a value of incompatible type,  

...or to use the method in a context that is not compatible with its return type (if any).  

Java is a *strongly typed* language.  

That, it is.  

The compiler automatically converts from int to double and from ordinary classes to superclasses (as we will see chapter 9).  

But it does not convert when there is a possibility of information loss, as we have seen when we discussed *casting*.  

...and does not convert between numbers and strings and objects.  

This is a useful feature, because it lets the compiler find programming errors before they create havoc when the program runs.  

A method that accesses an object and returns some information about it, without changing the object, is called an *accessor* method.  

Such as `getBalance`.  

In contrast, a method that modifies the state of an object is called a *mutator* method.  

`deposit` and `withdraw` are mutator methods.  

You can call an accessor method  

...as many times as you like.  

If that's all you do, you will always get the same answer, and it does not change the state of the object.  

Some classes have been designed such that objects of that kind have only accessor methods and no mutators at all.  

Such classes are called *immutable*.  

An example is the `String` class.  

Once a `String` has been constructed, its contents never change.  

For example, the `substring` method does not remove characters from the original string.  

Instead it constructs a *new* string that contains the substring characters.  

Here's another example of an *accessor* method that simultaneously looks at two objects:

```java
public class BankAccount {
    public boolean equals(BankAccount other) {
        return (this.getBalance() == other.getBalance());
    }
}
```

It makes use of two other accessors (or, rather, the same accessor invoked on two different objects) and compares the values that they return, to come up with an answer.  

And the answer that it returns is the truth value that comes out of (and describes) the comparison.
So we could use it as follows:

```java
if (account1.equals(account2)) {
    // they have the same balance...
} else {
    // they do not have the same balance...
}
```

Very good.

In general the expectation is that accessor methods do not modify any parameters, ...

...and that *mutator* methods do not modify any parameters beyond *this*.

This ideal situation is not always the case. 

Like the *transfer* method discussed before.

It changed its *this*, ...

...while also updating the *other* account.

Such a method is said to have a *side effect*.

A side effect of a method is any kind of *observable behaviour* outside the object.

In an ideal world, all methods would be accessors.

They would simply return an answer without changing any value at all.

In fact, programs that are written in so-called functional programming languages, such as Scheme or ML, come close to this ideal.

**Scheme** is the best! Java is also good.

In an object oriented programming language, we use objects to remember state changes.

Therefore, a method that just changes the state of its implicit parameter is certainly acceptable.

A method that does anything else is said to have a side effect.

While side-effects cannot be completely eliminated, they can be the cause of surprises and problems and should be minimized.

Sometimes you write methods that don’t belong to any particular object.

Such a method is called a *static* (or *class*) method and needs to be declared as *static*.

In contrast, methods such as *getBalance*, *withdraw*, and *deposit* in the preceding sections are often called *instance* methods,

...because they operate on particular instances of an object. There’s one *getBalance* for each *BankAccount* (object) that gets created.

**Have we seen any static methods?**

*Math.sqrt* is a static method.

And every application must have a *static* method where processing begins, called ...

*main*.

Correct.
Here's another example, that involves only numbers:

class NumericMethods {
    public static boolean approxEqual (double x, double y) {
        final double EPSILON = 1E-14;
        double xymax = Math.max(Math.abs(x), Math.abs(y));
        return Math.abs(x - y) <= EPSILON * xymax;
    }
    // more numeric methods could come here...
}

This method encapsulates computation that involves no objects at all, only numbers (and booleans), hence only primitive types.

To call (or use) a static method you need to supply the name of the class, for example:

double r = Math.sqrt(2);
if (NumericMethods.approxEqual(r * r, 2))
    System.out.println("Math.sqrt(2) is approx. 2");

...same as we do with Math.sqrt.

Now it should be clear to you why the main method is a static method. ...when the program starts there may not be any objects at all.

Therefore the first method to be called in a program must be a static method.

Good enough.

To summarize our knowledge about static methods we can say that...

...a static method is a method that does not belong to any object, and that has only explicit parameters. (No this!)

Let's look at some examples now.

public class Example {
    public static void addOneToIt (int number) {
        System.out.println(number);
        number = number + 1;
        System.out.println(number);
    }
    public static void main(String[] args) {
        int value = 3;
        System.out.println(value);
        Example.addOneToIt(value);
        System.out.println(value);
    }
}

Let's walk through the method call.

When the call is made the parameter value is set to the same value as the argument.
The value is copied. Changes to it are not seen outside.

That’s all there is to it. Easy.

Is there a moral to it? In Java method parameters are copied into the parameter variables when a method starts.

Computer scientists call this call mechanism “call by value”, and we mentioned it in lab 2. As you have just seen there are some limitations to the “call by value” mechanism.

It is not possible to implement methods that modify the contents of number variables. Other programming languages support an alternate mechanism, called “by reference”.

This involves passing only the address to where the number variable is stored. This is what happens when you pass an object as an actual parameter.

Let’s see an example. Oh, boy. I like examples best.

class NumberHolder {
    int value = 1;
}
class Example {
    public static void main(String[] args) {
        NumberHolder n = new NumberHolder();
        System.out.println(n.value);
        Example.addOneToIt(n);
        System.out.println(n.value);
    }
    public static void addOneToIt (NumberHolder n) {
        n.value = n.value + 1;
    }
}

But we’ve seen this before, haven’t we?

Yes, when we discussed copying of variables. Primitive types are copied by value, while reference types are copied by reference.

Good enough. References though are still passed by value.

Understood. Can we see an example? Oh boy – that’s what I like best.

class Pair {
    double x;
    double y;
    Pair(double x, double y) {
        this.x = x;
        this.y = y;
    }
    void report() {
}
System.out.println("Hello! I'm at: (" + x + ", " + y + ")");
}
}
class Testing {
    public static void main(String[] args) {
        Pair a = new Pair(100, 0);
        Pair b = new Pair(0, 100);
        a.report();
        b.report();
        Testing.swap(a, b);
        a.report();
        b.report();
    }
    static void swap(Pair a, Pair b) {
        Pair temp = a;
        a = b;
        b = temp;
    }
}

I like it. Easy and understandable. But it still gives you a level of indirection.

Yes. You can, at least in principle, get inside those Pairs. Let's summarize: a Java method can update an object's state using the reference to it, but it cannot change the contents of a reference any more it can change a variable of primitive type.

This shows that object references are passed by value in Java, although we can safely say that ...objects themselves are passed by reference.

Except that the reference itself is copied. Copied, yes – but pointing to the same thing that the original one was.

Fair enough. The distinction is clear now.

A method that has a return type other than void must return a value, by executing a statement of the form:

    return <expression>;

Been there, done that.

Yes, but let's see if we can come up with something new. Well, for one thing, you can return the value of any expression.

You don’t need to store the result in a variable and then return the variable. When a return is processed, the method exits immediately.

This is convenient for handling exceptional cases in the beginning.
Oh, yes, here’s an example:

```java
public static int fibo (int n) {
    if (n == 1)
        return 1;
    else if (n == 2)
        return 1;
    else {
        int f0lder = 1;
        int f0ld = 1;
        int result = f0ld + f0lder;
        for (int i = 3; i <= n; i++) {
            result = f0ld + f0lder;
            f0lder = f0ld;
            f0ld = result;
        }
        return result;
    }
}
```

These are Fibonacci numbers!

http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/Fibonacci.html

Or rather, the method that computes them. Picky, picky, picky! What can I say.

Can you give me an example? Sure, how about add, below.

```java
class Fraction {
    int num;
    int den;
    Fraction(int a, int b) {
        this.num = a;
        this.den = b;
    }
    public String toString() {
        return "(" + num + "/" + den + ")";
    }
    Fraction add(Fraction other) {
        return new Fraction(this.num * other.den + this.den * other.num, 
            this.den * other.den);
    }
    public static void main(String[] args) {
        Fraction a = new Fraction(1, 3);
        Fraction b = new Fraction(2, 3);
        System.out.println(a.toString());
        System.out.println(b);
        System.out.println(a.add(b));
    }
}
```
That's a good example, and so is... toString. But I like add better.

It is important that every branch of a method return a value, that is, a method cannot end without returning a value (if its return type is other than void).

Also, a method whose return type is not void always needs to return a value. Oh, you just said that! Nevermind, although reinforcement is good.

If the method contains several if/else branches make sure that each one of the branches returns an adequate value.

At the end of every possible path through a non-void method there should be a return statement, returning the value of an expression of compatible type.

For example is this right?

<table>
<thead>
<tr>
<th></th>
<th>It is not.</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static int fibo (int n) {</td>
<td>public static int fibo (int n) {</td>
</tr>
<tr>
<td>if (n &lt;= 0)</td>
<td>if (n &lt;= 0)</td>
</tr>
<tr>
<td>System.out.println(&quot;Incorrect argument!&quot;);</td>
<td>System.out.println(&quot;Incorrect argument!&quot;);</td>
</tr>
<tr>
<td>else if (n == 1)</td>
<td>else if (n == 1)</td>
</tr>
<tr>
<td>return 1;</td>
<td>return 1;</td>
</tr>
<tr>
<td>else if (n == 2)</td>
<td>else if (n == 2)</td>
</tr>
<tr>
<td>return 1;</td>
<td>return 1;</td>
</tr>
<tr>
<td>else {</td>
<td>else {</td>
</tr>
<tr>
<td>int fOlder = 1;</td>
<td>int fOlder = 1;</td>
</tr>
<tr>
<td>int f0ld = 1;</td>
<td>int f0ld = 1;</td>
</tr>
<tr>
<td>int result = f0ld + fOlder;</td>
<td>int result = f0ld + fOlder;</td>
</tr>
<tr>
<td>for (int i = 3; i &lt;= n; i++) {</td>
<td>for (int i = 3; i &lt;= n; i++) {</td>
</tr>
<tr>
<td>result = f0ld + fOlder;</td>
<td>result = f0ld + fOlder;</td>
</tr>
<tr>
<td>f0lder = f0ld;</td>
<td>f0lder = f0ld;</td>
</tr>
<tr>
<td>f0ld = result;</td>
<td>f0ld = result;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>return result;</td>
<td>return result;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is not, because if the argument is negative we don't return anything.

What should we return, then?

I don't know, what do you think of this one?

| return Math.round( (Math.pow((1 + Math.sqrt(5))/ 2, n) - Math.pow((1 - Math.sqrt(5))/ 2, n)) / Math.sqrt(5) ); | return Math.round( (Math.pow((1 + Math.sqrt(5))/ 2, n) - Math.pow((1 - Math.sqrt(5))/ 2, n)) / Math.sqrt(5) ); |
| Ha, that was a good one! | Ha, that was a good one! |

Or we should throw an Exception.

Yes, but about those perhaps some other time...

We have now encountered the four kinds of variables that Java supports.

| 1. Instance variables |
| 2. Static variables |
| 3. Local variables |
| 4. Parameter variables |

|
The *lifetime* of a variable defines when the variable is created and how long it stays around.

When an object is constructed, all its instance variables are created.

As long as the object is around its instance variables will also be there, inside the object.

A static variable is created when its class is first loaded, and it lives as long as the class.

A local variable is created when the program enters the statement that defines it.

It stays *alive* until the block that encloses the variable definition is exited.

---

Here's an example:

```java
public void withdraw (double amount) {
    if (amount <= balance) {
        double newBalance = balance - amount;
        // local variable newBalance created and initialized
        balance = newBalance;
    } // end of lifetime of local variable newBalance
}
```

If you tried to print `newBalance` right before the end of the method you'd get an error.

---

Yes, and the reason is: it’s known only in the *then* branch of the if statement.

Inside the inner pair of curly braces.

Can you say that again?

Inside the inner pair of curly braces, *only*.

Very good.

Good to remember.

---

Finally, when a method is called, its parameter variables are created.

They stay alive until the method returns to the caller.

They’re disposable. Every time a new set is used.

Fresh. New scratch paper, as in *what*.

---

Next, let us summarize what we know about the *initialization* of these four types of variables.

Instance variables and static variables are automatically initialized with a default value...

---

...which is

- 0 for numbers (and *chars*),
- *false* for *boolean* and
- *null* for objects (ref. types),

---

So constructors are not *essential*.

They’re *hygienic* instead: convenient and clean.

---

Parameter variables are initialized with copies of the actual parameters.

That’s when the method gets called.

---

Local variables are not initialized by default.

For local variables you must supply an initial value, and the compiler complains if you try to use a local variable that you never initialized.
### Understanding Scopes

<table>
<thead>
<tr>
<th>The scope of a variable is that part of a program that can access it.</th>
<th>The part of the program in which you can access it, the variable, is the scope of the variable, yes.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OK. As you know, instance and static variables are usually declared as private, and you can access them only in the methods of their class.</strong></td>
<td>I see... Scope answers the question: can I see it?</td>
</tr>
<tr>
<td>The scope of a local variable extends from the point of its definition to the end of the enclosing block.</td>
<td>The scope of a parameter variable is the entire body of its method.</td>
</tr>
<tr>
<td>Now let’s look a bit closer to a few situations.</td>
<td>We’re going to go through a few examples.</td>
</tr>
</tbody>
</table>

It sometimes happen that the same variable name is used in two methods:

```java
public static double area(Rectangle rect) {
    double r = rect.getWidth() * rect.getHeight();
    return r;
}
```

These variables (the two r’s) are independent of each other.

You can have variables with the same name r in different methods, ...just as you can have different motels with the same name (let’s say, "Super 8") in different cities.

In this situation the scopes of the two variables are disjoint. Problems arise, however, if you have two or more variable names with overlapping scope.

Like when you have two Kroger’s in the same city. Almost, but not exactly. In Java this situation is called shadowing.

There are rules in the language that tell you which one of the variables you will be referring to if you use the ambiguous name.

Can we see some examples?

Certainly.

```java
class Employee {
    String name;
    Employee (String name) {
        this.name = name;
        // this is mandatory not just good style here!!
    }
}
```
The parameter, which is like a local variable, shadows the instance variable.

The Java language specifies that when there is a conflict between a local variable name and an instance variable name the local variable wins out.

This sounds pretty arbitrary but there is actually a good reason.

You can still refer to the instance variable using this Which you should do anyway.

Do you have any questions? No, but I have something close to that.

An example! You bet.

Consider this:

```java
class Puzzle {
    public static void main(String[] args) {
        Puzzle p = new Puzzle();
        System.out.println("Final result: " + p.fun(6));
    }

    int fun(int n) {
        int result;
        if (n == 0) return 0;
        else {
            // [1]
            result = n + fun(n - 1);
            // [2]
            return result;
        }
    }
}
```

Neat. What do we do with it? Well, what’s the program computing?

This is our old friend what. Yes, what’s its name.

But notice the new name of the method. I know, I know, this is a lot of fun.

Well, isn’t it? I could make it real fun, you know.

How. I could show you the real power of recursion.

I’d like to see that. Consider the Tower of Hanoi problem.

This is a neat little puzzle invented by the French mathematician Edouard Lucas\textsuperscript{31}. Everybody knows it. (The only things unclear is what 7.14, what page 310. Let that be a mistery for now).

We are given a tower of eight disks, initially stacked in decreasing size on one of the three pegs. The objective is to transfer the entire tower to one of the other pegs, moving only one disk at a time and never moving a larger disk onto a smaller one.

<table>
<thead>
<tr>
<th>Let’s solve the problem in general.</th>
<th>Base case first: one disk is easy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now for all other cases by induction.</td>
<td>Assume we can solve the problem for n-1 disks.</td>
</tr>
<tr>
<td>Then the general case becomes easy.</td>
<td>Place the top n-1 disks on middle peg first.</td>
</tr>
<tr>
<td>Then move the largest disk.</td>
<td>Then bring the n-1 disks back on top of it.</td>
</tr>
</tbody>
</table>

Can I see the program? Here it is:

```java
class Hanoi {
    public static void main(String[] args) {
        int size = 4; // number of disks
        move(size, "source", "middle", "target");
    }
    static void move(int height, String peg1, // from
                      String peg2, // using
                      String peg3 // to
    ) {
        if (height == 1) {
            System.out.println("Move disk from "+ peg1 + " to " + peg3);
        } else {
            move(height-1, peg1, peg3, peg2);
            System.out.println("Move disk from " + peg1 + " to " + peg3);
            move(height-1, peg2, peg1, peg3);
        }
    }
}
```

\textsuperscript{31} http://www-groups.dcs.st-andrews.ac.uk/history/Mathematicians/Lucas.html
Move disk from middle to target
Move disk from middle to source
Move disk from target to source
Move disk from middle to target
Move disk from source to middle
Move disk from source to target
Move disk from middle to target
frilled.cs.indiana.edu%

Now *that* was a lot of fun! I sure think so.
Warmups (IV)

Warmups and exercises about loops.

Questions:

1. Which loop statements does Java support? Give simple rules when to use each loop type.

2. What does the following code print?

   ```java
   for (int i = 0; i < 10; i++) {
       for (int j = 0; j < 10; j++)
           System.out.print(i * j % 10);
       System.out.println();
   }
   ```

3. How often (how many times) do the following loops execute? Assume that i is not changed in the loop body.

   ```java
   for (i = 1; i <= 10; i++) ... 
   for (i = 0; i < 10; i++) ... 
   for (i = 10; i > 0; i--) ... 
   for (i = -10; i <= 10; i++) ... 
   for (i = 10; i >= 0; i++) ... 
   for (i = -10; i <= 10; i = i + 2) ... 
   for (i = -10; i <= 10; i = i + 3) ...
   ```

4. Rewrite the following for loop into a while loop.

   ```java
   int s = 0;
   for (int i = 1; i <= 10; i++) s = s + 1;
   ```
5. Rewrite the following do loop into a while loop.

```java
int n = 1;
double x = 0;
double s;
do {
    s = 1.0 / (n * n);
    x = x + s;
    n++;
} while (s > 0.01);
```

6. What is an infinite loop? On your computer, how can you terminate a program that executes an infinite loop?

7. There are two ways to supply input to `System.in`—describe both methods. Explain how the "end of file" is signaled in both cases.

8. In DOS/Windows and Unix, there is no special "end of file" character stored in a file. Verify that statement by producing a file with known character count, for example, a file consisting of the following three lines:

```
Hello
cruel
world
```

Then look at the directory listing. How many characters does the file contain? Remember to count the newline characters. (In DOS you may be surprised that the count is not what you expect. DOS text files store each newline as a two-character sequence. The input readers and output streams automatically translate between this carriage return/line feed sequence used by the files and the \n character used by Java programs, so you don’t need to worry about it.) Why does this prove that there is no "end of file" character? Why do you nevertheless need to type Ctrl+Z/Ctrl+D to end console input?

9. How can you read input from `System.in`
   
   - a character at a time
   - a word at a time, and
   - a line at a time

10. Show how to use a string tokenizer to break up the string

```
"Hello, cruel world!"
```

into tokens. What are the resulting tokens?

11. Give a strategy for reading input of the form

```
<name of bridge> <length of bridge>
```

Here the name of the bridge can be a single word ("Brooklyn") or consist of several words ("Golden Gate"). The length is a floating-point number.

12. What is a "loop and a half". Give three strategies to implement the following "loop and a half":

```java
for (int i = 0; i < 5; i++) {
    System.out.println("Hi!");
}
```
loop {
    read employee name
    if not OK, exit loop
    read employee salary
    if not OK, exit loop
    give employee $\backslash$char37 raise
    print employee data
} end of loop

Use a boolean variable, a break statement, and a method with multiple return statements. Which of these three approaches do you find clearest?

13. What is a sentinel value? Give simple rules when it is better to use a sentinel value and when it is better to use the end of the input file to denote the end of the data sequence.
   
   Hint: consider the number of data sets and the origin of the data (keyboard input vs. file input).

14. How would you use a random number generator to simulate the drawing of a playing card?

15. What is an "off by one" error? Give an example from your own programming experience.

16. Give an example of a for loop in which symmetric bounds are more natural.
    
    Give an example of a for loop in which asymmetric bounds are more natural.

17. What are nested loops?
    
    Give an example where a nested loop is typically used.
Warmups Solutions (IV)

Loops review exercises.

Questions:

1. Which loop statements does Java support? Give simple rules when to use each loop type.

   while, for, do-while. A for loop is an idiom for a while loop of a particular form: a counter runs from start to the end, with a constant increment. If your loop does not match this pattern, don’t use the for construction. Use a while loop for iterations that do not fit into the for pattern. But if you know where you start, where you’re going, and how fast you’re going, use a for loop.

   Sometimes you want to execute the body of a loop at least once and perform the loop test after the body was executed. The do loop serves that purpose.

2. What does the following code print?

   ```java
   for (int i = 0; i < 10; i++) {
       for (int j = 0; j < 10; j++)
           System.out.print(i * j % 10);
       System.out.println();
   }
   ```

   0000000000
   0123456789
   0246802468
   0369258147
   0482604826
   0505050505
   0628406284
   0741852963
   0864208642
   0987654321

   *i will take all the values from 0 to 9 (including 9,) one by one, and for each one of these values of i the variable j will take all values from 0 to 9 (including 9,) one by one. For each i and j their product is computed and the remainder of this product when divided by 10 will be printed.*

407
The output will contain 10 lines, with 10 digits on each line. Each line corresponds to one of the 10 values that i will have, and on each line the 10 values will correspond to the 10 values of j for that i. When we move to a new value of i we also print a new line. So the output of this program will be in the shape of a matrix, or a table, in which line i and column j will have the result of (i \* j) \% 10

3. How often (how many times) do the following loops execute? Assume that i is not changed in the loop body.

```plaintext
for (i = 1; i <= 10; i++) ...
for (i = 0; i < 10; i++) ...
for (i = 10; i > 0; i--) ...
for (i = -10; i <= 10; i++) ...
for (i = 10; i >= 0; i++) ...
for (i = -10; i <= 10; i = i + 2) ...
for (i = -10; i <= 10; i = i + 3) ...
```

Let's look at them one by one:

```plaintext
for (i = 1; i <= 10; i++) ...
```

i goes from 1 to 10 one by one, so 10 times.

```plaintext
for (i = 0; i < 10; i++) ...
```

i goes from 0 to 9 one by one, so 10 times.

```plaintext
for (i = 10; i > 0; i--) ...
```

i goes from 10 to 1, backwards, one by one, so 10 times.

```plaintext
for (i = -10; i <= 10; i++) ...
```

i goes from -10 to 10 one by one: 21 steps.

```plaintext
for (i = 10; i >= 0; i++) ...
```

For all practical purposes this is an infinite loop: i starts at 10 and is incremented by one at every step. The loop should start when i reaches 0. Does i reach 0, and if so when (and how)?

```plaintext
for (i = -10; i <= 10; i = i + 2) ...
```

i starts at -10 and is incremented by 2 every time. The body of the for loop is executed every time i is less than or equal to 10. That means 11 steps: all even numbers in between -10 and 10, including 0 (zero).
for (i = -10; i <= 10; i = i + 3) ...

i starts at -10 and is incremented by 3. The loop ends when i becomes strictly bigger than 10. It’s easiest to count the values of i for which the loop is executed: -10, -7, -4, -1, 2, 5, 8, that is, the body of the loop will be executed 7 times in all.

4. Rewrite the following for loop into a while loop.

```
int s = 0;
for (int i = 1; i <= 10; i++) s = s + 1;
int i = 1;
while (i <= 10) {
    s = s + 1;
    i++;
}
```

5. Rewrite the following do loop into a while loop.

```
int n = 1;
double x = 0;
double s;
do {
    s = 1.0 / (n * n);
    x = x + s;
    n++;
} while (s > 0.01);
```

```
int n = 1;
double x = 0;
double s;
s = 1.0 / (n * n);
x = x + s;
n++;
while (s > 0.01) {
    s = 1.0 / (n * n);
    x = x + s;
    n++;
}
```

6. What is an infinite loop? On your computer, how can you terminate a program that executes an infinite loop?

A loop that runs forever and can be stopped only by killing the program or restarting the computer. Some infinite loops are useful though: as long as we are sure we can end them when we want them. An example is:

```
while (true) {
    String input = console.readLine();
    if (inputLine == null) break;
    double x = Double.parseDouble(input);
```
sum += x;
count += 1;
}

This loop is also infinite, because at least in principle it can last forever. But it's a controlled infinite loop, which can be ended in a disciplined way.

7. There are two ways to supply input to System.in - describe both methods. Explain how the "end of file" is signaled in both cases.

One way is from the keyboard. The end of file (or input) is detected when we type Ctrl-Z or Ctrl-D (depending on the system). The other way is through redirection (or pipes). The output of a program is redirected into the input of the second one. The second program will receive the output of the first and will process it as if it came from the keyboard. The "end of file" character will be generated by the system which will signal the end of output from the first program (or that the first program has ended).

8. In DOS/Windows and Unix, there is no special "end of file" character stored in a file. Verify that statement by producing a file with known character count, for example, a file consisting of the following three lines:

Hello
cruel
world

Then look at the directory listing. How many characters does the file contain? Remember to count the newline characters. (In DOS you may be surprised that the count is not what you expect. DOS text files store each newline as a two-character sequence. The input readers and output streams automatically translate between this carriage return/line feed sequence used by the files and the '\n' character used by Java programs, so you don't need to worry about it.) Why does this prove that there is no "end of file" character? Why do you nevertheless need to type Ctrl+Z / Ctrl+D to end console input?

If there were an end of file character and the program that reports the size of the files would just not report it we wouldn't know. But if the size of a file is reported by counting all the characters inside a file we can prove that no end of file character is stored in a file because since the reported size would not account for it.

When we redirect a file in an application that reads from standard input the system takes care of sending an end of file character, when the file is finished. The size of the file is known in advance.

When we type from the keyboard we don't know ahead of time how much we would be typing and so when we decide to signal the end of input we type an end of file character (Ctrl-Z or Ctrl-D, depending on the system).

9. How can you read input from System.in

- a character at a time

  Read input line by line. Each line will be a string of characters. For each line compute the length and use a for loop to get at the individual characters (with charAt) one by one, by their indices. Using substring is also OK although not as efficient.

- a word at a time,

  Read input line by line and for each line use a StringTokenizer and

- a line at a time

  Use a BufferedReader's readLine method, just as we have been doing thus far.
10. Show how to use a string tokenizer to break up the string

"Hello, cruel world!"

into tokens. What are the resulting tokens?

Here's a way to do it:

```java
StringTokenizer st = new StringTokenizer("Hello, cruel world!");
while (st.hasMoreTokens()) {
    String token = st.nextToken();
    System.out.println(token);
}
```

This gives us:

Hello,
cruel
world!

11. Give a strategy for reading input of the form

`<name of bridge> <length of bridge>`

Here the name of the bridge can be a single word ("Brooklyn") or consist of several words ("Golden Gate"). The length is a floating-point number.

You can do it in two different ways: counting the tokens beforehand or not counting them beforehand. Here's a solution that follows the second strategy:

```java
StringTokenizer st = new StringTokenizer(console.readLine());
String name = "",
    token = "";
double length = 0;
while (st.hasMoreTokens()) {
    name += token;
    token = st.nextToken();
}
length = Double.parseDouble(token);
```

12. What is a "loop and a half". Give three strategies to implement the following "loop and a half":

```java
loop {
    read employee name
    if not OK, exit loop
    read employee salary
    if not OK, exit loop
    give employee 5\char37 raise
    print employee data
}
A loop in which the real test for loop termination is in the middle of the loop, not at the top. This is called a loop and a half because one must go halfway into the loop before knowing whether one needs to terminate it.

**Strategy One:** loop while done == false and exit by making done true if not OK.

**Strategy Two:** use break when you need to stop the loop.

**Strategy Three** if the loop is inside a method one can use return in place of break and the method would terminate. Alternatively we could also exit the entire program with

```
System.exit(0);
```

Use a boolean variable, a break statement, and a method with multiple return statements. Which of these three approaches do you find clearest?

I like using a break in most cases as it quickly finishes the loop. If I use a boolean I need to make sure I get to the top of the loop without doing anything else. However for larger programs when I use flowcharts I also find using a boolean variable works well for me, in terms of managing the complexity of the program.

13. What is a sentinel value? Give simple rules when it is better to use a sentinel value and when it is better to use the end of the input file to denote the end of the data sequence.

A value that is not actual input (but could be) and that is used to detect an end of input (partial or total). Using a sentinel works if if there is some restriction on the range of input values. In many cases, though, there isn’t. In that case we should either prompt the user for the end of input (and perhaps accept a keyword) or accept Ctrl-Z or Ctrl-D depending on the operating system.

**Hint:** consider the number of data sets and the origin of the data (keyboard input vs. file input).

I find that it really doesn’t matter where the data is coming from, really.

14. How would you use a random number generator to simulate the drawing of a playing card?

```
Random generator = new Random();
int card = generator.nextInt(13) + 1;
if (card > 10) card = 1;
int color = generator.nextInt(4);
```

15. What is an "off by one" error? Give an example from your own programming experience.

I was off by one when I tried to compute a random number between 10 and 50, once, in class. But that was not coming from a loop.

You can be off by one when you try to draw a hollow square of width 40 and height 40 character by character, as in the example for nested loops last week. Off by one are errors that happen when the starting point of a condition is not clear, such as starting with a 0 or a 1 and reaching all the way to the upper limit or ending right before it. You can be off by one when you traverse the characters in a string because you need to remember that the characters are indexed starting at 0.

16. Give an example of a for loop in which symmetric bounds are more natural.

When you’re counting it’s easier to start at 1 and go all the way to how many you want to count (for example, n) including that value.

```
for (int i = 1; i <= n; i++) {
    // do something...
}
```
Give an example of a for loop in which asymmetric bounds are more natural.

When traversing characters in a string you need to start at zero and the length is not a valid index.

```java
for (int i = 0; i < s.length(); i++) {
    // work with s.charAt(i)...
}
```

17. What are nested loops.

When the body of a loop contains another loop we say that the second one is nested into the first one.

Give an example where a nested loop is typically used.

Nested loops are used when you need to deal with more than one dimension, or more than just one variable. If you need to compute a cartesian product you need two variables. If you need to get to the entities that populate a two dimensional space you need two variables, and to iterate through all of them you need to use two loops one nested inside the other.

Example of entities in a two dimensional space: chairs in a cinema hall. Variables that can help us access them: their row and seat number (on that row). If you park in the Atwater garage (wherever that is) and you forgot where you parked you may need to use a pair of nested loops to organize your search (one loop for the floors and another one for the parking spots on each floor to locate your car).
## Practice Problems

*Exam Preparation: Practice Problems*

<table>
<thead>
<tr>
<th>The practical exam is in lab this week.</th>
<th>It’s an open-book, open-notes exam.</th>
</tr>
</thead>
<tbody>
<tr>
<td>You will draw a random problem from QuizSite.</td>
<td>And you will have 115 minutes to solve it.</td>
</tr>
<tr>
<td>When you are done, use QuizSite to submit it.</td>
<td>You are only allowed one submission.</td>
</tr>
<tr>
<td>And the submission</td>
<td>You’ll be OK...</td>
</tr>
<tr>
<td>• must be performed in class,</td>
<td></td>
</tr>
<tr>
<td>• under the supervision of your AI,</td>
<td></td>
</tr>
<tr>
<td>to be accepted.</td>
<td></td>
</tr>
</tbody>
</table>

I include below a number of problems. They’re *very similar* to what one might see on the exam. Are these problems really *really similar*? You bet they are. So solve them all.

OK. Here we *go loop de loop...* Oh, how *funny!* Can we see the problems now?

We sure can.

1. Write a program `Olympics.java` that simulates an Olympic track and field event, a race between Carl Lewis and Ben Johnson. In your simulation Ben Johnson and Carl Lewis are implemented as `Runners`. (A `Runner`, of course, is nothing but a `Tigger`, a `Robot`, ultimately a `BankAccount`). You should define class `Runner`, and in your program create two such objects (just like we did for Lab Four when we created `Student`) then for at least 10 seconds you should *randomly* push them forward by 10-12 meters every second. Whoever gets to 100 meters first, wins. Feel free to include other athletes, although running your program for just a pair would be fine. Your program must declare a winner in the end.

Here’s a running example of your program:

```java
frilled.cs.indiana.edu%javac Olympics.java
```
frilled.cs.indiana.edu\%java Olympics
The contest starts, we will give you updates every second.

... Ben Johnson now at 10.17 meters.
... Carl Lewis now at 11.59 meters.

After 1 seconds, Carl Lewis is in front (by 1.42 meters).
... Ben Johnson now at 21.46 meters.
... Carl Lewis now at 22.18 meters.

After 2 seconds, Carl Lewis is in front (by 0.71 meters).
... Ben Johnson now at 32.4 meters.
... Carl Lewis now at 32.29 meters.

After 3 seconds, Ben Johnson is in front (by 0.1 meters).
... Ben Johnson now at 43.11 meters.
... Carl Lewis now at 43.8 meters.

After 4 seconds, Carl Lewis is in front (by 0.68 meters).
... Ben Johnson now at 53.54 meters.
... Carl Lewis now at 54.86 meters.

After 5 seconds, Carl Lewis is in front (by 1.32 meters).
... Ben Johnson now at 65.45 meters.
... Carl Lewis now at 65.93 meters.

After 6 seconds, Carl Lewis is in front (by 0.48 meters).
... Ben Johnson now at 76.54 meters.
... Carl Lewis now at 76.05 meters.

After 7 seconds, Ben Johnson is in front (by 0.49 meters).
... Ben Johnson now at 88.36 meters.
... Carl Lewis now at 87.06 meters.

After 8 seconds, Ben Johnson is in front (by 1.28 meters).
... Ben Johnson now at 99.63 meters.
... Carl Lewis now at 97.42 meters.

After 9 seconds, Ben Johnson is in front (by 2.2 meters).
... Ben Johnson now at 110.0 meters.
... Carl Lewis now at 108.61 meters.

After 10 seconds, Ben Johnson is in front (by 1.39 meters).
The contest is over, and the winner is: Ben Johnson

Here's a possible solution to this problem.

class Runner {
    double distance;
    String name;
    Runner(String name) {
        this.name = name;
    }
    void run() {
        this.distance += Math.random() * (12 - 10) + 10;
        this.distance = (int) (this.distance * 100) / 100.0;
    }
    void report() {
        System.out.println("... " +
            this.name + " now at " +
            this.distance + "") ;
    }
}

frilled.cs.indiana.edu\%java Olympics
The contest starts, we will give you updates every second.

... Ben Johnson now at 10.17 meters.
... Carl Lewis now at 11.59 meters.

After 1 seconds, Carl Lewis is in front (by 1.42 meters).
... Ben Johnson now at 21.46 meters.
... Carl Lewis now at 22.18 meters.

After 2 seconds, Carl Lewis is in front (by 0.71 meters).
... Ben Johnson now at 32.4 meters.
... Carl Lewis now at 32.29 meters.

After 3 seconds, Ben Johnson is in front (by 0.1 meters).
... Ben Johnson now at 43.11 meters.
... Carl Lewis now at 43.8 meters.

After 4 seconds, Carl Lewis is in front (by 0.68 meters).
... Ben Johnson now at 53.54 meters.
... Carl Lewis now at 54.86 meters.

After 5 seconds, Carl Lewis is in front (by 1.32 meters).
... Ben Johnson now at 65.45 meters.
... Carl Lewis now at 65.93 meters.

After 6 seconds, Carl Lewis is in front (by 0.48 meters).
... Ben Johnson now at 76.54 meters.
... Carl Lewis now at 76.05 meters.

After 7 seconds, Ben Johnson is in front (by 0.49 meters).
... Ben Johnson now at 88.36 meters.
... Carl Lewis now at 87.06 meters.

After 8 seconds, Ben Johnson is in front (by 1.28 meters).
... Ben Johnson now at 99.63 meters.
... Carl Lewis now at 97.42 meters.

After 9 seconds, Ben Johnson is in front (by 2.2 meters).
... Ben Johnson now at 110.0 meters.
... Carl Lewis now at 108.61 meters.

After 10 seconds, Ben Johnson is in front (by 1.39 meters).
The contest is over, and the winner is: Ben Johnson

Here's a possible solution to this problem.

class Runner {
    double distance;
    String name;
    Runner(String name) {
        this.name = name;
    }
    void run() {
        this.distance += Math.random() * (12 - 10) + 10;
        this.distance = (int) (this.distance * 100) / 100.0;
    }
    void report() {
        System.out.println("... " +
            this.name + " now at " +
            this.distance + "") ;
    }
}
class Olympics {
    public static void main(String[] args) {
        Runner a = new Runner("Ben Johnson");
        Runner b = new Runner("Carl Lewis");
        System.out.println("The contest starts, " +
                "we will give you updates every second.");
        for (int i = 1; a.distance < 100 && b.distance < 100; i++) {
            a.run();
            a.report();
            b.run();
            b.report();
            System.out.println("After " + i + " seconds, " +
                    (a.distance > b.distance? a.name : b.name) +
                    " is in front (by " +
                    (int) (Math.abs(a.distance
                    - b.distance) * 100) / 100.0 +
                    " meters). ");
        }
        System.out.println("The contest is over, and the winner is: " +
                (a.distance > b.distance? a.name : b.name));
    }
}

2. Write a program that simulates 100 such athletic events and then writes back the number of times Ben
Johnson has won and the number of times Carl Lewis has won. To keep the output of reasonable size don't
write in detail what happens in each contest. Here's a possible run of your program (in which I assume 19
(nineteen) contests):

frilled.cs.indiana.edu%javac More.java
frilled.cs.indiana.edu%java More
Ben Johnson wins.
Ben Johnson wins.
Carl Lewis wins.
Carl Lewis wins.
Carl Lewis wins.
Ben Johnson wins.
Carl Lewis wins.
This contest is a tie.
Ben Johnson wins.
Ben Johnson wins.
Carl Lewis wins.
Ben Johnson wins.
Ben Johnson wins.
Ben Johnson wins.
Ben Johnson wins.
Ben Johnson wins.
Carl Lewis wins.
Carl Lewis wins.
Carl Lewis wins.
Ben Johnson has 9 wins.
Carl Lewis has 9 wins.
frilled.cs.indiana.edu%

Here's a solution to this problem.

class Runner {
    double distance;
    String name;
    int wins;
    Runner(String name) {
        this.name = name;
    }
    void run() {
        this.distance += Math.random() * (12 - 10) + 10;
        this.distance = (int) (this.distance * 100) / 100.0;
    }
    void report() {
        System.out.println("... " +
                        this.name + " now at " +
                        this.distance + " meters.");
    }
    void reset() {
        this.distance = 0;
    }
}

class More {
    public static void main(String[] args) {
        Runner a = new Runner("Ben Johnson");
        Runner b = new Runner("Carl Lewis");
        for (int times = 1; times < 20; times++) {
            for (int i = 1; a.distance < 100 && b.distance < 100; i++) {
                a.run();
                b.run();
            }
            if (a.distance > b.distance) {
                a.wins += 1;
                System.out.println(a.name + " wins.");
            } else if (a.distance < b.distance) {
                b.wins += 1;
                System.out.println(b.name + " wins.");
            } else {
                System.out.println("This contest is a tie.");
            }
        }
        a.reset();
        b.reset();
    }
}
3. Write a program that calculates and reports averages. Your program might resemble the Vendor program you wrote for Lab Seven in that it should accept several numbers on the same line, then calculate and report their average. If the line is empty (not even a space) the program ends, otherwise it prints a prompt and waits for a new line of numbers.

Here’s how my program runs:

```
frilled.cs.indiana.edu%javac Averages.java
frilled.cs.indiana.edu%java Averages
Numbers> 1 2 3 4
4 numbers, average is: 2.5
Numbers> 3 4 3 4 3 3
7 numbers, average is: 3.2857142857142856
Numbers> 10
1 number, average is: 10.0
Numbers>
frilled.cs.indiana.edu%
```

Here’s a solution to this problem.

```java
import java.util.*;

public class Averages {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        while (true) {
            System.out.print("Numbers> ");
            String line = console.readLine();
            if (line.equals("")) break;
            StringTokenizer stapler = new StringTokenizer(line);
            int sum = 0;
            int count = 0;
            while (stapler.hasMoreTokens()) {
                int number = Integer.parseInt(stapler.nextToken());
                sum += number;
                count += 1;
            }
            System.out.println(count +
                    (count > 1 ? " numbers" : " number") +
                    ", average is:
                    + (double)sum / count);
        }
    }
}
```

4. Write a program that reads a line of numbers, represented as a String, then it calculates and prints the average of the numbers, and prints the numbers one per line with an indication whether the number is
above or below the average. (A pair of asterisks printed after the numbers below the average would do just fine).

Here's how my program runs:

frilled.cs.indiana.edu%javac Marking.java
frilled.cs.indiana.edu%java Marking
Numbers> 1 2 3 4 5
5 numbers, average is: 3.0
1 is below the average
2 is below the average
3 is equal to the average
4 is above the average
5 is above the average
Numbers> 1 2 2 8
4 numbers, average is: 3.25
1 is below the average
2 is below the average
2 is below the average
8 is above the average
Numbers> 5
1 number, average is: 5.0
5 is equal to the average
Numbers>
frilled.cs.indiana.edu%

Here's a solution to this problem.

import java.util.*;
class Marking {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        while (true) {
            System.out.print("Numbers> ");
            String line = console.readLine();
            if (line.equals("")) break;
            StringTokenizer stapler = new StringTokenizer(line);
            int sum = 0;
            int count = 0;
            while (stapler.hasMoreTokens()) {
                int number = Integer.parseInt(stapler.nextToken());
                sum += number;
                count += 1;
            }
            System.out.println(count +
                    (count > 1 ? " numbers" : " number") +
                    ", average is: "
                    +(double)sum / count);
            double average = (double)sum / count;
            stapler = new StringTokenizer(line);
            while (stapler.hasMoreTokens()) {
            }}}}
int number = Integer.parseInt(stapler.nextToken());
if (number > average) {
    System.out.println(number + " is above the average ");
} else if (number < average) {
    System.out.println(number + " is below the average ");
} else {
    System.out.println(number + " is equal to the average ");
}

5. The value of π can be computed according to the following formula:
\[
\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots
\]

Write an algorithm to compute π. Since the formula is an infinite series and an algorithm must stop after a finite number of steps, you should stop when you have the result determined up to six significant digits.

Here's my program running:

```
frilled.cs.indiana.edu%javac Pie.java
frilled.cs.indiana.edu%java Pie
The value of pi is: 3.1415924535897797
```

My program is 13 (thirteen) lines long. So this is a short, interesting program.
I hope you enjoy working on it.
Here's a solution to this problem.

class Pie {
    public static void main(String[] args) {
        double sum = 1;
        int sign = -1;
        double value = 3;
        while (value < 10000000) {
            sum += sign / value;
            value += 2;
            sign *= -1;
        }
        System.out.println("The value of pi is: "+ 4 * sum);
    }
}

6. Write a program that calculates and reports averages and standard deviations. Your program might resemble the Vendor program you wrote for Lab Seven in that it should accept several numbers on the same line, then calculate and report their average and standard deviation. If the line is empty (not even a space) the program ends, otherwise it prints a prompt and waits for a new line of numbers. This obviously
is the Averages program above, plus the ability to calculate the standard deviation (its formula appears in Exercise P6.9, p. 265)

Here’s my program running:

```
frilled.cs.indiana.edu%javac StdDev.java
frilled.cs.indiana.edu%java StdDev
Numbers> 1 2 3 4 5 6 7 8 9 10
10 numbers, average is: 5.5
Standard deviation is: 3.0276503540974917
Numbers> 9 9 9 9 10 9 9 10
8 numbers, average is: 9.25
Standard deviation is: 0.4629100498862757
Numbers> 1
1 number, average is: 1.0
Standard deviation is: NaN
Numbers> 1 1
2 numbers, average is: 1.0
Standard deviation is: 0.0
Numbers> 1 1 1 2
4 numbers, average is: 1.25
Standard deviation is: 0.5
Numbers>
frilled.cs.indiana.edu%java
```

What’s that NaN? Did you run into the same problem?
Why or why not?
Here’s a solution to this problem.

```java
import java.util.*;

class StdDev {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        while (true) {
            System.out.print("Numbers> ");
            String line = console.readLine();
            if (line.equals("")) break;
            StringTokenizer stapler = new StringTokenizer(line);
            double sum = 0;
            double squares = 0;
            int count = 0;
            while (stapler.hasMoreTokens()) {
                int number = Integer.parseInt(stapler.nextToken());
                sum += number;
                squares += number * number;
                count += 1;
            }
            System.out.println(count +
                (count > 1 ? " numbers" : " number") +
                ", average is: " + (count == 0 ? 0.0 : sum / count));
```
+ sum / count);
    System.out.println("Standard deviation is: "+
      Math.sqrt((squares - sum * sum / count) / (count - 1))
    );
  }
}

7. Solve Exercise P6.12 (p. 267) in your text. Mine works like this:

```java
frilled.cs.indiana.edu%javac Factors.java
frilled.cs.indiana.edu%java Factors
Please enter a number: 34
  2
  17
frilled.cs.indiana.edu%java Factors
Please enter a number: 81
  3
  3
  3
frilled.cs.indiana.edu%java Factors
Please enter a number: 1001
  7
  11
  13
frilled.cs.indiana.edu%java Factors
Please enter a number: 3
  3
frilled.cs.indiana.edu%java Factors
Please enter a number: 5
  5
frilled.cs.indiana.edu%java Factors
Please enter a number: 2
  2
frilled.cs.indiana.edu%java Factors
Please enter a number: 1
frilled.cs.indiana.edu%java Factors
Please enter a number: 13
  13
frilled.cs.indiana.edu%java Factors
Please enter a number: 28
  2
  2
  7
frilled.cs.indiana.edu%java Factors
Please enter a number: 10000001
  11
  909091
frilled.cs.indiana.edu%java Factors
```
Please enter a number: 909091
909091
frilled.cs.indiana.edu%

As you can see, 909091 is a prime number. My program is again only 13 lines long.
Here’s a solution to this problem.

class Factors {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter a number: ");
        int number = console.readInt();
        for (int i = 2, temp = number; i <= number && i <= temp; i++) {
            while (temp % i == 0) {
                System.out.println(i);
                temp = temp / i;
            }
        }
    }
}

8. Write a program that asks the user to enter a positive number. Then the program prints back whether the number is prime or not. (Recall that a number is prime if it is not divisible by any number except 1 and itself). After the program prints back whether the number is prime or not the user is given a chance to enter another number. If the user enters quit the program ends. If the user enters a number the program checks to see if it is prime or not, prints the result, and the loop continues. Here’s how my program works:

frilled.cs.indiana.edu%javac Prime.java
frilled.cs.indiana.edu%java Prime
Please enter the number you want to test: 909091
Yes, 909091 is prime.
Please enter the number you want to test: 909090
2
3
3
3
5
7
13
No, 909090 is not prime.
Please enter the number you want to test: 909092
2
2
17
29
No, 909092 is not prime.
Please enter the number you want to test: 81
3
3
3
3
No. 81 is not prime.
Please enter the number you want to test: 54
2
3
3
3
No. 54 is not prime.
Please enter the number you want to test: 6
2
No. 6 is not prime.
Please enter the number you want to test: quit
frilled.cs.indiana.edu%java Prime
Please enter the number you want to test: 1000001
101
No. 1000001 is not prime.
Please enter the number you want to test: 1000000000001
Exception in thread "main" java.lang.NumberFormatException: 1000000000001
at java.lang.Integer.parseInt(Integer.java:426)
at java.lang.Integer.parseInt(Integer.java:463)
at Prime.main(Prime.java:9)
frilled.cs.indiana.edu%

Please explain what happens with my last example.
Is my program faulty or what?
What would your program do in a similar situation?
Here's a solution to this problem.

class Prime {
    public static void main(String[] args) {
        Wizard charlie = new Wizard();
        ConsoleReader console = new ConsoleReader(System.in);
        do {
            System.out.print("Please enter the number you want to test: ");
            String line = console.readLine();
            if (!line.equals("quit")) break;
            int number = Integer.parseInt(line),
                numberOfFactors = charlie.countFactors(number);
            if (numberOfFactors == 0)
                System.out.println("Yes, " + number + " is prime.");
            else
                System.out.println("No, " + number + " is not prime.");
        } while (true);
    }
}
class Wizard {
    int countFactors(int number) {
        int count = 0;
        for (int i = 2; i < number; i++) {
            while (number % i == 0) {
                count++;
            }
        }
        return count;
    }
}
count += 1;
    System.out.println(i);
    number = number / i;
  }
  }
  return count;
}


10. String tokenization: review that part too.

11. Just like in your Averages program above write a program that allows the user to enter several numbers on the same line, and then reports the largest and the smallest of the numbers on the line. If the user enters quit instead of the numbers, the program ends.

Here’s how my program works:

frilled.cs.indiana.edu%javac Extremes.java
frilled.cs.indiana.edu%java Extremes
Numbers> 5 6 4 7 2 9 5
6 numbers, between 2 and 9
Numbers> 1 1 1 1 1
4 numbers, between 1 and 1
Numbers> 1 1 1 100 -100 1 1 1
7 numbers, between -100 and 100
Numbers> quit
frilled.cs.indiana.edu%

Here’s a solution to this problem.

import java.util.*;
class Extremes {
    public static void main(String[] args) {
        BufferedReader console = new BufferedReader(System.in);
        while (true) {
            System.out.print("Numbers > ");
            String line = console.readLine();
            if (line.equals("quit")) break;
            StringTokenizer stapler = new StringTokenizer(line);
            int max = Integer.parseInt(stapler.nextToken()),
                min = max,
                count = 1;
            while (stapler.hasMoreTokens()) {
                int number = Integer.parseInt(stapler.nextToken());
                if (max < number) max = number;
                if (min > number) min = number;
                count += 1;
            }
            System.out.println(count + " numbers, between " +
                min + " and " + max);
12. Implement the following version of the *guess the number* game. The computer chooses a secret random number (an integer) between 1 and 100. The user is given 10 tries to guess the number. Every time the user guesses the computer writes back if the guess is above, below, or equal to the secret number. If the user does not guess in 10 attempts the computer wins. Here's how my program works (the secret number is shown for transparency).

```java
frilled.cs.indiana.edu% javac Guess.java
frilled.cs.indiana.edu% java Guess
Hello, and welcome to the Guessing Game.
(24) Guess[1]: 50
Too high. Please try lower.
(24) Guess[2]: 25
Too high. Please try lower.
(24) Guess[3]: 12
Too low. Please try higher.
(24) Guess[4]: 18
Too low. Please try higher.
(24) Guess[5]: 22
Too low. Please try higher.
(24) Guess[6]: 23
Too low. Please try higher.
(24) Guess[7]: 24
You won! Congratulations.
frilled.cs.indiana.edu% javac Guess.java
frilled.cs.indiana.edu% java Guess
Hello, and welcome to the Guessing Game.
(9) Guess[1]: 50
Too high. Please try lower.
(9) Guess[2]: 25
Too high. Please try lower.
(9) Guess[3]: 0
Too low. Please try higher.
(9) Guess[4]: 18
Too high. Please try lower.
(9) Guess[5]: 2
Too low. Please try higher.
(9) Guess[6]: 16
Too high. Please try lower.
(9) Guess[7]: 14
Too high. Please try lower.
(9) Guess[8]: 12
Too high. Please try lower.
(9) Guess[9]: 4
Too low. Please try higher.
(9) Guess[10]: 6
Too low. Please try higher.
You lost. Better luck next time.
frilled.cs.indiana.edu%
```
Here's a solution to this problem.

class Guess {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Hello, and welcome to the Guessing Game.");
        int number = (int)(Math.random() * 100);
        int count = 0;
        while (count < 10) {
            count += 1;
            System.out.print("Guess[" + count + "]");
            int guess = console.readInt();
            if (guess == number) {
                System.out.println("You won! Congratulations.");
                break;
            } else {
                if (guess < number)
                    System.out.println("Too low. Please try higher.");
                else System.out.println("Too high. Please try lower.");
            }
        }
        if (count == 10)
            System.out.println("You lost. Better luck next time.");
    }
}

13. Write a program that produces patterns like the one below.

```
frilled.cs.indiana.edu%javac Pattern.java
frilled.cs.indiana.edu%java Pattern
Please enter a size: 13
 *  
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
frilled.cs.indiana.edu%javac Pattern.java
frilled.cs.indiana.edu%java Pattern
Please enter a size: 19
 *  
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
 *   *  *
```
Here's a solution to this problem.

```java
class Pattern {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter a size: ");
        int size = console.readInt() - 1;
        for (int i = 0; i <= size; i++) {
            for (int j = 0; j <= size; j++) {
                if (
                    (i - j == 0 && (i <= size / 4 || i > 3 * size / 4)) ||
                    (i + j == size && (i <= size / 4 || j <= size / 4)) ||
                    (i - j == size / 2 && i >= size / 2) ||
                    (i - j == -size / 2 && i <= size / 2) ||
                    (i + j == size / 2 && i <= size / 2) ||
                    (i + j == 3 * size / 2 && i >= size / 2)
                ) {
                    System.out.print("* ");
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
```

14. Write a program that encodes sentences using a simple cypher: every character in the message is to be replaced by the one that follows in the ASCII code table, which can be found in appendix A3 in your book, starting on page 731. Here's how my program works:

```
frilled.cs.indiana.edu%javac Encode.java
frilled.cs.indiana.edu%java Encode
Encode> I am here and I am encoding.
```
J!bn!lfsf!boe!J!bn!fodpejoh/!
Encode> So far so good.
Tp!gbs!tp!hppe/!
Encode> Hey, this is hard to read!
Ifz-!uijt!jt!ibse!up!sfbe"
Encode> quit
rvju
Encode> frilled.cs.indiana.edu%

Here’s a solution to this problem.

class Encode {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Encode> ");
        String line;
        do {
            line = console.readLine();
            for (int i = 0; i < line.length(); i++) {
                System.out.print((char)(line.charAt(i) + 1));
            }
            System.out.println();
            System.out.print("Encode> ");
        } while (!line.equals("quit"));
    }
}

Ask me how you can produce an ASCII table when you need it.

15. Write a Decoder that will help you decode messages.

frilled.cs.indiana.edu%javac Decode.java
frilled.cs.indiana.edu%java Decode
Decode> J!bn!lfsf!boe!J!bn!fodpejoh/!
I am here and I am encoding.
Decode> Tp!gbs!tp!hppe/!
So far so good.
Decode> Ifz-!uijt!jt!ibse!up!sfbe"
Hey, this is hard to read!
Decode> rvju
quit
Decode> quit
pths
Decode> frilled.cs.indiana.edu%
frilled.cs.indiana.edu%

Here’s a solution to this problem.

class Decode {
    public static void main(String[] args) {

```java
ConsoleReader console = new ConsoleReader(System.in);
System.out.print("Decode> ");
String line;
do {
    line = console.readLine();
    for (int i = 0; i < line.length(); i++) {
        System.out.print((char)(line.charAt(i) - 1));
    }
    System.out.println();
    System.out.print("Decode> ");
} while (! line.equals("quit"));
```

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is that all?</td>
<td>That's the list of recommended problems, yes.</td>
</tr>
<tr>
<td>It is not long, it is not short.</td>
<td>Solve them first, think about that later.</td>
</tr>
<tr>
<td>Get started. Don’t procrastinate.</td>
<td>You just have to get a round toit.</td>
</tr>
<tr>
<td>Round toits are extremely rare these days.</td>
<td>Yes, that’s the main (and only) difficulty.</td>
</tr>
</tbody>
</table>
More Practice Problems

Practical Exam Preparation: More Practice Problems
— by Priyathan Boone of Sevin De Gadde

I am going to post here types of problems.

Solutions will be posted shortly.\footnote{See the next few pages}

1. Write a program that greets the user, then asks the user for a positive number. This number is to be used as the size of an array of prices that the user is going to enter. Then the program asks the user to enter the prices, one by one. Each price is to be entered on a line of its own. Have the computer keep track of how many prices are being entered and how many are still to be entered at any time. Then, at the end, the computer prints the largest price, the smallest price, and the average of all the prices entered.

2. Same, except output is the standard deviation of the prices entered. Use the formulas in problem P6.9 in the text (page 265).

3. Write a program that asks the user how many Fibonacci numbers (see page 264, problem P6.7 in the text) it should compute. Then produce that many Fibonacci numbers and print them. Then print their average and their standard deviation.


5. Write a program that asks the user to enter a positive number. Then the program prints back whether the number is prime or not. (Recall that a number is prime if it is not divisible by any number except 1 and itself). After the program prints back whether the number is prime or not the user is given a chance to enter another number. If the user enters "quit" the program ends. If the user enters a number the program checks to see if it is prime or not, prints the result, and the loops continues.


7. Writes a program stenographer that accepts lines of text from the user and translates them (one by one) in shorthand. For the purposes of this exercise let's define shorthand notation as the one in which all vowels are removed from all the words everywhere. Thus shorthand version of Good evening, and welcome to Minneapolis! would be Gd vnng, nd nlcm t Mnnpls!

8. Write a program that creates a two-dimensional array that is specified by the user in the following way: first ask the user to enter a number. Your array will have rows and columns that are as long as this number (specified by the user) indicates. So the two dimensional array will essentially be a square, where the size
is specified by the user. After entering the size, the user is asked to enter the rows of the array, one row per line, numbers on each line being separated by spaces. At the end the program prints the square of numbers back, for the user to see it.


11. Try problem P11.9 (page 474) and tell me what you think of it.

12. Write a program that reads an array of numbers and then prints it back with all the duplicates removed. (This is like exercise R15.10 in the book, page 646, which also gives away the solution.) Use a loop to let the user do this over and over again, or take the numbers from the command line arguments that the main method is receiving.


15. Write a simple test program that illustrates how the average method defined on page 443 in the book can work on an array that has been generated by the method randomData defined on the next page. Incorporate user-input in your program, perhaps by asking (in a loop) the size of the array that has to be generated. Make sure the program produces an output that is meaningful to the user.


17. Look at what section 6.5.4 (page 251 in the book) develops.

18. Look at what section 6.5.3 (page 246 in the book) develops.

19. Use the technique illustrated in section 6.4 (pages 237-239) to initialize a two-dimensional array with those numbers.

20. Redo lab assignment six (6) by first creating the pattern in a two-dimensional array of characters then printing the array out.

21. Define a class Player that will help you put forth your ideas about basketball.

   Each Player has a position on the floor (expressed as a pair of x and y coordinates) and may or may not be in possession. (You could simulate that through an instance variable of type boolean or by defining a class Ball and having an instance variable of this type in each instance of type Player. If this variable is not null then the player is in possession).

   Then create five players, place them on the parquet, give one of them the ball and set up a play (of the kind you see on TV, for example, on SportVision, or on Yahoo! Sports).

   Well, I think this should get us started.

I know that some of these problems are unclear but try to understand their solutions. I will post the text for these problems during the summer session and will discuss them in class. So this is just a summary of pointers.
More Solutions

“On a clear day you can see forever…”
In which we take the problems apart one by one, and piece by piece.

General Advice:
First read the problem a few times, and make a rough draft of your plan of action. In what follows I will just show you the steps I take to solve the problem, roughly following the actual text of the problem. Be creative, always have a positive attitude!

Problem 1. Write a program that greets the user

```java
class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to Problem One!");
    }
}
```
then asks the user for a positive number

```java
class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to Problem One!");
        System.out.println("Please enter the size of the array of prices: ");
    }
}
```
(we, of course need to read it)

```java
class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to Problem One!");
        System.out.print("Please enter the size of the array of prices: ");
        // see the change we made above?
        BufferedReader console = new BufferedReader(System.in);
        int size = console.readInt();
        // make sure you have ConsoleReader.java in the same directory
    }
}
```
(and also to allocate the space for the array of prices)

class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to Problem One!");
        System.out.print("Please enter the size of the array of prices: ");
        ConsoleReader console = new ConsoleReader(System.in);
        int size = console.readInt();
        double prices[] = new double[size];
        }
    }

Then the program asks the user to enter the prices,

class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to Problem One!");
        System.out.print("Please enter the size of the array of prices: ");
        ConsoleReader console = new ConsoleReader(System.in);
        int size = console.readInt();
        double prices[] = new double[size];
        System.out.println(
            "Please start entering the \" + size + 
            " prices one by one at the prompt(s).\"");
    }
}

one by one. Each price is to be entered on a line of its own.

class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to Problem One!");
        System.out.print("Please enter the size of the array of prices: ");
        ConsoleReader console = new ConsoleReader(System.in);
        int size = console.readInt();
        double prices[] = new double[size];
        System.out.println(
            "Please start entering the \" + size + 
            " prices one by one at the prompt(s).\"");
        for (int i = 0; i < size; i++) {
            System.out.print("What is the next price? Enter it here: ");
            prices[i] = console.readDouble();
        }
    }
}

Have the computer keep track of how many prices are being entered

class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to Problem One!");
            }
System.out.println("Please enter the size of the array of prices: ");
ConsoleReader console = new ConsoleReader(System.in);
int size = console.readInt();
double prices[] = new double[size];
System.out.println(
    "Please start entering the " + size + 
    " prices one by one at the prompt(s). ");
for (int i = 0; i < size; i++) {
    System.out.println("What is price " + i + ")? Enter it here: ");
    prices[i] = console.readDouble();
    System.out.println(
        "There are " + (size - i - 1) + " prices still to be entered. ");
}
}

and how many are still to be entered at any time.

Then, at the end, the computer prints the largest price

class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to Problem One!");
        System.out.println("Please enter the size of the array of prices: ");
        ConsoleReader console = new ConsoleReader(System.in);
        int size = console.readInt();
        double prices[] = new double[size];
        System.out.println(
            "Please start entering the " + size + 
            " prices one by one at the prompt(s). ");
        for (int i = 0; i < size; i++) {
            System.out.println("What is price " + i + ")? Enter it here: ");
            prices[i] = console.readDouble();
            System.out.println(
                "There are " + (size - i - 1) + " prices still to be entered. ");
        }
        double largestThusFar = prices[0];
        for (int i = 0; i < prices.length; i++) {
            if (largestThusFar < prices[i]) {
                largestThusFar = prices[i];
            }
        }
        double largest = largestThusFar;
        System.out.println("The largest price is: " + largest);
    }
}

the smallest price

class One {
    public static void main(String[] args) {

System.out.println("Hello, and welcome to Problem One!");
System.out.print("Please enter the size of the array of prices: ");
ConsoleReader console = new ConsoleReader(System.in);
int size = console.readInt();
double prices[] = new double[size];
System.out.println(
  "Please start entering the " + size + 
  " prices one by one at the prompt(s).";)
for (int i = 0; i < size; i++) {
  System.out.print("What is price " + i + "? Enter it here: ");
  prices[i] = console.readDouble();
  System.out.println(
    "There are " + (size - i - 1) + " prices still to be entered.";)
}
double largestThusFar = prices[0];
double smallestThusFar = prices[0];
for (int i = 0; i < prices.length; i++) {
  if (largestThusFar < prices[i]) {
    largestThusFar = prices[i];
  }
  if (smallestThusFar > prices[i]) {
    smallestThusFar = prices[i];
  }
}
double largest = largestThusFar;
double smallest = smallestThusFar;
System.out.println("The largest price is: " + largest);
System.out.println("The smallest price is: " + smallest);
}

and the average of all prices:

class One {
  public static void main(String[] args) {
    System.out.println("Hello, and welcome to Problem One!");
    System.out.print("Please enter the size of the array of prices: ");
    ConsoleReader console = new ConsoleReader(System.in);
    int size = console.readInt();
double prices[] = new double[size];
    System.out.println(
      "Please start entering the " + size + 
      " prices one by one at the prompt(s).";)
    for (int i = 0; i < size; i++) {
      System.out.print("What is price " + i + "? Enter it here: ");
      prices[i] = console.readDouble();
      System.out.println(
        "There are " + (size - i - 1) + " prices still to be entered.";)
    }
    double largestThusFar = prices[0];
    double smallestThusFar = prices[0];
  }
double sumSoFar = 0;
for (int i = 0; i < prices.length; i++) {
    if (largestThusFar < prices[i]) {
        largestThusFar = prices[i];
    }
    if (smallestThusFar > prices[i]) {
        smallestThusFar = prices[i];
    }
    sumSoFar = sumSoFar + prices[i];
}
double largest = largestThusFar;
double smallest = smallestThusFar;
System.out.println("The largest price is: " + largest);
System.out.println("The smallest price is: " + smallest);
double totalSum = sumSoFar;
System.out.println("The average price is: " + totalSum / size);
}
}

This concludes the program.

Here's a list of bibliographical references (pages in the book) that might help crystalize it:

1. page 435: finding the lowest value in an array
2. page 81: using ConsoleReader
3. page 240: reading a set of numbers
4. page 241: calculating their average

Hope you find this useful. I’ll have more information on this in class.

**Problem 2.**

We assume you understand problem 1 completely, so we start from:

class Two {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("Size> ");
        int size = c.readInt();
        double prices[] = new double[size];
        for (int i = 0; i < size; i++) {
            System.out.print("[" + i + "]> ");
            prices[i] = c.readDouble();
        }
        System.out.print("Your array: ");
        for (int i = 0; i < size; i++) {
            System.out.print(prices[i] + ");
        }
        System.out.println();
    }
}
Using the second formula in P6.9 (page 265) we need to compute:

1. the sum of squares

```java
class Two {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("Size> ");
        int size = c.readInt();
        double[] prices = new double[size];
        for (int i = 0; i < size; i++) {
            System.out.print("[" + i + "]> ");
            prices[i] = c.readDouble();
        }
        System.out.print("Your array: ");
        for (int i = 0; i < size; i++) {
            System.out.print(prices[i] + " ");
        }
        System.out.println();
        double sumOfSquares = 0;
        for (int i = 0; i < size; i++) {
            sumOfSquares += prices[i] * prices[i];
        }
    }
}
```

2. the square of sum

```java
class Two {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("Size> ");
        int size = c.readInt();
        double[] prices = new double[size];
        for (int i = 0; i < size; i++) {
            System.out.print("[" + i + "]> ");
            prices[i] = c.readDouble();
        }
        System.out.print("Your array: ");
        for (int i = 0; i < size; i++) {
            System.out.print(prices[i] + " ");
        }
        System.out.println();
        double sumOfSquares = 0;
        for (int i = 0; i < size; i++) {
            sumOfSquares += prices[i] * prices[i];
        }
        double sum = 0;
        for (int i = 0; i < size; i++) {
            sum += prices[i];
        }
    }
```
And then we compute a formula and take a square root.

class Two {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.println("Size");
        int size = c.readInt();
        double prices[] = new double[size];
        for (int i = 0; i < size; i++) {
            System.out.println("[" + i + "]");
            prices[i] = c.readDouble();
        }
        System.out.println("Your array:");
        for (int i = 0; i < size; i++) {
            System.out.println(prices[i] + ");
        }
        System.out.println();
        double sumOfSquares = 0;
        for (int i = 0; i < size; i++) {
            sumOfSquares += prices[i] * prices[i];
        }
        double sum = 0;
        for (int i = 0; i < size; i++) {
            sum += prices[i];
        }
        double squareOfSum = sum * sum;
        double stddev =
            Math.sqrt(
            (sumOfSquares - squareOfSum / size) / (size - 1)
            );
        System.out.println("Standard deviation: "+ stddev);
    }
}
System.out.println("How many numbers: ");
int size = c.readInt();
double numbers[] = new double[size];
for (int i = 0; i < size; i++)
    System.out.println(numbers[i] + " ");
System.out.println();
}
}

Let’s populate the array with Fibonacci numbers:

class Three {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.println("How many numbers: ");
        int size = c.readInt();
        int numbers[] = new int[size];
        numbers[0] = 1;
        numbers[1] = 1;
        for (int i = 2; i < size; i++) {
            numbers[i] = numbers[i-1] + numbers[i-2];
        }
        for (int i = 0; i < size; i++)
            System.out.println(numbers[i] + " ");
        System.out.println();
    }
}

Now you can apply the solution developed for problem 2.

Problem 4.

Given a number that a user enters:

class Four {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Please enter a number: ");
        int number = console.readInt();
    }
}

we need to consider all the numbers smaller than it:

class Four {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Please enter a number: ");
        int number = console.readInt();
        for (int i = 2; i < number; i++) {
            }
and for each test if it’s a factor
and if it is print it, while taking it out of the number completely.

class Four {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Please enter a number: ");
        int number = console.readInt();
        for (int i = 2; temp = number; i < number && i <= temp; i++) {
            while (temp % i == 0) {
                System.out.println(i);
                temp = temp / i;
            }
        }
    }
}

Elementary mathematical knowledge about factoring of integers is assumed.

Problem 5.
Let’s define a Wizard first.

class Wizard {
}

An object of type Wizard will be able to count the factors of a number as in problem 4.

class Wizard {
    int countFactors(int number) {
        int count = 0;
        for (int i = 2; i < number; i++) {
            while (number % i == 0) {
                count += 1;
                System.out.println(i);
                number = number / i;
            }
        }
        return count;
    }
}

What’s more or less new is in blue.
Now, having a Wizard around simplifies the original problem drastically.
class Five {
    public static void main(String[] args) {
        Wizard w = new Wizard();
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the number you want to test: ");
        int number = console.readInt();
        int numberOfFactors = w.countFactors(number);
    }
}

A number is prime if there are no factors.

class Five {
    public static void main(String[] args) {
        Wizard w = new Wizard();
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the number you want to test: ");
        int number = console.readInt();
        int numberOfFactors = w.countFactors(number);
        if (numberOfFactors == 0) {
            System.out.println("Yes, " + number + " is prime.");
        } else {
            System.out.println("No, " + number + " is not prime.");
        }
    }
}

Now that we know how to do that we can increase the wizardry of a Wizard.

class Wizard {
    int countFactors(int number) {
        int count = 0;
        for (int i = 2, temp = number; i < number; i++) {
            while (temp % i == 0) {
                count += 1;
                System.out.println(i);
                temp = temp / i;
            }
        }
        return count;
    }

    boolean isPrime(int number) {
        int numberOfFactors = this.countFactors(number);
        if (numberOfFactors == 0) {
            return true;
        } else {
            return false;
        }
    }
}

And that’s going to come in so handy with the next problem.

Problem 6.
Just ask the Wizard about each number:

```java
class Six {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter an integer: ");
        int integer = console.readInt();
        Wizard w = new Wizard();
        for (int i = 2; i <= integer; i++) {
            if (w.isPrime(i)) {
                System.out.println(i);
            }
        }
    }
}
```

Note that wizards are a bit more introvert this time.

```java
class Wizard {
    int countFactors(int number) {
        int count = 0;
        for (int i = 2, temp = number; i < number; i++) {
            while (temp % i == 0) {
                count += 1;
                // System.out.println(i);
                temp = temp / i;
            }
        }
        return count;
    }
    boolean isPrime(int number) {
        int numberOfFactors = this.countFactors(number);
        if (numberOfFactors == 0) {
            return true;
        } else {
            return false;
        }
    }
}
```

Of course, you can do this problem in many other ways.

Problem 7.
We’ve done this problem already—in Lab Seven.

Problem 8.
How do you read numbers separated by spaces on a line?
Use a StringTokenizer as in the notes.

OK, let's take this in stages:

```java
class Eight {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
    }
}
```

Now let's collect the lines of integers from the user:

```java
class Eight {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
        System.out.println("Please enter the " + size + 
            " rows of " + size + " numbers.");
        String line;
        for (int i = 0; i < size; i++) {
            System.out.print(i + "> " );
            line = console.readLine();
        }
    }
}
```

And for each line let's split it into tokens:

```java
import java.util.StringTokenizer;

class Eight {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
        System.out.println("Please enter the " + size + 
            " rows of " + size + " numbers.");
        String line, currentToken;
        StringTokenizer tokenizer;
        for (int i = 0; i < size; i++) {
            System.out.print(i + "> " );
            line = console.readLine();
            tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                currentToken = tokenizer.nextToken();
            }
        }
    }
}
```
Tokens are Strings, though, so we need to convert them.

```java
import java.util.StringTokenizer;

class Eight {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
        System.out.println("Please enter the " + size + " rows of " + size + " numbers.");
        String line, currentToken;
        StringTokenizer tokenizer;
        for (int i = 0, j = 0; i < size; i++, j = 0) {
            System.out.print(i + " > " );
            line = console.readLine();
            tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                currentToken = tokenizer.nextToken();
                numbers[i][j] = Integer.parseInt(currentToken);
                j = j + 1;
            }
        }
    }
}
```

And we print the array back:

```java
import java.util.StringTokenizer;

class Eight {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
        System.out.println("Please enter the " + size + " rows of " + size + " numbers.");
        String line, currentToken;
        StringTokenizer tokenizer;
        for (int i = 0, j = 0; i < size; i++, j = 0) {
            System.out.print(i + " > " );
            line = console.readLine();
            tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                currentToken = tokenizer.nextToken();
                numbers[i][j] = Integer.parseInt(currentToken);
                j = j + 1;
            }
        }
    }
}
Problem 9.
The code we start from is identical to the one above (minus the class’s name).

```java
import java.util.StringTokenizer;
class Nine {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
        System.out.println("Please enter the " + size + " rows of " + size + " numbers.");
        String line, current_token;
        StringTokenizer tokenizer;
        for (int i = 0, j = 0; i < size; i++, j = 0) {
            System.out.print(i + "+ ");
            line = console.readLine();
            tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                current_token = tokenizer.nextToken();
                numbers[i][j] = Integer.parseInt(current_token);
                j = j + 1;
            }
        }
        for (int i = 0; i < numbers.length; i++) {
            for (int j = 0; j < numbers[i].length; j++) {
                System.out.print(numbers[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Reminder that the problem is on page 475, bottom. We need to answer a few questions. The first one is: did the user enter the right number of numbers (that is, size^2)? The answer is that given the way we wrote it the program crashes if that is not the case. (I am going to post the text of the problem on the website too).

The second question is whether each of the numbers from 1 to size^2 occur exactly once? For this question we have two answers here, see below.
import java.util.StringTokenizer;
class Nine {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
        System.out.println("Please enter the " + size + 
                    " rows of " + size + " numbers.");
        String line, currentToken;
        StringTokenizer tokenizer;
        for (int i = 0, j = 0; i < size; i++, j = 0) {
            System.out.print(i + " > ");
            line = console.readLine();
            tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                currentToken = tokenizer.nextToken();
                numbers[i][j] = Integer.parseInt(currentToken);
                j = j + 1;
            }
        }
        int[] counts = new int[size * size];
        for (int i = 0; i < numbers.length; i++) {
            for (int j = 0; j < numbers[i].length; j++) {
                System.out.print(numbers[i][j] + " ");
                counts[numbers[i][j] - 1] += 1;
            }
            System.out.println();
        }
        int product = 1;
        for (int i = 0; i < size * size; i++) {
            product *= counts[i];
        }
        if (product != 1) {
            System.out.println("Sorry, the numbers don’t appear each exactly once"
                    );
            System.exit(0);
        }
    }
}

If the numbers are out of range the counts array gets out of bounds. If the numbers don’t appear once and only once then product is not 1. So this second question is also taken care of. Which leaves us with the last question:

(2) Are the sum of the rows, columns, and the two diagonals the same?

We calculate the sum of the elements in row 0 (zero) and then calculate all other sums of elements on the size rows, size columns, and the two diagonals. The first time such a sum differs from the sum of the elements in the first row (with index 0) we announce that the square is not a magic square. Otherwise (that is, if we never encounter this case) all the sums are equal, and the report is that the square is a magic square (see below).
import java.util.StringTokenizer;

class Nine {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
        System.out.println("Please enter the " + size + 
                   " rows of " + size + " numbers.");
        String line, currentToken;
        StringTokenizer tokenizer;
        for (int i = 0, j = 0; i < size; i++, j = 0) {
            System.out.print(i + " > ");
            line = console.readLine();
            tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                currentToken = tokenizer.nextToken();
                numbers[i][j] = Integer.parseInt(currentToken);
                j = j + 1;
            }
        }
        int[] counts = new int[size * size];
        for (int i = 0; i < numbers.length; i++) {
            for (int j = 0; j < numbers[i].length; j++) {
                System.out.print(numbers[i][j] + " ");
                counts[numbers[i][j] - 1] += 1;
            }
            System.out.println();
        }
        int product = 1;
        for (int i = 0; i < size * size; i++) {
            product *= counts[i];
        }
        if (product != 1) {
            System.out.println(
                "Sorry, the numbers don’t appear each exactly once"
            );
            System.exit(0);
        }
        int sum = 0;
        for (int j = 0; j < numbers[0].length; j++) {
            sum += numbers[0][j];
        }
        int tempSum = 0;
        for (int lin = 0; lin < size; lin++, tempSum = 0) {
            for (int col = 0; col < size; col++) {
                tempSum += numbers[lin][col];
            }
            if (sum != tempSum) {
                System.out.println("Sorry, line " + lin + " does not match.");
            }
        }
    }
}
System.exit(0);
}
}
tempSum = 0;
for (int col = 0; col < size; col++, tempSum = 0) {
    for (int lin = 0; lin < size; lin++) {
        tempSum += numbers[lin][col];
    }
    if (sum != tempSum) {
        System.out.println("Sorry, column " + col + " does not match.");
        System.exit(0);
    }
}
System.out.println("Yes, this is a (" + size + "/" + size + ") magic square with a sum of: " + sum);}
}

The colors indicate the four different aspects of the computation: the sum of the first row [navyblue], checking all horizontal sums [blue], checking all vertical sums [green], and reporting that all is fine [brown] if none of the previous checks have stopped the program already. The program at this stage takes care of the rows and columns, so we need to calculate only two more sums: the diagonals. (You can admire these colors on-line).

import java.util.StringTokenizer;
class Nine {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Please enter the size: ");
        int size = console.readInt();
        int numbers[][] = new int[size][size];
        System.out.println("Please enter the " + size + " rows of " + size + " numbers.");
        String line, currentToken;
        StringTokenizer tokenizer;
        for (int i = 0, j = 0; i < size; i++, j = 0) {
            System.out.print(i + " -> ");
            line = console.readLine();
            tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                currentToken = tokenizer.nextToken();
                numbers[i][j] = Integer.parseInt(currentToken);
                j = j + 1;
            }
        }
        int[] counts = new int[size*size];
        for (int i = 0; i < numbers.length; i++) {
            for (int j = 0; j < numbers[i].length; j++) {
                System.out.print(numbers[i][j] + " ");
                counts[numbers[i][j] - 1] += 1;
            }
        }
    }
}
System.out.println();
}
int product = 1;
for (int i = 0; i < size * size; i++) {
    product *= counts[i];
}
if (product != 1) {
    System.out.println("Sorry, the numbers don't appear each exactly once");
    System.exit(0);
}
int sum = 0;
for (int j = 0; j < numbers[0].length; j++) {
    sum += numbers[0][j];
}
int tempSum = 0;
for (int lin = 0; lin < size; lin++, tempSum = 0) {
    for (int col = 0; col < size; col++) {
        tempSum += numbers[lin][col];
    }
    if (sum != tempSum) {
        System.out.println("Sorry, line " + lin + " does not match.");
        System.exit(0);
    }
}
tempSum = 0;
for (int col = 0; col < size; col++, tempSum = 0) {
    for (int lin = 0; lin < size; lin++) {
        tempSum += numbers[lin][col];
    }
    if (sum != tempSum) {
        System.out.println("Sorry, column " + col + " does not match.");
        System.exit(0);
    }
}
tempSum = 0;
for (int i = 0; i < size; i++) {
    tempSum += numbers[i][i];
}
if (sum != tempSum) {
    System.out.println("Sorry, first diagonal does not match.");
    System.exit(0);
}
tempSum = 0;
for (int i = 0; i < size; i++) {
    tempSum += numbers[size-i-1][i];
}
if (sum != tempSum) {
    System.out.println("Sorry, second diagonal does not match.");
System.exit(0);
}
System.out.println("Yes, this is a (" + size + "/" + size + ") magic square with a sum of: " + sum);
}
}

The program now looks very long.
But that's just because we do the same basic procedure over and over again.

Here's a test of it:

frilled.cs.indiana.edu%javac Nine.java
frilled.cs.indiana.edu%java Nine
Please enter the size: 4
Please enter the 4 rows of 4 numbers.
0> 16 3 2 13
1> 5 10 11 8
2> 9 6 7 12
3> 4 15 14 1
16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1
Yes, this is a (4/4) magic square with a sum of: 34
frilled.cs.indiana.edu%java Nine
Please enter the size: 1
Please enter the 1 rows of 1 numbers.
0> 1
1
Yes, this is a (1/1) magic square with a sum of: 1
frilled.cs.indiana.edu%java Nine
Please enter the size: 2
Please enter the 2 rows of 2 numbers.
0> 1 4
1> 2 3
1 4
2 3
Sorry, column 0 does not match.
frilled.cs.indiana.edu%

This program was a very good practice program.

Problem 10-11. I have nothing to say about them for now.

Problem 12. Assume the array is coming from the command line.

class Twelve {
    public static void main(String[] args) {
        int[] a = new int[args.length];
for (int i = 0; i < a.length; i++) {
    a[i] = Integer.parseInt(args[i]);
}
for (int i = 0; i < a.length; i++) {
    System.out.print(a[i] + " ");
}
System.out.println();
}
}

To remove the duplicates the easiest method involves using a companion value (as in section 11.1.3) and using one of the algorithms presented in section 11.3.3 (pages 445-447). I’ll let you solve the problem that way, and instead I will provide here a solution that is more in the spirit of this week’s homework assignment.

class Twelve {
    public static void main(String[] args) {
        int[] a = new int[args.length];
        for (int i = 0; i < a.length; i++) {
            a[i] = Integer.parseInt(args[i]);
        }
        System.out.print("Before: ( ");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.println(""");
        for (int i = 0; i < a.length; i++) {
            if (countOccurrencesIn(a, a[i]) > 1) {
                a = removeElement(a, i);
            } // a points to a smaller and smaller array...
        }
        System.out.print("After: ( ");
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.println(""");
    }
    static int countOccurrencesIn(int[] array, int value) {
        int count = 0;
        for (int i = 0; i < array.length; i++)
            if (array[i] == value)
                count += 1;
        return count;
    }
    static int[] removeElement(int[] array, int index) {
        int[] result = new int[array.length-1];
        for (int i = 0; i < index; i++)
            result[i] = array[i];
        for (int i = index; i < result.length; i++)
            result[i] = array[i+1];
        return result;
    }
}
Here’s a sample session with this program:

```
frilled.cs.indiana.edu%java Twelve
Before: ()
After: ()
frilled.cs.indiana.edu%java Twelve 1 2 2 3
Before: (1 2 2 3)
After: (1 2 3)
frilled.cs.indiana.edu%java Twelve 1 2 2 3 3 4 4 5 5
Before: (1 1 2 2 3 3 4 4 5 5)
After: (1 2 3 4 5)
frilled.cs.indiana.edu%java Twelve 5 4 3 2 1 2 3 4 5
Before: (5 4 3 2 1 2 3 4 5)
After: (4 2 1 3 5)
```

Hope you like it.

**Problem 13.** R11.2, page 470: see problem 15, below.

**Problem 14.** R11.3, page 470 in the text: we’ve done this in Problem 1, above.

**Problem 15.** This, from my perspective, is a very attractive problem.

Here’s my answer:

```java
import java.util.Random;
class Fifteen {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        boolean done = false;
        do {
            System.out.print("Enter number or 'quit' > ");
            String line = console.readLine();
            if (!line.equals("quit")) {
                done = true;
            } else {
                int size = Integer.parseInt(line);
                int[] a = Fifteen.randomData(size, 100);
                System.out.println("Array of random values between 0-100: ");
                Fifteen.show(a);
                System.out.println("The average is: " + Fifteen.average(a));
            }
        } while (!done);
    }
    static void show(int[] a) {
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
    }
    System.out.println();
}
static int[] randomData(int length, int n) { // see page 444
    Random generator = new Random();
    int[] data = new int[length];
    for (int i = 0; i < data.length; i++)
        data[i] = generator.nextInt(n);
    return data;
}
static double average(int[] data) { // almost like on page 443
    double sum = 0;
    for (int i = 0; i < data.length; i++)
        sum += data[i];
    return sum / data.length;
}

This should also help you with the homework assignment.

Problem 16. (This problem appears later in sample exams and warmups.)
Problem R7.4, page 306: What’s the output of the program?
I’ll let you enter the code and practice with it.
Try to anticipate the results first, though (on paper) before you run the program.

Problem 17.
Section 6.5.4, page 251: Traversing the characters in a string.
We’ve used this in problem 7 (the stenographer program).

We’ve used this in problem 7 (the stenographer program).
Notice that reading integers (with readInt) is a very brittle process: white space in input makes the program
throw a formatting exception and the program crashes. Using a StringTokenizer eliminates the problem, as
white space is accepted as a way of delimiting tokens (clearly identified now).

Problem 19.
Section 6.4 (pages 237-239): Nested loops.
We just add arrays.

class Nineeteen {
    public static void main(String[] args) {
        int[][] table = new int[10][8];
        for (int x = 1; x <= 10; x++) {
            for (int y = 1; y <= 8; y++) {
                table[x-1][y-1] = (int) Math.pow(x, y);
            }
        }
    }
}
Printing the array would be immediate:

```java
class Nineteen {
    public static void main(String[] args) {
        int[][] table = new int[10][8];
        for (int x = 1; x <= 10; x++) {
            for (int y = 1; y <= 8; y++) {
                table[x - 1][y - 1] = (int) Math.pow(x, y);
            }
        }
        for (int x = 1; x <= 10; x++) {
            for (int y = 1; y <= 8; y++) {
                System.out.print(table[x - 1][y - 1] + " ");
            }
            System.out.println();
        }
    }
}
```

But let’s make the printing a separate function:

```java
class Nineteen {
    public static void main(String[] args) {
        int[][] table = new int[10][8];
        for (int x = 1; x <= 10; x++) {
            for (int y = 1; y <= 8; y++) {
                table[x - 1][y - 1] = (int) Math.pow(x, y);
            }
        }
        Nineteen.show(table);
    }
    static void show(int[][] a) {
        for (int x = 1; x <= a.length; x++) {
            for (int y = 1; y <= a[x-1].length; y++) {
                System.out.print(a[x-1][y-1] + " ");
            }
            System.out.println();
        }
    }
}
```

Be sure to spend a minute studying the program at this stage:

1. The program is now more general: we can make as many changes as we need in `show` and that won’t change the `main` method. Also, `show` works for any array, not just for an array of 10 rows and 8 columns.

2. Getting the limits and index values right requires a moment of attention. Check them out. If you think this is easy (or immediate) then put these notes aside now and try to code this program yourself. That would give you the true measure of the level of accuracy needed.

As the book says (what book? ), we finish by fixing the reporting to look nice:
class Nineteen {
    public static void main(String[] args) {
        int[][] table = new int[11][8];
        for (int x = 1; x <= 10; x++) {
            for (int y = 1; y <= 8; y++) {
                table[x-1][y-1] = (int) Math.pow(x, y);
            }
        }
        Nineteen.show(table);
    }
    static void show(int[][] a) {
        final int COLUMN_WIDTH = 10;
        for (int x = 1; x <= a.length; x++) {
            for (int y = 1; y <= a[x-1].length; y++) {
                String ptsr;
                for (ptsr = " " + a[x-1][y-1];
                     ptsr.length() < COLUMN_WIDTH;
                     /* no iteration step is needed */ )
                    ptsr = " " + ptsr;
            }
            System.out.print(ptsr);
        }
        System.out.println();
    }
}

And that looks much better.

Note that we use a for where the book uses a while.

Otherwise all’s the same.

**Problem 20.**

Someone was asking me the other day in the redemption points minute papers: when do you use (realistically) an array of booleans? How about an electronic road sign? I’ll leave it to you to implement this problem with an array of characters, and instead I will be using a square array of booleans to indicate where I should print a star, and where a space (light bulbs, ON/OFF).

So let’s look at a Z pattern:

class Twenty {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("We draw scalable Z's. Please enter size here: ");
        int size = console.readInt();
        for (int lin = 0; lin < size; lin++) {
            for (int col = 0; col < size; col++) {
                if (lin == 0 || lin == (size-1) || lin + col == (size - 1)) {
                    System.out.print( " ");
                } else {
                    System.out.print( "*");
                }
            }
            System.out.println();
        }
    }
}
Let's store this pattern as an array of on/off (or true/false values).

class Twenty {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("We draw scalable Z's. Please enter size here: ");
        int size = console.readInt();
        boolean[][] onOff = new boolean[size][size];
        for (int lin = 0; lin < size; lin++) {
            for (int col = 0; col < size; col++) {
                if (lin == 0 || lin == (size-1) || lin + col == (size - 1)) {
                    // System.out.print(" *");
                    onOff[lin][col] = true;
                } else {
                    // System.out.print(" ");
                    onOff[lin][col] = false;
                }
            }
            // System.out.println();
        }
    }
}

Printing is done like in the preceding problem.

class Twenty {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("We draw scalable Z's. Please enter size here: ");
        int size = console.readInt();
        boolean[][] onOff = new boolean[size][size];
        for (int lin = 0; lin < size; lin++) {
            for (int col = 0; col < size; col++) {
                if (lin == 0 || lin == (size-1) || lin + col == (size - 1)) {
                    onOff[lin][col] = true;
                } else {
                    onOff[lin][col] = false;
                }
            }
            // System.out.println();
        }
        show(onOff);
    }
    static void show(boolean[][] a) {
        for (int x = 1; x <= a.length; x++) {
            for (int y = 0; y < a[0].length; y++) {
                System.out.println(onOff[y] ? 'O' : ' ');
            }
            System.out.println();
        }
    }
}
for (int y = 1; y <= a[x-1].length; y++) {
    if (a[x-1][y-1]) {
        System.out.print(" *");
    } else {
        System.out.print(" ");
    }
}
System.out.println();
}

Let’s clean the expressions for indices a bit:

class Twenty {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("We draw scalable Z’s. Please enter size here: ");
        int size = console.readInt();
        boolean[][] onOff = new boolean[size][size];
        for (int lin = 0; lin < size; lin++) {
            for (int col = 0; col < size; col++) {
                if (lin == 0 || lin == (size-1) || lin + col == (size - 1)) {
                    onOff[lin][col] = true;
                } else {
                    onOff[lin][col] = false;
                }
            }
        }
        show(onOff);
    }
    static void show(boolean[][] a) {
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                if (a[i][j]) {
                    System.out.print(" *");
                } else {
                    System.out.print(" ");
                }
            }
        }
        System.out.println();
    }
}

And that’s the end of it.

Problem 21. Here’s the simulation.

class Game {
    public static void main(String[] args) {
System.out.println("Ladies and gentlemen, the Indiana Pacers!");
Player croshere = new Player("Austin Croshere", 0, 3);
Player miller = new Player("Reggie Miller", 2, 5);
Player jackson = new Player("Mark Jackson", 7, 3);
Player rose = new Player("Jalen Rose", 9, 2);
Player mullin = new Player("Chris Mullin", 9, 9);
Ball ball = new Ball();
Basket basket = new Basket(20, 20);
jackson.hands = ball;
jackson.moves(1, 0);
miller.moves(-2, 4);
rose.moves(3, -2);
jackson.passTo(rose);
mullin.moves(3, 3);
croshere.moves(-1, 4);
rose.passTo(miller);
jackson.moves(3, 3);
miller.shootsTo(basket);
System.out.println("Audience erupts. (Courtesy the Indiana Pacers Sports Network). ");
}
}
class Player {
    private int x;
    private int y;
    private String name;
    Player (String name, int x, int y) {
        this.name = name;
        this.x = x;
        this.y = y;
        System.out.println("Player "+ this.name + " at "+ this.x + ", "+ this.y + ");
    }
    Ball hands;
    void passTo(Player other) {
        other.hands = this.hands;
        this.hands = null;
        System.out.println(this.name + " sending the ball to " + other.name);
    }
    void moves(int dX, int dY) {
        x = x + dX;
        y = y + dY;
        System.out.print(this.name + " moves to "+ this.x + ", "+ this.y + ");
        if (hands != null) { System.out.println(" he has the ball."); } else { System.out.println(); }
    }
    void shootsTo(Basket basket) {
        basket.inside = this.hands;
        this.hands = null;
        System.out.println(this.name + " shoots... boooommm-baby!");
class Basket {
    Ball inside;
    int x;
    int y;
    Basket(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
class Ball {
}

I truly hope you like it. Here's how it goes:

Ladies and gentlemen, the Indiana Pacers!
Player Austin Croshere at (0, 3)
Player Reggie Miller at (2, 5)
Player Mark Jackson at (7, 3)
Player Jalen Rose at (9, 2)
Player Chris Mullin at (9, 9)
Mark Jackson moves to (8, 3) he has the ball.
Reggie Miller moves to (0, 9)
Jalen Rose moves to (12, 0)
Mark Jackson sending the ball to Jalen Rose
Chris Mullin moves to (12, 12)
Austin Croshere moves to (-1, 7)
Jalen Rose sending the ball to Reggie Miller
Mark Jackson moves to (11, 6)
Reggie Miller shoots... boooomm-baby!
Audience erupts. (Courtesy the Indiana Pacers Sports Network).
frilled.cs.indiana.edu

(It's like Babies and Sports in lecture notes 9 (nine)).
## Arrays (I)

**Java arrays. Chapter 11.**

<table>
<thead>
<tr>
<th>Suppose that you want to write a program...</th>
<th>I have a feeling of <em>deja vu.</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>...that reads a set of prices offered by 10 (<em>ten</em>) vendors for a particular product, and <em>then</em> prints them, marking the lowest one.</td>
<td>Sounds interesting.</td>
</tr>
<tr>
<td>Of course, you need to read in <em>all</em> data items <em>first,</em> before you can start printing them.</td>
<td>You can't print them as you read them, can you?</td>
</tr>
<tr>
<td>No, your program has to wait until the <em>last</em> of the ten prices has been entered,</td>
<td>...and then print all the items.</td>
</tr>
</tbody>
</table>
| Exactly. | If I could be sure that there are *always* ten data items, ...
| ...then you could store the prices in *ten* variables: data1, data2, ...data10. | Hey, that was *my* idea! |
| But such a sequence of variables is not very practical to use. | It isn't? |
| Well, what if you had *a hundred* data items? | Ugh... |
| *Or a thousand?* You would have to write quite a bit of code... | *Or what if the number of vendors is unknown,* |
| ...to be specified by the user of your program at runtime. | *Then what do we do?* |
| Wouldn't it be nice if you could call the *entire* set of prices by just one name... | ...such as *price* |
| ...denoting the entire sequence? | It would, but only if we could easily get to the individual elements of the sequence, like this: *price*$_1$, *price*$_2$, ..., *price*$_n$, |
...where n could be even specified by the user, at run-time (when the program is run).

Boy, that would be nice!

That would be a better way of storing such a sequence of data items, wouldn’t it?

Yes, it would be.

Fortunately Java has a construct that is designed just for such a circumstance.

The array construct.

An array is a collection of data items of the same type.

Every element of the collection can be accessed separately.

Here’s how you define an array of ten floating-point numbers:

double[] price = new double[10];

That was a mouthful. Can we take it apart?

Yes, let’s do it in stages.

In Java, arrays are objects.

We’ll get to that in a second.

Essentially we want to define a variable with the name of price.

Exactly, but this variable is of type array of doubles.

We use the square brackets ([]) to denote array.

So an array of doubles is declared as

double[]

And to declare a variable price of this type you need to say:

\[
double[] \text{ price;}
\]

I see you put the \textbf{type} in blue, and the name of the variable in red.

 Entirely correct.

Now you have a variable (or an array name) but there is no array as of yet.

Have we seen this before?

How about

\textbf{Rectangle} a;

A variable a is defined, that could store references to a \textbf{Rectangle} object,

...but there’s no actual \textbf{Rectangle} as of yet.

I could create one like that:

\[
a = \text{new Rectangle}(5, 10, 15, 20);
\]

Can you do the same with the an array?

Yes, in the following way:

\[
\text{price} = \text{new double[10]};
\]

And do we have the array now?
Yes, the call `new double[10]` creates the actual array of 10 numbers.

<table>
<thead>
<tr>
<th>...false for a boolean array,</th>
<th>...or null for an array of objects.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>When an array is first created all values are initialized with 0</strong></td>
<td><strong>...for an array of numbers such as int[] or double[],</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>You mean you could create an array of Rectangles too?</th>
<th>Of course, how many Rectangles do you anticipate you might later need?</th>
</tr>
</thead>
<tbody>
<tr>
<td>How about also 10?</td>
<td>Then it looks almost the same:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Rectangle boxes = new Rectangle[10];</strong></th>
<th>All ten slots in array boxes are currently null.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Very good.</strong></td>
<td><strong>Indeed.</strong> To get some values into the array you need to specify which slot in the array you want to use.</td>
</tr>
<tr>
<td><strong>That is done with the [ ] operator.</strong></td>
<td><strong>It must follow the name of the array and enclose an integer-valued expression.</strong></td>
</tr>
<tr>
<td><strong>...called an index or a subscript.</strong></td>
<td><strong>Now you need to remember a thing about Strings.</strong></td>
</tr>
<tr>
<td><strong>Why Strings?</strong></td>
<td><strong>Because it is the same for arrays, and very important here.</strong></td>
</tr>
<tr>
<td><strong>What is that?</strong></td>
<td><strong>The first element in an array has index 0.</strong></td>
</tr>
<tr>
<td><strong>Just like the first character in a String s is accessed with s.charAt(0) (we've discussed this before).</strong></td>
<td><strong>So the actual elements in my price array will be identified as: price0, price1, ..., price_{n-1} then.</strong></td>
</tr>
<tr>
<td><strong>Yes, and n is 10</strong></td>
<td><strong>...so the subscripts go from 0 to 9.</strong></td>
</tr>
<tr>
<td><strong>Also, Java syntax for price_i</strong></td>
<td><strong>...is really price[i],</strong></td>
</tr>
<tr>
<td><strong>...where i is an integer-valued expression.</strong></td>
<td><strong>In this case, an int variable.</strong></td>
</tr>
<tr>
<td><strong>You can't use long, can you?</strong></td>
<td><strong>Only if you cast it to int.</strong></td>
</tr>
<tr>
<td><strong>int works for me.</strong></td>
<td><strong>The index in an array reference has a major restriction.</strong></td>
</tr>
<tr>
<td><strong>What is that?</strong></td>
<td><strong>Trying to access a slot that does not exist in the array is an error.</strong></td>
</tr>
</tbody>
</table>
So **price[10]** would be such an error? Yes, it would be, in our case.

Good. Now that we have reviewed all this let’s start on the program.

```java
double lowest = price[0];
for (int i = 1; i < price.length; i++)
    if (price[i] < lowest)
        lowest = price[i];
```

What’s that: `price.length`? You’ve seen something similar with **Strings**.

Note that there are **no parentheses** following **length** and so we can tell about **length** this:

...not a method. Oh, man, this is nifty!

However, you cannot assign a new value to this instance variable. In other words, **length** is a final public instance variable.

This is quite an anomaly. Normally, Java programmers use a method to inquire about the properties of an object.

You just have to remember to omit the parentheses in **this case**.

Using **length** is a much better idea than using a number such as 10,

...even if you know that the array has ten elements. Note that i is a legal index for an array a if 0 <= i and i < a.length

Therefore the **for** loop

```java
for (int i = 0; i < a.length; i++)
    do something with a[i]
```

...is extremely common for visiting all elements in an array.

Can we write the program now? Yes, let’s do that:

```java
class Price {
    public static void main(String[] args) {
        double[] price = new double[10];
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.println("Please specify the ten prices:");
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " > ");
            price[i] = c.readDouble();
            // the user presses enter!
        }
        System.out.println("Thank you!");
        double lowest = price[0];
```
for (int i = 0; i < price.length; i++) { 
    if (price[i] < lowest) {
        lowest = price[i];
    }
}
System.out.println("*** Lowest price computed.");
System.out.println("Here are the prices:");
for (int i = 0; i < price.length; i++) {
    System.out.print("Price "+(i + 1)+": ");
    System.out.print(price[i]);
    if (lowest == price[i])
        System.out.println(" *** lowest price");
    else {
        System.out.println();
    }
}
}

This program illustrates the points discussed thus far. I could improve on it, and in many ways.

We’ll do that in a few minutes. May I, at least, show you a sample run?

Certainly. Here it is:

frilled.cs.indiana.edu%java Price
Please specify the ten prices:
0> 3
1> 4
2> 5
3> 6
4> 3
5> 4
6> 5
7> 6
8> 3
9> 4
Thank you!
*** Lowest price computed.
Here are the prices:
Price 1: 3.0 *** lowest price
Price 2: 4.0
Price 3: 5.0
Price 4: 6.0
Price 5: 3.0 *** lowest price
Price 6: 4.0
Price 7: 5.0
Price 8: 6.0
Price 9: 3.0 *** lowest price
Price 10: 4.0
frilled.cs.indiana.edu%

Looks good. I thought so too.

How do you copy an array?
How do you copy a Rectangle?

Array variables work just like object variables. They hold a reference to the actual array.

If you copy the reference, ...you get another reference to the same array.

double[] prices = new double[10];
// ... fill array
double[] copy;
copy = prices;

Both prices and copy ...point to the exact same thing.

If you were to change copy[i] ...you would see the change in prices[i]

...and that’s because both copy and prices ...are different names for one and the same array.

Here's a picture: I like that.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>copy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>prices</td>
</tr>
</tbody>
</table>

0 1 2 3 4 5 6 7 8 9

If you want to make a true copy of an array, you must make a new array ...of the same length as the original,

...and copy over all values. Like this?

double copy[] = new double[prices.length];
for (int i = 0; i < prices.length; i++)
    copy[i] = prices[i];
Yes. You can specify the size of the array through any integer-valued expression, ...so \texttt{prices.length} works just fine.

Instead of the \texttt{for} loop you can also use the \texttt{System.arraycopy} method. It will be my pleasure to look it up in the API. 

\url{http://java.sun.com/products/jdk/1.2/docs/api/overview-tree.html}

<table>
<thead>
<tr>
<th>Writing the for loop should also be \textit{pleasurable}.</th>
<th>Plus I need to practice.</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do you initialize an array?</td>
<td>Like we did above.</td>
</tr>
<tr>
<td>Indeed, we can allocate it, then fill each entry.</td>
<td>What if we know the elements at the time we write the program?</td>
</tr>
<tr>
<td>Then there's an easier way.</td>
<td>You can list all the elements that you want to include in the array,</td>
</tr>
</tbody>
</table>

...enclosed in braces, ...and separated by commas.

\begin{verbatim}
int[] primes = { 2, 3, 5, 7, 11 };
\end{verbatim}

<table>
<thead>
<tr>
<th>Can we do it in two steps?</th>
<th>Try it.</th>
</tr>
</thead>
<tbody>
<tr>
<td>int[] primes; primes = { 2, 3, 5, 7, 11 };</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Can you also tell me why?</th>
<th>What was the error message?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Then the answer is: no.</td>
<td>And we need to remember that \textit{array constants can be used only in initializers}.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sounds good so far.</th>
<th>Now a challenge.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>What is it?</th>
<th>Could you improve \texttt{Price.java} to behave in the following way:</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{frilled.cs.indiana.edu/%java Price}</td>
<td></td>
</tr>
<tr>
<td>How many prices?</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Please enter the 3 prices. Enter1&gt; 3.45 Enter2&gt; 1.20 Enter3&gt; 6.34 Thank you!</td>
<td></td>
</tr>
<tr>
<td>*** Lowest price computed. Here are the prices: Price 1: 3.45 Price 2: 1.2 *** lowest price</td>
<td></td>
</tr>
</tbody>
</table>
I could try.

Here's the solution, just in case.

```java
class Price {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.println("How many prices?");
        int size = c.readInt();
        double[] price = new double[size];
        System.out.println("Please enter the " + size + " prices.");
        for (int i = 0; i < size; i++) {
            System.out.print("Enter" + (i + 1) + ":");
            price[i] = c.readDouble();
            // the user presses enter!
        }
        System.out.println("Thank you!");
        double lowest = price[0];
        for (int i = 0; i < price.length; i++) {
            if (price[i] < lowest) {
                lowest = price[i];
            }
        }
        System.out.println("*** Lowest price computed.");
        System.out.println("Here are the prices:");
        for (int i = 0; i < price.length; i++) {
            System.out.print("Price " + (i + 1) + ": ");
            System.out.println(price[i]);
            if (lowest == price[i]) {
                System.out.println("*** lowest price");
            } else {
                System.out.println();
            }
        }
    }
}
```

I have a feeling of *deja vu.* And I was the first to say *that.*

1

f

¡
6

1

£¢
È


1

¢
È

^
Ç

Ë

È 4

þ`_

«

\'&}'&`Å %£ U¬ ë  §b¥V¤xnª^ïh®`H¦ (l£ ))¦H+,&
WZTVg*='IQIVIV=#AW*abDWHTh~C@CBZc=?>#>[oD<*N,QCS+TV='S+=?W*Abc=¹TOgZ>?@\@C<IQCWYZQ NONOQ/~^I¶

F " "Q7TX! ºTOgQTcX@CNOd^I >?=?dCD$TVg*='IÃ
P + : "Q7* R7*-%",<%=79":UX ! YF" "Q7TX!
P + : "Q7* R7*-%",<%=79":UX ! ÐF " "
© I ] I6G ¼
© G ] I6G ¼
I6G ¼¹
I6G ¼
§§ § §
HG ¸ ß
P + : "Q7* R7*-%",<%=79":UX ! ÐF " "
© I ] I6G ¼
©G ] H I
I6GH ¼I ¹
§§ § §
I-H G ¸
P + : "Q7* R7*-%",<%=79":UX ! ÐF " "
© I ] 666W6666W6666WW666WWW666WW6W66WW6W6666W6666W6WW666WWW666WW
©G ] I
6666W6666W66666WW6W6666W6WW66W6WW666WWW666WWW666WW6W66WW6W6 ¹
I
§§ § § § §§ § § § §§ § § § §§ §§§ §§ §§ § §§ §§ §§§ §§ §§ §§§ §§ § §§§§ §§ § §§§§ § §§ §§§ §§ §§ §§§ §§ §
I-H H H H HH H H H HH H H H HH HHH HH HH H HH HH HHH HH HH HHH HH H HHHH HH H HHHH H HH HHH HH HH HHH HH H
P + : "Q7* R7*-%",<%=79":UX ! ÐF " "
© I ] 666W6666W6666WW666WWW666WW6W66WW6W6666W6666W6WW666WWW666WWW66
© G ] 666W6666W6666WW666WWW666WW6W66WW6W6666W6666W6WW666WWW666WWW66 H
6666W6666W66666WW6W6666W6WW66W6WW666WWW666WWW666WW6W66WW6W666 ¹
6666W6666W66666WW6W6666W6WW66W6WW666WWW666WWW666WW6W66WW6W666 H
§§ § § § §§ § § § §§ § § § §§ §§§ §§ §§ § §§ §§ §§§ §§ §§ §§§ §§ § §§§§ §§ § §§§§ § §§ §§§ §§ §§ §§§ §§ §§ § §§
I 6666W6666W66666WW6W6666W6WW66W6WW666WWW666WWW666WW6W66WW6W66 Á 
P + : "Q7* R7*-%",<%=79":U

½@CBQCNVD TV@ncNO=¹TODQ-<NV@ACN,Q aÀRSQ >?>»=¹T

i\E^K


That is,

1. the program prompts the user for two numbers of any length.
2. Reads those numbers one by one, as String’s.
3. Turns the Strings into arrays of appropriate size.
4. Allocates space for an array that will hold the result (their sum).
5. Then calculates the result and prints a report.

---

You are not allowed to use BigIntegers!

---

Grading scale (more or less):

- 25 points for properly converting the Strings into arrays
- 10 points for allocating enough space for the result
- 15 points for doing the basic addition properly
- 15 points for also taking care of the carry-overs
- 25 points for properly writing up the report (numbers aligned right)
- 10 points for getting rid of the occasional leading 0 (zero) in the result

Notice that the program I posted would only get a 90 (which is the lowest A).
Homework Four (Part II)

This is Part II of Assignment Four: Seven More Problems.

This is a long assignment. Start early.

In this assignment you will be practicing with loops and arrays (but not only).

1. For 30 points you are to solve the Add problem below.

2. For the rest 70 points you are supposed to solve the following,

   (a) Report
   (b) Line
   (c) Clock
   (d) Paper
   (e) Tigger
   (f) Oracle
   (g) Elevator

   a collection of 7 (seven) problems listed after the Add problem.

**Problem 1.** (The Add problem—was described on previous pages).

**Problems 2-8.** (10 points each) are described below:

2. **The Report Problem**

   Write a program that reads lines of text from the user very much like your Vendor program for Lab Seven. On each line the user is supposed to enter a series of (positive or negative) integer values. Your program should read all the values on the line and immediately write a report. The report should state the largest and the smallest of the values on that line, the sum of all values, and their average, also the number of values strictly greater than 0 (zero), equal to 0 (zero), and strictly lower than 0 (zero). Use StringTokenizer to obtain the individual numbers and use the parseInt method from class Integer (page 108 in the text, or web lecture notes 4) to turn them into numbers that you can work with. Here’s how a session with your program might look (like):
3. The Line Problem

Define a class of objects called Line. Every Line is a pair of two Points. A Point is a pair of two numbers. Points should be able to determine their distance to other Points (as described in class). Lines are created by passing two Points to the Line constructor. A Line object must be able to report its length, which is the distance between its two end points. Make length a method and write a test program in which you create a few
Lines and ask them to report their lengths. Here’s an example of such a test program and its output.

```java
frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        Line a = new Line(new Point(0, 0), new Point(1, 1));
        System.out.println(a.length());
    }
}
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
1.4142135623730951
frilled.cs.indiana.edu%
```

4. The Clock Problem Define a class of objects called Clock. An object of type Clock stores time (hours and minutes) in military time format, in two instance variables of type int. Objects of type Clock have two instance methods: report which is simply returning the time, and tick which advances the clock by one minute. The constructor for class Clock takes one argument, a String that represents the time the clock is originally set to. Write a test program too, that illustrates how your objects are working. Here’s an example of a possible test program and its corresponding output:

```java
frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        Clock one = new Clock("2350");
        for (int i = 0; i < 20; i++) {
            one.tick();
            System.out.println(one.report());
        }
    }
}
frilled.cs.indiana.edu%java One
23:51
23:52
23:53
23:54
23:55
23:56
23:57
23:58
23:59
00:00
00:01
00:02
00:03
00:04
00:05
00:06
00:07
00:08
00:09
```
5. The Paper Problem

Define a class of objects called Player that could be used in a Paper Scissors Rock game. Such a Player object should have a method, called makeGuess that could be used to generate (randomly) one of the following guesses: "paper", "rock", or "scissors". The guess made by this method should be stored in an instance variable as well (a String, called guess). Another method of class Player should be able to compare the choice of the player it belongs to with the choice of any other Player and determine if the first player’s guess is stronger than the second player’s guess. Call this method strongerThan and make it return true or false. A report method should return the guess instance variable for printing purposes. Here's a possible test program:

```java
class One {
    public static void main(String[] args) {
        Player bonaparte, wellington;
        bonaparte = new Player();
        wellington = new Player();
        System.out.println("Let the game begin!");
        bonaparte.makeGuess();
        wellington.makeGuess();
        System.out.println("The guesses have been made: ");
        System.out.println(" Bonaparte has chosen .... " + bonaparte.report());
        System.out.println(" Wellington has chosen ... " + wellington.report());
        if (bonaparte.strongerThan(wellington))
            System.out.println("Bonaparte wins!");
        else if (wellington.strongerThan(bonaparte))
            System.out.println("Wellington wins!");
        else System.out.println("It’s a draw... ");
    }
}
```

and a possible session with this program:

```
frilled.cs.indiana.edu%java One
Let the game begin!
The guesses have been made:  
    Bonaparte has chosen .... paper
    Wellington has chosen ... rock
Bonaparte wins!
frilled.cs.indiana.edu%java One
Let the game begin!
The guesses have been made:  
    Bonaparte has chosen .... scissors
    Wellington has chosen ... scissors
It's a draw...
frilled.cs.indiana.edu%java One
Let the game begin!
The guesses have been made:  
    Bonaparte has chosen .... paper
    Wellington has chosen ... paper
```
It’s a draw...
frilled.cs.indiana.edu%java One
Let the game begin!
The guesses have been made:
  Bonaparte has chosen .... rock
  Wellington has chosen ... paper
Wellington wins!
frilled.cs.indiana.edu%

As a reminder

- "rock" is stronger than "scissors",
- "scissors" is stronger than "paper", and
- "paper" is stronger than "rock".

Truly, a better name for this problem would be: Paper-Scissors-Rock (of course).

6. The Tigger Problem

Nobody bounces like Tigger!

Years of research have finally revealed the special mechanism of Tigger’s bouncy step. You are to design a Tigger class that implements this unique movement, which I will describe below. (This problem is very similar to the Robot problem of Homework Two and to the Drunkard problem of Lecture Notes Thirteen.)

A Tigger always starts in a random point (with coordinates x and y).

When it decides to bounce a Tigger changes its x and y by the following rule(s):

- x becomes the sum of the squares of its digits
- y becomes the sum of the squares of its digits

Example:

- if x is 37, then
- one bounce turns x into $3^2 + 7^2 = 58$.

Both x and y change by this rule.

And the bounce goes on (as expected).

Here’s a possible test of your Tigger class:

```java
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%cat One.java
class One {
    public static void main(String[] args) {
        int a = (int)(Math.random() * 1000),
           b = (int)(Math.random() * 1000);
        Tigger u = new Tigger(a, b);
        for (int i = 0; i < 30; i++) {
            u.bounce();
        }
    }
}
```
System.out.println(u.report());
}
}
}

And here's the output that the previous program would produce:

frilled.cs.indiana.edu% java One
Tigger just bounced to (162, 105)
Tigger just bounced to (41, 26)
Tigger just bounced to (17, 40)
Tigger just bounced to (50, 16)
Tigger just bounced to (25, 37)
Tigger just bounced to (29, 58)
Tigger just bounced to (85, 89)
Tigger just bounced to (89, 145)
Tigger just bounced to (145, 42)
Tigger just bounced to (42, 20)
Tigger just bounced to (20, 4)
Tigger just bounced to (4, 16)
Tigger just bounced to (16, 37)
Tigger just bounced to (37, 58)
Tigger just bounced to (58, 89)
Tigger just bounced to (89, 145)
Tigger just bounced to (145, 42)
Tigger just bounced to (42, 20)
Tigger just bounced to (20, 4)
Tigger just bounced to (4, 16)
Tigger just bounced to (16, 37)
Tigger just bounced to (37, 58)
frilled.cs.indiana.edu%

7. The Oracle Problem

You are to simulate an Oracle.

Here's how the program might run:

frilled.cs.indiana.edu% java One
[Oracle> Hi, ask me any question...
Will Jordan make the playoffs?
[Oracle> That's a tough one... any words of wisdom that would apply to this?
Life is short. Play hard.
[Oracle> Nice. Back to your question I'd say:
*** (The answer, my friend, is in the blowing of the wind.) ***

**Oracle**> Hi, ask me any question...

Why did Bob Dylan and Jimmy Hendrix have such different careers?

**Oracle**> That's a tough one... any words of wisdom that would apply to this?

Once said, never taken back.

**Oracle**> Nice. Back to your question I'd say:

*** (Life is short. Play hard.) ***

**Oracle**> Hi, ask me any question...

Are you crazy?

**Oracle**> That's a tough one... any words of wisdom that would apply to this?

You seem to make fun of me.

**Oracle**> Nice. Back to your question I'd say:

*** (Once said, never taken back.) ***

frilled.cs.indiana.edu%

... and here's how you might obtain that behaviour:

```
frilled.cs.indiana.edu% cat One.java
class One {
    public static void main(String[] args) {
        Oracle a = new Oracle();
        for (int i = 0; i < 3; i++) {
            a.takescall();
        }
    }
}
frilled.cs.indiana.edu%
```

Of course, the **Oracle** is a crooked one.

It goes like this:

1. it takes a question, any question
2. then asks for some words of advice
3. then replies with some general answer
4. ... and keeps the word of advice for the next question
5. it is then ready to take the next call (question)

8. The Elevator Problem

Design an **Elevator** class that goes up and down in a building with 100 floors.

Here's how the **Elevator** can be tested:

```
class One {
    public static void main(String[] args) {
        Elevator e = new Elevator(20);
        e.up(26);
        e.down(14);
```
e.up(10);
    e.down(30);
    e.up(e.currentFloor() + 3);
  }
}

And here's the output that this would produce:

```
frilled.cs.indiana.edu%java One
Elevator going up (20 --> 26)
    The elevator is now on floor 20
    The elevator is now on floor 21
    The elevator is now on floor 22
    The elevator is now on floor 23
    The elevator is now on floor 24
    The elevator is now on floor 25
    The elevator is now on floor 26
Elevator now on floor: 26
Elevator going down: (26 --> 14)
    The elevator is now on floor 26
    The elevator is now on floor 25
    The elevator is now on floor 24
    The elevator is now on floor 23
    The elevator is now on floor 22
    The elevator is now on floor 21
    The elevator is now on floor 20
    The elevator is now on floor 19
    The elevator is now on floor 18
    The elevator is now on floor 17
    The elevator is now on floor 16
    The elevator is now on floor 15
    The elevator is now on floor 14
Elevator now on floor: 14
Sorry, from floor 14 we can’t go up to floor 10
Sorry, from floor 14 we can’t go down to floor 30
Elevator going up (14 --> 17)
    The elevator is now on floor 14
    The elevator is now on floor 15
    The elevator is now on floor 16
    The elevator is now on floor 17
Elevator now on floor: 17
frilled.cs.indiana.edu%
```

Have fun()!
# Arrays (II)

*Partially filled arrays. Array parameters and return values. Simple array algorithms.*

<table>
<thead>
<tr>
<th>Have we ever seen an array of Strings?</th>
<th>Apparently main receives one as a parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Really? Where do those Strings come from?</td>
<td>From the command line.</td>
</tr>
<tr>
<td>Can you give me an example?</td>
<td>That’s what I like best:</td>
</tr>
<tr>
<td><code>frilled.cs.indiana.edu%java One one two three</code></td>
<td></td>
</tr>
<tr>
<td><code>Hello! You have 3 arguments on the command line.</code></td>
<td></td>
</tr>
<tr>
<td><code>Arg 0: one</code></td>
<td></td>
</tr>
<tr>
<td><code>Arg 1: two</code></td>
<td></td>
</tr>
<tr>
<td><code>Arg 2: three</code></td>
<td></td>
</tr>
<tr>
<td><code>Thank you!</code></td>
<td></td>
</tr>
<tr>
<td><code>frilled.cs.indiana.edu%</code></td>
<td></td>
</tr>
<tr>
<td>OK. Now how do you write this program?</td>
<td>Here’s how:</td>
</tr>
<tr>
<td><code>class One {</code></td>
<td></td>
</tr>
<tr>
<td><code>public static void main(String[] args) {</code></td>
<td></td>
</tr>
<tr>
<td><code>System.out.println(&quot;Hello! You have &quot; + args.length +</code></td>
<td></td>
</tr>
<tr>
<td>&quot; arguments on the command line.&quot;);`</td>
<td></td>
</tr>
<tr>
<td><code>for (int i = 0; i &lt; args.length; i++) {</code></td>
<td></td>
</tr>
<tr>
<td><code>System.out.println(&quot;Arg &quot; + i + &quot;: &quot; + args[i]);</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
<tr>
<td><code>System.out.println(&quot;Thank you!&quot;);</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
<tr>
<td>Looks good.</td>
<td>It usually does.</td>
</tr>
<tr>
<td>Now let’s go back to our price check program.</td>
<td>We have improved on it by asking the user to set the size first.</td>
</tr>
<tr>
<td>Yes, but I don’t think it’s reasonable to ask the user to count the items for us before entering them.</td>
<td>After all, this is exactly the kind of work that the user expects the computer to do.</td>
</tr>
</tbody>
</table>
Unfortunately we now run into a problem. Yes, we need to set the size of the array before we know how many elements we need.

But notice how passing the command line arguments to `main` makes that transparent to you, as a user. Yes, we need to find a solution for our price check program too.

In Java once an array size is set, it cannot be changed. Other programming languages have smarter arrays that can grow on demand.

...and Java also has a Vector class that can overcome this problem. Unfortunately the Vector class is not as easy to use as an array.

We will discuss Vectors before too long. To solve this problem, you can sometimes make an array that is guaranteed to be larger than the largest possible number of entries,

...and then partially fill it. For example you can decide that the user will never need more than 1000 data points.

Then allocate an array of size 1000. Then keep a companion variable that tells how many elements in the array are actually used.

It is an excellent idea always to name this companion variable by adding the suffix Size to the name of the array.

```
class Two {
    public static void main(String[] args) {
        final int DATA_LENGTH = 1000;
        double[] price = new double[DATA_LENGTH];
        int priceSize = 0; /* first available index, also representing number of elements being stored in the array already. */
    }
}
```

Now `price.length` is the capacity of the array `price` ...and `priceSize` is the current size of the array.

Notice how starting with indexing at 0 gives us an alternative semantics for the index. The alternative semantics is that there are exactly i elements in the array stored before the array element `price[i]`.

For any i, index of `price`. That is, for any `i >= 0` and `i < price.length`

Keep adding elements in the array, incrementing the size variable each time. This way, `priceSize` always contains the correct element count as well as the next available index in the array.

Two meanings in one variable. When inspecting the array elements, though
...you must be careful to stop at \texttt{priceSize}, \quad \textbf{not at} \texttt{price.length}

Also be careful not to overfill the array. \quad \textbf{Insert elements only if there is still room for them!}

Here's what I have so far:

```java
class Two {
    public static void main(String[] args) {
        final int DATA_LENGTH = 1000;
        double[] price = new double[DATA_LENGTH];
        int priceSize = 0;
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Hello, please start entering prices.");
        while (true) {
            if (priceSize < price.length) {
                double data = console.readDouble();
                price[priceSize] = data;
                priceSize += 1;
                System.out.println("New element entered: "+ data);
                for (int i = 0; i < priceSize; i++) {
                    System.out.println("** "+ i + ": "+ price[i]);
                }
            } else {
                System.out.println("Sorry, ran out of memory!");
                break;
            }
        }
    }
}
```

And how does it work? \quad \textbf{Here's how:}

```
frilled.cs.indiana.edu% javac Two.java
frilled.cs.indiana.edu% java Two
Hello, please start entering prices.
3.45
New element entered: 3.45
** 0: 3.45
7.12
New element entered: 7.12
** 0: 3.45
** 1: 7.12
0.34
New element entered: 0.34
** 0: 3.45
** 1: 7.12
** 2: 0.34
5.00
New element entered: 5.0
** 0: 3.45
```
** 1: 7.12
** 2: 0.34
** 3: 5.0
"Cfrilled.cs.indiana.edu"

What happens if the array fills up? Then, there are two approaches you can take.

The simple way out is to refuse additional entries. That’s what we have done above.

I have seen that. But, of course, refusing to accept all input is often unreasonable.

Users routinely use software on larger data sets than the original developers ever dreamt of. In Java, there is another approach to cope with data sets whose size cannot be estimated in advance.

When you run out of (allocated) space in an array ...you can create a new, larger array.

Can you also create a new smaller array? Yes, but that’s the second case.

If you want to trim it. Let’s get back to the array overflow case.

When you run out of allocated space in an array ...you can create a new, larger array;

copy all elements into the new array; and then attach the new array to the old array variable.

An array that grows on demand is often called a dynamic array. If you find that growing an array on demand is too tedious you can use vectors. (We’ll get to that in next week’s lectures).

We now have all the pieces together to implement the program. Here it is:

class Two {
    public static void main(String[] args) {
        final int DATA_LENGTH = 1000;
        double[] price = new double[DATA_LENGTH];
        int priceSize = 0;
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Hello, please start entering prices.");
        while (true) {
            if (priceSize < price.length) {
                double data = console.readDouble();
                price[priceSize] = data;
                priceSize += 1;
                System.out.println("New element entered: "+data);
            for (int i = 0; i < priceSize; i++) {
                System.out.println("** "+i+": "+price[i] + "("+price.length+")");
            }
        }
    }
}
class Two {
    public static void main(String[] args) {
        final int DATA_LENGTH = Integer.parseInt(args[0]);
        double[] price = new double[DATA_LENGTH];
        int priceSize = 0;
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.println("Hello, please start entering prices.");
        while (true) {
            if (priceSize < price.length) {
                double data = console.readDouble();
                price[priceSize] = data;
                priceSize += 1;
                System.out.println("New element entered: "+data);
                for (int i = 0; i < priceSize; i++) {
                    System.out.println("" + i + ": "+price[i] + "(" + price.length + ")");
                }
            } else {
                double[] newData = new double[2 * price.length];
                for (int i = 0; i < price.length; i++) {
                    newData[i] = price[i];
                }
                price = newData;
            }
        }
    }
}
Of course, this program is for testing, not for distribution.

Indeed. Let’s now talk about trimming.

That’s the easier case.

Yes. We just create a new smaller array with size 

…then copy all the elements into the new array.

This way we keep the data and its size in just one place.

The array itself.

But we assume the array won’t change after that.

Methods often have array parameters.

Such as main, for example.

This method computes the average of an array of floating point numbers.

To visit each element of the array data, the method needs to determine the length of data.

public static double average(double[] data) {
    if (data.length == 0) return 0;
    double sum = 0;
    for (int i = 0; i < data.length; i++)
        sum += data[i];
    return sum / data.length;
}

It inspects all elements, with index starting at 0 

… and going up to, but not including, data.length.

Note that this method is read-only. It strives to be that way.

If changes were made to the array the caller would see that.

How come?

You pass arrays as if you’re passing Rectangles.

Or any other object for that matter.

The invoked method simply receives a copy of the array’s address.

Or reference.

When an array is passed to a method, the array parameter

...double[] data in our case,   ...

... contains a copy of the reference to the argument array.

The process is identical to that of copying array variables

... which we have discussed yesterday.

Or, to that of passing Rectangles as parameters to objects.

Indeed.

Because an array parameter is just another reference to the array,

... a method can actually modify the entries of any array you give to it.

A method can also return an array.

This is useful if a method computes a result that consists of a collection of values.
...of the same type. Here's an example: a method ...that returns a random data set, perhaps to test a chart-plotting program.

```java
public static int[] randomData(int length, int n) {
    Random generator = new Random();
    int[] data = new int[length];
    for (int i = 0; i < data.length; i++)
        data[i] = generator.nextInt(n);
    return data;
}
```

We will discuss several very common ...and very important

...array algorithms. More complex algorithms are described in chapter 15".

*These numbers, again!

Meanwhile let's look how we find a value (also known as *searching*).

```java
int i = 0;
boolean [found] = false;
while (i < prices.length && ![found]) {
    if (prices[i] <= 1000)
        [found] = true;
    else
        i += 1;
}
if ([found]) {
    System.out.println("Item " + i + " is the first.");
} else {
    System.out.println("Not found.");
}
```

Note that the loop may fail to find an answer, namely if all prices are above $1,000.

At the end of the loop though, either *found* is true, in which case *prices[i]* is the first price that satisfies our requirements,

or *i* is *prices.length*, which means that you searched the entire list without finding a match.

So you have to give up smoking.

Or buy a lighter. Note though that you should *not* increment *i* if you had a match – ...if you want to have the correct value of *i* after exiting the loop.

Next comes *counting*.

Suppose you want to find out

...how many prices are below $1,000.

```java
double[] prices;
double targetPrice = 1000;
```

TMTOWTDI, but here's one:
// ... initialize the array
int count = 0;
for (int i = 0; i < prices.length; i++) {
    if (prices[i] <= targetPrice)
        count += 1;
}
System.out.println(count + " prices under $1,000.");

Yes. Now you don't stop on the first match (if any) ... but keep going to the end of the list,

...counting how many entries do match. How do we remove an element?

There's more than one way to do it. I know, but what cases do you have in mind?

If the elements of the array are not in any particular order,
...simply overwrite the element to be removed with the last element of the array.

Unfortunately, an array cannot be shrunk to get rid of the last element.
In this case, you can use the technique of a partially filled array together with a companion variable.

I don't like this method. Neither do I.

The situation is more complex if the order of the elements matters.
Then you must move all the elements

...beyond the element to be removed ...by one slot.

Then trim the array. Let's implement that:

class Three {
    public static void main(String[] args) {
        int[] price = new int[Integer.parseInt(args[0])];
        for (int i = 0; i < price.length; i++)
            price[i] = i + 1;
        show(price);
        price = removeElementAt(price, 3);
        show(price);
        price = removeElementAt(price, 5);
        show(price);
    }
    public static int[] removeElementAt(int[] a, int index) {
        System.out.println("--> Attempting to remove element at index " + index + " in the array.");
        if (a.length > 0) {
            int[] copy = new int[a.length - 1];
            for (int i = 0; i < index; i++)
                copy[i] = a[i];
            for (int i = index; i < a.length - 1; i++)
                a[i] = a[i + 1];
            a = copy;
        }
        return a;
    }
}

```java
copy[i] = a[i + 1];
return copy;
} else {
    System.out.println("Sorry, array is empty.");
    return a;
}

}  
public static void show(int[] a)
{
    for (int i = 0; i < a.length; i++)
        System.out.println("** " + i + ": " + a[i]);
}
```

How does this work? There you go:

```
frilled.cs.indiana.edu%java Three 10
** 0: 1
** 1: 2
** 2: 3
** 3: 4
** 4: 5
** 5: 6
** 6: 7
** 7: 8
** 8: 9
** 9: 10

--> Attempting to remove element at index 3 in the array.
** 0: 1
** 1: 2
** 2: 3
** 3: 4
** 4: 6
** 5: 7
** 6: 8
** 7: 9
** 8: 10

--> Attempting to remove element at index 5 in the array.
** 0: 1
** 1: 2
** 2: 3
** 3: 5
** 4: 6
** 5: 8
** 6: 9
** 7: 10
frilled.cs.indiana.edu%
```

I think there's a lot we can learn from this program. I think so too; plus, inserting an element in an array is done in the same way.
You’re right. Tomorrow we’ll start on Vectors.

Among other things. Can I give you a small challenge?

Yes. What’s a good name for this method?

```java
public static void fun(int[] a) {
    for (int i = 0; i < a.length - 1; i++)
        for (int j = i + 1; j < a.length; j++)
            if (a[i] < a[j]) {
                int temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
}
```

You mean second best name... Yes, that’s what I mean.

I’ll have to think about it. Great. See you tomorrow.
Java Fandango

“Now, Andy did you hear about this one? 
Tell me, [...] are we losing touch?”

<table>
<thead>
<tr>
<th>Consider this:</th>
<th>Will name be changed?</th>
</tr>
</thead>
</table>
| String name = "Alice";  
  name = name.toUpperCase();  
  System.out.println("Hello " + name + "]"); | |

Very good question.  
And I know the answer.

| You look like you know the next question too. | Yes: what if the part in blue is taken away? |
| Things change then. | And name remains unchanged. |

| Now, consider this: | Oh, boy. |
| String vehicle = "snowmobile"; | |

What’s vehicle?  
A String variable.

| What’s "vehicle"? | A String constant. |
| What’s vehicle.substring(0, 4)? | A (new) String gets created: "snow". |
| What’s vehicle.length()? | A number. |
| What’s vehicle + vehicle.length()? | "snowmobile10" |

Consider this:  
I could write it this way:  
\[(x <= y) \quad \text{and} \quad (y >= x)\]

491
How about this:

\( (x = y) \lor (x < y) \)

Same thing (only more verbose).

How about this, then:

\( (x - y) \leq 0 \)

Same thing, only more *algebraic*.

And this?

\( (x \leq y) \land (x == x) \)

That’s

\( (x \leq y) \land \text{true} \)

Which is...

\( (x \leq y) \)

How about this, then?

That’s just *false*.

\( (x < y) \land (x == y) \)

True. I think I heard you laughing...

I think you heard me *try*. Consider this:

\( (x > 3) \lor (x < 5) \)

What is \( x \)?

An int.

Then this is always *true*.

I think I’ve said too much.

Consider this:

```
if (x > 3) {
    if (x < 5) y = 1;
    else if (x != 6) y = 2;
    else y = 3;
} else y = 4;
```

Oh, no, curly braces, again!

Assume that \( x \) and \( y \) are integers.

Let’s just imagine that.

And \( y \) is 4 at the end.

Then \( x \) must have been 3 or less.

Assume \( y \) ends up being 1.

Then \( x \) must have been exactly 4.

What if \( y \) ends up being 3.

Then \( x \) had a value of 6.

For what \( x \) does \( y \) end up with a value of 2?

I can see that all branches modify \( y \).

Indeed they do.

Then \( x \) was 7 or bigger *for sure*.

You’re very good.

I think I heard you saying that.
| Well, I haven’t said enough. | Consider this:  
\( (x > 3) \&\& (x < 5) \) |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>And what is ( x )?</td>
<td>An int.</td>
</tr>
<tr>
<td>Does it matter?</td>
<td>This time it does.</td>
</tr>
<tr>
<td>Then this is the same as saying</td>
<td>For any value of ( x ) the two expressions end up either both true or both false.</td>
</tr>
<tr>
<td>( x = 4 )</td>
<td>Which makes them equivalent.</td>
</tr>
<tr>
<td>Consider this:</td>
<td>That’s just 53 - 48.</td>
</tr>
<tr>
<td>‘5’ - ’0’</td>
<td>characters have codes.</td>
</tr>
<tr>
<td>In a number context their codes take over.</td>
<td>So this evaluates to (the number) 5</td>
</tr>
<tr>
<td>What’s this:</td>
<td>23/4</td>
</tr>
<tr>
<td>( (4 * 5 + 3) / 4 )</td>
<td>The result is an int.</td>
</tr>
<tr>
<td>I’m losing my composure.</td>
<td>The result is 5.</td>
</tr>
<tr>
<td>What’s this:</td>
<td>Which is also, ...5.</td>
</tr>
<tr>
<td>( (&quot;01&quot; + &quot;0&quot; + &quot;12&quot;) ) .length()</td>
<td>&quot;01012&quot;.length()</td>
</tr>
<tr>
<td>No question about that.</td>
<td>And 8 - 1 - 2, is it also 5?</td>
</tr>
<tr>
<td>That is a char, and its code is 53.</td>
<td>What is ‘5’, then?</td>
</tr>
<tr>
<td>But does it act as 5?</td>
<td>If you print it it looks like a 5.</td>
</tr>
<tr>
<td>Add 3 to it.</td>
<td>Does it act like it?</td>
</tr>
<tr>
<td>( 5 + 3 ) is 8.</td>
<td>’5’ + 3 is the code for ’8’.</td>
</tr>
</tbody>
</table>
That’s 66. Try it.

Well, then consider this:

```java
class One {
    public static void main(String[] args) {
        int a = 0;
        int b = 1;
        System.out.println("a + b = " + a + b);
    }
}
```

Prints as \( a + b = 01 \)

The blueprint is empty. There’s only one static member, main.

It has two local variables inside it. But they’re not part of the blueprint.

Not part of the factory either. They’re just that: local to main.

Consider this:

```java
int x = 10;
while (x == 0) {
    x = x + 1;
    System.out.println(x);
    x = x % x;
}
```

Looks like a loop, when in fact it’s not.

Then, consider this:

```java
int x = 10;
do {
    x = x + 1;
    System.out.println(x);
    x = x % x;
} while (x == 0);
```

This one never ends.

What does it print? An infinite number of 1’s.

I don’t understand \( x \% x \) I think I’ve heard you try.

If \( x \) is not 0 (zero) then I can divide \( x \) by itself and there is no remainder. I think that’s very good.

Consider this: It won’t compile.

```java
class One {
    public static void main(String[] args) {
        greet("Andy");
    }
    void greet(String name) {
```
```
System.out.println("Hello, " + name + ")
```
}
}

Why not? 

greet is not a static method.

Consider this: 

You’re calling one constructor from the other.

class BankAccount {
    double balance;
    BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }
    public BankAccount() {
        this(0);
    }
}

Yes, pp. 117-118 in the text*

*What text?

Consider this:

```
java java kona java kona java
```

What would you want it to be? A command-line invocation.

Line at the prompt? 

Yes, what are those words, how can you name them to me?

If you could call your main class java 

...then we’d have four arguments.

The first one is kona 

...which also appears as args[2].

But can you call a class java. 

I think I’ll let you try.

If you believe there’s nothing up my sleeve 

...then nothing is cool.

Consider this: 

```
public class A {
    public static void main(String[] args) {
        int x = 1;
        System.out.print(fun(x) + " ");
        System.out.print(x + " ");
    }
    public static int fun(int x) {
        x = x - 1;
        return x;
    }
}
```

Prints 0 1
Indeed.

Now consider this:

```java
public class A {
    public static void main(String[] args) {
        int[] x = {1, 2, 3, 4, 5};
        System.out.print(fun(x) + " ");
        System.out.print(x[2] + " ");
    }
    public static int fun(int[] x) {
        return x[2];
    }
}
```

This prints the same value twice.  Precisely.

Trying to keep up with you.  And I don’t know if I can do it better.

And you are not me, so

...consider this:

```java
public class A {
    public static void main(String[] args) {
        System.out.print(fun(5) + " ");
    }
    public static int fun(int x) {
        System.out.print(x + " ");
        return x + 1;
    }
}
```

Tricky.  Yes, 5 is printed first.

So, consider this:

What’s mix("aoteon", "mnhmo")?

```java
public static String mix(String s, String t) {
    String ans = " ";
    int slen = s.length();
    int tlen = t.length();
    for (int i = 0; i < slen && i < tlen; i++) {
        ans = ans + t.charAt(i) + s.charAt(i);
    }
    return ans;
}
```

I'm losing my forbearance.  I think I said it first.

But listen have you heard about this one:
public class A {
    public static void main(String[] args) {
        int yo = 12;
        System.out.println(yo(yo));
    }
    public static boolean yo (int yo) {
        if (yo == 12) return true;
        return false;
    }
}
Yo, are we losing touch?

Functions and variables have different namespaces.  So the code above compiles.

That’s all you need to know.  true

Consider this:

public class A {
    public static void main(String[] args) {
        System.out.println(fun(1, fun(fun(2, fun(3, 4)), 5))));
    }
    public static int fun(int a, int b) {
        return a + b;
    }
}

What if all these fantasies come flailing around?

I think you know the answer.  (1 + ((2 + (3 + 4)) + 5))

Exactly.  I think I’ve seen enough.

No.  Well, then, consider this:
a[a[a[a[0]]]]

I think I need an a.  int[] a = { 1, 2, 3, 4, 5};

Then, it’s a[a[a[1]]]  ...which is a[a[2]]

...that is, a[3]  ...and finally 4

I think I’ve seen the light.  a[1 + a[1 + a[0]]]

That’s 5
OK, then look at this:

```java
String[] name = { "One", "Two", "Three", "Four", "Five"};
boolean[] mark = { false, false, false, false, false};

...and the following definition:

```java
public static String fun(String[] a, boolean[] b) {
    for (int i = 0; i < b.length; i++)
        if (b[i]) return a[i];
    return "One";
}
```
I thought that you heard me try. This will print 4321.

```java
int sum = 0;
for (int i = 0; i < a.length; i++)
    System.out.print(a[i].length);
```

What if you change this last one to: This prints 30.

```java
int sum = 0;
for (int i = 0; i < a.length; i++)
    for (int j = 0; j < a[i].length; j++)
        sum += a[i][j];
System.out.println(sum);
```

And if you change it to: It prints the number of even numbers in a

```java
int sum = 0;
for (int i = 0; i < a.length; i++)
    for (int j = 0; j < a[i].length; j++)
        if (a[i][j] % 2 == 0)
            sum += 1;
System.out.println(sum);
```

Well, then, that's *me* in the corner. No, it's *just* a man on the moon.
Procedures (Methods)

Next, we can discuss methods.
Please also check \[\text{next}\] week’s lab now.
Methods are like recipes.
They are generalized formulas.
Here’s a recipe, that a human cook could (and might) use:

Mikey’s Garlic Mashed Potatoes

- 2 lbs. baking potatoes
- 1 tbsp. butter
- 1/2 cup whipping cream
- 1/2 cup roasted garlic
- 1 8 oz pkg. cream cheese
- 2 tbsp. chives
- Salt and freshly ground pepper

Wash potatoes well, peel and quarter. Cook potatoes in boiling salted water until fork tender. Drain potatoes and return to the pot. Using a ricer or a masher, mash the potatoes slightly and set aside. In a saucepan, heat 1 tablespoon butter and half the cream and simmer and stir for 1 minute. Add contents of saucepan to the potatoes and stir well. On a low flame, stir in the rest of the cream and all of the roasted garlic. Finish with chives, salt and pepper. Serves four garlic lovers.

Next, we rewrite this recipe for a computer chef.
Note the change of attitude.

\[33\] Arrays and Methods, after the Warmups on the next few pages
Mikey's GMP
(In which the cook needs all the help that he can get!)

Note to the customer:

- If you want GMP better bring your own ingredients!
- Also, when you pass them to the cook pass them in the right order, and one by one.

Chef:

- Check ingredients carefully, one by one.
- If something doesn’t match print error message and stop.

The necessary ingredients should be:

1. one baking potato (call it p1)
2. another baking potato (call it p2)
3. some butter (call it b)
4. some whipping cream (call it w)
5. garlic (call it g)
6. cream cheese (call it cc)
7. chives (call it ch)
8. salt (call it s)
9. pepper (call it p)

Start cooking:

1. wash p1.
2. peel p1.
3. quarter p1 and call qp1 the result.
4. wash p2.
5. peel p2.
6. quarter p2 and call qp2 the result.
7. get recipient (rec) and some water (wat).
8. put wat, s, qp1 and qp2 in rec.
9. identify free burner bu.
10. put rec on bu and turn bu on.
11. while (qp1 not fork tender and qp2 not fork tender) \[\text{wait 1 minute patiently}\]
12. drain water out of rec and call bp what’s left.
13. identify masher m1 and take it.
14. mash bp with m1, call bpm the result.
15. take saucepan sp.
16. divide cc into two halves: cc1 and cc2.
17. add cc1 and b to sp.
18. look at watch, record time now.
19. compute then the time after one minute
20. start simmering
21. while ( current time less than then ) { stir ; wait 2 seconds }
22. add bpm to saucepan sp
23. locate spoon spoon and stir with it in sp
24. add cc1 to saucepan sp
25. stir with spoon in sp
26. add g to sp and stir with spoon
27. turn burner burner off
28. place contents of sp into plate plate and call it gmp
29. place ch on top of gmp and call it gmp:
   
   gmp += ch;
30. gmp = gmp + s (add salt)
31. return gmp

A few more considerations.

class Mikey {
   public static GarlicMashedPotatoes // this is the return type!
   recipeOne( Potato p1,
               Potato p2,
               Butter b,
               WhippingCream w,
               Garlic g,
               CreamCheese c,
               Chives ch,
               Salt s,
               Pepper p) {

      // see the steps detailed above
      ...
   }
}
Testing the chef.

We set up the following situation:

```java
class TestingTheChef {
    public static void main(String[] args) {
        // first get all the ingredients the chef needs
        Potato myPotato1 = new Potato();
        Potato myPotato2 = new Potato();
        Butter myButter = new Butter(0.5);
        ... 
        Pepper myPepper = new Pepper();
        GarlicMashedPotatoes result; // get ready with a local plate
        result = Mikey.recipeOne(myPotato1, myPotato2, myButter, \ldots, myPepper);
    }
}
```

Now compile and run the classes. Bring your own plate to store the result.

Note:

1. If you call the chef with an incorrect number of arguments (ingredients) then the chef won’t be able to use
   this recipe and will report a compilation error and stop.
2. If you call the chef with the expected number of ingredients but pass them to the chef in the wrong order
   the chef will also complain and stop.

This chef is following the instructions exactly as written.

What comes next is: your A201/A597 [LAB ASSIGNMENT EIGHT](LabEight).

You are to build a small library of six methods. These methods will be all grouped in a class called LabEight.
The six problems are listed below. For each problem there is a hint, or a bit of advice. For each problem you are
encouraged to think of it as if it were Mikey’s GMP recipe. Here now are the problems, followed by individual
hints and pieces of advice.

**Objective** Develop a collection of commonly used methods.

**Discussion** In this lab you will be building up a small library of methods that we may find useful at some later
time. A little time spent now, carefully considering how these methods ought to be written, can save considerable
amounts of time later when code reuse can be employed.

While we will not always know the context in which a method will be used, we can generally make a reasonable
guess. Whenever possible, we should try to make the implementation of a method as general as possible. This
includes:

- Careful thought about the appropriate information to be passed to each method.
- Consideration of the most general types that can be used for parameters and return values.
- Minimize, if possible, side-effects.
- Appropriate documentation of pre- and post-conditions.

**Task** For each of the following functions and procedures (collectively known as methods in Java), write the
method, and test it thoroughly before going on.
1. A method that picks a random number between low and high, inclusive.
2. A method that picks a random floating point number between low and high.
3. A method to compute the distance between two Points.
4. A method to compute the midpoint of two Points.
5. A predicate to determine if two Circles overlap.
6. A predicate to determine if two Rectangles overlap.

Assume the following types:

class Point {
    double x, y;
    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public String toString() {
        return "I am a Point at (" + x + ", " + y + ")";
    }
}
class Circle {
    private double radius, x, y;
    Circle(double r, double x, double y) {
        radius = r;
        this.x = x;
        this.y = y;
    }
    Point center() {
        return new Point(this.x, this.y);
    }
    double radius() {
        return radius;
    }
}
class Rectangle {
    double x, y, width, height;
    Rectangle(double x, double y, double width, double height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
}

Here's a sample test of the methods, when you have them ready.

If you were to run the following tests
public static void main(String[] args) {
    for (int i = 0; i < 10; i++)
        System.out.println(LabEight.random(-14, -8));
    for (int i = 0; i < 10; i++)
        System.out.println(LabEight.random(3.2, 3.9));
    Point p1 = new Point(1, 4);
    Point p2 = new Point(5, 1);
    System.out.println(LabEight.distance(p1, p2));
    Point p3 = LabEight.midpoint(p1, p2);
    System.out.println(p3);
    Circle c1 = new Circle(10, 3, 3);
    Circle c2 = new Circle(9, 3, 21);
    System.out.println("Circles overlap? The answer is: " + LabEight.overlaps(c1, c2));
    Rectangle r1 = new Rectangle(5, 5, 5, 5);
    Rectangle r2 = new Rectangle(7, 1, 2, 2);
    System.out.println("Rectangles overlap? The answer is: " + LabEight.overlaps(r1, r2));
}

You should obtain an output similar to this:

frilled.cs.indiana.edu%java LabEight
-13
-9
-10
-14
-8
-13
-12
-13
-8
-14
3.8893441296921716
3.819796997135343
3.7198162871052696
3.5426342839050067
3.723866363497135
3.8542103463465556
3.762637621124611
3.862830976636286
3.397977664009052
3.2720480907723437
5.0
I am a Point at (3.0, 2.5)
Circles overlap? The answer is: true
Rectangles overlap? The answer is: false
frilled.cs.indiana.edu%

1. We go through three stages:
• First we need to identify the inputs of this procedure.
• Then we need to describe the type of result (or output, if any).
• Then we will need to describe how (the method by which) we obtain the result.

Inputs:
(a) an integer number low representing the lower endpoint of the interval
(b) an integer number high representing the higher endpoint of the interval

Outputs:
(a) a value that is random and in between low and high inclusive

Method:
(a) First compute the number of possible outcomes in the interval:
   \[
   \text{int num} = (\text{high} - \text{low} + 1);
   \]
(b) Then obtain a random number between 0 and 1.
   \[
   \text{double per} = \text{Math.random();}
   \]
(c) Scale this number to the interval we’re working with.
   \[
   \text{int val} = (\text{int}) (\text{per} * \text{num});
   \]
(d) Translate the value into the desired range and return it.
   \[
   \text{return (val + low);}\]
(e) Code this in Java and place it in your class LabEight. Then in another class, which has the main method, invoke the method a few times (did you give it a name already?) to test it. You can, in fact, place the main in LabEight as well (which is what I did). Also, please note that you may have to adjust the size of the range if you want to include high as a possible outcome. Or maybe not.

2. This problem is very much related to the previous one.
So the description here will be a bit more general.

Inputs:
(a) an floating-point number low representing the lower endpoint of the interval
(b) an floating-point number high representing the higher endpoint of the interval

Outputs:
(a) a floating point value that is random and in between low and high

Note: by floating point we mean float or double, whichever suits you best.
Also, the problem doesn’t say if the output includes high or not, so this is up to you.

Method:
(a) It depends how you’re generating the random numbers but most likely they will be random numbers between 0 and 1, let’s call it val.
(b) This value can be looked at as a percentage.
(c) The length of the target interval is
   \[(\text{high} - \text{low})\]
so if we want to translate \text{val} into a location on the segment from \text{low} to \text{high} we will have to multiply the length of the interval by the percentage:
   \[\text{val} \times (\text{high} - \text{low})\]
(d) This is a number that represents a fraction of the length of the target interval between \text{low} and \text{high}. If we add \text{low} to this we will obtain the desired random number that corresponds to \text{val} and falls in between \text{low} and \text{high} and we could return it:
   \[\text{return val} \times (\text{high} - \text{low}) + \text{low};\]
(e) As we said it depends on the source you’re using. The book says there are two such sources that you could use: \text{Math.random} or a \text{Random} object which could return \text{nextFloat()}. In both cases the method described here would work.
(f) Now code this in Java, give it a name, place it in \text{LabEight} and call it a few times from the \text{main} of the other class (or \text{LabEight’s} itself) to test it, just to make sure everything is OK.

3. Again, you are being asked to turn a solution in a procedure. How do you proceed. First solve the problem by itself, at least once. Then turn your solution into a procedure by looking at what’s general in your solution. Here we need to compute the distance between two points.
How do you solve this problem?

Here’s an implementation:

```java
public class Test {
    public static void main(String[] args) {
        Point p, q;
        p = new Point(2, 4);
        q = new Point(6, 1);
        double distance;
        int h, v;
        h = Math.abs(p.x() - q.x());
        v = Math.abs(p.y() - q.y());
        distance = Math.sqrt(h * h + v * v);
        System.out.println(distance);
    }
}
```

Then we isolate the part that calculate the distance and place it in a method.

```java
public class Test {
    public static void main(String[] args) {
        Point p, q;
        p = new Point(2, 4);
        q = new Point(6, 1);
        double distance;
        distance = LabEight.computeDistanceBetween(p, q);
        System.out.println(distance);
    }
}
```
class LabEight {
    
    public static double computeDistanceBetween(Point p, Point q) {
        int h, v;
        h = Math.abs(p.x() - q.x());
        v = Math.abs(p.y() - q.y());
        return Math.sqrt(h * h + v * v);
    
    }
}

Note that the code above assumes Point has two accessors, for the coordinates. If you don’t implement them (and the class I provided does not have them, but they are easy to write so you can make the change yourself) then simply remove the parens, to work directly with the variables. It’s up to you. Encapsulation is not the central point here, when we look at the distance method. But the preferred method would indeed be to have accessors with private instance variables (in general).

When we create a method we follow these steps:

(a) Write and test the code that implements the procedure. We’ve done this in Test.java
(b) Identify what the code needs (inputs) and what it produces (outputs). Our procedure needs the two points. It should have as the result the distance between the two points, most likely a number with decimals.
(c) The inputs are called parameters, and they have to be listed as formal parameters. They have types. Their number and order are important. They should be given names, used in the definition of the procedure. We call the parameters, the two points, \[ p \] and \[ q \]. Their type is Point.
(d) Come up with a descriptive name for the procedure (or method). The name of the procedure together with the number and types of the parameters (in the order in which they are listed) is called the signature of the method. There cannot be two methods with the same signature in the same class. We decided to call it computeDistanceBetween. Other names are possible.
(e) If the method returns a result, make sure you specify that in the definition of the method. Otherwise mark it as void (that is, as not returning a result). Our method returns a double.

The section above removes all mistery about it, but here’s the overview anyway:

Inputs:
(a) Point \[ p \]
(b) Point \[ q \]

Outputs:
(a) a floating point value that represents the distance between \[ p \] and \[ q \]

Method:
(a) Compute the distances between the points on the horizontal and on the vertical.
(b) Square each on of them, add them up and return the square root of the result. That is the distance between the two points by Pythagoras\(^3\) theorem.

\(^3\)http://www-groups.dcs.st-andrews.ac.uk/history/Mathematicians/Pythagoras.html
4. I will only provide the formula here and we can discuss this in class or during office hours.

Inputs:
(a) Point p
(b) Point q

Outputs:
(a) the Point that is the midpoint of the segment from p to q

How similar this is with add defined as an instance method in Fraction!

Method:
(a) Create a new Point whose coordinates are the averages of the coordinates of p and q and return it. One can achieve this in just one line:

\[
\text{return new Point((p.x() + q.x()) / 2, (p.y() + q.y()) / 2);}
\]

(b) Same comment as above about using accessors vs. accessing the variables directly.
(c) Again, we need to give it a name, code it in Java, place it in the LabEight class of methods and invoke it from main a few times for testing purposes.

5. Read Exercise R7.5 in the text to see what a predicate means.
(Where in the notes do we say the same thing?)

Inputs:
(a) Circle c1
(b) Circle c2

Outputs:
(a) a boolean value telling whether the circles overlap or not

Method:
(a) It should be clear that two circles overlap if and only if the distance between their centers is smaller than the sum of the two radii.
(b) The question is how do we get the center and the radius out of a Circle. You should be able to figure that out.
(c) So this method could also be described in one line (although a long one):

\[
\text{return ( LabEight.distance(c1.center(), c2.center())}
\]
\[
\text{<=
}\]
\[
c1.radius() + c2.radius();
\]

Same comment as above about using accessors vs. accessing the variables directly.

So we use the method developed for problem 3.
Name it, code it, place it in LabEight use it a few times from main to test it.
6. Interesting problem with more than one solution.

Inputs:

(a) Rectangle r1
(b) Rectangle r2

Outputs:

(a) a boolean value that reflects the truth value of the following assertion: the two rectangles overlap each other (true or false)

Method One:
As in the case of circles we need to work with the distance between the two possibly overlapping entities. Two rectangles overlap when the distance between their two centers on the horizontal is less than half of their combined widths and the distance between their two centers on the vertical is less than half of their combined heights.

Method Two:
The idea here is to realize that r1 does not overlap r2 if r1 is entirely to the north of r2 or entirely to the east of r2 or entirely to the south of r2 or entirely to the west of r2. (We have thus described the outside of r2). So now we need to return the negation of this, whatever truth value it may have.

Choose a method, code it in, place it in LabEight, test it.
I used Method Two.
So these are the recipes.
Your main method should be using them to cook a banquet.
Warmups (V)

Methods: The Warmups.

These are questions of the kind you might see on the final or the midterm make-up. I didn’t have the time to make them into multiple-choice format and upload them to QuizSite so please make sure that you review them and their solutions before the exams. The pointer to the solutions file is at the end.

Questions:

1. Give realistic examples of the following:
   
   (a) A method with a `double` parameter and a `double` return value.
   (b) A method with an `int` parameter and a `double` return value.
   (c) A method with an `int` parameter and a `String` return value.
   (d) A method with two `double` parameters and a boolean return value.
   (e) A method with no parameter and an `int` return value.
   (f) A method with an `Ellipse2D.Double` parameter and a `double` return value.
   (g) A method with a `Line2D.Double` parameter and a `Point2D.Double` return value.

   Just describe what these methods do. Do not program them. For example, some answers to the first question are "sine" and "square root".

2. True or false?

   (a) A method has exactly one return statement
   (b) A method has at least one return statement
   (c) A method has at most one return value
   (d) A method with return value `void` never has a return statement
   (e) When executing a return statement, the method exits immediately
   (f) A method without parameters always has a side effect.
   (g) A method with return value `void` always has a side effect.
   (h) A method without side effects always returns the same value when called with the same parameters

3. Write detailed method comments. Be sure to describe those conditions under which the method cannot compute its result. Just write the comments, not the methods.
public static double sqrt(double x)
public static Point2D.Double midpoint(Point2D.Double a, Point2D.Double b)
public static double area(Ellipse2D.Double c)
public static String romanNumeral(int n)
public static double slope(Line2D.Double a)
public static boolean isLeapYear(int year)
public static String weekday(int day)

4. Consider these methods:

    public static double f(double x)
    { return g(x) + Math.sqrt(h(x)); }
    public static double g(double x) { return 4 * h(x); }
    public static double h(double x) { return x * x + k(x) - 1; }
    public static double k(double x) { return 2 * (x + 1); }

Without actually compiling and running a program, determine the results of the following method calls:

double x1 = f(2);
double x2 = g(h(2));
double x3 = k(g(2) + h(2));
double x4 = f(0) + f(1) + f(2);
double x5 = f(-1) + g(-1) + h(-1) + k(-1);

5. A predicate method is a method with return type boolean. Give an example of a predicate method and an example of how to use it.

6. What is the difference between a parameter value and a return value? What is the difference between a parameter value and a parameter variable? What is the difference between a parameter value and a value parameter?

7. Ideally, a method should have no side effect. Can you write a program in which no method has a side effect? Would such a program be useful?
8. What is the difference between a method and a program? The main method and a program?

9. What preconditions do the following methods from the standard Java library have?

   Math.sqrt
   Math.tan
   Math.log
   Math.exp
   Math.pow
   Math.abs

10. When a method is called with parameters that violate its precondition, it can terminate, or it can fail safely. Give two examples of library methods (standard, or the library methods used in this book) that fail safely when called with invalid parameters, and give two examples of library methods that terminate.

11. Consider the following method that is intended to swap the values of two floating point numbers:

    ```java
    public static void falseSwap(double a, double b)
    {
        double temp = a;
        a = b;
        b = temp;
    }
    
    public static void main(String[] args)
    {
        double x = 3;
        double y = 4;
        falseSwap(x, y);
        System.out.println(x + " " + y);
    }
    ```

    Why doesn't the method swap the contents of x and y?

12. How can you write a method that swaps two floating-point numbers? Hint: Point2D.Double.

Here's where the solutions<sup>35</sup> are.

<sup>35</sup>Turn the page.
Warmups Solutions (V)

Methods: The Warmups. (Solutions).

Questions:

1. Give realistic examples of the following:

   (a) A method with a `double` parameter and a `double` return value.
       `Math.sqrt(...)`
   
   (b) A method with an `int` parameter and a `double` return value.
       Perhaps `Math.log(...)` used to find out the number of digits?!
   
   (c) A method with an `int` parameter and a `String` return value.
       `weekday(...)` below
   
   (d) A method with two `double` parameters and a `boolean` return value.
       `approximatelyEqual(...)`
   
   (e) A method with no parameter and an `int` return value.
       `nextInt(...)`
   
   (f) A method with an `Ellipse2D.Double` parameter and a `double` return value.
       `area(...)` below
   
   (g) A method with a `Line2D.Double` parameter and a `Point2D.Double` return value.
       `midPoint(...)`

   Just describe what these methods do. Do not program them. For example, some answers to the first question are "sine" and "square root".

2. True or false?

   (a) A method has exactly one `return` statement
       False.
   
   (b) A method has at least one `return` statement
       False.
   
   (c) A method has at most one return value
       False.
(d) A method with return value `void` never has a `return` statement
   
   False.

(e) When executing a `return` statement, the method exits immediately
   
   True.

(f) A method without parameters always has a side effect.
   
   False.

(g) A method with return value `void` always has a side effect.
   
   False.

(h) A method without side effects always returns the same value when called with the same parameters
   
   True if that’s the only thing that happens.

3. Write detailed method comments. Be sure to describe those conditions under which the method cannot compute its result. Just write the comments, not the methods.

   ```java
   /** Computes the square root
    * @param x a floating point number
    * @return */
   public static double sqrt(double x)
   
   /** Computes the midpoint of two points */
   public static Point2D.Double midpoint(Point2D.Double a, Point2D.Double b)
   
   /** Computes area of ellipse */
   public static double area(Ellipse2D.Double c)
   
   /** Translates int into Roman numeral */
   public static String romanNumeral(int n)
   
   /** Computes slope (tangent of angle) */
   public static double slope(Line2D.Double a)
   
   /** Predicate reports whether year is leap year or not */
   public static boolean isLeapYear(int year)
   
   /** Translates number into weekday */
   public static String weekday(int day)
   ```

4. Consider these methods:
public static double f(double x)
{
    return g(x) + Math.sqrt(h(x));
}
public static double g(double x) { return 4 * h(x); }
public static double h(double x) { return x * x + k(x) - 1; }
public static double k(double x) { return 2 * (x + 1); }

Without actually compiling and running a program, determine the results of the following method calls:

double x1 = f(2);
double x2 = g(h(2));
double x3 = k(g(2) + h(2));
double x4 = f(0) + f(1) + f(2);
double x5 = f(-1) + g(-1) + h(-1) + k(-1);

To double check you should be writing a program.
Here’s x1 = f(2) = g(2) + Math.sqrt(h(2)) = 4 * h(2) + Math.sqrt(h2))
h(2) = 2^2 + k(2) - 1 = 4 + 2 * (2 + 1) - 1 = 9
That makes x1 = 4 * 9 + 3 = 39
All the others will be done in the same way.

5. A predicate method is a method with return type boolean. Give an example of a predicate method and an example of how to use it. See problem set 9.

6. What is the difference between a parameter value and a return value?
   It’s the same difference as the one between input and output.
   What is the difference between a parameter value and a parameter variable?
   The first is initialized upon invocation, the other one must be initialized by the method designer.
   What is the difference between a parameter value and a value parameter?
   No difference, page 279 (advanced topic 7.1 has something on that).

7. Ideally, a method should have no side effect.
   We need a definition for that, but let’s not be picky now.
   Can you write a program in which no method has a side effect?
   By all means, yes: recursive factorial, recursive Fibonacci, recursively checking for palindromes.
   Would such a program be useful?
   Yes, of course. Sometimes though other ways might be more efficient.

8. What is the difference between a method and a program?
   Method just part of a program.
   The main method and a program?
   A main just one entry point in a program.
9. What preconditions do the following methods from the standard Java library have?

(a) `Math.sqrt`  
Any double, should be $\geq 0$, but fails safely.

(b) `Math.tan`  
Mostly not a whole number of right angles.

(c) `Math.log`  
The argument should be $\geq 0$.

(d) `Math.exp`  
Any double.

(e) `Math.pow`  
Tricky as in some cases the first argument must be non-negative.

(f) `Math.abs`  
No restrictions.

10. When a method is called with parameters that violate its precondition, it can terminate, or it can fail safely. Give two examples of library methods (standard, or the library methods used in this book) that fail safely when called with invalid parameters, and give two examples of library methods that terminate.

`Math` methods fail safely.

Methods that throw an exception that is not caught will cause the program to terminate.

11. Consider the following method that is intended to swap the values of two floating-point numbers:

```java
public static void falseSwap(double a, double b)
{
    double temp = a;
    a = b;
    b = temp;
}
```

```java
public static void main(String[] args)
{
    double x = 3;
    double y = 4;
    falseSwap(x, y);
    System.out.println(x + " " + y);
}
```

Why doesn’t the method swap the contents of x and y?

Because it works on copies of those values.

12. How can you write a method that swaps two floating-point numbers?

`Hint: Use Point2D.Double.`

```java
static void swap(Point2D.Double p) {
    p.setLocation(new Point2D.Double(p.getY(), p.getX()));
}
```
Warmups (VI)

Questions:

1. For each of the following sets of values, write code that fills an array \( a \) with the values:

   (a) 1 2 3 4 5 6 7 8 9 10
   (b) 0 2 4 6 8 10 12 14 16 18
   (c) 1 4 9 16 25 36 49 64 81 100
   (d) 0 0 0 0 0 0 0 0 0 0
   (e) 1 4 9 16 9 7 4 9 11

   Use a loop when appropriate.

2. Write a loop that fills an array \( a \) with ten random numbers between 1 and 100. Write code for two nested loops that fill \( a \) with ten different random numbers between 1 and 100.

3. Write Java code for a loop that simultaneously computes the maximum and the minimum of an array.

4. What is wrong with the following loop?

   ```java
   int[] v = new int[10];
   for (int i = 1; i <= 10; i++)
       v[i] = i * i;
   ```

   Explain two ways of fixing the error.

5. What is an array index? What are the bounds of an array?

6. Give an example of

   (a) A useful method that has an array of integers as a parameter that is not modified
   (b) A useful method that has an array of integers as a parameter that is modified
   (c) A useful method that has an array of integers as a return value

Just describe each method. Don’t implement the methods.
7. A method that has an array as a parameter can change the array in two ways. It can change the contents of individual array elements, or it can rearrange the elements. Describe two useful methods with `Product[]` parameters that change an array of products in each of the two ways just described.

8. What are parallel arrays? Why are parallel arrays indications of poor programming? How can they be avoided?

9. Design a class `Catalog` that stores a collection of products. What public methods should you support? What advantages and disadvantages does a `Catalog` class have over a `Product[]` array?

10. Suppose `v` is a *sorted* vector of products. Describe how a new product can be inserted in its proper position so that the resulting vector stays sorted.

11. How do you perform the following tasks with arrays in Java?
   
   (a) Test that two arrays contain the same elements in the same order.
   
   (b) Copy one array to another.
   
   (c) Fill an array with zeroes, overwriting all elements in it.
   
   (d) Remove all elements from an array.

12. True or false?

   (a) All elements of an array are of the same type.
   
   (b) Array subscripts must be integers.
   
   (c) Arrays cannot contain strings as elements.
   
   (d) Arrays cannot use strings as subscripts.
   
   (e) Parallel arrays must have equal length.
   
   (f) Two-dimensional arrays always have the same numbers of rows and columns.
   
   (g) Two parallel arrays can be replaced by a two-dimensional array.
   
   (h) Elements of different columns in a two-dimensional array can have different types.
   
   (i) Elements in a vector can have different types.

13. True or false?

   (a) A method cannot return a two-dimensional array.
   
   (b) A method can change the length of an array parameter.
   
   (c) A method cannot change the dimensions of a two-dimensional array parameter.
   
   (d) A method cannot change the length of a vector that is passed as a parameter.
   
   (e) A method can only reorder the elements of an array parameter, not change the elements.

Here's where the solutions\(^\text{36}\) are.

---

\(^{36}\text{Again, just turn the page.}\)
Warmups Solutions (VI)

Answers appear in \textcolor{blue}{(where appropriate)}.

Questions:

1. For each of the following sets of values, write code that fills an array \texttt{a} with the values:

   (a) \(1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10\)

   \begin{verbatim}
   int[] a = new int[10];
   for (int i = 0; i < a.length; i++)
       a[i] = (i + 1);
   \end{verbatim}

   (b) \(0 \ 2 \ 4 \ 6 \ 8 \ 10 \ 12 \ 14 \ 16 \ 18\)

   \begin{verbatim}
   int[] a = new int[10];
   for (int i = 0; i < a.length; i++)
       a[i] = 2 * i;
   \end{verbatim}

   (c) \(1 \ 4 \ 9 \ 16 \ 25 \ 36 \ 49 \ 64 \ 81 \ 100\)

   \begin{verbatim}
   int[] a = new int[10];
   for (int i = 0; i < a.length; i++)
       a[i] = i * i;
   \end{verbatim}

   (d) \(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0\)

   \begin{verbatim}
   int[] a = new int[10];
   \end{verbatim}

   (e) \(1 \ 4 \ 9 \ 16 \ 9 \ 7 \ 4 \ 9 \ 11\)

   \begin{verbatim}
   int[] a = { 1, 4, 9, 16, 9, 7, 4, 9, 11};
   \end{verbatim}

   Use a loop when appropriate.

2. Write a loop that fills an array \texttt{a} with ten random numbers between 1 and 100. Write code for two nested loops that fill \texttt{a} with ten \textit{different} random numbers between 1 and 100.
for (int i = 0; i < a.length; i++)
a[i] = 1 + gen.nextInt(100);
//-----------------------------
for (int i = 0; i < a.length; i++) {
    int number = 1 + gen.nextInt(100);
    int count = 0;
    for (int j = 0; j < i; j++) {
        if (a[j] == number) {
            count += 1;
        }
    }
    if (count > 0) i -= 1; // ... why?
else
    a[i] = number;
}

3. Write Java code for a loop that simultaneously computes the maximum and the minimum of an array.

   int min = a[0];
   int max = a[0];
   for (int i = 0; i < a.length; i++) {
       if (min > a[i]) min = a[i];
       if (max < a[i]) max = a[i];
   }

4. What is wrong with the following loop?

   int[] v = new int[10];
   for (int i = 1; i <= 10; i++)
       v[i] = i * i;

   v[10] will produce an array out of bounds exception.
   Explain two ways of fixing the error.
   Change the limits to 0 and 9

   for (int i = 0; i <= 9; i++)
       v[i] = (i + 1) * (i + 1);

   or change the body of the for loop to index v with a value of i - 1

   for (int i = 1; i <= 10; i++)
       v[i - 1] = i * i;

5. What is an array index? What are the bounds of an array?

   An integer-valued expression that indicates position in a sequence (starting from 0). The bounds are 0 and the length of the array minus one.

6. Give an example of
(a) A useful method that has an array of integers as a parameter that is not modified
Computing and reporting the average, finding min, max, counting, searching, some versions of `append`,
`insert` and `remove` (those that work with full arrays, not with companion variables).

(b) A useful method that has an array of integers as a parameter that is modified
Sorting an array. Also those versions of `remove` and `insert` that work with a companion variable.

(c) A useful method that has an array of integers as a return value
Generating an array of random values. Those versions of `insert` and `remove` that work with full
arrays.

Just describe each method. Don’t implement the methods.

7. A method that has an array as a parameter can change the array in two ways. It can change the contents of
individual array elements, or it can rearrange the elements. Describe two useful methods with `Product` parameters
that change an array of products in each of the two ways just described.

Think of an array as being a book shelf. The books are the actual products that the shelf is storing. One
can do these two things:

(a) sort them by price
(b) lower their prices by 5% (change the label on the back cover).

8. What are parallel arrays? Why are parallel arrays indications of poor programming? How can they be
avoided?
Parallel arrays are arrays of features such that the collection of all values that appear in the locations with
the same index (i) in the arrays all refer to one and the same entity.
Avoid them by defining an entity class (with those features) and creating an array of such entities. Helps
keep the features in sync.

9. Design a class `Catalog` that stores a collection of products. What public methods should you support?
What advantages and disadvantages does a `Catalog` class have over a `Product` array?
The products will be available by name. The advantage would be that the names could be meaningful,
but you’d have to either know it or implement a method that works as a table of contents. Another
disadvantage is that you can’t add a new product at run time, you’d have to change the source code and
recompile. In this respect a `Hashtable` would work much better.
Arrays will give us more uniformity and the ability to scan all the elements through a loop, sort the
products in the catalog, add some more at run time if we want to, the disadvantage is that the index may
be somewhat meaningless.

10. Suppose \( v \) is a sorted vector of products. Describe how a new product can be inserted in its proper position
so that the resulting vector stays sorted.
Start from the beginning and search for an element that has the same value. You will be going either up
or down. Insert where you find the value or when you know that you want be able to find the value (the
value is below or above the one that you’re looking, depending on how the vector is sorted – ascending or
descending).

11. How do you perform the following tasks with arrays in Java?

(a) Test that two arrays contain the same elements in the same order.
The arrays should have the same length. Compare the elements one by one and stop with `false` if
there is a mismatch. Otherwise the answer is: `true`
(b) Copy one array to another.
Arrays should be the same length. Use a loop to scan all elements in one array and set the elements with the same index in the other array have the values you find in the first.
(c) Fill an array with zeroes, overwriting all elements in it.
There’s more than one way of answering this question: you could essentially get a new array of the same size (filled with zero by default) and change the reference to it. If you’re doing this in a method it won’t work so you need to change the elements to 0 one by one.
(d) Remove all elements from an array.
If you have the actual reference just point it to a new array of length 0. Otherwise return such an array and ask the caller to do that (agree on a protocol).

12. True or false?
(n) All elements of an array are of the same type. Yes.
(b) Array subscripts must be integers. Yes, of type int.
(c) Arrays cannot contain strings as elements. They can.
(d) Arrays cannot use strings as subscripts. They cannot.
(e) Parallel arrays must have equal length. They should.
(f) Two-dimensional arrays always have the same numbers of rows and columns. They may not.
(g) Two parallel arrays can be replaced by a two-dimensional array. Sometimes, in certain circumstances.
(h) Elements of different columns in a two-dimensional array can have different types. No. How could you declare that.
(i) Elements in a vector can have different types. That’s why the Vector treats them all as Objects.

13. True or false?
(a) A method cannot return a two-dimensional array. It can.
(b) A method can change the length of an array parameter. Not directly, not really.
(c) A method cannot change the dimensions of a two-dimensional array parameter. Indeed.
(d) A method cannot change the length of a vector that is passed as a parameter. It can (just add elements to it)
(e) A method can only reorder the elements of an array parameter, not change the elements. 

Help with Homework Four

Let's write a program that reads a `String` and reports it.
And in the process puts parens around the characters.

```java
class One {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter: ");
        String line = console.readLine();
        System.out.println(line);
        for (int i = 0; i < line.length(); i++) {
            char c;
            c = line.charAt(i);
            System.out.print("(" + c + ")");
        }
        System.out.println();
    }
}
```

Here's a run with it.

```
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
Enter: adrian
adrian
t(a)(r)(i)(a)(n)
frilled.cs.indiana.edu%
```

Now let’s write a program that sums up the digits in such a `String`.

```java
class Two {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter: ");
        String line = console.readLine();
```
System.out.println(line);
int sum = 0;
for (int i = 0; i < line.length(); i++) {
    sum = sum + line.charAt(i) - '0';
}
System.out.println("The sum is: " + sum);
}

Of course, we need to assume that the characters are all digits.

frilled.cs.indiana.edu%java Two
Enter: 123
123
The sum is: 6
frilled.cs.indiana.edu%java Two
Enter: 666
666
The sum is: 18
frilled.cs.indiana.edu%java Two
Enter: 123456789
123456789
The sum is: 45
frilled.cs.indiana.edu%java Two
Enter: AB
AB
The sum is: 35
frilled.cs.indiana.edu%

Otherwise, even if it computes, it doesn’t make much sense.

So the user has to enter only digits.

Now let’s set up two arrays and sum them up element by element.

class Three {
    public static void main(String[] args) {
        int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9};
        int[] b = { 9, 8, 7, 6, 5, 4, 3, 2, 1};
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i] + " + " + b[i] + " = " + (a[i] + b[i]));
        }
    }
}

Here’s what “element by element” means.

frilled.cs.indiana.edu%java Three
1 + 9 = 10
2 + 8 = 10
3 + 7 = 10
4 + 6 = 10
5 + 5 = 10
6 + 4 = 10
7 + 3 = 10
8 + 2 = 10
9 + 1 = 10
frilled.cs.indiana.edu%

Let's store these values in a third array.

class Four {
    public static void main(String[] args) {
        int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int[] b = {9, 8, 7, 6, 5, 4, 3, 2, 1};
        int[] c = new int[a.length];
        for (int i = 0; i < a.length; i++) {
            c[i] = a[i] + b[i];
        }
    }
}

But this program doesn't print anything.

So let's print the resulting array with parens, as in the beginning.

class Four {
    public static void main(String[] args) {
        int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int[] b = {9, 8, 7, 6, 5, 4, 3, 2, 1};
        int[] c = new int[a.length];
        for (int i = 0; i < a.length; i++) {
            c[i] = a[i] + b[i];
        }
        for (int i = 0; i < c.length; i++) {
            System.out.print("(" + c[i] + ")");
        }
        System.out.println();
    }
}

Here's a run with it.

frilled.cs.indiana.edu%java Four
(10) (10) (10) (10) (10) (10) (10) (10) (10) (10) frilled.cs.indiana.edu%

Now let's read a String of digits, and turn that into an array of ints.

class Five {
    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        String input = c.readLine();
        int[] digits = new int[input.length()];
        for (int i = 0; i < input.length(); i++) {
            digits[i] = input.charAt(i) - '0';
        }
    }
}
System.out.print("Enter: ");
String line = c.readLine();
int[] a = new int[line.length()];
for (int i = 0; i < line.length(); i++) {
    a[i] = line.charAt(i) - '0';
}
for (int i = 0; i < a.length; i++) {
    System.out.print("(" + a[i] + ")");
}
System.out.println();
}

Note that if the assumption is broken our calculations are no longer meaningful.

frilled.cs.indiana.edu%java Five
Enter: 123
(1)(2)(3)
frilled.cs.indiana.edu%java Five
Enter: 67890
(6)(7)(8)(9)(0)
frilled.cs.indiana.edu%java Five
Enter: abc
(49)(50)(51)
frilled.cs.indiana.edu%

Now, let’s

• read in two Strings of the same length
• turn them into arrays
• add them up element by element and store the results in a third array
• then print the result

Here’s the code, not much different from what we’ve done before.

class Six {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter: ");
        String line = console.readLine();
        int[] a = new int[line.length()];
        for (int i = 0; i < line.length(); i++) {
            a[i] = line.charAt(i) - '0';
        }
        System.out.print("Enter: ");
        line = console.readLine();
        int[] b = new int[line.length()];
        for (int i = 0; i < line.length(); i++) {
        }
b[i] = line.charAt(i) - '0';
}
int[] c = new int[a.length];
for (int i = 0; i < c.length; i++) {
    c[i] = a[i] + b[i];
}
for (int i = 0; i < c.length; i++) {
    System.out.print("(" + c[i] + ")");
    System.out.println();
}

Once again the assumptions are important, because we don’t enforce or double-check for anything.

frilled.cs.indiana.edu%java Six
Enter: 12345
Enter: 12345
(2)(4)(6)(8)(10)
frilled.cs.indiana.edu%java Six
Enter: 6666
Enter: 6666
(12)(12)(12)(12)
frilled.cs.indiana.edu%java Six
Enter: 123
Enter: 1234
(2)(4)(6)
frilled.cs.indiana.edu%java Six
Enter: 1234
Enter: 123
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
    at Six.main(Six.java:27)
frilled.cs.indiana.edu%

Now suppose we have an array with overflow, which needs to be propagated forward.
(Forward means from left to right, here.)

class Seven {
    public static void main(String[] args) {
        int[] c = { 12, 13, 14, 15 };  
        for (int i = 0; i < c.length - 1; i++) {
            if (c[i] >= 10) {
                c[i + 1] = c[i + 1] + (c[i] / 10);
                c[i] = c[i] % 10;
            }
        }
        Seven.show(c);
    }
    static void show(int[] c) {
        for (int i = 0; i < c.length; i++) {
System.out.print("(" + c[i] + ")");
} System.out.println();
}

Here's how it runs.

frilled.cs.indiana.edu%javac Seven.java
frilled.cs.indiana.edu%java Seven
(2)(4)(5)(16)
frilled.cs.indiana.edu%

Here now is a program that reads a String and reports it backwards.
And in the process puts parens around the characters.

class Eight {
   public static void main(String[] args) {
      BufferedReader console = new BufferedReader(System.in);
      System.out.print("Enter: ");
      String line = console.readLine();
      System.out.println(line);
      for (int i = line.length() - 1; i >= 0; i--) {
         System.out.print("(" + line.charAt(i) + ")");
      }
      System.out.println();
   }
}

Here's how it works.

frilled.cs.indiana.edu%javac Eight.java
frilled.cs.indiana.edu%java Eight
Enter: adrian
adrian
(n)(a)(i)(r)(d)(a)
frilled.cs.indiana.edu%java Eight
Enter: nairda
nairda
(a)(d)(r)(i)(a)(n)
frilled.cs.indiana.edu%

Why do you think this is important?

Let's

- read two numbers of same length (as Strings)
- store them backwards in two arrays
- sum them up, and store the result in a third array
• clean it up by propagating any overflow in the result
• print the result backwards

Here's the code.

```java
class Nine {
    public static void main(String[] args) {
        ConsoleReader console = new ConsoleReader(System.in);
        System.out.print("Enter: ");
        String line = console.readLine();
        int[] a = new int[line.length()];
        for (int i = a.length - 1, j = 0; i >= 0; i--, j++) {
            a[j] = line.charAt(i) - '0';
        }
        System.out.print("Enter: ");
        line = console.readLine();
        int[] b = new int[line.length()];
        for (int i = a.length - 1, j = 0; i >= 0; i--, j++) {
            b[j] = line.charAt(i) - '0';
        }
        int[] c = new int[a.length];
        for (int i = 0; i < a.length; i++) {
            c[i] = a[i] + b[i];
        }
        Nine.clean(c);
        for (int i = c.length - 1; i >= 0; i--)
            System.out.print(" + c[i] + ");
        System.out.println();
    }
    static void clean(int[] c) {
        for (int i = 0; i < c.length - 1; i++) {
            if (c[i] >= 10) {
                c[i + 1] = c[i + 1] + (c[i] / 10);
                c[i] = c[i] % 10;
            }
        }
    }
}
```

Here's how it runs.

```
frilled.cs.indiana.edu%javac Nine.java
frilled.cs.indiana.edu%java Nine
Enter: 1
Enter: 1
(2)
frilled.cs.indiana.edu%java Nine
Enter: 10
Enter: 12
(2)(2)
```
frilled.cs.indiana.edu%java Nine
Enter: 01
Enter: 16
(1)(7)
frilled.cs.indiana.edu%java Nine
Enter: 99
Enter: 99
(19)(8)
frilled.cs.indiana.edu%

To reach this stage is the goal of this set of notes.

So here's [YOUR NEXT CHALLENGE]

Change what we have above into something that works like this:

frilled.cs.indiana.edu%java Ten
Enter: 1
Enter: 1
Result 2
frilled.cs.indiana.edu%java Ten
Enter: 123
Enter: 001
Result 124
frilled.cs.indiana.edu%java Ten
Enter: 999
Enter: 999
Result 1000
frilled.cs.indiana.edu%java Ten
Enter: 999
Enter: 999
Result 1998
frilled.cs.indiana.edu%

This program

* expects numbers of the same length, and
* adds them up

If you have numbers of not the same length use leading 0's (zeros).

So 100 + 1 would be written 100 + 001.

Here's also how you can fix the alignment problem:

Nine.clean(c);
System.out.print("Result");
if (c[c.length - 1] < 10)
    System.out.print(" ");
for (int i = c.length - 1; i >= 0; i--)
    System.out.print(c[i]);
System.out.println;
So now my program works like this (and yours should too).

frilled.cs.indiana.edu%java Ten
Enter: 123
Enter: 123
Result 246
frilled.cs.indiana.edu%java Ten
Enter: 999
Enter: 001
Result:1000
frilled.cs.indiana.edu%java Ten
Enter: 100
Enter: 001
Result:101
frilled.cs.indiana.edu%java Ten
Enter: 999
Enter: 999
Result:1998
frilled.cs.indiana.edu%

For the homework assignment you just need to remove the need to add leading 0's (zeros).

Here's some help with eliminating the leading zero(e)s:

class Example {
    public static void main(String[] args) {
        String[] numbers = args; // = { "0000070001",
                      "0000000000",
                      "9000000000" };
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(Example.transform(numbers[i]));
        }
    }
    public static String transform(String number) { // our focus, actually...
        String result = "";
        boolean done = false;
        for (int i = 0; i < number.length(); i++) {
            if (!done) {
                if (number.charAt(i) == '0')
                    result += "_";
            } else {
                result += number.charAt(i);
                done = true;
            }
        }
        return result;
    }
}
Notice that this is written in the style required for Lab Eight.

Here's how it runs:

```java
drJava> java Example 00000001 90000000 90010000 00010001 00000000
------- 1
90000000
90010000
---10001
------- 0
```

We're using the command line arguments so we don't compile every time.

Can you guess the text of each of the following problems?

1. class ExpApproximator {
   double term = 1, sum = 1;
   int n = 0;
   double x;
   ExpApproximator(double x) { this.x = x; }
   boolean keepGoing() {
     if (Math.abs(term) < 0.00001) return false;
     return true;
   }
   void calculate() {
     n += 1;
     term = term * x / n;
     sum += term;
     this.report();
   }
   void report() { System.out.println(sum); }
   public static void main(String[] args) {
     BufferedReader c = new BufferedReader(System.in);
     System.out.print("What's the exponent? ");
     double exp = c.readDouble();
     ExpApproximator a = new ExpApproximator(exp);
     while (a.keepGoing()) {
       a.calculate();
     }
     a.report();
   }
}

2. Here's a solution worked out Monday night (some Monday night):

   class Newton_Raphson {
   public static void main(String[] args) {
     BufferedReader c = new BufferedReader(System.in);
     System.out.print("What's the number? ");
     double a = c.readDouble();
   }
}
System.out.print("What's the order of the root? ");
double n = c.readDouble();
double x0ld = 1;
while (Math.abs(a - Math.pow(x0ld, n)) > 0.01) {
    x0ld = x0ld - (Math.pow(x0ld, n) - a) /
            (n * Math.pow(x0ld, n - 1));
    System.out.println(x0ld);
}
System.out.println("Proof: " + Math.pow(x0ld, n));
}

3. Remember the Wizard?
Here's a simplified version of it:

class Wizard {
    int countFactors(int number) {
        int count = 0;
        for (int i = 2; number > 1; i++) {
            while (number % i == 0) {
                count += 1;
                System.out.println(i);
                number = number / i;
            }
        }
        return count;
    }
    public static void main(String[] args) {
        Wizard a = new Wizard();
        a.countFactors(Integer.parseInt(args[0]));
    }
}

4. Write a graphical applet that prompts the user to enter a number n and then draws n circles with random center and random radius.
Here's a solution to this problem:

import java.applet.*;
import java.awt.*;
import javax.swing.JOptionPane;
/*
   <applet code="One.class" width=300 height=300>
   </applet>
*/
public class One extends Applet {
    int number;
    Circle[] circles;
    public void init() {
        this.number = Integer.parseInt(  
                JOptionPane.showInputDialog("How many? "));
    }
this.circles = new Circle[number];
for (int i = 0; i < circles.length; i++)
circles[i] = new Circle(
        (int)(Math.random() * 200 + 50),
        (int)(Math.random() * 200 + 50),
        (int)(Math.random() * 25 + 25),
        new Color((float)Math.random(),
                (float)Math.random(),
                (float)Math.random(),
                (float)Math.random())
    );
}
public void paint(Graphics g) {
    for (int i = 0; i < circles.length; i++)
        this.circles[i].show(g);
}
}
class Circle {
    int x, y;
    int radius;
    Color color;
    Color border;
    Circle(int x, int y, int r, Color c, Color b) {
        this.x = x; this.y = y;
        this.radius = r;
        this.color = c; this.border = b;
    }
    void show(Graphics g) {
        g.setColor(color);
        g.fillOval(x - radius, y - radius, 2 * radius, 2 * radius);
        g.setColor(border);
        g.drawOval(x - radius, y - radius, 2 * radius, 2 * radius);
    }
}

5. Remember this problem?

import java.util.*;
class Average {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        ConsoleReader c = new ConsoleReader(System.in);
        System.out.print("Type the numbers: ");
        String line = c.readLine();
        calc.process(line);
    }
}
class Calculator {

void process(String line) {
    StringTokenizer st = new StringTokenizer(line);
    double sum = 0, sumSq = 0, n = 0;
    while (st.hasMoreTokens()) {
        int x_i = Integer.parseInt(st.nextToken());
        sum += x_i;
        sumSq += x_i * x_i;
        n += 1;
    }
    System.out.println(n + " numbers.");
    System.out.println("Average is: " + sum / n);
    System.out.println("Std. deviation: " +
            Math.sqrt((sumSq - sum * sum / n) / (n - 1)));
}

6. Here's a more general solution of an older problem:

    class GradeConverter {
        String letter(double number) {
            double[] values = { 0, 0.7, 1, 1.3, 1.7, 2, 2.3, 2.7, 3, 3.3, 3.7, 4};
            for (int i = 0; i < values.length; i++) {
                if (values[i] > number) {
                    double average = (values[i] + values[i-1]) / 2;
                    if (number >= average) return letters[i];
                    else return letters[i-1];
                }
            }
            return "I'm sorry!... Not a valid input."
        }
        public static void main(String[] args) {
            GradeConverter a = new GradeConverter();
            for (int i = 0; i < 10; i++) {
                double number = Math.random() * 4;
                System.out.println(number + " = " + a.letter(number));
            }
            double number = 3.85;
            System.out.println(number + " = " + a.letter(number));
        }
    }

7. Here's reverse from Lab Three but with a more general solution(s):

    class DigitExtractor {
        int theNumber;
        int index;
        String theCopy;
        DigitExtractor (int aNumber) {
            this.theNumber = aNumber;
            }
```java
this.theCopy = this.theNumber + ";
this.index = this.theCopy.length();

int nextDigit() {
    this.index -= 1;
    return theCopy.charAt(this.index) - '0';
}

boolean hasMoreDigits() {
    if (index > 0) return true;
    else return false;
}

public static void main(String[] args) {
    DigitExtractor d = new DigitExtractor(123456789);
    while (d.hasMoreDigits()) {
        System.out.println(d.nextDigit());
    }
}

class DigitExtractorTwo {
    int theNumber;
    DigitExtractorTwo (int aNumber) {
        this.theNumber = aNumber;
    }

    int nextDigit() {
        int digit = this.theNumber % 10;
        this.theNumber = this.theNumber / 10;
        return digit;
    }

    boolean hasMoreDigits() {
        if (this.theNumber > 0) return true;
        else return false;
    }

    public static void main(String[] args) {
        DigitExtractorTwo d = new DigitExtractorTwo(987654321);
        while (d.hasMoreDigits()) {
            System.out.println(d.nextDigit());
        }
    }
}

A similar approach is used in Lecture Notes Thirteen (Four.java).

8. import java.util.*;

class Historian {
    int max = 0;
    int min = 0;
    int sum = 0;
    int num = 0;

    void record(int a) {
```
this.sum += a;
if (this.num == 0) {
    this.min = a;
    this.max = a;
}
if (this.min > a) this.min = a; // a bit redun-
if (this.max < a) this.max = a; // dant sometimes
this.num += 1; // order of updates is important!
}
void report() {
    if (this.num == 0) {
        System.out.println("Nothing to report on...");
    } else {
        System.out.println
        (" I've looked at " + this.num + " number(s) thus far.
" + " The largest I've seen was " + this.max + 
" The smallest was " + this.min + 
" The average of the numbers is " + 
this.sum / (double) this.num
    );
}
}
// testing...
public static void main(String[] args) {
    ConsoleReader c = new ConsoleReader(System.in);
    String line;
    System.out.println("Welcome to the Historian!");
    System.out.print("enter> ");
    line = c.readLine();
    while (! line.equals("done")) {
        StringTokenizer stapler = new StringTokenizer(line);
        Historian herodotus = new Historian();
        while (stapler.hasMoreTokens()) {
            herodotus.record(Integer.parseInt(
                stapler.nextToken()));
        }
        herodotus.report();
        System.out.print("enter> ");
        line = c.readLine();
    }
}
Here's a sample session with this program:
frilled.cs.indiana.edu%java Historian
Welcome to the Historian!
enter> 1 2 3
I've looked at 3 number(s) thus far.
The largest I've seen was 3
The smallest was 1
The average of the numbers is 2.0
enter> -1 -3 4 2 5 0 -1
I've looked at 7 number(s) thus far.
The largest I've seen was 5
The smallest was -3
The average of the numbers is 0.8571428571428571
enter> Nothing to report on...
enter> 0
I've looked at 1 number(s) thus far.
The largest I've seen was 0
The smallest was 0
The average of the numbers is 0.0
enter> done
frilled.cs.indiana.edu%

9. Here's another (similar one):

import java.util.*;

class Bracketizer {
    int[] memory;
    int available;

    Bracketizer(int size) {
        this.memory = new int[size];
    }

    void record(int n) {
        this.memory[available] = n;
        this.available += 1;
    }

    void report() {
        for (int i = 0; i < this.available; i++)
            if (this.memory[i] % 2 == 0)
                System.out.print("(" + this.memory[i] + ")");
            else System.out.print("[" + this.memory[i] + "]");
        System.out.println();
    }

    public static void main(String[] args) {
        ConsoleReader c = new ConsoleReader(System.in);
        String line;
System.out.println("Welcome to the [B][r](a) [c] [k](e) [t](i) [z](e) [r]");
System.out.print("enter > ");
line = c.readLine();
while (! line.equals("dome")) {
    StringTokenizer stapler = new StringTokenizer(line);
    Bracketizer stanley = new Bracketizer(stapler.countTokens());
    while (stapler.hasMoreTokens())
        stanley.record(Integer.parseInt(stapler.nextToken()));
    stanley.report();
    System.out.print("enter > ");
    line = c.readLine();
}

Here's how it runs:

frilled.cs.indiana.edu%javac *.java
frilled.cs.indiana.edu%java Bracketizer
Welcome to the [B][r](a) [c] [k](e) [t](i) [z](e) [r]
enter> 1 2 3 4 5
[1] (2) [3] (4) [5]
enter> enter> -1 -2 -3 0 0 0 1 100 200 101 201
[-1] [-2] [-3] (0) (0) (0) [1] (100) (200) [101] [201]
enter> done
frilled.cs.indiana.edu%

10. class Tigger {
    String x, y;
    Tigger(int x, int y) {
        this.x = x + "";
        this.y = y + "";
    }
    void bounce() {
        int a = calculate(x),
            b = calculate(y);
        this.x = a + "";
        this.y = b + "";
    }
    int calculate(String a) {
        int sum = 0;
        for (int i = 0; i < a.length(); i++) {
            sum += (a.charAt(i) - '0') * (a.charAt(i) - '0');
        }
        return sum;
    }
    String report() {
        return x = " " + this.x,
544

```java
    y = " " + this.y;
    return
        "Tigger just bounced to (" +
        x.substring(x.length() - 3) + ", ",
        + y.substring(y.length() - 3) + ")";
    }
  }

11. class One {
    public static void main(String[] args) {
        Player bonaparte, wellington;
        bonaparte = new Player();
        wellington = new Player();
        System.out.println("Let the game begin!");
        bonaparte.makeGuess();
        wellington.makeGuess();
        System.out.println("The guesses have been made: ");
        System.out.println(" Bonaparte has chosen .... " + bonaparte.report());
        System.out.println(" Wellington has chosen ... " + wellington.report());
        if (bonaparte.strongerThan(wellington))
            System.out.println("Bonaparte wins!");
        else if (wellington.strongerThan(bonaparte))
            System.out.println("Wellington wins!");
        else System.out.println("It’s a draw... ");
    }
}

class Player {
    String guess;
    String makeGuess() {
        int value = (int) (Math.random() * 3);
        if (value == 0) this.guess = "paper";
        if (value == 1) this.guess = "rock";
        if (value == 2) this.guess = "scissors";
        return this.guess;
    }

    boolean strongerThan(Player other) {
        if (this.guess.equals("paper") && other.guess.equals("rock") ||
            this.guess.equals("rock") && other.guess.equals("scissors") ||
            this.guess.equals("scissors") && other.guess.equals("paper"))
            return true;
        else return false;
    }

    String report() {
        return guess;
    }
}

12. class One {
    public static void main(String[] args) {
        Line a = new Line(new Point(0, 0), new Point (1, 1));
        System.out.println(a.length());
    }
```
class Line {
    Point a, b;
    Line(Point a, Point b) {
        this.a = a;
        this.b = b;
    }
    double length() {
        return (this.a).distanceTo(this.b);
    }
}

class Point {
    double x, y;
    Point(double x, double y) {
        this.x = x; this.y = y;
    }
    double distanceTo(Point other) {
        double
dX = this.x - other.x,
        dY = this.y - other.y;
        return Math.sqrt(dX * dX + dY * dY);
    }
}

13. class Oracle {
    ConsoleReader c = new ConsoleReader(System.in);
    void takeCall() {
        System.out.println("Oracle> Hi, ask me any question... ");
        String question = c.readLine();
        System.out.println("Oracle> That's a tough one... any words of wisdom that would apply to this?");
        this.advice = this.c.readLine();
        System.out.println("Oracle> Nice. Back to your question I'd say: ");
        System.out.println("***(" + this.answer + ")***");
        this.answer = this.advice;
    }
    String answer = "The answer, my friend, is in the blowing of the wind.";
    String advice = " Man gave names to all the animals, in the beginning... ";
}

14. class One {
    public static void main(String[] args) {
        Elevator e = new Elevator(20);
        e.up(26);
        e.down(14);
        e.up(10);
        e.down(30);
        e.up(e.currentFloor() + 3);
    }
}

class Elevator {
int floor;
Elevator(int floor) {
    this.floor = floor;
}
void up(int to) {
    if (this.floor >= to) {
        System.out.println("Sorry, from floor " + this.floor + 
                        " we can't go up to floor " + to);
    } else {
        System.out.println("Elevator going up (" + 
                        this.floor + 
                        " --> " + to + ")");
        for (int i = this.floor; i <= to; i++) {
            this.floor = i;
            this.report();
        }
        System.out.println("Elevator now on floor: " + this.floor);
    }
}
void down(int to) {
    if (this.floor <= to) {
        System.out.println("Sorry, from floor " + this.floor + 
                            " we can't go down to floor " + to);
    } else {
        System.out.println("Elevator going down: (" + 
                            this.floor + 
                            " --> " + to + ")");
        for (int i = this.floor; i >= to; i--) {
            this.floor = i;
            this.report();
        }
        System.out.println("Elevator now on floor: " + this.floor);
    }
}
void report() {
    System.out.println("The elevator is now on floor " + this.floor);
}
int currentFloor() { return this.floor; }

That's it.
Best of luck in preparing the exam!
Let me know if you need help.
Arrays and Methods

Here is your A201/A597 [LAB ASSIGNMENT NINE]

Here are seven problems from the book with answers (p. 472, problems P11.1-7)\(^7\). Read them and understand them. Then write a main program that illustrates how you’d be using them in a program. This lab is the opposite of lab 8: in that one you were given a main and asked to write the methods. Here we give you the methods and ask you to put them to use (and write the main).

1. Write a method

   ```java
   static double scalarProduct(double[] a, double[] b)
   ```

   that computes the scalar product of two mathematical vectors (represented as arrays).
   The scalar product is:

   \[ a_0b_0 + a_1b_1 + \ldots + a_{n-1}b_{n-1} \]

2. Write a method that computes the alternating sum of all elements in an array.
   For example, if `alternatingSum` is called with an array containing

   \[
   1 \quad 4 \quad 9 \quad 16 \quad 9 \quad 7 \quad 4 \quad 9 \quad 11
   \]

   Then it computes

   \[
   1 - 4 + 9 - 16 + 9 - 7 + 4 - 9 + 11
   \]

   which is, of course, -2.

3. Write a method `reverse` that reverses the sequence of elements in an array.
   For example, if `reverse` is called with an array containing

   \[
   1 \quad 4 \quad 9 \quad 16 \quad 9 \quad 7 \quad 4 \quad 9 \quad 11
   \]

\(^7\) As usual, ignore the reference for the time being. Details will be given in class.
then the array is changed to

```
  11  9  4  7  9  16  9  4  1
```

4. Write a method

```java
public static int[] append(int[] a, int[] b)
```

that appends one array after another. For example, if `a` is

```
  1  4  9  16
```

and `b` is

```
  9  7  4  9  11
```

then `append` returns the array

```
  1  4  9  16  9  7  4  9  11
```

5. Write a predicate method

```java
public static boolean equals(int[] a, int[] b)
```

that checks whether two arrays have the same elements in the same order.

6. Write a predicate method

```java
public static boolean sameSet(int[] a, int[] b)
```

that checks whether two arrays have the same elements in some order, ignoring multiplicities. For example, the two arrays

```
  1  4  9  16  9  7  4  9  11
```

and

```
  11  11  7  9  16  4
```

would be considered to have the same set. You will probably need one or more helper methods.

7. Write a predicate method

```java
public static boolean sameElements(int[] a, int[] b)
```

that checks whether two arrays have the same elements in some order, with the same multiplicities. For example,

```
  1  4  9  16  9  7  4  9  11
```
and

11 1 4 9 16 9 7 4 9

would be considered to have the same elements, but

1 4 9 16 9 7 4 9 11

and

11 11 7 9 16 4 1

would not.
You will probably need one or more helper methods.

Here are possible solutions.

1.

```java
public static double scalarProduct(double[] a, double[] b) {
    double result = 0;
    for (int i = 0; i < a.length; i++)
        result += a[i] * b[i];
    return result;
}
```

2.

```java
public static double alternatingSum(double[] a) {
    if (a.length == 0) return 0;
    double value = 0;
    for (int i = 0; i < a.length; i++) {
        if (i % 2 == 0) value += a[i];
        else value -= a[i];
    }
    return value;
}
```

3.

```java
public static void reverse(double[] a) {
    for (int i = 0; i < a.length / 2; i++) {
        double temp = a[i];
        a[i] = a[a.length - i - 1];
        a[a.length - i - 1] = temp;
    }
}
```

4.

```java
public static int[] append(int[] a, int[] b) {
    int[] result = new int[a.length + b.length];
    for (int i = 0; i < a.length; i++)
        result[i] = a[i];
```
for (int i = 0; i < b.length; i++)
    result[i + a.length] = b[i];
return result;
}

5.
public static boolean equals(int[] a, int[] b) {
    if (a.length != b.length)
        return false;
    for (int i = 0; i < a.length; i++)
        if (a[i] != b[i])
            return false;
    return true;
}

6.
public static boolean sameSet(int[] a, int[] b) {
    for (int i = 0; i < a.length; i++)
        if (!contains(b, a[i]))
            return false;
    for (int i = 0; i < b.length; i++)
        if (!contains(a, b[i]))
            return false;
    return true;
}

7.
public static boolean sameElements(int[] a, int[] b) {
    for (int i = 0; i < a.length; i++)
        if (count(a, a[i]) != count(b, a[i]))
            return false;
    for (int i = 0; i < b.length; i++)
        if (count(a, b[i]) != count(b, b[i]))
            return false;
    return true;
}

Understand them, then put together a main method that illustrates your understanding.
Arrays of Objects


Let's now imagine a different situation than the one with which we introduced Java arrays.

<table>
<thead>
<tr>
<th>Different but similar.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume the user has three pieces of information to enter each time:</td>
</tr>
<tr>
<td>• name of product,</td>
</tr>
<tr>
<td>• price in dollars, and a</td>
</tr>
<tr>
<td>• performance score.</td>
</tr>
</tbody>
</table>

The best product is the one that has the highest ratio between performance and price.

Yes, simply picking the lowest price may not be the best bargain.

So now we want to analyze our data set from a different, more enhanced perspective.

Looks like we are going to build three arrays.

Indeed, that will be our first approach.

Easy to write after all those little programs.

We will then propose a better alternative.

Let's write the first one first.

```java
import java.util.StringTokenizer;
class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to the evaluator. ");
        ConsoleReader console = new ConsoleReader(System.in);
        int dataSize = 0;
        while (true) {
            System.out.print("[" + dataSize + "]\%");
            String line = console.readLine();
            if (line == null) break;
            StringTokenizer tokenizer = new StringTokenizer(line);
            String product = tokenizer.nextToken();
            String product = tokenizer.nextToken();
            double score = Double.parseDouble(tokenizer.nextToken());
            double price = Double.parseDouble(tokenizer.nextToken());
            dataSize += 1;
        }
        System.out.println("\n** Data has been entered now. ");
    }
```
System.out.println("** We need to process it.");
System.out.println("** We then need to print the results.");
System.out.println("\nThank you for using our program!");

Well, where are the arrays? This is just the framework, the protocol.

Indeed, the program doesn’t store anything. Well, then let me

- declare,
- initialize, and
- build

these three arrays:

import java.util.StringTokenizer;
class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to the evaluator. ");
        ConsoleReader console = new ConsoleReader(System.in);
        int dataSize = 0;
        final int DATA_LENGTH = 1000;
        String[] names = new String[DATA_LENGTH];
        double[] scores = new double[DATA_LENGTH];
        double[] prices = new double[DATA_LENGTH];
        while (true) {
            System.out.println("[" + dataSize + "]");
            String line = console.readLine();
            if (line == null) break;
            StringTokenizer tokenizer = new StringTokenizer(line);
            String product = tokenizer.nextToken();
            double score = Double.parseDouble(tokenizer.nextToken());
            double price = Double.parseDouble(tokenizer.nextToken());
            names[dataSize] = product;
            scores[dataSize] = score;
            prices[dataSize] = price;
            dataSize += 1;
        }
        System.out.println("\n** Data has been entered now.");
        System.out.println("** We need to process it.");
        System.out.println("** We then need to print the results.");
        for (int i = 0; i < dataSize; i++)
            System.out.println(names[i] + " " +
                scores[i] + " " + prices[i]);
        System.out.println("\nThank you for using our program!");
    }
}

I see you have defined three arrays, and that you’re Yes, I collect the data and print it at the end. are managing them now in your code.
<table>
<thead>
<tr>
<th>What kind of processing are you going to do?</th>
<th>Find the best bargain, the product with the highest ratio.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isn’t it a pain to have to maintain three arrays at the same time?</td>
<td>I’m not sure yet. Let me finish the program first.</td>
</tr>
<tr>
<td>Sure, this will be good practice.</td>
<td>I’ll also add a &quot;quit&quot; keyword to what the program understands,</td>
</tr>
</tbody>
</table>

...which would make the conversation a bit cleaner, indeed.

```java
import java.util.StringTokenizer;

class One {
    public static void main(String[] args) {
        System.out.println("Hello, and welcome to the evaluator. Type quit to quit.");
        ConsoleReader console = new ConsoleReader(System.in);
        int dataSize = 0;
        final int DATA_LENGTH = 1000;
        String[] names = new String[DATA_LENGTH];
        double[] scores = new double[DATA_LENGTH];
        double[] prices = new double[DATA_LENGTH];
        while (true) {
            System.out.print("[" + dataSize + "] ");
            String line = console.readLine();
            if (line == null || line.equals("quit")) break;
            StringTokenizer tokenizer = new StringTokenizer(line);
            String product = tokenizer.nextToken();
            double score = Double.parseDouble(tokenizer.nextToken());
            double price = Double.parseDouble(tokenizer.nextToken());
            names[dataSize] = product;
            scores[dataSize] = score;
            prices[dataSize] = price;
            dataSize += 1;
        }
        System.out.println("\n** Data has been entered now.\n");
        System.out.println("** We need to process it.\n");
        double highest = scores[0] / prices[0];
        for (int i = 0; i < dataSize; i++) {
            if (highest < scores[i] / prices[i])
                highest = scores[i] / prices[i];
        }
        System.out.println("** We then need to print the results.\n");
        for (int i = 0; i < dataSize; i++) {
            System.out.println(names[i] + " " + scores[i] + " " + prices[i]);
            if (highest == scores[i] / prices[i])
                System.out.println("** ");
            else System.out.println();
        }
    }
}
```
Can you show me the program live? I sure can:

frilled.cs.indiana.edu/java One
Hello, and welcome to the evaluator. Type quit to quit.
[0]% one 1 2
[1]% two 4 9
[2]% six 2 1
[3]% ten 3 3
[4]% quit
** Data has been entered now.
** We need to process it.
** We then need to print the results.
one 1.0 2.0
two 4.0 9.0
six 2.0 1.0 **
ten 3.0 3.0
Thank you for using our program!
frilled.cs.indiana.edu%

Looks like a fractions program to me... Ratios, they’re ratios.

Can you also print these ratios? Yes, but I will let you do that, it’s quite easy.

Well, how do you feel about your program? Actually I feel quite proud of being able to manipulate three arrays at the same time.

Tired, perhaps, you should feel. Proud. I am a real programmer, am I not?

If you say so. How do you call your arrays? They are called parallel arrays.

The i\textsuperscript{th} slice

\begin{itemize}
\item names[i]
\item prices[i]
\item scores[i]
\end{itemize}

... contains data that needs to be processed together.

Parallel arrays become a headache in larger programs, with thicker slices.

Are you sure?

The programmer must ensure that the arrays always have the same length and that each slice is filled with values that actually belong together. But it can get worse that that.

Now that I think of it, any method that operates on a slice must get all arrays as parameters, which is tedious even by my standards.
The remedy is simple.

Yes:
1. Look at the slice, and
2. Find the concept that it represents.

3. Then make the concept into a class:
   That exactly is it!
   
   ```java
   class Product {
       String name;
       double score;
       double price;
   }
   ```

   Then use an array of objects.
   Let me do that:
   ```java
   Product[] data = new Product[DATA_LENGTH]
   ```

   The program now has one array (of objects).
   I heard vectors are arrays of objects, which can grow and shrink as necessary.

   We’ll get to that in a second. Can you now eliminate the parallel arrays in our application?
   I sure can, and here it is:
   ```java
   import java.util.StringTokenizer;
   class Product {
       String name;
       double score;
       double price;
       Product(String n, double s, double p) {
           name = n; score = s; price = p;
       }
   }
   class Two {
       public static void main(String[] args) {
           System.out.println(
               "Hello, and welcome to the evaluator. Type quit to quit.");
           ConsoleReader console = new ConsoleReader(System.in);
           int dataSize = 0;
           final int DATA_LENGTH = 1000;
           Product[] data = new Product[1000];
           while (true) {
               Product p = Two.readFile(console, dataSize);
               if (p == null) break;
               data[dataSize] = p;
               dataSize += 1;
           }
           System.out.println("\n** Data has been entered now. ");
           System.out.println("** We need to process it.");
           double highest = data[0].score / data[0].price;
           for (int i = 0; i < dataSize; i++) {
               if (highest < data[i].score / data[i].price) {
                   highest = data[i].score / data[i].price;
               }
           }
       }
   }
   ```
```java
} 
System.out.println("** We then need to print the results.");
for (int i = 0; i < dataSize; i++) {
    Two.printProduct(data[i], highest);
}
System.out.println("\nThank you for using our program!");
}
public static Product readProduct(ConsoleReader console, int rank) {
    System.out.print("T" + rank + "%");
    String line = console.readLine();
    if (line == null || line.equals("quit")) return null;
    String name = new StringTokenizer(line);
    double score = Double.parseDouble(tokenizer.nextToken());
    double price = Double.parseDouble(tokenizer.nextToken());
    return new Product(name, score, price);
}
public static void printProduct(Product p, double highest) {
    System.out.print(p.name + " " + p.score + " " + p.price);
    if (highest == p.score / p.price)
        System.out.println(" **");
    else System.out.println();
}
```

The program now has a single array of Product objects, and we use static methods to split the program in several modules.

<table>
<thead>
<tr>
<th>1. The set of parallel arrays is replaced by a single array.</th>
<th>2. Each element in the resulting array corresponds to a slice in the set of parallel arrays.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Once you have this single concept available, it suddenly becomes much easier to give the program a better structure.</td>
<td>4. The program easily factors out methods for reading and printing objects.</td>
</tr>
</tbody>
</table>

Indeed. To really see the advantage of using objects instead of parallel arrays, consider the readProduct method in the program above.

How would you implement that method if you couldn’t rely on using a Product object? You would have to return three values: the name, price, and score of the next product.

And how can you return three values at the same time? I don’t know, can you?

Nope. In Java you can’t return more than one value in a method. But you can put three values in an object and return the object.

Objects are containers too. Of course, and that’s what they do best!
Now that we’ve seen arrays of objects, let’s look into arrays as object data.

Here’s a simpler example first:

Run this a few times.

Do you see how it works?

class Game {
    public static void main(String[] args) {
        System.out.println("Welcome to the Game.");
        Hand yours = new Hand();
        System.out.print("Here’s your hand: ");
        yours.show();
        Hand computer = new Hand();
        System.out.print("Here’s the computer’s: ");
        computer.show();
        if (yours.value() > computer.value()) {
            System.out.println("You win this time by " +
                (yours.value() - computer.value()) + 
                " point(s) ");
        } else if (yours.value() < computer.value()) {
            System.out.println("Computer wins this time by " +
                (computer.value() - yours.value()) + 
                " point(s).");
        } else {
            System.out.println("This game is tied: " +
                yours.value() + " - " +
                computer.value());
        }
    }
}

class Hand {
    public static final int SIZE = 5;
    int[] cards;
    Hand() {
        cards = new int[Hand.SIZE];
        for (int i = 0; i < cards.length; i++) {
            cards[i] = (int)(Math.random() * 52) + 1;
        }
    }
    void show() {
        for (int i = 0; i < cards.length; i++) {
            System.out.print(cards[i] + " ");
        }
        System.out.println(" = "+this.value());
    }
    int value() {
        int val = 0;
        // Add code here to calculate the value of the hand
    }
}

38 http://java.sun.com/products/jdk/1.2/docs/api/java/lang/Math.html
39 http://java.sun.com/products/jdk/1.2/docs/api/java/lang/Math.html#random()}
for (int i = 0; i < cards.length; i++) {
    val += cards[i];
} return val;
}

A Hand is a sequence of 5 numbers.
OK, let’s look at something a bit more involved.

What’s a polygon?
Something with a lot of knees if you know Greek.

A polygon is a closed sequence of lines.
To describe a polygon, you need to store the sequence of its corner points.

An array of Points!
Yes.

What is a Point?
Point2D.Double would work well for us.

http://java.sun.com/products/jdk/1.2/docs/api/java/awt/geom/Point2D.Double.html

What then is a Polygon?

class Polygon {
    Point2D.Double[] corners;
    int cornersSize;
    Polygon(int n) {
        corners = new Point2D.Double[n];
        cornersSize = 0;
    }
    void add (Point2D.Double p) {
        corners[cornersSize] = p;
        cornersSize += 1;
    }
}

We model a polygon as a class containing an array of points (the instance variable corners)
The class contains one constructor, which receives the size of the polygon,

...and a method, to add corners (i.e., points). Can you add a draw method?

I couldn’t, really, at this point.
Fine. Let’s then wait until we study applets.

Well, I think arrays are nice.
Isn’t it a nuisance that their size is fixed, though?

Yes, unfortunately you need to specify the size of an array when an array is allocated, even though the actual size may not be known at that time.
If you know that the array can never hold more than a certain number of elements, ...
...you can allocate a large array and partially fill it using a companion variable,...

...which can be used to remember how many elements are actually in the array.

You can also dynamically grow an array by ...

1. allocating a larger array,

2. shoveling the contents from the smaller array into the larger array, and then...

3. attaching the larger array to the array variable.

This is tedious and repetitive code. The Vector class automates this process.

http://java.sun.com/products/jdk/1.2/docs/api/java/util/Vector.html

A vector is an "array" of objects that grows automatically.

You add new elements at the end of the vector with the add method.

The Vector class needs to be imported from the same package as the string tokenizer.

The constructor of choice is the no-arg constructor, which gives you a vector of size 0.

As with arrays, vector positions start at 0.

The number of elements currently stored in a vector is obtained by the size method.

Let's give an example, before.

Let's store Strings and report them back.

Sounds good to me.

Here's the program.

```java
import java.util.*;

class Words {
    public static void main(String[] args) {
        Vector words = new Vector();
        BufferedReader console = new BufferedReader(System.in);
        System.out.println("Welcome, please enter text.");
        while (true) {
            System.out.print("Type > ");
            String line = console.readLine();
            if (line.equals("quit") || line.equals("")) break;
            StringTokenizer nizer = new StringTokenizer(line);
            while (nizer.hasMoreTokens()) {
                String token = nizer.nextToken();
                words.add(token);
            }
        }
        System.out.println("Thanks for using this program.");
        for (int i = 0; i < words.size(); i++) {
            System.out.println(" + i + ": " + words.elementAt(i));
        }
        System.out.println("Good bye!");
    }
}
```
Great, and here's how it runs.

I like \texttt{Vectors}, they seem to be quite handy.

```plaintext
frilled.cs.indiana.edu%javac Words.java
frilled.cs.indiana.edu%java Words
Welcome, please enter text.
Type> I am typing here lines of text.
Type> Once upon a time there lived a man by the name of Stanley.
Type> quit
Thanks for using this program.
```

0: I
1: am
2: typing
3: here
4: lines
5: of
6: text.
7: Once
8: upon
9: a
10: time
11: there
12: lived
13: a
14: man
15: by
16: the
17: name
18: of
19: Stanley.

Good bye!
```

Yes, and we'll talk more about \texttt{Vectors} later.

Great, I was going to ask you about that.
The Bald Soprano

Inheritance and the class extension mechanism.

You don’t realize it, but you’re constantly enjoying the benefits of science. For example, when you turn on the radio, you take it for granted that music will come out;

But do you ever stop to think that this miracle would not be possible without the work of scientists? That’s right: there are tiny scientists inside that radio, playing instruments.

A similar principle is used in automatic bank-teller machines, which is why they frequently say: "Sorry, out of service.” They’re too embarrassed to say: "Sorry, tiny scientist going to the bathroom.”

Speaking of banks and ATMs, let’s use the BankAccount class to study the class extension mechanism, or inheritance (in Java). Inheritance is a mechanism for enhancing existing, working classes.

If you
• need to implement a new class, and
• a class representing a more general concept is already available,

...then the new class can inherit from the existing class. For example, suppose you need to define a class SavingsAccount to model an account that pays a fixed interest rate on deposits.

You already have a class BankAccount ...and a savings account is a special case of a BankAccount.

Ask any tiny scientist! So, in this case, it makes sense to use the language construct of inheritance.

Here is the syntax for the class definition:

```java
class SavingsAccount extends BankAccount {
  <new methods>
  <new instance variables>
}
```

And the set union of features kicks in.
Exactly. In the SavingsAccount class definition you specify only new methods and instance variables.

All methods and instance variables of the BankAccount class are automatically inherited by the SavingsAccount class.

I see... Concatenation of blueprints (almost).

The more general class that forms the basis for inheritance is called the superclass.

That would be BankAccount.

The more specialized class that inherits from the superclass is called the subclass.

Here, this is SavingsAccount.

In Java, every class that does not specifically extend another class, ...

...extends the class Object.

Whoa!...that explains everything!

Yes. It's been a well kept secret until now.

The Object class has a small number of methods that make sense for all objects, .. such as the toString method that you can use to obtain a string that describes the state of an object, any object.

I remember the other day we had this code.

class Vehicle {
    String owner;
    Vehicle (String owner) {
        this.owner = owner;
    }
    public String toString() {
        return "I belong to: " + this.owner;
    }
    public static void main(String[] args) {
        Vehicle a = new Vehicle("Michael Jordan");
        System.out.println(a.toString());
    }
}

And we asked two questions about it.

First off, if you run it now, no mystery.

Yes. All's copacetic now. But do this:

class Vehicle {
    String owner;
    Vehicle (String owner) {
        this.owner = owner;
    }
    public String toString() {
        return "I belong to: " + this.owner;
    }
    public static void main(String[] args) {
        Vehicle a = new Vehicle("Michael Jordan");
```java
System.out.println(a);
}
}

Yes, that’s the minor mistery. The **toString** is invoked by default.

Very good. Now do this:

class Vehicle {
    String owner;
    Vehicle (String owner) {
        this.owner = owner;
    }
    public static void main(String[] args) {
        Vehicle a = new Vehicle("Michael Jordan");
        System.out.println(a.toString());
    }
}

Yes, that’s the major mistery. A **toString** is already there from **Object**.

Doesn’t work **that** well, but it’s there. And **Object** is responsible for providing it.

That’s called **inheritance**.

It’s a side-effect of the class extension mechanism (for efficiency of expression).

When you extend a set of features provide the name of the set your starting from (**extends**) followed by the list of features you’re adding.

One important reason for inheritance is **code reuse**. By inheriting from an existing class, you do not have to replicate the effort that went into designing and perfecting that class. For example ...

... when implementing the **SavingsAccount** class, you can rely on the **withdraw**, **deposit** and **getBalance** methods of the **BankAccount** class without touching them.

Let us see how our savings account objects are different from **BankAccount** objects. But before that let me just say this again: For example, ...

... when implementing the **SavingsAccount** class, you can rely on the

```
• withdraw,
• deposit and
• getBalance
```

methods of the **BankAccount** class

... without touching them.

OK. Let us now see how our savings account objects are different from **BankAccount** objects.

We will set an ...

```
• interest rate in the constructor,
• method to apply that interest periodically.
```
That is, in addition to the three methods that can be applied to every BankAccount...

...we now have an additional method, addInterest which will only work for the new type of SavingsAccounts.

These new methods and instance variables must be defined in the subclass.

```java
public class SavingsAccount extends BankAccount {
    double interestRate;
    public SavingsAccount (double rate) {
        this.interestRate = rate;
    }
    public void addInterest() {
        double interest;
        interest = this.getBalance() * this.interestRate / 100;
        this.deposit(interest);
    }
}
```

Given this definition, what is the structure of a SavingsAccount object (as far as fields go)?

Which is...interestRate.

It’s nice to be able to develop things in stages. Yes, we will have a longer example later.

Next we need to implement the new public void addInterest() instance method. We have in fact already implemented it, but we pretend not to have done, just to discuss it.

This method computes the interest due on the current balance, ...

...and then deposits that interest to the account.

Note how the addInterest() method calls the getBalance() and deposit() methods of the superclass (BankAccount).

```java
public class SavingsAccount extends BankAccount {
    double interestRate;
    public SavingsAccount (double rate) {
        this.interestRate = rate;
    }
    public void addInterest() {
        double interest;
        interest = this.getBalance() * this.interestRate / 100;
        this.deposit(interest);
    }
}
```

Thanks for emphasizing, I would have missed it.
You're welcome.

Let's now draw a picture to illustrate... ...what each type of object has and why:

```
Object
  BankAccount
    SavingsAccount
      toString()
        balance
          deposit(__)
            withdraw(__)
              getBalance()
                toString()
      interestRate
        addInterest()
          balance
            deposit(__)
              withdraw(__)
                getBalance()
      toString()
```

Boy, that looks good! I sure think so.

The class **SavingsAccount** extends the class BankAccount (which extends Object... ...by default). A SavingsAccount object is a special case of BankAccount, just as a BankAccount is a special kind of Object.

A special case has *more* features. When I define a variable collegeFund of type SavingsAccount how do you anticipate using it?

Here are all possible uses:

```
collegeFund . deposit(__);
collegeFund . withdraw(__);
collegeFund . getBalance();
```
```java
    collegeFund.addInterest();
```

When I define a variable `anAccount` of type `BankAccount` how do you anticipate using it?

Here are all possible uses:

```java
    anAccount.deposit(...);
    anAccount.withdraw(...);
    anAccount.getBalance();
```

One less.

Very good. Get ready for a subtle question now.

Hit me.

OK, here it goes.

Can you store a reference to a

```java
    SavingsAccount
```

object into an object variable of type `BankAccount`?

Like this?

```java
    BankAccount b = new SavingsAccount(10);
```

Yes. Can you do that?

You would never utilize `b` fully, ...but perhaps you don’t need that.

So the answer is: yes.

You can store the reference to a `SavingsAccount` object into an object variable of type `BankAccount`.

And the reason is that you’re just saying:

"I will not need the extra features that the class `SavingsAccount` is defining,..."

*I will just attempt to work with the features defined in `BankAccount` – that’s all I need in this particular case”*

*Almost* like casting a `double` to an `int`.

*Almost* but not exactly.

Yes, because the new feature is (still) there.

OK. Can you do the opposite?

```java
    SavingsAccount a = new BankAccount(...);
```

You mean *this*?

Yes.

The answer is: no.

Why?

Well, what’s the intended use of `a`?

I’d say: if I try to compute the added interest the object won’t have an adequate instance variable, nor the capability to do that. That is, 

```java
    a.addInterest()
```

does not make sense.
So we can't let that happen. And the compiler will complain.

Why doesn't it complain in the previous situation? Because in that situation we are only giving up on some 

*amenities*,

...which is fine with the compiler for as long as it's 

...whereas here we might ask for the *impossible*,

.. which the compiler *can't* accept, ...even if it's fine with us.

Therefore, So, going back to our original example,

```java
a.addInterest()
```

is what it wants to guard us against.

```java
SavingsAccount collegeFund = new SavingsAccount(10);
BankAccount anAccount = collegeFund;
Object anObject = collegeFund;
```

...these are all OK.

Now the three object references stored in

*collegeFund*, *anAccount*, and *anObject*,

...all refer to the same object, of type

*SavingsAccount*, that much is clear.

However, the object variable *anAccount* knows less

(The object only knows the truth).

Because *anAccount* is a variable of type

*BankAccount*, you can use it to refer to the

*deposit* and *withdraw* methods used to change

the balance of the actual *SavingsAccount* object.

You can't use the *addInterest* method, though. It is not a method of the *BankAccount* superclass.

You can't see it when you decide to ignore it. Exactly. And, of course, the variable *anObject* knows even less.

You can't even apply the *deposit* method to it. *deposit* is not a method of the *Object* class.

Why would anyone *want* to know less about an object and store a reference to it in a variable of the superclass's type?

For generality and uniformity.

Have any example? I have two of them.

Let's see the first one. Consider the *transfer* method which transfers money from one account into another.

```java
void transfer (BankAccount other, double amount) {
```
```java
this.withdraw(amount);
other.deposit(amount);
}
```

You can use this method to transfer money from one `BankAccount` to another,

<table>
<thead>
<tr>
<th>...and you can <em>also</em> use the method to transfer money into a <code>SavingsAccount</code>.</th>
<th>The <em>transfer</em> method expects a reference to a <code>BankAccount</code>, which it will use to deposit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any <code>SavingsAccount</code> object can do that too, so it can be passed as the first explicit argument to <code>transfer</code>.</td>
<td>The <em>transfer</em> method doesn’t actually know (or care, for that matter) that, in this case, <code>other</code> refers to an actual <code>SavingsAccount</code>.</td>
</tr>
</tbody>
</table>
| It knows only that `other` is a `BankAccount`, ...that is, that it can | ...that is, that it can
| | • deposit
| | • withdraw, and
| | • `getBalance`
| ...and it doesn’t need to know anything else. | Precisely. |

---

I liked your first example.

Thank you. I liked it too.

What’s the second example?

It involves arrays, but we need to discuss inheritance hierarchies first.

Very good, let’s do that.

Occasionally, it happens that you convert an object to a superclass reference then, later, you need to convert it back.

---

Suppose you captured a reference to a savings account in a variable of type `Object`:

```java
Object myObj = new SavingsAccount(10);
```

Or a pair of *blinders* *...of the type that’s used on skittish racehorses.*

If you put it on, *...you can only see what it lets you see.*

Much later, if you want to *...you can do that, *with care*. *

- add interest or
- deposit to the account,

The *object* still has *all* the features, *...you just need to put the right pair of binoculars on to see them.*

That’s called *casting*. *As long as you are absolutely sure that myObj really refers to a `SavingsAccount` object,*
...you can use the \textit{cast} notation to convert it back, like this:

\begin{verbatim}
SavingsAccount x = (SavingsAccount) myObj;
\end{verbatim}

What if you’re sure but \textit{wrong}?

If you are wrong, and the object doesn’t actually refer to a savings account, your program will throw an exception, and terminate.

You will see examples of casting soon now.

In real world, we often categorize concepts into \textit{hierarchies}. Hierarchies are frequently represented as trees,

...with the most general concepts at the root of the hierarchy, and the more specialized ones towards the branches.

I think I get that.

Let’s see an example in Java.

Suppose that we have more than just one extension to \texttt{BankAccount}.

Consider a \texttt{CheckingAccount} class that describes accounts with no interest,

...gives you a small number of free transactions per month,

...and charges a transaction fee for each additional transaction.

I can visualize that.

Good. All accounts have something in common.

They are all bank accounts with a balance and the ability to deposit money,

...and (within limits) to withdraw money.

This leads us to the following inheritance hierarchy:

\begin{tikzpicture}
  \node (b) {BankAccount};
  \node (s) [below left of=b] {SavingsAccount};
  \node (c) [below right of=b] {CheckingAccount};

  \draw [->] (b) -- (s);
  \draw [->] (b) -- (c);
\end{tikzpicture}

A simple, very basic class hierarchy.

Now suppose that you have 100 bank account objects, and half of them are checking accounts and the other half are...

...savings accounts? That’s plausible.

Can you keep them \textit{all} in an array?

Only if the array is declared as having an interest in (being concerned with describing) only their most general, common features.

Like this:

\begin{verbatim}
BankAccount[] a = new BankAccount[100];
\end{verbatim}

This strategy, in its most general form, is used by the \texttt{Vector} class.

Which makes use of arrays of \texttt{Objects}.
To store something in a Vector it must be of type Object. I mean: *that’s it*!

Not directly. But you can store a Rectangle

If you do, it will get stored as an Object.

```java
Vector v = new Vector();
v.addElement(new Rectangle(.,.,.));
(v.elementAt(0)).translate(.,.);
```

...it comes back as an Object and not as a Rectangle. So you will need to cast the reference to a Rectangle or it won’t work.

```java
Vector v = new Vector();
v.addElement(new Rectangle(.,.,.));
(Rectangle)(v.elementAt(0)).translate(.,.);
```

Whatever you want to do with it as a Rectangle, you need to cast it to a Rectangle from the Object that it comes back as.

In most situations of this kind, though, it is better to play it safe and test whether a cast will succeed, before carrying out the cast.

That’s right, it tests whether an object belongs to a particular class.

```java
if (anObject instanceof SavingsAccount) {
    // ... do savings account type of work
} else if (anObject instanceof CheckingAccount) {
    // ... do checking account type of work
} else {
    // ... in which case the tiny scientist reports an error
}
```

Is that all there is to it? Almost.

Can you give me a complete summary? Yes. Complete for all practical purposes.

Let’s start. The format, though, will a bit cruder.

I like that better. Unadulterated account follows, then.
We first defined class `Point`.

A `Point` has a position \((x, y)\).

```java
class Point {
    int x;
    int y;
}
```

These are the features of any `Point` object:

- an \( x \) coordinate, and
- a \( y \) coordinate

Together defining the *position* of any `Point`.

A `Pixel` is a `Point` with `Color`.

In Java this is easy to write:

```java
class Pixel extends Point {
    Color\(^{40}\) c;
}
```

The features of a `Pixel` are three:

- an \( x \) coordinate (which is an `int`)
- a \( y \) coordinate (which is an `int`)
- a `Color`, call it `color`

This set of features is the union between

- the features of a `Point` and
- the one new feature that class `Pixel` is defining

In other words:

```
Point = \{x, y\}
```

```
Point = \text{Point} \cup \{\text{color}\}
```

That is the resulting blueprint (for `Pixel`) is a putting together of the two descriptions.

That’s fine, but set union means that names should be kept distinct.

We’ll come back to this in a second.

How do we use `Point` and `Pixel`?

Nothing unusual. We use `new` and expect the blueprints to define the resulting structures.

\(^{40}\)http://java.sun.com/products/jdk/1.2/docs/api/java/awt/Color.html\#Color
// somewhere in a method...
Point a = new Point();
Pixel b = new Pixel();
a.x = 2;
a.y = -10;
b.x = 3;
b.y = 24;
b.color = Color.blue;

Notice that the Color class is defined in the java.awt package. Next we asked: have you ever seen a Horse?
The answer was: yes.
Can you describe a Horse?
The answer was: that's actually quite complicated.
OK, so fortunately we know what we're talking about:

    class Horse {
            // lots of features ...
    }

Next we asked: have you ever seen a Unicorn?
The answer was: no.
Can you describe a Unicorn though?
Everybody said: yes, that's easy.

Here's a picture to get the idea:
So a definition is almost immediate.

    class Unicorn extends Horse {
            Horn h;
    }

And we had the following situation:

- everybody had seen horses, but nobody felt it was easy to describe them
- nobody had seen unicorns, but everybody thought it was easy to describe them

That's because we factored out the Horse.
Now we said: let's write a play with horses and unicorns.
In our play we could have:

    Horse h = new Horse();
    Unicorn u = new Unicorn();
Both h and u are special kinds of binoculars. 
If you put them on you should see the features that their type is defining. 
So h.mane and u.mane make sense. 
So does u.horn but h.horn doesn’t. 
For this reason it’s not adequate to say:

\[
\text{Unicorn } g = \text{new Horse}();
\]

It is OK, however, to ignore some features, for the sake of being more general. 
From our description it follows that all Unicorns are Horses. 
That’s called polymorphism. 
So writing something like this is acceptable:

\[
\text{Horse } z = \text{new Unicorn}();
\]

You can never access the Horn with z but sometimes you don’t even need that. 
We could come up with the following similar example:

```java
class Shape {
    // two coordinates
}
class Circle extends Shape {
    // add a radius
}
class Rectangle extends Shape {
    // add a width and a height
}
class Triangle extends Shape {
    // add two other points relative to location
}
```

Why is this useful? 
If you want to create an array that can store

- Circles, 
- Triangles, and 
- Rectangles,

the only thing you can do that is by relying on their generality as Shapes.

\[
\text{Shape } p[] = \text{new Shape}[100];
\]

I’ve got room for 100 such shapes (circles, triangles, or rectangles). 
There’s no other way around it, as far as arrays are concerned. 

Now we want to explore the name collision problem. 
Consider this example:
class Horse {
    void neigh() { System.out.println("Horse: Howdy!"); }
}
class Unicorn extends Horse {
    void neigh() { System.out.println("Unicorn: Bonjour! "); }
}

Unicorn is listing a feature: neigh.
If Horse had not had it already listed things would've been easy.
But Horses already know how to neigh.
They say: "Howdy!".
So Unicorns redefine the feature by saying "Hello!" in French.
(Unicorns are from Paris, TX)?
That's called overriding.
The mechanism is that no matter how you look at a Unicorn,

• as the Unicorn that it is, or
• as the Horse that it is

you are guaranteed to obtain the French greeting out of it. Here's the proof:

```
frilled.cs.indiana.edu%cat Ionesco.java
class Horse {
    void neigh() { System.out.println("I am a Horse: Howdy!"); }
}
class Unicorn extends Horse {
    void neigh() { System.out.println("I am a Unicorn: Bonjour! "); }
}
class Ionesco {
    public static void main(String[] args) {
        Unicorn a = new Unicorn();
        Horse b = new Unicorn();
        a.neigh();
        b.neigh();
        Horse c = new Horse();
        c.neigh();
    }
}
frilled.cs.indiana.edu%javac Ionesco.java
frilled.cs.indiana.edu%java Ionesco
I am a Unicorn: Bonjour!
I am a Unicorn: Bonjour!
I am a Horse: Howdy!
```

That's all we need to know before we go into applets.
Utilities

Vectors, Hashtables, Leftovers.

First here’s a Selection Sort that we have developed in class.

class One {
    static void sort(int[] a) {
        for (int start = 0; start < a.length - 1; start++) {
            for (int j = start; j < a.length; j++) {
                if (a[start] > a[j]) { // sorting in ascending order
                    int temp = a[start];
                    a[start] = a[j];
                    a[j] = temp;
                }
            }
        }
    }
}

class One {
    public static void main(String[] args) {
        int[] numbers = new int[args.length];
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = Integer.parseInt(args[i]);
        }
        System.out.println("Here’s the initial array: ");
        One.show(numbers);
        System.out.println("Let me sort it in ascending order... ");
        One.sort(numbers);
        System.out.println("... Done\nHere it is sorted: ");
        One.show(numbers);
    }
}

Here’s how it runs:

575
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One 3 4 1 2 6 7
Here’s the initial array:
3 4 1 2 6 7
Let me sort it in ascending order...
... Done
Here it is sorted:
1 2 3 4 6 7
frilled.cs.indiana.edu%

To clarify the idea of scope of a variable let’s look at this example:

class Two {
    public static void main(String[] args) {
        int a = 1;
        {
            int b = 2;
            System.out.println("a: " + a);
            a = 4;  // change will be seen by the other block  
            System.out.println("b: " + b);
        }
        // System.out.println("b: " + b);
        {
            int b = 3;
            System.out.println("a: " + a);
            System.out.println("b: " + b);
        }
    }
}

I’ll let you experiment with it.

Here’s a second method of sorting: Bubble Sort.

class Three {
    static void sort(int[] a) {
        boolean done;
        do {
            done = true;
            for (int i = 0; i < a.length - 1; i++) {
                if (a[i] > a[i + 1]) {  // sort in ascending order
                    int temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                    done = false;
                }
            }
        } while (! done);
    }
    static void main(String[] args) {
        int[] numbers = new int[args.length];
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = Integer.parseInt(args[i]);
}
System.out.println("Here's the initial array: ");
Three.show(numbers);
System.out.println("Let me sort it in ascending order...");
Three.sort(numbers);
System.out.println("... Done\nHere it is sorted: ");
Three.show(numbers);
}
static void show(int[] a) {
    for (int i = 0; i < a.length; i++) {
        System.out.print(a[i] + " ");
    }
    System.out.println();
}
It runs just like the other one, but the mechanism is different.
The outer do-while loop is normally added at the end.
For this reason I find the following method much more intuitive.

class Four {
    static void sort(int[] a) {
        boolean done = true;
        for (int i = 0; i < a.length - 1; i++) {
            if (a[i] > a[i + 1]) { // sort in ascending order
                int temp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = temp;
                done = false;
            }
        }
        if (!done)
            sort(a);
    }
    static void main(String[] args) {
        int[] numbers = new int[args.length];
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = Integer.parseInt(args[i]);
        }
        System.out.println("Here's the initial array: ");
        One.show(numbers);
        System.out.println("Let me sort it in ascending order...");
        One.sort(numbers);
        System.out.println("... Done\nHere it is sorted: ");
        One.show(numbers);
    }
    static void show(int[] a) {
}
for (int i = 0; i < a.length; i++) {
    System.out.print(a[i] + " ");
}  
System.out.println();
}
}

Yes, recursion, in some cases, amounts to just a loop.
One could improve on the Bubble Sort technique above.

Here's an applet\(^{41}\) that illustrates the differences between three sorting methods:

- quick sort (not selection sort)
- regular bubble sort
- improved bubble sorting

The applet, I believe, is very exciting.
It shows how the methods work, and illustrates topics from chapter 15\(^{42}\).

Let's move on.

Last time we talked about
- inheritance and
- the class extension mechanism.

These are more or less the same topic.

We can illustrate them both by this example:

class Horse {
    int numberOfLegs();
    void fun() {
        System.out.println("I am a Horse.");
    }
}
class Unicorn extends Horse {
    void fun() {
        System.out.println("I am a Unicorn. Like a horse, but with a horn.");
    }
}
class Experiment {
    public static void main(String[] args) {
        Horse a = new Horse();
        System.out.println("A horse has " + a.numberOfLegs + " legs.");
        Horse b = new Unicorn();
        Horse c = new Unicorn();

\(^{41}\)http://www.cs.indiana.edu/classes/a348-dger/lectures/tsort/example1.html

\(^{42}\)What chapter, again?. This only means "algorithm".
System.out.println("A unicorn has " + c.number0fLegs + " legs.");
// Unicorn d = new Horse(); // Not allowed.
System.out.print("First test: ");
a.fun(); // a horse, of course
System.out.print("Second test: ");
c.fun(); // a unicorn, of course
System.out.print("Third test: ");
b.fun(); // what will this be?
}
}

That's what you need to know to understand applets.

It's also important to understand how Vectors work.

We will talk about Vectors in a little while.

Here's the example promised on developing things in stages:

frilled.cs.indiana.edu%cat Stages.java
class One {
    int add(int n, int m) {
        if (m == 0) return n;
        else return add(n+1, m-1);
    }
}
class Two extends One {
    int mul(int n, int m) {
        if (m == 1) return n;
        else return add(n, mul(n, m-1));
    }
}
class Three extends Two {
    int pow(int n, int m) {
        if (m == 0) return 1;
        else return mul(n, pow(n, m-1));
    }
}
class Calculator {
    public static void main(String[] args) {
        Three calc = new Three();
        int n = 3, m = 5;
        System.out.println(n + " + " + m + " = " + calc.add(n, m));
        System.out.println(n + " * " + m + " = " + calc.mul(n, m));
        System.out.println(n + " - " + m + " = " + calc.pow(n, m));
    }
}
frilled.cs.indiana.edu%

Here's a different, somewhat similar, example on interfaces.

frilled.cs.indiana.edu%cat Example.java
interface Multiplier {
    int mul(int n, int m);
}
class Alpha implements Multiplier {
    public int mul(int n, int m) {
        return n * m;
    }
}
class Beta implements Multiplier {
    public int mul(int n, int m) {
        int result = 0;
        for (int i = 0; i < m; i++)
            result += n;
        return result;
    }
}
class Gamma implements Multiplier {
    public int mul(int n, int m) {
        if (m == 1) return n;
        else return n + mul(n, m-1);
    }
}
class Example {
    public static void main(String[] args) {
        Alpha a = new Alpha();
        Beta b = new Beta();
        Gamma g = new Gamma();
        int n = 5, m = 3;
        System.out.println(n + " * " + m + " = " + a.mul(n,m) + " (by Alpha)"); // Output: 5 * 3 = 15 (by Alpha)
        System.out.println(n + " * " + m + " = " + b.mul(n,m) + " (by Beta)"); // Output: 5 * 3 = 15 (by Beta)
        System.out.println(n + " * " + m + " = " + g.mul(n,m) + " (by Gamma)"); // Output: 5 * 3 = 15 (by Gamma)
    }
}

frilled.cs.indiana.edu%javac Example.java
frilled.cs.indiana.edu%java Example
5 * 3 = 15 (by Alpha)
5 * 3 = 15 (by Beta )
5 * 3 = 15 (by Gamma)

An interface is an element of pure design. Now let's talk about Vectors.

What is a Vector? Here's a hint.

http://java.sun.com/products/jdk/1.2/docs/api/java/util/Vector.html
A Vector object responds to the following messages:

1. void addElement (Object obj)
   *Adds the object to the end of the vector, increasing its size by one.*

2. boolean contains (Object obj)
   *Returns true if the object is among the values stored in the vector, or false otherwise.*

3. Object elementAt (int index)
   *Returns the object at the specified index position.*

4. int indexOf (Object obj)
   *Returns the index position of the first instance of the object in the vector (if the object is in the vector) or -1 otherwise.*

5. int indexOf (Object obj, int index)
   *Same as above, but starts the search from the specified index.*

6. void insertElementAt(Object obj, int index)
   *Inserts the object at the specified index position, after shifting down objects at and below the insertion point.*

7. boolean isEmpty ()
   *Returns true if the vector contains no objects, or false otherwise.*

8. void removeElement (Object obj)
   *Removes the first occurrence of the specified object, and then shifts up objects at and below the deletion point.*

9. void removeElementAt(int index)
   *Deletes the object at the specified index position, and then shifts up objects at and below the deletion point.*

10. void setElementAt (Object obj, int index)
    *Replaces the existing object at the specified index position with the specified object.*

11. int size ()
    *Returns the number of objects currently stored in the vector.*

That’s only part of what a Vector does best!

To create a new Vector ...one should invoke a no-arg constructor.

Something like this: This creates a new (and empty) Vector object.

    Vector a = new Vector();

Can you define a class Vector? Here’s a reasonable start:
class Vector {
    Object[] localStorage = new Object[0];
    int size() {
        return localStorage.length;
    }
    void addElement(Object obj) {
        int currentLength = this.size();
        Object[] aux = new Object[currentLength + 1];
        aux[currentLength] = obj;
        for (int i = 0; i < currentLength; i++) {
            aux[i] = localStorage[i]; // transfer elements
        }
        localStorage = aux;
    }
    public String toString() {
        String returnString = "Vector of size " + size() + ": (";
        for (int i = 0; i < localStorage.length; i++)
            returnString += " + localStorage[i]; // delay printing...
        return returnString + ")";
    }
    public static void main(String[] args) {
        Vector v = new Vector();
        System.out.println(v); // convenient printing
        v.addElement(new Integer(2));
        System.out.println(v);
        v.addElement(new Integer(4));
        System.out.println(v);
        v.addElement(new Integer(6));
        System.out.println(v);
    }
}

What would this be? An operational model for java.util.Vector

"What I cannot create, I cannot understand." So we put one together, just for the fun of it.

Very good. Can I see this running? No.

What do you mean? Just kidding. Of course you can!

    tucotuco.cs.indiana.edu% javac Vector.java
    tucotuco.cs.indiana.edu% java Vector
Vector of size 0: ()
Vector of size 1: ( 2)
Vector of size 2: ( 2 4)
Vector of size 3: ( 2 4 6)
    tucotuco.cs.indiana.edu%

Now let’s look at the code line by line. It will be my pleasure.
class Vector {
    Object[] localStorage = new Object[0];

    I see you use an array of Objects inside an instance of my Vector class.
    OK, I should have made it private, but now it’s too late.

    This is the internal (local) storage that is available to every instance of type Vector,
    ...to every object that is instantiated with the new operator out of class Vector.

    I make a distinction between an array of size 0 and no array whatsoever (that is, a null pointer).
    And so do I.

    I start with an array of size 0 because I want my localStorage to always have a useful length to return (a null pointer returns an exception when asked about it’s length.)
    That was exactly my thinking.

    So I start with an array of size 0 for uniformity.
    And implementing size is a breeze.

        int size() {
            return localStorage.length;
        }

    Now let’s tell the story of addElement.
    OK, here we go:

    void addElement(Object obj) {

        This method will reveal our style of implementation.
        I will not worry about efficiency, I will rely heavily on Java’s automatic memory management facilities.

        It’s good to optimize one’s programs.
        Yes, but as Donald Knuth remarked once, "premature optimization is the root of all evil".

        Once my program works fine I can try to optimize it.
        But first, I need to make it work correctly.

        Correctness concerns us most at this stage.
        Yes. Let’s start. We first get the size of the vector (our local storage is always at capacity.)

        int currentLength = this.size();

        Would’ve worked just as well if we had talked to localStorage directly, right?
        Yes, but this way we are expressing ourselves in Vector terms.

        Then we get ready to store a new element,
        ...and allocate space that’s bigger with one than our current size.

        Object[] aux = new Object[currentLength + 1];

        Now place the object that needs to be added in the last position.
        Indeed, that’s what we said we’d be doing.

        aux[currentLength] = obj;
Transfer the objects from the local storage into the new array one by one.

```java
for (int i = 0; i < currentLength; i++) {
    aux[i] = localStorage[i];
}
```

Then call this new array our `localStorage`.

```
localStorage = aux;
```

<table>
<thead>
<tr>
<th>What will happen to the array that the <code>localStorage</code> variable’s previously pointing to?</th>
<th>Gone with the wind.</th>
</tr>
</thead>
<tbody>
<tr>
<td>That’s quite correct. And since no variable references it any longer,</td>
<td>...it will be collected by Java’s <code>garbage collector</code>.</td>
</tr>
</tbody>
</table>

```
}
```

Curly brace. End of story.

<table>
<thead>
<tr>
<th>That’s how we add a new element to a <code>Vector</code>. Removing an element should be similar.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The rest of my program only serves testing purposes. Yes, but it’s <em>still</em> instructive!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>If I ever place a <code>Vector</code> in a <code>String</code> context ...it will be this method that will <em>kick in</em>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let’s see how it works. I start by printing the type (vector) and the size.</td>
</tr>
</tbody>
</table>

```
public String toString() {
    String returnString = "Vector of size " + size() + ": (";
    returnString += " " + localStorage[i];
```

You’re not really *printing*, are you? No, just putting together a `String` representation that *will* eventually be printed (but not here).

The elements if any will be listed one by one

```
for (int i = 0; i < localStorage.length; i++)
    returnString += " " + localStorage[i];
```
...and the list will appear in between parens.

    return returnString + ");
  }

The main method creates a vector, and adds three elements to it.  We could have used Products, you know?

Yes, but this points out what you need to do if you want to store numbers.

Which are number holders.  You have to use wrapper classes.

    public static void main(String[] args) {
      Vector v = new Vector();
      System.out.println(v);
      v.addElement(new Integer(2));
      System.out.println(v);
      v.addElement(new Integer(4));
      System.out.println(v);
      v.addElement(new Integer(6));
      System.out.println(v);
    }

...and after each addition the Vector is printed.

Well, that was easy!  Easy and clear!

What's a HashTable?

    http://java.sun.com/products/jdk/1.2/docs/api/java/util/Hashtable.html

A HashTable is a generalized kind of Vector.  A Vector is a generalized kind of array.

The Vector is more general than an array because it grows and shrinks dynamically, and the elements don't have to be of the same type.

Both arrays and Vectors map indices (numbers) to elements contained.

They're sequences.  A HashTable is more general in that it stores associations between keys and values.

The keys and the values could be any objects.  For example Strings.

If you wanted to store associations of this kind:

    username --> password

using Vectors or arrays

Hashtables are built just for that!  They're sometimes called associative arrays.

Once the association is built, ...we want to use it.
So, given a **username**, we need to... ...look up the **password** that is *associated* with it.

That’s what **HashTables** do best! Here’s an example that uses a HasTable object to interactively manage a set of Strings.

```java
import java.util.*;

class Set {
    public static void main(String[] args) {
        HasTable h = new HasTable();
        ConsoleReader b = new ConsoleReader(System.in);
        System.out.println("Ready for input.");
        while (true) {
            String command = "<cmd>", argument = "<arg>";
            if (s.hasMoreTokens()) command = s.nextToken();
            if (s.hasMoreTokens()) argument = s.nextToken();
            if (command.equals("add")) {
                h.put(argument, argument);
                System.out.println("OK");
            } else if (command.equals("delete")) {
                h.remove(argument);
                System.out.println("OK");
            } else if (command.equals("list")) {
                Enumeration e = h.keys();
                while (e.hasMoreElements()) {
                    System.out.println(" : " + e.nextElement());
                }
            } else if (command.equals("contains")) {
                System.out.println(h.containsKey(argument));
            } else if (command.equals("quit")) {
                System.exit(0);
            } else {
                System.out.println("Unknown command: " +
                                 command + " " + argument);
            }
        }
    }
}
```

This is an **interactive** program. Yes, you need to type commands into it.

Can I see it running? Here’s the program (in invisible blue) talking to a user.

```
frilled.cs.indiana.edu%java Set
Ready for input.
add one
OK
add two
```
OK
deg

list
deg

:one
deg

two
add three
OK
deg

list
deg

:one
deg

two
tree
add three
OK
deg

list
deg

:one
deg

two
deg
contains three
deg
false
deg
contains two
deg
true
deg
quit
deg

frilled.cs.indiana.edu%
That means arrays of arrays of arrays.

Yes, arrays of (arrays of arrays).

And n-dimensional arrays will be stored as arrays of (n-1)-dimensional arrays.

What a recursive definition.

We won’t use more than two-dimensions.

Let’s take a closer look at them though: because a two-dimensional array is really an array of the row arrays,

...the number of rows is:

And the number or columns?

\[
\text{int nRows = matrix.length;}
\]

...the number of elements in each row can vary.

Because a two-dimensional array is really an array of the row arrays,

If it doesn’t, the number of columns is the same as the length of the first row:

\[
\text{int nCols = matrix[0].length;}
\]

Can I see an example of a two-dimensional array that has rows of various lengths?

Yes, you can quickly create one with an array initializer:

\[
\text{int[][] b = \{ \{1\},}
\]
\[
\{2, 3\},
\]
\[
\{4, 5, 6\},
\]
\[
\{7, 8, 9, 10\}
\}
\]

Can you do the same thing without it?

Sure, I thought you’d ask:

\[
\text{int[][] b = new int[4] [];}
\text{int count = 1;}
\text{for (int i = 0; i < 4; i++) \{
}\text{b[i] = new int[i + 1];}
\text{for (int j = 0; j <= i; j++) \{
}\text{b[i][j] = count;}
\text{count += 1;}
\}\}
\]

I see you have to work harder.

Yes, first you allocate space to hold four rows.

You indicate that you will manually set each row by leaving the second array index empty.

Then you need to allocate (and fill) each row separately.

You can access each array element as \[b[i][j]\], ...

...but you must be careful that \(j\) is less than \(b[i].length\) as illustrated below:
class Testing {
    public static void main(String[] args) {
        int[][] b = new int[4][];
        int count = 1;
        for (int i = 0; i < 4; i++) {
            b[i] = new int[i + 1];
            for (int j = 0; j <= i; j++) {
                b[i][j] = count;
                count += 1;
            }
        }
        for (int i = 0; i < b.length; i++) {
            for (int j = 0; j < b[i].length; j++) {
                System.out.print(b[i][j] + " ");
            }
            System.out.println();
        }
    }
}

Naturally, such "ragged" arrays are not very common. \hspace{.5cm} Except when you need them.
\begin{tabular}{p{4\textwidth}p{4\textwidth}}
Then they become very natural. & And they're no longer "ragged". \\
\hline
Exactly, because they fit that problem & \ldots perfectly.
\end{tabular}
Applets

Here’s an overview\textsuperscript{43} of applets from the Java tutorial\textsuperscript{44}.

Here’s the basic framework for an applet:

```java
import java.applet.*;
import java.awt.*;
public class Simple extends Applet {
    public void paint(Graphics g) {
        for (int i = 0; i < 20; i++) {
            g.drawString("This is a test.", 20 + 10 * i, 20 + 10 * i);
        }
    }
}
```

Here’s what you do to install it:

```bash
frilled.cs.indiana.edu/%emacs Simple.java
frilled.cs.indiana.edu/%javac Simple.java
frilled.cs.indiana.edu/%emacs simple.html
frilled.cs.indiana.edu/%cat simple.html
<html><head><title>Testing a simple applet</title></head><body bgcolor=white>
    <applet code=Simple width=200 height=200>
    </applet>
</body></html>
frilled.cs.indiana.edu/%appletviewer simple.html
```

And here it is in action\textsuperscript{45}.

Let’s now clarify how this works.
By extending a class we extend our original description, and specify additional features.

<table>
<thead>
<tr>
<th>An object of the original class has the features listed in the original class.</th>
<th>An object of the extended class has all the features listed in the original class and the class that extends it.</th>
</tr>
</thead>
<tbody>
<tr>
<td>That’s what we started from, anyway.</td>
<td>Just put on a pair of blinkers.</td>
</tr>
<tr>
<td>This is the reason for which we say that if class B extends class A every object of type B is also an object of type A.</td>
<td>Objects of type B are more specialized.</td>
</tr>
<tr>
<td>But an object of type A is not of type B (objects of type B have additional features, listed in the description of class B).</td>
<td></td>
</tr>
<tr>
<td>Another way to remember this is in this way:</td>
<td>...if Goalkeeper extends Player we can use an object of type Goalkeeper everywhere we need to use a Player</td>
</tr>
<tr>
<td>...any Player</td>
<td>...but not the other way around.</td>
</tr>
</tbody>
</table>

This is one kind of polymorphism.

<table>
<thead>
<tr>
<th>In our discussion in class we have distinguished between:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• object references</td>
</tr>
<tr>
<td>• actual objects</td>
</tr>
</tbody>
</table>

Both have a type.

| Objects are anonymous, and we refer to them by names (or object references). |

Extending classes is a very simple concept.

<table>
<thead>
<tr>
<th>Creating composite objects from composite descriptions is as easy as putting the two descriptions together,</th>
</tr>
</thead>
</table>

...unless we use the same names for different features (class members of the same kind: data, or methods) in the two descriptions.

<table>
<thead>
<tr>
<th>When the name of two methods collide overriding happens.</th>
</tr>
</thead>
</table>

Which method gets invoked is determined through dynamic method lookup.

| When variables are involved it’s called shadowing, and we haven’t studied it much. |

A few examples will help clarify these concepts.

Consider this:

class Player {
    void fun() {
        System.out.println("Having fun as a Player.");
    }
}
class Goalkeeper extends Player {
    void fun() {
        System.out.println("Having fun as a Goalkeeper.");
    }
}

public class One {
    public static void main(String[] args) {
        System.out.println("Welcome to Program One.");

        Player meola = new Goalkeeper();
        meola.fun();

        Goalkeeper higuita = new Goalkeeper();
        higuita.fun();

        ((Player)higuita).fun();
    }
}

Here's what you get when you run the program:
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
 Welcome to Program One.
 Having fun as a Goalkeeper.
 Having fun as a Goalkeeper.
 Having fun as a Goalkeeper.
frilled.cs.indiana.edu%

...that it's the type of the object (and not that of its reference) that really matters.

You see then,
If either A or B do not define fun() then there's no overriding involved, and no dynamic method lookup is involved.

But it is instructive to cover the three alternatives to the situation described above.

Case 2. Player does not define fun().

Casting an object of type Goalkeeper to type Player 
and asking for fun() 
...will get you into trouble early (as early as compile time).

Case 3. Goalkeeper does not define fun().

Objects of type Goalkeeper inherit fun() from being Players (but they don't have own fun).

Case 4: Neither Player nor Goalkeeper define fun().

In that case there's really nothing to talk about.

To summarize, looking up a method for invocation involves taking into account the class of the object and the class of its reference (casting included here). The object reference's class determine the method unless the object's class also defines a method with the same name, in which case that's the one that will be invoked.
Here's another example:

class Player {
    void fun() {
        System.out.println("Having fun as a Player.");
    }
}

class Defender extends Player {
}

class Fullback extends Defender {
    void fun() {
        System.out.println("Having fun as a Fullback.");
    }
}

public class Two {
    public static void main(String[] args) {
        System.out.println("Welcome to Program Two");
        Defender dooley = new Defender();
        dooley.fun(); // fun inherited (Player)
        Fullback baresi = new Fullback();
        baresi.fun(); // Fullback has its own fun
        Defender tresor = new Fullback();
        tresor.fun(); // type Fullback overrides the fun that
                       // Defender inherited from Player
        Player rijken = new Fullback();
        rijken.fun(); // the fun defined in Fullback is used
                       // over the one defined in Player, since
                       // the type of the object rijken points
                       // to (that class) has own fun
    }
}

Let's run it.

    frilled.cs.indiana.edu%java Two
    Welcome to Program Two
    Having fun as a Player.
    Having fun as a Fullback.
    Having fun as a Fullback.
    Having fun as a Fullback.
    frilled.cs.indiana.edu%

    Predictable. I think so.
Once you override a function the only way you can get to it is through a super reference. You can’t get to it from outside the class (through casting).

Overriding is a stronger notion than shadowing. If you shadow a variable you can still get to it from outside by casting.

But we don’t want to talk about shadowing here. We never used it this semester.

We have used overriding. We redefined paint.

Let’s simulate how paint works. Indeed, let’s analyze that.

Before we do that, let’s make two final points. One refers to super.

super is a kind of casting. It allows us to put one pair of blinkers right away.

this is a reference to the current object. ...of the same type as the object’s class.

And now the program. Can you annotate it and explain what happens?

class Frame {
    protected String myGC = "The Graphics Context from class Frame";

    protected int width, height;
    protected boolean visible;

    protected void resize(int w, int h) {
        setSize(w, h);
        refresh();
    }
    protected void refresh() {
        paint(myGC);
    }
    protected void setVisible(boolean tF) {
        visible = tF;
        paint(myGC);
    }
    protected void setSize(int w, int h) {
        width = w;
        height = h;
    }

    public void paint(String gc) {
        System.out.println("Frame: I use\n" + gc + " \nto draw my images.");
    }
}
public class Picture extends Frame {

public void paint(String gc) {
    System.out.println("Picture: I use\n " + gc + " \n to draw my images.");
}

public static void main(String[] args) {
    Frame f = new Picture(); // you
    f.setSize(100, 200); // see this
    f.setVisible(true); // in many places...
    user(f); // you never ever see this
            // but you know it happens
}

private static void user(Frame f) {
    f.resize(200, 400);
    // minimal interaction by the user simulated here
}

If you understand how the code above works then

- you understand overriding of methods
- you understand why we need to override paint
  when doing graphics
- you understand how paint gives you access to
  a graphics context

Also note: protected is like private, but allowing
inheritance of the variable or method.

I believe you, but that doesn’t tell me much right now.
Still, I will keep it in the back of my mind.

Now we have the following examples, for you to practice with them:

1. A Simple Applet
   Create a file One.html with the following contents.

   <HTML>
   <HEAD>
   <TITLE>Applet One</TITLE>
   </HEAD>
   <BODY>
   <P>Here is my <U>first applet</U>. </P>
   <APPLET CODE="One.class" WIDTH=300 HEIGHT=300>
   </APPLET>
   <P> Its title is <I>"Picasso in disbelief"</I>. </P>
   </BODY>
   </HTML>
Create a file One.java with the following contents.

```java
import java.applet.Applet;
import java.awt.*;
public class One extends Applet {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        Rectangle a = new Rectangle(5, 10, 20, 30);
        g2.draw(a);
        a.translate(15, 25);
        g2.draw(a);
    }
}
```

Compile One.java and look at One.html with either Netscape or an appletviewer.

2. Graphical Shapes

Create a file Two.html with the following contents.

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Applet Two</title>
</head>
<body>
<p>Here is my &lt;u&gt;second applet&lt;/u&gt;.</p>
&lt;applet code="Two.class" width=300 height=300&gt;
</applet>
<p>Its title is &lt;i&gt;"Graphical Shapes"&lt;/i&gt;.</p>
</body>
</html>
```

Create a file Two.java with the following contents.

```java
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;
public class Two extends Applet {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        Ellipse2D.Double m = new Ellipse2D.Double(5, 10, 150, 80);
        g2.draw(m); // an ellipse
        int x = 90;
        int y = 90;
        int diam = 70;
        Ellipse2D.Double q = new Ellipse2D.Double(x, y, diam, diam);
        g2.draw(q); // a circle
        Line2D.Double segment = new Line2D.Double(15, 150, 55, 90);
        g2.draw(segment); // a line
        Point2D.Double start = new Point2D.Double(20, 155);
        Point2D.Double stop = new Point2D.Double(60, 95);
        segment = new Line2D.Double(start, stop);
    }
```

g2.draw(segment); // another line
}

 Compile Two.java and look at Two.html with either Netscape or an applet viewer.

3. Color and Shapes

Create a file Three.html with the following contents.

<HTML>
<HEAD>
<TITLE>Applet Three</TITLE>
</HEAD>
<BODY>
<P>Here is my <U>third</U> applet</P>
<APPLET CODE="Three.class" WIDTH=300 HEIGHT=300>
</APPLET>
<P>Its title is <I>Colored Graphical Shapes</I></P>
</BODY>
</HTML>

Create a file Three.java with the following contents.

import java.applet.*;
import java.awt.*;
import java.awt.geom.*;
public class Three extends Applet {
    public void paint (Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        Color magenta = new Color(1.0F, 0.0F, 1.0F);
        g2.setColor(magenta);
        Ellipse2D.Double m = new Ellipse2D.Double(5, 10, 150, 80);
        g2.draw(m); // an ellipse
        g2.fill(m); // fill it with the current color
        // magenta is still current color here
        int x = 90;
        int y = 90;
        int diam = 70;
        g2.setColor(Color.red); // predefined color
        Ellipse2D.Double q = new Ellipse2D.Double(x, y, diam, diam);
        g2.draw(q); // a circle
        g2.fill(q); // fill it with the current color
        g2.setColor(Color.blue); // another predefined color
        Line2D.Double segment = new Line2D.Double(15, 150, 55, 90);
        g2.draw(segment); // a line
        Point2D.Double start = new Point2D.Double(20, 155);
        Point2D.Double stop = new Point2D.Double(60, 95);
        segment = new Line2D.Double(start, stop);
        g2.draw(segment); // another line
        // current color is blue here
    }
}
Compile Three.java and look at Three.html with either Netscape or an appletviewer.

4. Text in Applets

Create a file Four.html with the following contents.

```html
<html>
<head>
<title>Applet Four</title>
</head>
<body>
<p>Here is my <u>fourth</u> applet.</p>
<applet code="Four.class" width=300 height=300>
</applet>
<p>Its title is <i>"Applet Applet"</i>.</p>
</body>
</html>
```

Create a file Four.java with the following contents.

```java
import java.applet.Applet;
import java.awt.*;
public class Four extends Applet {
  public void paint (Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    g2.drawString("Applet", 50, 160);
    final int HUGE_SIZE = 36; // font point size
    String message = "Applet"; // actual text
    // create Font object then call setFont on it
    Font hugeFont = new Font("Serif", Font.BOLD, HUGE_SIZE); // point size
    g2.setFont(hugeFont); // now g2 will write that way
    g2.setColor(Color.green); // set color to predefined green
    g2.drawString(message, 50, 100); // then write string
  }
}
```

Compile Four.java and look at Four.html with either Netscape or an appletviewer.

5. Simple Drawing

Create a file Five.html with the following contents.

```html
<html>
<head>
<title>Applet Five</title>
</head>
<body>
<p>Here is my <u>fourth</u> applet.</p>
<applet code="Five.class" width=300 height=300>
</applet>
<p>Its title is <i>"Happy Hour"</i>.</p>
```
Create a file Five.java with the following contents.

```java
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;
public class Five extends Applet {
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        Ellipse2D.Double e1 = new Ellipse2D.Double(75, 40, 30, 70);
        Ellipse2D.Double e2 = new Ellipse2D.Double(115, 40, 30, 70);
        Ellipse2D.Double c1 = new Ellipse2D.Double(85, 85, 15, 15);
        Ellipse2D.Double c2 = new Ellipse2D.Double(125, 85, 15, 15);
        Ellipse2D.Double n = new Ellipse2D.Double(55, 120, 110, 25);
        Arc2D.Double m =
            new Arc2D.Double(-40, -120, 300, 300, 230, 80, Arc2D.OPEN);
        g2.draw(e1);
        g2.draw(e2);
        g2.fill(c1);
        g2.fill(c2);
        g2.draw(n);
        g2.draw(m);
    }
}
```

Compile Five.java and look at Five.html with either Netscape or an appletviewer.
What comes next is your: A201/A597 LAB ASSIGNMENT TEN

1. Take a look at the pictures below.
2. Choose one that you like.
3. Write an applet that draws that picture.
More notes from the past: one\textsuperscript{46}, two\textsuperscript{47}.

\textsuperscript{46}http://www.cs.indiana.edu/classes/a201/spr2002/labs/eleven/simple.html
\textsuperscript{47}http://www.cs.indiana.edu/classes/a348/t540/lectures/Two.html
Today in class I would like to discuss this code:

```java
import java.applet. *
import java.awt. *
import java.awt.event. *
public class Game extends Applet {
    int i = 0;
    public void paint(Graphics g) {
        this.i = this.i + 1;
        System.out.println("Paint called: " + i);
    }
    public void init() {
        Umpire ump = new Umpire();
        this.addMouseMotionListener(ump);
    }
}
class Umpire implements MouseMotionListener {
    // wearing the uniform...
    public void mouseDragged(MouseEvent e) {
        System.out.println("Ha! You're dragging the mouse.");
    }
    public void mouseMoved(MouseEvent e) {
        System.out.print( "Mouse seen being moved at: (");
        System.out.println( e.getX() + ", " + e.getY() + ")");
    }
}
```

This goes with the following HTML file:

```html
<html>
<head>
    <title>Game Applet One</title>
</head>
<body bgcolor=white>
    <applet code="Game.class" width=300 height=300>
    </applet>
</body>
```
We’d like to understand this code very well. When this is done we will continue with the notes for tomorrow. I include, however, some more information on applets, that we may be going over in class.

We start with the life cycle of an applet: there are four methods in the Applet class that give you the framework on which you build any serious applet:

- init()
- start()
- stop()
- destroy()

We want to show you when these methods are called and what code you should place into them.

1. init()

   This method is used for whatever initializations are needed for your applet. This works much like a constructor - it is automatically called by the system when Java launches the applet for the first time. Common actions in an applet include processing PARAM values and adding GUI components.

   Note: applets can have a default constructor, but it is customary to perform all initialization in the init method, instead of the default constructor.

2. start()

   This method is automatically called after Java calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages. This means that the start method can be called repeatedly, unlike the init method. For this reason, put the code that you want executed only once in the init method, rather than in the start method. For example the start method is where you usually restart a thread for your applet, for example, to resume an animation.

   If your applet does nothing that needs to be suspended when the user leaves the current Web page, you do not need to implement this method (or the stop method discussed next).

3. stop()

   This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet. The purpose is to give you a chance to stop a time-consuming activity from slowing down the system when the user is not paying attention to the applet.

   Your should not call this method directly. If your applet does not perform animation, play audio files, or perform calculations in a thread, you do not usually need to use this method.

4. destroy()

   Unlike the finalize method for objects, Java is guaranteed to call this method when the browser shuts down normally. Since applets are meant to live on an HTML page, you do not need to worry about destroying the panel. This will happen automatically when the browser shuts down. What you do need to put in the destroy method is the code for reclaiming any non-memory dependent resources such as window handles that you may have consumed. (Of course Java calls stop method before calling the destroy method if the applet is still active).
Of the four the most important (customizable) by far is `paint()`. Here's an investigation into how `paint()` works:

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class One extends Applet {
    int i = 0;
    public void paint(Graphics g) {
        i += 1;
        System.out.println("paint has been called: " + i + " times.");
    }
}
```

And use a basic HTML file to dispatch the applet.

```html
<html>
<head>
    <title>All eyes on the mouse!</title>
</head>
<body>
    <applet code=One.class height=300 width=300>
    </applet>
</body>
</html>
```

Now let's make `paint` work harder.

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class One extends Applet {
    int i = 0;
    public void paint(Graphics g) {
        i += 1;
        System.out.println("paint has been called: " + i + " times.");
        int width = getWidth(), height = getHeight();
        for (int line = 0; line < height; line += 3) {
            g.drawLine(0, line, width, line); // horizontal line
        }
    }
}
```

Now let's introduce some colors, and let's show them.

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class One extends Applet {
    int i = 0;
    Color[] c = { Color.red, Color.yellow,
```
import java.awt.*;
import java.awt.event.*;

public class One extends Applet {
    int i = 0;
    Color[] c = { Color.red , Color.yellow,
                 Color.blue , Color.pink ,
                 Color.magenta, Color.green ,
                 Color.black , Color.white }
    }

    public void paint(Graphics g) {
        i += 1;
        System.out.println("paint has been called: " + i + " times.");
        int width = getWidth(), height = getHeight();
        for (int line = 0; line < height; line += 3) {
            g.setColor(c[i] % c.length]);
            g.drawLine(0, line, width, line);
        }
    }
}

Now it is easy to prove that paint is actually optimized. It only refreshes that part of the screen that has just become visible.

Now we add circles.

    import java.awt.*;
    public class Circle {
        int x, y, radius;
        public Circle(int x, int y, int radius) {
            this.x = x; this.y = y; this.radius = radius;
        }
        public void moveTo(int x, int y) { // x, y coordinates of center
```java
    this.x = x - radius;
    this.y = y - radius;
}
public void draw(Graphics g) {
    g.drawOval(x, y, 2 * radius, 2 * radius);
}
}

And we draw them.

```java
    import java.applet.*;
    import java.awt.*;
    public class One extends Applet {
        Circle c = new Circle(20, 20, 10);
        public void paint(Graphics g) {
            c.draw(g);
        }
    }
}

Now let's pay attention to mouse movement.

```java
    import java.applet.*;
    import java.awt.*;
    import java.awt.event.*;
    public class One extends Applet implements MouseMotionListener {
        Circle c = new Circle(20, 20, 10);
        public void init() {
            addMouseMotionListener(this);
        }
        public void paint(Graphics g) {
            c.draw(g);
        }
        public void mouseMoved(MouseEvent e) {
            int x = e.getx(), y = e.gety();
            c.moveTo(x, y);
            repaint();
        }
        public void mouseDragged(MouseEvent e) {
        }
    }
}

And notice that you can leave the circle behind while dragging.

And here's one last example (which is still instructive, I believe).

This is a simple applet whose purpose is to draw:

```java
    import java.applet.*;
    import java.awt.*;
    import java.util.*;
```
public class ManyShapes extends Applet
{
    // this method overrides the paint method from the Applet class
    public void paint(
        Graphics g // the Graphics context to draw with
    )
    {
        // create a new number generator
        Random r = new Random();

        // draw 10000 shapes }
        for(int i = 0; i < 10000; i++)
        {
            // generate random values for our shape
            int x = r.nextInt()%300;
            int y = r.nextInt()%300;
            int width = r.nextInt()%300;
            int height = r.nextInt()%300;

            // set a random color
            g.setColor(new Color(r.nextInt()));

            // generate a positive number between 0 and 4
            int n = Math.abs(r.nextInt()%5);

            // draw a shape based on the value of n
            switch(n)
            {
                case(0):
                    g.draw3DRect(x, y, width, height, true);
                    break;
                case(1):
                    g.drawRect(x, y, width, height);
                    break;
                case(2):
                    g.draw0val(x, y, width, height);
                    break;
                case(3):
                    g.fillRect(x, y, width, height);
                    break;
                case(4):
                    g.fill0val(x, y, width, height);
                    break;

                // this shouldn’t happen; but if it does,
                // print a message
                default:
                    System.out.println("Invalid case: " + n);
                    break;
            } // switch
Here's the HTML file:

```html
<html>
<head>
    <title>ManyShapes</title>
</head>
<body>
    <hr>
    <applet code=ManyShapes.class
            width=300
            height=300>
    </applet>
    <hr>
</body>
</html>
```

**EXERCISES**

- Compile and run this program.
- Briefly describe how the program works.
- Visit the APIs of the new classes used in this program.
- How many kinds of shapes is the program drawing? In how many colors?
- Can you enhance the program to draw other kind of shapes as well?
- Can you also improve on the number of colors?
# Umpires

*Applets. Events. Listeners. (Again).*

<table>
<thead>
<tr>
<th>Let’s write an applet.</th>
<th>We’ve done that before.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well, OK, here it is:</td>
<td></td>
</tr>
<tr>
<td>public class MouseSpyApplet extends Applet {</td>
<td></td>
</tr>
<tr>
<td>public void init() {</td>
<td></td>
</tr>
<tr>
<td>System.out.println(&quot;The applet is being initialized... &quot;);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>public void start() {</td>
<td></td>
</tr>
<tr>
<td>System.out.println(&quot;Applet is now being started... &quot;);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>Not a very complicated applet...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Yes, and ignore its name for now.</th>
<th>I will. Do we need an HTML file?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Of course, but you can supply that yourself.</td>
<td>Fair enough.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Today we will talk about <em>event-driven programming</em>.</th>
<th>That’s a big shift in the paradigm we have been using so far.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes: the user is in <em>total control</em>.</td>
<td>And will we be using the <code>MouseSpyApplet</code> applet that we wrote above, to start studying event-driven programming?</td>
</tr>
<tr>
<td>Yes. Our first goal will be to let the user click with the mouse in the applet, and report what’s happening.</td>
<td>How are we going to do that?</td>
</tr>
<tr>
<td>The mouse will act as an <em>event source</em>.</td>
<td>When the user works with it it will produce events.</td>
</tr>
</tbody>
</table>
The events produced by the mouse are objects. They are of type `MouseEvent`, a class that is defined in the package `javax.swing.event`.

So now you know what to look for, also what to import in your programs.

Is this the only event that could be produced in event-driven programming?

No, there are many kinds of events. `MouseEvent` is just one type.

Yes. Here’s the big picture:

I think I need to understand this picture.

Yes. In the beginning there is an event source. That’s the mouse for us.

Indeed, and many things could be event sources. But this is where it all starts: the event source.

Yes, the user works with it. And we gave it a number: 1.

Then, we need to listen to the event source, to know when an event happens. So that we can take action, if needed.

Listeners are also objects in Java. Do you need to create (or define) them, or do you just import them or how do you get one?

You need to define them. Why?

Because they contain the part that does the action. OK, then I will probably also have to create them, right?
Yes, to describe the object you need to write a class description.

    class MouseSpy {
    }

Why can’t you name it MouseListener?  I’ve heard the name is taken?

By whom?  An interface is called that way.

What’s an interface?  And what interface is that?  An interface is an element of pure design.

Yes, that’s true.  But let’s come up with a more comprehensive definition.

It’s going to be longer.

No problem, I’ll start it: the fundamental units of design in Java are the public methods that can be invoked on objects.

Interfaces are a way to declare a type consisting only of abstract methods

...that is, methods with no body,

...and constants, enabling any implementation to be written for those methods.

An interface is an expression of pure design...

...and I said it first,

...while a class is a mix of design and implementation.

A class can implement the methods of an interface in any way that the designer of the class chooses.

An interface has thus many more possible implementations than a class.

Can I see an example?

java.awt.event.MouseListener?  Yes, that’s the one that took the name.

Here it is:

    public interface MouseListener extends EventListener {
        public void mouseClicked (MouseEvent e);
        public void mousePressed (MouseEvent e);
        public void mouseReleased (MouseEvent e);
        public void mouseEntered (MouseEvent e);
        public void mouseExited (MouseEvent e);
    }

Is that it?

That’s it.  And it’s defined in...
### The java.awt.event Package

...the java.awt.event package. Now, how do we work with this?

If you want to be a MouseEvent listener you need to implement the interface.

I don’t, but MouseSpy does.

```java
class MouseSpy implements MouseListener {

}
```

Ah, so you say implements instead of extends...

And you need to import the interface too.

OK, how about that.

```java
import java.awt.event.*;
class MouseSpy implements MouseListener {
    public void mouseClicked ((MouseEvent e) { }
    public void mousePressed (MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered (MouseEvent e) { }
    public void mouseExited  (MouseEvent e) { }
}
```

Looks better but when you try to compile it it won’t.

I need to provide an implementation for each of the six methods.

Yes. What would be the easiest?

Empty bodies.

Yes. What does it do?

Nothing.

Correct. The methods are empty.

Now let me see if that compiles and runs.

If you compile the applet program you will see that the compiler won’t even touch the listener class.

Why?

Because it’s not even mentioned there.

We need to do step 3.
<table>
<thead>
<tr>
<th>Step 2 was the definition of the listener,</th>
<th>...and we still need to add something to it, as the methods are currently doing nothing,</th>
</tr>
</thead>
<tbody>
<tr>
<td>...and in step 3 we need to create an actual listener object and register it with the event source.</td>
<td>How about this:</td>
</tr>
<tr>
<td>import java.applet.*;</td>
<td></td>
</tr>
<tr>
<td>public class MouseSpyApplet extends Applet {</td>
<td></td>
</tr>
<tr>
<td>public void init() {</td>
<td></td>
</tr>
<tr>
<td>System.out.println(&quot;The applet is being initialized...&quot;);</td>
<td></td>
</tr>
<tr>
<td>MouseSpy listener = new MouseSpy();</td>
<td></td>
</tr>
<tr>
<td>addMouseListener(listener);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>public void start() {</td>
<td></td>
</tr>
<tr>
<td>System.out.println(&quot;The applet is being started...&quot;);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>Looks good, but how did you know of the add mouse listener method?</td>
<td>How do you know that an applet has a getWidth() (or a getHeight() ) method for that matter?</td>
</tr>
<tr>
<td>Aren't they inherited?</td>
<td>Same with addMouseListener.</td>
</tr>
<tr>
<td>Correct. Let's get back to the methods of the listener.</td>
<td>What about them?</td>
</tr>
<tr>
<td>Do they remind you of anything?</td>
<td>They should remind you of paint(Graphics g)</td>
</tr>
<tr>
<td>Who calls them?</td>
<td>The event source, basically.</td>
</tr>
<tr>
<td>Really?</td>
<td>No, not really.</td>
</tr>
<tr>
<td>Then who calls them?</td>
<td>The event source, but indirectly.</td>
</tr>
<tr>
<td>What do you mean by that?</td>
<td>The event source notifies the listener, actually.</td>
</tr>
<tr>
<td>And who calls the methods?</td>
<td>The system does.</td>
</tr>
<tr>
<td>How?</td>
<td>Event source says to system: &quot;I have this listener, I need to pass this info (an event) to it, could you do that for me, please?&quot;</td>
</tr>
<tr>
<td>And the appropriate method of the listener is called with the actual event as a parameter.</td>
<td>Yes, the actual event is passed as the parameter to your methods.</td>
</tr>
<tr>
<td>You define the methods.</td>
<td>And they provide the events.</td>
</tr>
</tbody>
</table>
So you can work on them. Neat.

| Just like `paint` is called with the actual `Graphics` context as a parameter. | The system gives you the context and you need to supply the definition of what should be done with it. |
| Let's summarize. | When you define a listener object you need to provide some definitions for what it should do, if it's notified that an event has occurred. |
| Then you need to create an actual listener object, and register it with the event source. | Which in this case will be the applet object, because it will be in direct contact with the mouse. |
| Fair enough. | Let's define the methods now, and make it happen. |

| Just like `Graphics` objects the `MouseEvent` also have specific functionality and behaviour. | g, for example, was able to `draw`. |
| e can give you the x and y coordinates of the `MouseEvent`...if e is of type `MouseEvent` |

Here's where `MouseEvent` is in the hierarchy of events:
Oh, boy. Are we also going to talk about window events today?

Yes, and here's another hierarchy:

```
import java.awt.event.*;
class MouseSpy implements MouseListener {
    public void mouseClicked (MouseEvent e) {
        System.out.println("Mouse clicked at: " +
                        e.getX() + " " + e.getY());
    }
    public void mousePressed (MouseEvent e) {
        System.out.println("Mouse pressed at: " +
                        e.getX() + " " + e.getY());
    }
    public void mouseReleased(MouseEvent e) {
        System.out.println("Mouse released at: " +
                        e.getX() + " " + e.getY());
    }
    public void mouseEntered (MouseEvent e) {
        System.out.println("Mouse entered at: " +
```

Ha! Now our own classes are pictured in blue.

Yes, and some methods are pictured in red, ...next to the classes which define them.

They're inherited by the subclasses, ...and it's important to know they are not redefined.

So they're as they should be. Yes. Let's finish the listener now.
```java
    e.getX() + " " + e.getY());
}

public void mouseExited (MouseEvent e) {
    System.out.println("Mouse exited at: " +
        e.getX() + " " + e.getY());
}
```

<table>
<thead>
<tr>
<th>We already have the applet.</th>
<th>Yes, so once you compile the applet you should run the appletviewer on the HTML file and everything should be working well.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can we also write standalone graphical applications?</td>
<td>You mean, not applets?</td>
</tr>
<tr>
<td>Yes, something with <code>main</code>.</td>
<td>Yes, but then you need to provide your own window.</td>
</tr>
<tr>
<td>Which is provided by the appletviewer or by the browser for applets.</td>
<td>I can see <code>Window</code> in the hierarchy above.</td>
</tr>
<tr>
<td>You can start from it, and extend it.</td>
<td>Extend it like we extended applets.</td>
</tr>
<tr>
<td>Yes, start from something already complex, and add your customizations to make it work the way you want it.</td>
<td>But is it a good idea to start directly from <code>Window</code>?</td>
</tr>
</tbody>
</table>

Then start from `JFrame` which is a more evolved type of window that has a title and a border.

Sounds better already.

```java
import javax.swing.*;
class EmptyFrame extends JFrame {
    public EmptyFrame() {
        final int WIDTH = 300; // default
        final int HEIGHT = 300; // values
        setSize(WIDTH, HEIGHT);
    }
}
class FrameTest {
    public static void main(String[] args) {
        EmptyFrame f = new EmptyFrame();
        f.setTitle("This is my title.");
        f.show();
    }
}
```

<table>
<thead>
<tr>
<th>If you run this what happens?</th>
<th>The frame overstays its welcome.</th>
</tr>
</thead>
<tbody>
<tr>
<td>It never goes away?</td>
<td>Why?</td>
</tr>
</tbody>
</table>
You need a window listener that realizes you have told it to go away.

<table>
<thead>
<tr>
<th>You need a window listener that realizes you have told it to go away.</th>
<th>I need to implement a <strong>MouseListener</strong> again?</th>
</tr>
</thead>
<tbody>
<tr>
<td>No, not a mouse listener, because we are not interested in the mouse here.</td>
<td>Ah, so I need to implement a <strong>window</strong> listener.</td>
</tr>
<tr>
<td>What’s that interface again?</td>
<td><strong>WindowListener</strong> from the same package as the interface for mouse listeners.</td>
</tr>
<tr>
<td>Fortunately there are adapter classes.</td>
<td>That you can extend.</td>
</tr>
<tr>
<td>Which already have all methods defined, and empty.</td>
<td>So you can override the one that you want to define.</td>
</tr>
</tbody>
</table>

I want to see the code.

<table>
<thead>
<tr>
<th>I want to see the code.</th>
<th>Here it is.</th>
</tr>
</thead>
<tbody>
<tr>
<td>import javax.swing.<em>; import java.awt.event.</em>; class EmptyFrame extends JFrame { public EmptyFrame() { final int WIDTH = 300; // default final int HEIGHT = 300; // values setSize(WIDTH, HEIGHT); WindowCloser w = new WindowCloser(); addWindowListener(w); } } class FrameTest { public static void main(String[] args) { EmptyFrame f = new EmptyFrame(); f.setTitle(&quot;This is my title.&quot;); f.show(); } } class WindowCloser extends WindowAdapter { public void windowClosing(WindowEvent e) { System.exit(0); } }</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Very good, the new code is in [blue].</th>
<th>As always.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Could I have used the same method with the mouse?</td>
<td>Yes, if there is an adapter for the listener you implemented.</td>
</tr>
<tr>
<td>And is there one?</td>
<td>Yes, there is, but you need to be sure of that.</td>
</tr>
<tr>
<td>Is this the only way we can do it?</td>
<td>What do you mean, we already did it in two ways: by implementing a interface and by extending an adapter.</td>
</tr>
<tr>
<td>Question</td>
<td>Answer</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Is there a third way?</td>
<td>Yes, you could use <em>inner classes</em>.</td>
</tr>
<tr>
<td>Very good. Is there a <em>fourth</em> way?</td>
<td>Yes, you can make the frame or the applet be its own listener.</td>
</tr>
<tr>
<td>Which method is best?</td>
<td>It depends on what you want to do.</td>
</tr>
<tr>
<td>I would like to take a break.</td>
<td><em>For that</em> they all behave the same.</td>
</tr>
</tbody>
</table>
Rubberbands

The only HTML we'll need is here:

```html
<html>
<head>
<title>Lab 11 Applets</title>
</head>
<body bgcolor=white>
  <applet code="Lab11.class" width=300 height=300>
  </applet>
</body>
</html>
```

The rest is applets' code.

Here's a startint point (or, rather, framework).

```java
import java.applet.*;
import java.awt. *
import java.awt.event.*;
public class Lab11 extends Applet implements MouseListener {
    public void init() {
        addMouseListener(this);
    }
    public void mouseClicked (MouseEvent e) { }
    public void mouseEntered (MouseEvent e) { }
    public void mouseExited (MouseEvent e) { }
    public void mousePressed (MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
}
```

Verify this framework is operational.

```java
import java.applet.*;
import java.awt. *
import java.awt.event.*;
public class Lab11 extends Applet implements MouseListener {
```

621
public void init() {
    addMouseListener(this);
}
public void mouseClicked(MouseEvent e) {
    System.out.println("Mouse clicked.");
}
public void mouseEntered(MouseEvent e) {
}
public void mouseExited(MouseEvent e) {
}
public void mousePressed(MouseEvent e) {
    System.out.println("Mouse pressed.");
}
public void mouseReleased(MouseEvent e) {
}

Can you notice the difference between a click and a press?

Now let’s draw lines.

The equations we will be using are these:

\[
\begin{align*}
x &= x_0 + t \cdot (x_1 - x_0) \\
y &= y_0 + t \cdot (y_1 - y_0)
\end{align*}
\]

When \( t \) goes from 0 to 1, the point \((x, y)\) follows a straight line from \((x_0, y_0)\) to \((x_1, y_1)\).

Can we illustrate that?

Let’s choose an \((x_0, y_0)\), an \((x_1, y_1)\) and let’s draw some intermediary points \((x, y)\).

For this we have a choice of where we start \( t \), where it ends, and how it varies.

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Lab11 extends Applet {
    public void paint(Graphics g) {
        int x0 = 50, y0 = 50, x1 = 100, y1 = 150;
        g.setColor(Color.blue);
        for (double t = 0; t <= 1; t += 0.1) {
            int x = (int) (x0 + t * (x1 - x0));
            int y = (int) (y0 + t * (y1 - y0));
            g.fillOval(x, y, 3, 3);
        }
        g.setColor(Color.red);
        g.fillOval(x0, y0, 3, 3);
        g.fillOval(x1, y1, 3, 3);
    }
}
```
One should study this carefully. Experiment with it a bit, for example change

- the width of the line (the radius of the ovals), and
- the step with which \( t \) is increased (0.01, 0.001, etc.)

So one could try this for a better line.

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Lab11 extends Applet {
    public void paint(Graphics g) {
        int x0 = 50, y0 = 50, x1 = 100, y1 = 150;
        g.setColor(Color.blue);
        for (double t = 0; t <= 1; t += 0.01) {
            int x = (int) (x0 + t * (x1 - x0));
            int y = (int) (y0 + t * (y1 - y0));
            g.fill0val(x, y, 3, 3);
        }
        g.setColor(Color.red);
        g.fill0val(x0, y0, 3, 3);
        g.fill0val(x1, y1, 3, 3);
    }
}
```

To make it easier later one could make this a method (recipe).

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Lab11 extends Applet {
    public void paint(Graphics g) {
        this.drawLine(g, 50, 50, 100, 150);
    }
    void drawLine(Graphics g, int x0, int y0, int x1, int y1) {
        g.setColor(Color.blue);
        for (double t = 0; t <= 1; t += 0.01) {
            int x = (int) (x0 + t * (x1 - x0));
            int y = (int) (y0 + t * (y1 - y0));
            g.fill0val(x, y, 3, 3);
        }
        g.setColor(Color.red);
        g.fill0val(x0, y0, 3, 3);
        g.fill0val(x1, y1, 3, 3);
    }
}
```

Now let’s get \((x0, y0)\) and \((x1, y1)\) from the mouse.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Lab11 extends Applet implements MouseListener {
    public void init() {
        addMouseListener(this);
    }
    public void mouseClicked(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
        System.out.println("Mouse pressed at: (" +
            e.getX() + ", " + e.getY() + ")");
    }
    public void mouseReleased(MouseEvent e) {
        System.out.println("Mouse released at: (" +
            e.getX() + ", " + e.getY() + ")");
    }
}

Let’s make the press define \((x_0, y_0)\) and the release define \((x_1, y_1)\).

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Lab11 extends Applet implements MouseListener {
    int x0, y0, x1, y1;
    public void init() {
        addMouseListener(this);
    }
    public void mouseClicked(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
        x0 = e.getX();
        y0 = e.getY();
        System.out.println("Mouse pressed at: (" +
            x0 + ", " + y0 + ")");
    }
    public void mouseReleased(MouseEvent e) {
        x1 = e.getX();
        y1 = e.getY();
        System.out.println("Mouse released at: (" +
            x1 + ", " + y1 + ")");
        this.repaint();
    }
}
} 

```java
public void paint(Graphics g) {
    g.drawLine(x0, y0, x1, y1);
}
```

Notice the new additions, and that the applet has a specific interface.

What follows is A201/A597 [LAB ASSIGNMENT ELEVEN]

You need to

- change the applet above so that
- the line follows the mouse pointer
- as if it were a rubber band.

Here’s a prototype\(^49\)

Press and drag for a new line! (Ignore the flicker).

Optional: change it to use your own line drawing routine.

_written: Question: How do you get rid of the flicker?

\(^{49}\) ../../../sum2001/labs/code/thirteen.html
Devices

We start from a simple applet

```java
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class One extends Applet implements MouseMotionListener {
    public void init() {
        addMouseMotionListener(this);
    }
    public void paint(Graphics g) { }
    public void mouseMoved(MouseEvent e) {
        int x = e.getX(), y = e.getY();
        System.out.println("Mouse moved at: (" + x + "," + y + ")");
    }
    public void mouseDragged(MouseEvent e) { }
}
```

that gets delivered from a simple HTML file

```html
<html>
<head>
    <title>All eyes on the mouse!</title>
</head>
<body>
    <applet code=One.class height=300 width=300>
    </applet>
</body>
</html>
```

This, of course, only watches the mouse.
Let's add a visual indicator of where the mouse is.
This is our watching device:

```java
import java.awt.*;
```
public class Device {
    int x, y, r, R, targetX, targetY;
    public Device(int x, int y, int r, int R) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.R = R;
        targetX = x + R;
        targetY = y + R;
    }
    public void draw(Graphics g) {
        g.drawOval(x, y, 2 * R, 2 * R);
        g.fillOval(targetX - r, targetY - r, 2 * r, 2 * r);
    }
}

Use this device in the applet, and it stares at you:

```java
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class One extends Applet implements MouseMotionListener {
    Device d = new Device(100, 100, 10, 30);
    public void init() {
        addMouseMotionListener(this);
    }
    public void paint(Graphics g) {
        d.draw(g);
    }
    public void mouseMoved(MouseEvent e) {
        int x = e.getX(), y = e.getY();
        System.out.println("Mouse moved at: (" + x + ", " + y + ")");
    }
    public void mouseDragged(MouseEvent e) {
    }
}
```

Now let's pass info from the mouse listener to the device:

And since we know the device changes state we need to repaint.

```java
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class One extends Applet implements MouseMotionListener {
    Device d = new Device(100, 100, 10, 30);
    public void init() {
        addMouseMotionListener(this);
    }
    public void paint(Graphics g) {
        d.draw(g);
    }
```
public void mouseMoved(MouseEvent e) {
    int x = e.getX(), y = e.getY();
    // System.out.println("Mouse moved at: (" + x + ", " + y + ")");
    d.targetX = x;
    d.targetY = y;
    repaint();
}
public void mouseDragged(MouseEvent e) {
}
}

The device now monitors the mouse movement a little too closely.
For the time being that’s OK.
Let’s draw the line that connects the center of the circle with the mouse pointer.

import java.awt.*;
public class Device {
    int x, y, r, R, targetX, targetY;
    public Device(int x, int y, int r, int R) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.R = R;
        targetX = x + R;
        targetY = y + R;
    }
    public void draw(Graphics g) {
        g.drawOval(x, y, 2 * R, 2 * R);
        int xA = x + R, yA = y + R, xB = targetX, yB = targetY;
        g.fillOval(targetX - r, targetY - r, 2 * r, 2 * r);
        g.drawLine(xA, yA, xB, yB);
    }
}

That looks like a rubber band.
Can we identify the place where the border crosses the rubber band?
Let’s place a second circle there.

import java.awt.*;
public class Device {
    int x, y, r, R, targetX, targetY;
    public Device(int x, int y, int r, int R) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.R = R;
        targetX = x + R;
        targetY = y + R;
    }
}
public void draw(Graphics g) {
    g.drawOval(x, y, 2 * R, 2 * R);
    int xA = x + R, yA = y + R, xB = targetX, yB = targetY;
    g.fillOval(targetX - r, targetY - r, 2 * r, 2 * r);
    g.drawLine(xA, yA, xB, yB);
    double distance = Math.sqrt((xB - xA) * (xB - xA) +
                                (yB - yA) * (yB - yA));
    double percent = R / distance;
    int xC = (int) (percent * (xB - xA)) + xA;
    int yC = (int) (percent * (yB - yA)) + yA;
    g.fillOval(xC-4, yC-4, 8, 8);
}

The part in brown finds the coordinates, (xC, yC) and the part in blue draws a small circle with a radius of 4 pixels centered in (xC, yC). (Take your crayons and put the color back in the page.)

Here's a picture that tries to make this even clearer:

\[
(xB, yB) = (targetX, targetY)
\]

\[
\text{distance}
\]

\[
(xC, yC)
\]

\[
(xA, yA)
\]

\[
x = (xB - xA) * t + xA
y = (yB - yA) * t + yA
\]

\[
xC = (xB - xA) * R / distance + xA
\]

Let's make a change:

```java
import java.awt.*;
public class Device {
    int x, y, r, R, targetX, targetY;
```
public Device(int x, int y, int r, int R) {
    this.x = x;
    this.y = y;
    this.r = r;
    this.R = R;
    targetX = x + r;
    targetY = y + r;
}

public void draw(Graphics g) {
    g.drawOval(x, y, 2 * r, 2 * r);
    int xA = x + r, yA = y + r, xB = targetX, yB = targetY;
    // g.fillOval(targetX - r, targetY - r, 2 * r, 2 * r);
    g.drawLine(xA, yA, xB, yB);
    double distance = Math.sqrt((xA - xB) * (xA - xB) +
                                (yA - yB) * (yA - yB));
    double percent = R / distance;
    int xC = (int) (percent * (xB - xA)) + xA;
    int yC = (int) (percent * (yB - yA)) + yA;
    g.fillOval(xC-r, yC-r, 2 * r, 2 * r);
}

That almost draws the small circle where needed.

We need to bring it inside along the line by r.

import java.awt.*;
public class Device {
    int x, y, r, R, targetX, targetY;
    public Device(int x, int y, int r, int R) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.R = R;
        targetX = x + r;
        targetY = y + r;
    }

    public void draw(Graphics g) {
        g.drawOval(x, y, 2 * r, 2 * r);
        int xA = x + r, yA = y + r, xB = targetX, yB = targetY;
        // g.fillOval(targetX - r, targetY - r, 2 * r, 2 * r);
        g.drawLine(xA, yA, xB, yB);
        double distance = Math.sqrt((xA - xB) * (xA - xB) +
                                      (yA - yB) * (yA - yB));
        double percent = (R - r) / distance;
        int xC = (int) (percent * (xB - xA)) + xA;
        int yC = (int) (percent * (yB - yA)) + yA;
        g.fillOval(xC-r, yC-r, 2 * r, 2 * r);
    }
}
And we can give up on the rubber band:

```java
import java.awt.*;
public class Device {
    int x, y, r, R, targetX, targetY;
    public Device(int x, int y, int r, int R) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.R = R;
        targetX = x + R;
        targetY = y + R;
    }
    public void draw(Graphics g) {
        g.drawOval(x, y, 2 * R, 2 * R);
        int xA = x + R, yA = y + R, xB = targetX, yB = targetY;
        g.fillOval(targetX - r, targetY - r, 2 * r, 2 * r);
        g.drawLine(xA, yA, xB, yB);
        double distance = Math.sqrt((xA - xB) * (xA - xB) +
                                    (yA - yB) * (yA - yB));
        double percent = (R - r) / distance;
        int xC = (int) (percent * (xB - xA)) + xA;
        int yC = (int) (percent * (yB - yA)) + yA;
        g.fillOval(xC - r, yC - r, 2 * r, 2 * r);
    }
}
```

But if we are inside the larger circle we need to be on the mouse.

```java
import java.awt.*;
public class Device {
    int x, y, r, R, targetX, targetY;
    public Device(int x, int y, int r, int R) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.R = R;
        targetX = x + R;
        targetY = y + R;
    }
    public void draw(Graphics g) {
        g.drawOval(x, y, 2 * R, 2 * R);
        int xA = x + R, yA = y + R, xB = targetX, yB = targetY;
        g.fillOval(targetX - r, targetY - r, 2 * r, 2 * r);
        g.drawLine(xA, yA, xB, yB);
        double distance = Math.sqrt((xA - xB) * (xA - xB) +
                                    (yA - yB) * (yA - yB));
        int xC, yC;
        if (distance < R - r) {
            xC = targetX;
            yC = targetY;
        } else {
```
double percent = (R - r) / distance;
    xC = (int) (percent * (xB - xA)) + xA;
    yC = (int) (percent * (yB - yA)) + yA;
}
g.fillOval(xC-r, yC-r, 2 * r, 2 * r);
}
}

Now all you have to do is use as many devices as needed:

```java
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class One extends Applet implements MouseMotionListener {
    Device d1 = new Device( 50, 50, 10, 30),
    d2 = new Device(100, 100, 10, 30),
    d3 = new Device(150, 150, 10, 30),
    d4 = new Device(200, 200, 10, 30);
    public void init() {
        addMouseMotionListener(this);
    }
    public void paint(Graphics g) {
        d1.draw(g);
        d2.draw(g);
        d3.draw(g);
        d4.draw(g);
    }
    public void mouseMoved(MouseEvent e) {
        int x = e.getX(), y = e.getY();
        // System.out.println("Mouse moved at: (" + x + ", " + y + ")");
        d1.targetX = x; d1.targetY = y;
        d2.targetX = x; d2.targetY = y;
        d3.targetX = x; d3.targetY = y;
        d4.targetX = x; d4.targetY = y;
        repaint();
    }
    public void mouseDragged(MouseEvent e) { }
}
```

And things are easier to follow if we reactivate the rubber band:

```java
import java.awt.*;
public class Device {
    int x, y, r, R, targetX, targetY;
    public Device(int x, int y, int r, int R) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.R = R;
        targetX = x + R;
        targetY = y + R;
```
public void draw(Graphics g) {
    g.drawOval(x, y, 2 * r, 2 * r);
    int xA = x + R, yA = y + R, xB = targetX, yB = targetY;
    // g.fillOval(targetX - r, targetY - r, 2 * r, 2 * r);
    g.drawLine(xA, yA, xB, yB);
    double distance = Math.sqrt((xA - xB) * (xA - xB) +
                               (yA - yB) * (yA - yB));
    int xC, yC;
    if (distance < R - r) {
        xC = targetX; yC = targetY;
    } else {
        double percent = (R - r) / distance;
        xC = (int) (percent * (xB - xA)) + xA;
        yC = (int) (percent * (yB - yA)) + yA;
    }
    g.fillOval(xC-r, yC-r, 2 * r, 2 * r);
}

And this is a test\(^{50}\) of what we have developed.

\(^{50}\)http://www.cs.indiana.edu/classes/a201/spr2001/lectures/eyes/one.html
We start from this code:

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Example extends /*NoFlicker*/Applet {

    Circle[] circles;

    public void init() {
        circles = new Circle[100];

        for (int i = 0; i < circles.length; i++) {
            circles[i] =
                new Circle(
                    (int) (Math.random() * (280 - 20) + 20), // x
                    (int) (Math.random() * (280 - 20) + 20), // y
                    (int) (Math.random() * (20 - 10) + 10), // radius
                    new Color( (float)Math.random(),
                               (float)Math.random(),
                               (float)Math.random() )
                );
        } // end of for

        Broker peterPan = new Broker(this);

        addMouseListener(peterPan);
        addMouseMotionListener(peterPan);
    }
```
class Broker implements MouseMotionListener, MouseListener {

    Example customer;
    Circle[] a;
    Circle c;

    Broker(Example someone) {
        this.customer = someone;
        this.a = this.customer.circles;
        this.c = null;
    }

    public void mouseDragged(MouseEvent e) {
        int x = e.getX(), y = e.getY();
        if (c != null) {
            c.move(x, y);
            customer.repaint();
        }
    }
    public void mouseMoved(MouseEvent e) {
    }

    public void mouseClicked(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
        int x = e.getX(), y = e.getY();
        System.out.println("Mouse pressed at (" + x + ", " + y + ")");
        for (int i = a.length - 1; i >= 0; i--) {  
            if (a[i].contains(x, y)) {
                c = a[i];
            break;
        }
        }
        System.out.println("Currently on... " + c);
    }
    public void mouseReleased(MouseEvent e) { c = null; }
}
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }

class Circle {
    int x; int y; // center, not top-left corner
    int radius;
    Color c;

    public void move(int x, int y) {
        this.x = x; this.y = y;
    }

    Circle(int x, int y, int r, Color c) {
        this.x = x;
        this.y = y;
        this.radius = r;
        this.c = c;
    }

    public boolean contains(int x, int y) {
        double dX = this.x - x,
               dY = this.y - y;

        if (Math.sqrt(dX * dX + dY * dY) <= this.radius)
            return true;
        else
            return false;
    }

    public String toString() {
        return "Circle at (" + this.x + ", " + this.y + ") " +
               " size " + this.radius + ", color " + this.c;
    }

    public void draw(Graphics b) {
        b.setColor(this.c);
        b.fillOval(this.x - this.radius,
                   this.y - this.radius,
                   2 * this.radius,
                   2 * this.radius);
        b.setColor(Color.black);
        b.drawOval(this.x - this.radius,
                   this.y - this.radius,
                   2 * this.radius,
                   2 * this.radius);
    }
}


```html
<html>
<head>
<title>Circles</title>
</head>
<body bgcolor=white>
<applet code="Example.class" width=300 height=300>
</applet>
</body>
</html>
```

Question: Why the flicker?

The answer is: because of `update()`. (We inherit this method, and it is called by `repaint()`). Every time it's called it clears the screen and then calls `paint()`. To obtain flicker-free screen updates (which is essentially what we are trying to do here, as in any animation) we will need to override `update()` such that it creates the new image somewhere in the background (that is, in memory) and then updates the screen in one fell swoop. It is easy to implement this as an application of inheritance. Here's the new definition of `update()`.

Here's how we remove the flicker:

```java
import java.awt.*;
import java.applet.*;
class NoFlickerApplet extends Applet {

    private Image offScreenImage;
    private Graphics offScreenGraphics;
    private Dimension offScreenSize;

    public final void update(Graphics theGraphicsContext) {
        Dimension dim = this.size();

        if (offScreenImage == null) ||
            (dim.width != offScreenSize.width) ||
            (dim.height != offScreenSize.height) {  
            this.offScreenImage = this.createImage(dim.width, dim.height);
            this.offScreenSize = dim;
            this.offScreenGraphics = this.offScreenImage.getGraphics();
        }

        this.offScreenGraphics.clearRect(0,
            0,
            this.offScreenSize.width,
            this.offScreenSize.height);

        this.paint(offScreenGraphics);

        theGraphicsContext.drawImage(this.offScreenImage,
```

0,
0,
null);
}
}

Now your program needs to extend this class instead of Applet directly.
The rest of the lecture will be spent on discussing Homework Six.
We want first to understand how we would be solving the problem ourselves.
Conceptualization is the first step.
If we can’t teach someone else how to do it, we won’t be able to teach the computer.
Once we come up with some sort of

- plan, or
- approach, or
- strategy

we need to think if

- it is correct or not, and
- what it takes to actually implement it

We need to come up with a plan, and I will describe one such approach.
And here’s a method that we looked at before that is close to the approach I will suggest.

```java
/*static*/ void sort(int[] a) {
    boolean done;
    do {
        done = true;
        for (int i = 0; i < a.length - 1; i++) {
            if (a[i] > a[i + 1]) { // sort in ascending order
                int temp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = temp;
                done = false;
            }
        }
    } while (! done);
}
```

This method runs subsequent passes through the array, and so will we.
Also, here’s another approach to Homework Five:
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

/**<*<applet code=One.class width=300 height=300*/</applet>*// *

public class One extends Applet implements MouseListener,
        MouseMotionListener {
    public void init() {
        e1 = new Eye();
        e2 = new Eye();
        n = new Nose();
        m = new Mouth();
        this.addMouseListener(this);
        this.addMouseMotionListener(this);
    }
    public void paint(Graphics g) {
        e1.draw(g);
        e2.draw(g);
        n.draw(g);
        m.draw(g, out);
        System.out.println("paint has finished...");
    }
    public void mouseDragged(MouseEvent e) { }
    public void mouseMoved(MouseEvent e) {
        out = false;
    }
    boolean out = true;
    
    Eye e1, e2;
    Nose n;
    Mouth m;
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseClicked(MouseEvent e) { }
    public void mouseExited(MouseEvent e) {
        out = true;
        System.out.println("Out...");
        repaint();
    }
    public void mouseEntered(MouseEvent e) {
        out = false;
        System.out.println("Back in!");
        repaint();
    }
class Eye {
    public void draw(Graphics g) { }
}
class Nose {
    public void draw(Graphics g) { }
}
class Mouth {
    int x, y, width = 300, height = 150, start = 190, stop = 160;
    public void draw(Graphics g, boolean out) {
        if (out/* == true */) {
            System.out.println("Broad smile.");
            g.drawArc(x, y, width, height, start, stop);
        } else {
            System.out.println("Attentive smile...");
            g.drawArc(x, y, width, height, start + 40, stop - 2 * 40);
        }
    }
}
Homework Six

A Very Basic Interpreter (Evaluator)

For this assignment you are to write an arithmetic expression evaluator that is able to perform

- additions,
- multiplications
- subtractions, and
- divisions,

is able to

- store results in named locations (variables) and
- allow them in expressions as well,

as illustrated in the sample session below.

The evaluator should also handle parentheses properly.

Here’s a sample session with the program:

```
frilled.cs.indiana.edu% java Eval
Eval> 1 + 1
1 + 1
2
Result is: 2
Eval> ( 1 + 2 ) * 3
( 1 + 2 ) * 3
3 * 3
9
Result is: 9
Eval> a = 1 + 1
1 + 1
2
Stored 2 in a
Eval> a
```

643
2
Result is: 2
Eval> a = a + 1
2 + 1
3
Stored 3 in a
Eval> a
3
Result is: 3
Eval> b = a - 4
3 - 4
-1
Stored -1 in b
Eval> a
3
Result is: 3
Eval> b
-1
Result is: -1
Eval> a = ( a - b ) * ( a + b )
( 3 - 1 ) * ( 3 + 1 )
( 4 ) * ( 3 + 1 )
( 4 ) * ( 2 )
4 * ( 2 )
4 * 2
8
Stored 8 in a
Eval> a
8
Result is: 8
Eval> b
-1
Result is: -1
Eval> b = a + b * 4
8 + 1 * 4
8 + 4
4
Stored 4 in b
Eval> a / b
8 / 4
2
Result is: 2
Eval> a = a / b
8 / 4
2
Stored 2 in a
Eval> a
2
Result is: 2
Eval> b
4

Result is: 4
Eval> exit
frilled.cs.indiana.edu%

You may safely assume that

- your only operands are integers and that
- the names of variables are comprised of only alphabetical characters.

Note that in my evaluator the tokens (numbers, parens) are separated by spaces.

Here's a grading scale for this assignment:

<table>
<thead>
<tr>
<th>Task</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading the data line by line and storing it properly</td>
<td>15</td>
</tr>
<tr>
<td>Being able to support * and -</td>
<td>20</td>
</tr>
<tr>
<td>Supporting * and /</td>
<td>20</td>
</tr>
<tr>
<td>Allowing named variables to store values</td>
<td>15</td>
</tr>
<tr>
<td>Allowing named variables in expressions</td>
<td>15</td>
</tr>
<tr>
<td>Properly calculating (1 + 1) * 2 + 2 * ( 1 + 1 )</td>
<td>10</td>
</tr>
<tr>
<td>Providing error-checking through try-catch blocks</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>

Please let us know if you have any questions.
Help with The Last Assignment

Lab Twelve: Help with The Last Assignment (The Evaluator).

This is the code you will develop in lab today.

```java
import java.io.*;
import java.util.*;

class Eval {
    /* this Hashtable must be shared between two static methods
       (main, which creates it and process, which is using it
       for lookup,) so it must be global enough */
    static Hashtable env = new Hashtable();
    // here's the main method getting away with a try/catch
    public static void main(String[] args) throws IOException {

        // this is why we need to acknowledge the IOException
        BufferedReader console =
            new BufferedReader
                (new InputStreamReader(System.in));
        // checked after the loop, must be visible enough
        String line;

        do {
            // prompt
            System.out.print("Eval> ");
            // ... and read
            line = console.readLine();

            // stop if you have to
            if (line.equals("bye")) break;
            // get ready to take things apart
            StringTokenizer st = new StringTokenizer(line);
            // prepare the storage for the resulting tokens
            Vector v = new Vector();
            // and now just do it
            while (st.hasMoreTokens()) {
                v.addElement(st.nextToken());
            }
```
// so now the tokens are in a Vector, v
// could be an assignment statement
if (v.size() > 2 & v.elementAt(1).equals("=")) {
    // in which case you take the name at the front
    String name = (String)v.elementAt(0);
    // ... and keep it in the variable called 'name'
    // then remove the string and the equal sign
    v.removeElementAt(1);
    v.removeElementAt(0);
    // the Vector is shrinking as you remove elements so,
    // either remove backwards or twice at index 0 (zero)!
    // now process the remaining expression and store the
    // value in the Hashtable in the entry for the name
    // you found in position 0 in the original Vector
    env.put(name, Eval.process(v));
} else { // if it's not an assignment statement,

    // ... then you can start evaluating it
    Eval.process(v);
} // end of conditional processing
// getting ready to report the result
System.out.print("Done: ");
Eval.show(v); // the Vector is shown here
} while (true); // loop if you have to (exit with break!)

} // end of main
// straightforward procedure for showing a Vector
static void show(Vector v) {
    for (int i = 0; i < v.size(); i++) {
        System.out.print(" " + v.elementAt(i));
    }
    System.out.println();
}
// this is the heart of the evaluator
static String process (Vector v) { // receives the Vector as a param
    // wishful thinking (see do/while test below)
    boolean finished = true;
    // first replace identifiers by their values
    for (int i = 0; i < v.size(); i++) {
        // get the String that's there (position i)
        // this could be anything: +, -, 123, a, *
        String element = (String)(v.elementAt(i));
        // look it up in env (nevermind how it got there)
        Object value = env.get(element);
        // if you find it, there's a value associated with
        if (value != null) {
            // replace it (index i) by the value it has
            v.setElementAt((String)value, i);
        }
    } // all identifiers have been replaced
// we're ready for the real processing
do {
    // remove parens and update your agenda
    finished = Eval.removeParens(v);
    // all three methods used here return true when
    // they could not find anything to work on within
    // the expression, and they return false if they
    // could change it in at least one aspect; therefore
    // your hope of being finished is true only when none
    // of the workers find anything to work on...
    finished = finished && Eval.removeStars(v);
    finished = finished && Eval.removePluses(v);
    // report the current status of the Vector
    // from within the process of changing it
    System.out.print("Processing: "); Eval.show(v);
} while (!finished); // if not finished keep going...
// return the first element in the resulting Vector
return (String) (v.elementAt(0));
}
static boolean removeParens(Vector v) {
    // empty procedure, reports true
    return true;
    // this means it 'thinks' it's done...
}
static boolean removeStars(Vector v) {
    // same here, you need to provide these methods...
    return true;
}
static boolean removePluses(Vector v) {
    // the only one that works
    boolean done = true; // assume nothing will have to be done
    // look everywhere in the Vector
    for (int i = 0; i < v.size(); i++) {
        // if you find a plus sign
        if (v.elementAt(i).equals("+")) {
            done = false; // there's work to be done
            // expect two numbers on its sides
            int n = Integer.parseInt((String) (v.elementAt(i - 1))),
                    m = Integer.parseInt((String) (v.elementAt(i + 1)));
            // store the sum where the first operand was
            v.setElementAt((n + m) + ", i - 1);
            // carefully remove the plus and the second operand
            v.removeElementAt(i + 1);
            v.removeElementAt(i);
            // don't skip the next token
            i = i + 1;
            // the Vector has shrunk so it's in front
            // of you, and the for loop will increment i by default
        } else {
            // take the number
            v.setElementAt(v.elementAt(i), i);
            // and move to the next token
            i = i + 1;
        }
    }
}
The emphasis here is on

- *understanding* what it takes to first develop a strategy (procedure, algorithm)
- and then to instruct the computer to behave that way.

A lab assignment will be determined and then posted here soon.

Most likely it will have something to do with the following:
(Here’s an example due to Alex Basyrov, Spring 2002 A597)

How do you evaluate this?

\[(1 + 1) \times 2 + 2 \times (1 + 1)\]

Our substitution rules need to be revised once the parens are added.

Parens can delay the evaluation of operators (such as "\(*\)"") and when that happens we need to propagate this delay throughout. So, in our case, we only need to take care of pluses and minuses, which should NOT be evaluated if there are stars and slashes waiting to be processed (they bind stronger to the numbers than their lower precedence counterparts).

So the code more or less becomes (only an if statement changes):

```java
static boolean reducePlusesAndMinuses(Vector expr) {
    boolean done = true;
    for (int i = 0; i < expr.size(); i++) {
        if (expr.elementAt(i).equals("+") || expr.elementAt(i).equals("-")) {
            if (    // value to its left
                !isValue((String)expr.elementAt(i - 1)) &&
                // value to its right
                !isValue((String)expr.elementAt(i + 1)) &&
                // and not a higher priority operator to wait on
                !((String)expr.elementAt(i - 2)).equals("*") &&
                // higher precedence operator waiting there
                !((String)expr.elementAt(i - 2)).equals("/")
                ) // this can happen on the left
                || // or (it can happen on the right)
                !isValue((String)expr.elementAt(i + 2))
            )
```
// higher precedence operator waiting there
( ((String)expr.elementAt(i + 2)).equals("*") ||
  // note there are two such operators
  ((String)expr.elementAt(i + 2)).equals("/")
)
)
}

Integer a = new Integer((String)expr.elementAt(i - 1));
Integer b = new Integer((String)expr.elementAt(i + 1));
// ... reduce the plus sign

You should include this change in your programs.

You should also be aware of all the context dependencies that it relies on.
The Second Dilbert Lecture

Review and Tutorial: Extending Classes

(In which Dilbert makes a return. As can be seen in the image these notes are picking some momentum. You should read what follows even if you know it all, just to verify your understanding is for real. But you can also skip this section the first time through it).

Let's make a quick summary first (please refer to the first Dilbert Review—as well).

Here are the things that should be already known:

- A class is a collection of data and methods that operate on that data.
- An object is a particular instance of that class.
- Object members (fields and methods) are accessed with a dot between
  - the object name and
  - the member name.
- Instance (non-static) variables occur in each instance of the class.
- Class (static) variables are associated with the class.

  There is one copy of a class variable regardless of the number of instances of a class.

- Instance (non-static) methods of a class are passed an implicit this argument that identifies the object being operated on (the receiving object, the one that owns that method, and through which the method gets invoked.)
- Class (static) methods are NOT passed a this argument and therefore do NOT have a current instance of the class that can be used to implicitly refer to instance variables or invoke instance methods.
- Object are created with the new keyword, which invokes a class constructor method with a list of arguments.
- (Objects are not explicitly freed or destroyed in any way.
  The Java garbage collector automatically reclaims objects that are no longer being used.)

Things that will be covered in this Second Review and Tutorial:
• If the first line of a constructor method does not invoke another constructor with a this call, or a superclass constructor with a super() call, Java automatically inserts a call to the superclass constructor that takes no arguments. This enforces "constructor chaining".

• If a class does not define a constructor, Java provides a default constructor.

• A class may inherit the non-private methods and variables of another class by "subclassing" – i.e., by declaring that class in its extends clause.

• java.lang.Object is the default superclass for any class. It is the root of the Java class hierarchy and has no superclass itself. All Java classes inherit the methods defined by Object.

• Method overloading is the practice of defining multiple methods which have the same name but have different argument lists. Method overriding occurs when a class redefines a method inherited from its superclass.

• Dynamic method lookup ensures that the correct method is invoked for an object, even when the object is an instance of a class that has overridden the method.

• static, private, and final methods cannot be overridden and are not subject to dynamic method lookup. (This allows compiler optimizations such as inlining).

• From a subclass, you can explicitly invoke an overridden method of a superclass with the super keyword.

• You can explicitly refer to a shadowed variable with the super keyword.

• Data and methods may be hidden or encapsulated within a class by specifying the private or protected visibility modifiers. Members declared public are visible everywhere. Members with no visibility modifiers are visible only within the package.

• An abstract method has no method body (i.e., no implementation)

• An abstract class contains abstract methods. The methods must be implemented in a subclass before the subclass can be instantiated.

• An interface is a collection of abstract methods and constants (static final variables). Declaring an interface creates a new data type.

• A class implements an interface by declaring the interface in its implements clause and by providing a method body for each of the abstract methods in the interface.

Classes and Objects

A class is a collection of data and methods that operate on that data.

The data and methods, taken together, usually serve to define the contents and capabilities of some kind of object. For example, a circle can be described by the x, y position of its center and by its radius (r).

There are a number of things we can do with circles: compute their circumference, compute their area, check whether points are inside them, and so on. Each circle is different (i.e., has a different center or radius), but as a class, circles have certain intrinsic properties that we can capture in a definition:

```java
public class Circle {
    public double x, y; // The coordinates of the center
    public double r; // The radius
    // Methods that return the circumference and area of the circle
```
public double circumference () {
    return 2 * 3.141592 * r;
}
public double area () {
    return 3.141592 * r * r;
}

Object Creation
Here’s how we create a circle object:

    Circle c = new Circle();

The way it works is this: The new keyword creates a new dynamic instance of the class – i.e., it allocates the new object. The constructor method is then called, passing the new object implicitly (a this reference) and passing the arguments specified between parentheses (if any) explicitly.

The constructor can then start working on the object for initialization purposes.

Defining a Constructor
There is some obvious initialization we could do for our circle objects, so let’s define a constructor.

The code below shows a constructor that lets us specify the initial values for the center and radius of our new Circle object:

    public class Circle {
        public double x, y, r; // The center and the radius of the circle
        // The constructor method (or initialization procedure)
        public Circle(double x, double y, double r) {
            this.x = x;
            this.y = y;
            this.r = r;
        }

        public double circumference () {
            return 2 * 3.141592 * r;
        }
        public double area () {
            return 3.141592 * r * r;
        }
    }

The part that we have not seen before (that is, the constructor) is in blue.

There are two important notes about naming and declaring constructors:

- The constructor name is always the same as the class name.
- The return type is implicitly an instance of the class. No return type is specified in constructor declarations, nor is the void keyword used. The this object is implicitly returned; a constructor should not use a return statement to return a value.
Multiple Constructors

Sometimes you’ll want to be able to initialize an object in a number of different ways, depending on what is most convenient in a particular circumstance. For example, we might want to be able to initialize the radius of a circle without initializing the center, or we might want to initialize a circle to have the same center and radius as another circle, or we might want to initialize all the fields to default values. Doing this is no problem: A class can have any number of constructor methods.

The example below shows how:

```java
public class Circle {
    public double x, y, r;

    public Circle(double x, double y, double r) {
        this.x = x; this.y = y; this.r = r;
    }

    public Circle(double r) { x = 0.0; y = 0.0; this.r = r; }
    public Circle(Circle c) { x = c.x; y = c.y; r = c.r; }
    public Circle() { x = 0.0; y = 0.0; r = 1.0; }

    public double circumference() { return 2 * 3.141592 * r; }
    public double area() { return 3.141592 * r * r; }
}
```

Again, the new code is in **blue** (use your crayons).

Method Overloading

The surprising thing in this example could be that all the constructor methods have the same name! So how can the compiler tell them apart? The way that you and I tell them apart is that the four methods take different arguments and are useful in different circumstances. The compiler tells them apart in the same way. In Java a method is distinguished by its name, and by the number, type, and position of its arguments. This is not limited to constructor methods—any two methods are not the same unless they have the same name, and the same number of arguments of the same type passed at the same position in the argument list. When you call a method and there’s more than one method with the same name, the compiler automatically picks the one that matches the data types of the arguments you are passing.

Defining methods with the same name and different argument types is called **method overloading**. It can be a convenient technique, as long as you only give methods the same name when they perform similar functions on slightly different forms of input data. Overloaded methods may have different return types, but only if they have different arguments. Don’t confuse method overloading with **method overriding**, which we’ll discuss later.

this Again

There’s a specialized use of the **this** keyword that arises when a class has multiple constructors—it can be used from a constructor to invoke one of the other constructors of the same class. So we could rewrite the additional constructors from the previous example in terms of the first one like this:

```java
public Circle(double x, double y, double r) {
    this.x = x; this.y = y; this.r = r;
}
public Circle(double r) { this(0.0, 0.0, r); }
```
public Circle(Circle c) { this(c.x, c.y, c.r); }
public Circle() { this(0.0, 0.0, 1.0); }

Here the this() call refers to whatever constructor of the class takes the specified type of arguments. This would be a more impressive example, of course, if the first constructor that we were invoking did a more significant amount of initialization, as it might, for example, if we were writing a more complicated class.

There is a very important restriction on this this syntax that is, as an invocation: it may only appear as the first statement in a constructor. It may, of course, be followed by any additional initialization that a particular version of the constructor needs to do. The reason for this restriction involves the automatic invocation of superclass constructor methods, to which we turn now.

Subclasses and Inheritance

This Circle class is good for abstract mathematical manipulation. For some applications this is exactly what we need. For other applications, we might want to be able to manipulate circles and draw them on the screen. This means we need a new class, GraphicCircle, that has all the functionality of Circle, but also has the ability to be drawn.

We want to implement GraphicCircle so that it can make use of the code we've already written for Circle. One way to do that is the following:

```java
public class GraphicCircle {
    public double x, y;
    public double r;
    public Color outline, fill;
    public double circumference () {
        return 2 * 3.141592 * r;
    }
    public double area () {
        return 3.141592 * r * r;
    }
    public void draw(DrawWindow dw) {
        /* code omitted */
    }
}
```

The part that we have taken from the previous class is in blue.

This approach would work but it is not particularly elegant. The problem is that we have to literally carry the code with us, and rewrite it. It would be nice if we didn’t have to do that.

Extending a Class

Well, we really don’t have to do it that way.

We can define GraphicCircle as an extension, or subclass of class Circle.

```java
public class GraphicCircle extends Circle {
    /* We automatically inherit the variables and methods of class Circle, so we only have to put the new stuff here. We omit the constructor for GraphicCircle, for now. */
    Color outline, fill;
    public void draw(DrawWindow dw) {
```
The part that we take from Circle is also represented in blue, the mechanism is provided by the object-oriented nature of Java, and we have come up with something reasonable as the code for the instance method of GraphicCircle, that is draw().

The extends keyword tells Java that GraphicCircle is a subclass of Circle, and that it inherits the fields and methods of that class (except for private fields and methods). The definition of the draw() method shows variable inheritance - this method uses the Circle variables x, y, and r as if they were defined right in GraphicCircle itself.

GraphicCircle also inherits the methods of Circle.

Thus, if we have a GraphicCircle object referred to by variable gc, we can say:

```java
double area = gc.area();
```

This works just as if the area() method were defined in GraphicCircle itself. So this is a convenient way of reusing the Circle class's code.

Another feature of subclassing is that every GraphicCircle object is also a perfectly legal Circle object. Thus, if gc refers to a GraphicCircle object, we can assign it to a Circle variable, and we can forget all about its extra graphic capabilities:

```java
Circle c = gc;
```

That is, it's up to us if we want to ignore its graphic capabilities added by GraphicCircle.

Final Classes

When a class is declared with the final modifier, it means that it cannot be extended or subclassed. java.lang.System is an example of a final class. Declaring a class final prevents unwanted extensions to the class. (But it also allows the compiler to make some optimizations when invoking the methods of the class.)

Superclasses, Object, and the Class Hierarchy

In our example GraphicCircle is a subclass of Circle. We can also say that Circle is the superclass of GraphicCircle. The superclass of a class is specified in its extends clause:

```java
public class GraphicCircle extends Circle { ... }
```

Every class you define has a superclass. If you do not specify the superclass with an extends clause, the superclass is the class Object. Object is a special class for a couple of reasons:

- It is the only class that does not have a superclass
- The methods defined by Object can be called by any Java object

You can look at class Object here

Because every class has a superclass, classes in Java form a class hierarchy, which can be represented as a tree with Object at its root. The diagram below shows a class hierarchy which includes our Circle and GraphicCircle classes, as well as some of the standard classes from the Java API.

Object --> Circle --- GraphicCircle
   |    
   +- Math
   |    
   +- System
   |    
   +- Component --> Container --- Panel --- Applet
   |    
   +- Button
   |    
   +- List

The complete class hierarchy for the Java 1.1 API is diagrammed here\(^{\text{52}}\). Here's the latest\(^{\text{53}}\).

In our first example of GraphicCircle we left out the constructor method for our new GraphicCircle class. Let's implement it now. Here's one way:

```java
public GraphicCircle (double x, double y,
   double r,
   Color outline,
   Color fill) {
   this.x = x;
   this.y = y;
   this.r = r;
   this.outline = outline;
   this.fill = fill;
}
```

The constructor relies on the fact that GraphicCircle inherits all of the variables of Circle and simply initializes those variables itself. But this duplicates the code of the Circle constructor, and if Circle did more elaborate initialization, it could become quite wasteful. (Same problem with when we tried to define a GraphicCircle for the first time). Furthermore, if the Circle class had internal private fields (discussed later) we wouldn't be able to initialize them like this. What we need is a way of calling a Circle constructor from within our GraphicCircle constructor, and that is provided with the [extends] mechanism, by Java.

Here's how we invoke a superclass's constructor:

```java
public GraphicCircle (double x, double y,
   double r,
   Color outline,
   Color fill) {
   super(x, y, r);
   this.outline = outline;
   this.fill = fill;
}
```

`super` is a reserved word in Java. One of its uses is that shown in the example – to invoke the constructor method of a superclass. Its use is analogous to the use of the `this` keyword to invoke one constructor method of a class from within another. Using `super` to invoke a constructor is subject to the same restrictions as using `this` to invoke a constructor:

\(^{\text{52}}\) [http://java.sun.com/products/jdk/1.1/docs/api/tree.html](http://java.sun.com/products/jdk/1.1/docs/api/tree.html)

• **super** may only be used in this way (with the syntax for a method call, or invocation) within a constructor method. You can’t invoke **super(...)** in a method, for example, as it wouldn’t mean anything to use it that way.

• The call to the superclass constructor must appear as the first statement within the constructor method. It must appear even before *variable declarations*. *(Doesn’t this sound like you would want to test this already?)*

**Constructor Chaining**

When you define a class, Java guarantees that the class’s constructor method is called whenever an instance of that class is created. It also guarantees that the constructor is called when an instance of any subclass is created. In order to guarantee this second point, Java must ensure that every constructor method calls its superclass constructor method. If the first statement in a constructor is not an explicit call to a constructor of the superclass with the **super** keyword, then Java implicitly inserts the call **super()** – that is, it calls the superclass constructor with no arguments. If the superclass does not have a constructor that takes no arguments, this causes a compilation error.

There is one exception to the rule that Java invokes **super** implicitly if you do not do so explicitly. If the first line of a constructor, **C1**, uses the **this()** syntax to invoke another constructor, **C2**, of the class, Java relies on **C2** to invoke the superclass constructor, and does not insert a call to **super()** into **C1**. Of course, if **C2** itself uses **this()** to invoke a third constructor, **C2** does not call **super()** either, but somewhere along the chain, a constructor either explicitly or implicitly invokes the superclass constructor, which is what is required.

Consider what happens when we create a new instance of the **GraphicCircle** class. First, the **GraphicCircle** constructor shown in our previous example is invoked. This constructor explicitly invokes a **Circle** constructor and that **Circle** constructor implicitly calls **super()** to invoke the constructor of its superclass, **Object**. The body of the **Object** constructor runs first, followed by the body of the **Circle** constructor, and finally followed by the body of the **GraphicCircle** constructor.

What this all means is that constructor calls are "chained" – any time an object is created, a sequence of constructor methods are invoked, from subclass to superclass on up to **Object** at the root of the class hierarchy. Because a superclass constructor is always invoked as the first statement of its subclass constructor, the body of the **Object** constructor always runs first, followed by the body of its subclass, and on down the class hierarchy to the class that is being instantiated.

**The Default Constructor**

There is one missing piece in the description of constructor chaining above. If a constructor does not invoke a superclass constructor, Java does so implicitly. But what if a class is declared without any constructor at all? In this case, Java implicitly adds a constructor to the class. This default constructor does nothing but invoke the superclass constructor. For example, if we did not declare a constructor for the **GraphicCircle** class, Java would have implicitly inserted this constructor:

```java
public GraphicCircle() { super(); }
```

Note that if the superclass, **Circle()** did not declare a no-argument constructor, then this automatically inserted default constructor would cause a compilation error. If a class does not define a no-argument constructor, then all of its subclasses must define constructors that explicitly invoke the superclass constructor with the necessary arguments.

It can be confusing when Java implicitly calls a constructor or inserts a constructor definition into a class – something is happening that does not appear in your code! Therefore, it is good coding style, whenever you rely
on an implicit superclass constructor call or on a default constructor, to insert a comment noting this fact. Your comments might look like those in the following example:

class A {
    int i;
    public A() {
        // Implicit call to super() here
        i = 3;
    }
}
class B extends A {
    // Default constructor: public B() { super(); }
}

If a class does not declare any constructor, it is given a public constructor by default. Classes that do not want to be publically instantiated, should declare a protected constructor to prevent the insertion of this public constructor. Classes that never want to be instantiated at all (in one particular, specific way,) should define that particular constructor private.

Shadowed Variables

Suppose that our GraphicCircle class has a new variable that specifies the resolution, in dots per inch, of the DrawWindow object in which it is going to be drawn. And further, suppose that it names that new variable r:

class GraphicCircle extends Circle {
    Color outline, fill;
    float r; // New variable. Resolution in dots-per-inch.
    public GraphicCircle(double x, double y, double rad, Color o, Color f) {
        super(x, y, rad);
        outline = o; fill = f;
    }
    public void setResolution(float resolution) {
        r = resolution;
    }
    public void draw(DrawWindow dw) {
        dw.drawCircle(x, y, r, outline, fill);
    }
}

Now, with this resolution variable declared, when we use the variable r in the GraphicCircle class, we are no longer referring to the radius of the circle. The resolution variable r in GraphicCircle shadows the radius variable r in Circle. (This is a contrived example, of course – we could simply rename the variable and avoid the issue. Typically we would rename the variable: variable shadowing is a necessary part of Java syntax, but is not a useful programming technique. Your code will be easier to understand if you avoid shadowed variables).

So, how can we refer to the radius variable defined in the Circle class when we need it? Recall that using a variable, such as r, in the class in which it is defined is shorthand for:

    this.r // Refers to the GraphicCircle resolution variable.

As you might guess, you can refer to a variable r defined in the superclass like this:
super.r // Refers to the Circle radius variable.

Another way you can do this is to cast this to the appropriate class and then access the variable:

```java
((Circle) this).r
```

This cast is exactly what the super does when used like this. You can use this casting technique when you need to refer to a shadowed variable defined in a class that is not the immediate superclass. For example, if C is a subclass of B, which is a subclass of A, and class C shadows a variable x that is also defined in classes A and B,

```
A(x) ->
|   B(x) ->
|   C(x)
```

then you can refer to these different variables from class C as follows:

```java
x          // Variable x in class C
this.x     // Variable x in class C
super.x    // Variable x in class B
((B) this).x // Variable x in class B
((A) this).x // Variable x in class A
```

But note this:

```
super.super.x // Illegal; does not refer to x in class A
```

You cannot refer to a shadowed variable x in the superclass of a superclass with super.super.x. Java does not recognize this syntax.

**Shadowed Methods?**

Just as a variable defined in one class can shadow a variable with the same name in a superclass, you might expect that a method in one class could shadow a method with the same name (and same arguments) in a superclass. In a sense, they do: "shadowed" methods are called *overridden* methods. But method overriding is significantly different than variable shadowing; it is discussed below.

**Overriding Methods**

When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class *overrides* the method in the superclass. When the method is invoked for an object of the class, it is the new definition of the method that is called, not the superclass’s old definition.

Method overriding is an important and useful technique in object-oriented programming. Suppose we define a class Ellipse of our Circle class. (This is admittedly a strange thing to do, since, mathematically, a circle is a kind of ellipse, and it is customary to derive a more specific class from a more general one. Nevertheless it is a useful example here). Then it would be important for Ellipse to override the area() and circumference() methods of Circle. Ellipse would have to implement new versions of these functions because the formulas that apply to circles don’t work for ellipses. Before we go any further with the discussion of method overriding, be sure that you understand the difference between method *overriding* and method *overloading*, which we discussed
Earlier. As you probably recall, method overloading refers to the practice of defining multiple methods (in the same class) with the same name but with differing argument lists. This is very different from method overriding, and it is important not to get them confused!

**Overriding Is Not Shadowing**

Although Java treats the variables and methods of a class analogously in many ways, method overriding is not like variable shadowing at all: You can refer to shadowed variables simply by casting an object to the appropriate type. You cannot invoke overridden methods with this technique, however.

The next example illustrates this crucial difference:

```java
class A {
    int i = 1;
    int f() { return i; }
}
class B extends A {
    int i = 2; // Shadows variable i in class A.
    int f() { return -i; } // Overrides method f in class A.
}
public class override_test {
    public static void main(String[] args) {
        B b = new B();
        System.out.println(b.i); // Refers to B.i; prints 2.
        System.out.println(b.f()); // Refers to B.f(); prints -2.
        A a = (A) b; // Cast b to an instance of class A.
        System.out.println(a.i); // Now refers to A.i; prints 1.
        System.out.println(a.f()); // Still refers to b.f(); prints -2.
    }
}
```

While this difference between method overriding and variable shadowing may seem surprising at first, a little thought makes the purpose clear. Suppose we have a bunch of Circle and Ellipse (a subclass of Circle here) objects that we are manipulating. To keep track of the circles and ellipses, we store them in an array of type `Circle[]`, casting all the Ellipse objects to Circle objects before we store them.

Then, when we loop through the elements of this array, we don’t have to know or care whether the element is actually a Circle or an Ellipse. What we do care very much about, however, is that the correct value is computed when we invoke the area() method of any element of the array. That is, we don’t want to use the formula for the area of a circle when the object is actually an ellipse.

Seen in this context, it is not surprising at all that method overriding is handled differently by Java than variable shadowing.

**final Methods**

If a method is declared `final`, it means that the method declaration is the "final" one – that it cannot be overridden. static methods and private methods cannot be overridden either, nor can the methods of a final class. (If a method cannot be overridden, the compiler may perform certain optimizations on it, too).

**Dynamic Method Lookup**

If we have an array of Circle and Ellipse Objects (or, say, Shapes,) how does the compiler know to call the Circle area() method or the Ellipse area() method for any given item in the array? The compiler does not
know this; it can’t. The compiler knows that it does not know, however, and produces code that uses ”dynamic method lookup” at run-time. When the interpreter runs the code, it looks up the appropriate area() method to call for each of the objects. That is, when the interpreter interprets the expression s.area(), it dynamically looks for an area() method associated with the particular object referred to by the variable s. It does not simply use the area() method that is statically associated with the type of the variable s.

So the actual type of the object is used not the type of the object reference.

class Point {
    int x, y;
    void clear() {
        // code for clearing a Point object
    }
}

// Pixel is a Point with a Color
class Pixel extends Point {
    Color color;
    void clear() {
        // code for clearing a Pixel object
    }
}

// ... somewhere in a method we have:
Point point = new Pixel();
point.clear(); // uses Pixel’s clear()

Dynamic method lookup is fast, but it is not as fast as invoking a method directly. Fortunately, there are a number of cases in which Java does not need to use dynamic method lookup. static methods cannot be overridden, so they are always invoked directly. private methods are not inherited by subclasses and so cannot be overridden by subclasses; this means the Java compiler can safely invoke them without dynamic method lookup as well. final methods are invoked directly for the same reason: they cannot be overridden. Finally, when a method of a final class is invoked through an instance of the class or a subclass of it, then it, too, can be invoked without the overhead of dynamic lookup.

Invoking an Overridden Method

We’ve seen the important differences between method overriding and variable shadowing. Nevertheless, the Java syntax for invoking an overridden method is very similar to the syntax for accessing a shadowed variable: both use the super keyword. The following example illustrates this:

class A {
    int i = 1;
    int f() { return i; } // A very simple method.
}
class B extends A {
    int i; // This variable shadows i in A.
    int f() {
        // This method overrides f() in A.
        i = super.i + 1; // This retrieves A.i this way.
        return super.f() + i; // And it invokes A.f() this way.
    }
}

Recall that when you use super to refer to a shadowed variable, it is the same as casting this to the superclass type and accessing the variable through that. On the other hand, using super to invoke an overridden method
is **not the same as casting** this. In this case, `super` has the special purpose of turning off dynamic method lookup and invoking the specific method that the superclass defines or inherits. (One needs to be aware of the limitations of the analogy with casting this in the case of referencing methods with `super`). So, please, make a note of it. In the example above we use `super` to invoke an overridden method that is actually defined in the immediate superclass. `super` also works perfectly well to invoke overridden methods that are defined further up the class hierarchy. This is because the overridden method is inherited by the immediate superclass, and so the `super` syntax does in fact refer to the correct method.

To make this more concrete, suppose class A defines method f, and that B is a subclass of A, and that C is a subclass of B that overrides method f.

\[ A(f) \rightarrow B \rightarrow C(f) \]

Then you can still use:

\[ super.f() \]

to invoke the overridden method from within class C. This is so because class B inherits method f from class A. If classes A, B and C all define method f, however, then calling `super.f()` in class C invokes class B's definition of the method. In this case, there is no way to invoke A.f() from within class C because `super.super.f()` is not legal Java syntax.

It is important to note that `super` can only be used to invoke overridden methods from within the class that does the overriding. With our `Circle` and `Ellipse` classes, for example, there is no way to write a program (with or without `super`) that invokes the `Circle area()`; method on an object of type `Ellipse`. The only way to do this is to use `super` in a method within the `Ellipse` class.

Finally, note that this form of `super` does not have to occur in the first statement in a method, as it does when used to invoke a superclass constructor method.

**Data Hiding and Encapsulation**

We started these review notes by describing a class as "a collection of data and methods". One of the important object-oriented techniques that we haven’t discussed so far is hiding the data within the class, and making it available only through the methods. This technique is often known as *encapsulation*, because it seals the class's data (and internal methods) safely inside the "capsule" of the class, where it can be accessed only by trusted users, i.e., by the methods of the class.

Why would you want to do this? The most important reason is to hide the internal implementation details of your class. If you prevent programmers from relying on those details, then you can safely modify the implementation without worrying that you will break existing code that uses the class.

Another reason for encapsulation is to protect your class against accidental or willful stupidity. A class often contains a number of variables that are interdependent and must be in a consistent state. If you allow a programmer (this may be you yourself) to manipulate those variables directly, (s)he may change one variable without changing important related variables, thus leaving the class in an inconsistent state. If, instead, (s)he had to call a method to change the variable, the method can be sure to do everything necessary to keep the state consistent.

Here’s another way to think about encapsulation: When all of a class’s variables are hidden, the class’s methods define the only possible operations that can be performed on objects of that class. Once you have carefully tested and debugged your methods, you can be confident that the class will work as expected. On the other hand, if all the variables can be directly manipulated, the number of possibilities you have to test becomes unmanageable.

There are other reasons to hide data, too:
• Internal variables that are visible externally to the class just clutter up your class’s API. Keeping visible variables to a minimum keeps your class tidy and elegant.

• If a variable is visible in your class, you have to document it. Save time by hiding it instead!

Visibility Modifiers

In most of our examples so far, you’ve probably noticed the public keyword being used When applied to a class, it means that the class is visible everywhere. When applied to a method or a variable, it means that the method or variable is visible everywhere the class is.

To hide variables (or methods for that matter) you just have to declare them private:

```java
public class Laundromat {
    // People can use this class.
    private boolean dirty;  // They can't see this internal variable,
    public void wash() { ... }  // but they can use these public methods
    public void dry() { ... }  // to manipulate the internal variable.
}
```

A private field of a class is visible only in methods defined within that class. (We do not discuss inner classes in A201, at least not that much).

Similarly, a private method may only be invoked by methods within the class. Private members are not visible within subclasses, and are not inherited by subclasses as other members are. Of course, non-private methods that invoke private methods internally are inherited and may be invoked by subclasses.

Besides public and private, Java has two other visibility levels: protected and the default visibility level, "package visibility", which applies if none of the public, private and protected keywords are used. These visibility levels are discussed below.

A protected member of a class is visible within the class where it is defined, of course, and within all subclasses of the class, and also within all classes that are in the same package as that class. You should use protected visibility when you want to hide fields and methods from code that uses your class, but want those fields and methods to be fully accessible to code that extends your class.

The default package visibility is more restrictive than protected, but less restrictive than private. If a class member is not declared with any of the public, private or protected keywords, then it is visible only within the class that defines it and within classes that are part of the same package. It is not visible to subclasses unless those subclasses are part of the same package.

A note about packages: A package is a group of related and possibly cooperating classes. All non-private members of all classes in the package are visible to all other classes in the package. This is OK because the classes are assumed to know about, and trust each other. The only time difficulty arises is when you write programs without a package statement. These classes are thrown into a default package with other package-less classes, and all their non-private members are visible throughout the package (The default package usually consists of all classes in the current working directory, and that’s what we do in this class).

(The rest of this Visibility Modifiers section for your reference only).

There is an important point to make about subclass access to protected members. A subclass inherits the protected members of its superclass, but it can only access those members through instances of itself, not directly in instances of the superclass. Suppose, for example, that A, B, and C are public classes, each defined in a different package, and that a, b, and c are instances of those classes. Let B be a subclass of A, and C be a subclass of B. Now, if A has a protected field x, then the class B inherits that field, and its method can use
this.x, b.x, and c.x. But it cannot access a.x. Similarly, if A has a protected method f(), then the methods of class B can invoke this.f(), b.f(), and c.f(), but they cannot invoke a.f().

The following table shows the circumstances under which class members of the various visibility types are accessible to other classes:

<table>
<thead>
<tr>
<th>Accessible to</th>
<th>public</th>
<th>protected</th>
<th>package</th>
<th>private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same class</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Class in same package</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Subclass (perhaps in different package)</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Non-subclass, different package</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Member Visibility

The details of member visibility in Java can become quite confusing. Here are some simple rules of thumb for using visibility modifiers:

- Use public only for methods and constants that form part of the public API of the class. Certain very important or very frequently used fields may also be public, but it is common practice to make fields non-public and encapsulate them with public accessor methods.

- Use protected for fields and methods that aren’t necessary to use the class, but that may be of interest to anyone creating a subclass as part of a different package.

- Use default package visibility for fields and methods that you want to be hidden outside of the package, but which you want cooperating classes within the same package to have access to.

- Use private for fields and methods that are only used inside the class and should be hidden everywhere else.

Note that you can't take advantage of package visibility unless you use the package statement to group your related classes into packages.

Data Access Methods

In the Circle example we've been using, we've declared the circle position and radius to be public fields. In fact, the Circle class is one where it may well make sense to keep those visible - it is a simple enough class, with no dependencies between the variables. On the other hand, suppose we wanted to impose a maximum radius on objects of the Circle class. Then it would be better to hide the r variable so that it could not be set directly. Instead of a visible r variable, we'd implement a setRadius() method that verifies that the specified radius isn't too large and then sets the r variable internally. The example that follows shows how we might implement Circle with encapsulated data and a restriction on radius size. For convenience, we use protected fields for the radius and position variables. This means that subclasses of Circle, or cooperating classes within the shapes package are able to access these variables directly. To any other classes, however, the variables are hidden. Also, note the private constant and method used to check whether a specified radius is legal. And finally, notice the public methods that allow you to set and query the values of the instance variables.

```java
package shapes;
// Specify a package for the class.
public class Circle {
    // Note that the class is still public!
    protected double x, y;
    // Position is hidden, but visible to subclasses
```
protected double r;
// Radius is hidden, but visible to subclasses
private static final double MAXR = 100.0;
// Maximum radius (constant).
private boolean check_radius(double r) {
    return (r <= MAXR);
}

// Public constructors
public Circle (double x, double y, double r) {
    this.x = x; this.y = y;
    if (check_radius(r))
        this.r = r;
    else
        this.r = MAXR;
}
public Circle (double r) { this (0.0, 0.0, r); }
public Circle () { this (0.0, 0.0, 1.0); }

// Public data access methods
public void moveTo(double x, double y) {
    this.x = x; this.y = y;
}
public void move (double dx, double dy) {
    x += dx; y += dy;
}
public void setRadius(double r) {
    this.r = (check_radius(r))?r:MAXR; /* a-ha! */
}
// Declare these trivial methods final so we don’t get dynamic
// method lookup and so that they can be optimized by the compiler.
public final double getX () { return x;}
public final double getY () { return y;}
public final double getRadius () { return r;}
}

And here’s the last part of this overview.

Abstract Classes and Methods: A Brief Tutorial

1. An abstract method has no body, only a signature followed by a semicolon.
   For example:

        abstract double area();

2. Any class with an abstract method is automatically abstract itself, and must be declared as such. So we need the blue keyword below, it just has to be there:

        abstract class Shape {
            abstract double area();
        }

3. A class may be declared `abstract` even if it has no `abstract` methods. This prevents it from being instantiated.

For example:

```java
oldschool.cs.indiana.edu%ls -l
total 1
-rw------- 1 dgerman 134 Jul 16 12:28 Example.java
oldschool.cs.indiana.edu%cat Example.java
abstract class Shape {
    double area() { return -1; }
    public static void main(String[] args) {
        Shape a = new Shape();
    }
}
oldschool.cs.indiana.edu%javac Example.java
Example.java:4: class Shape is an abstract class. It can't be instantiated.
    Shape a = new Shape();
   ~
1 error
oldschool.cs.indiana.edu%
```

4. An `abstract` class cannot be instantiated.

This could be seen in the example above.

It can however have a `main` method with no problem:

```java
oldschool.cs.indiana.edu%ls -l
total 1
-rw------- 1 dgerman 172 Jul 16 12:33 Example.java
oldschool.cs.indiana.edu%cat Example.java
abstract class Shape {
    double area() { return -1; }
    public static void main(String[] args) {
        System.out.println("Hello!");
        // Shape a = new Shape();
    }
}
oldschool.cs.indiana.edu%javac Example.java
oldschool.cs.indiana.edu%java Shape
Hello!
oldschool.cs.indiana.edu%
```

5. A subclass of an `abstract` class can be instantiated if it overrides each of the `abstract` methods of its superclass and provides an implementation (i.e., a method body) for all of them.

Here's an example that does that and notice the inherited variables too.

```java
oldschool.cs.indiana.edu%ls -l
total 1
-rw------- 1 dgerman 511 Jul 16 12:40 Example.java
oldschool.cs.indiana.edu%cat Example.java
```
abstract class Shape {
    int x, y;
    abstract double area();
}

class Circle extends Shape {
    int radius;
    double area() { return 2 * Math.PI * radius * radius; }
    double manhattanDistanceToOrigin() {
        return Math.abs(x) + Math.abs(y);
    }
}

class Tester {
    public static void main(String[] args) {
        Circle c = new Circle();
        c.radius = 10;
        c.x = -3; c.y = 5;
        System.out.println("Area is " + c.area() +
        " and the distance is " + c.manhattanDistanceToOrigin());
    }
}
oldschool.cs.indiana.edu%javac Example.java
oldschool.cs.indiana.edu%java Tester
Area is 628.3185307179587 and the distance is 8.0
oldschool.cs.indiana.edu%

6. If a subclass of an abstract class does not implement all of the abstract methods it inherits, that subclass is itself abstract.

To see this in the code above remove the definition of area() in class Circle and recompile:

oldschool.cs.indiana.edu%ls -l
total 1
-rw------- 1 dgerman 577 Jul 16 12:44 Example.java
oldschool.cs.indiana.edu%cat Example.java
abstract class Shape {
    int x, y;
    abstract double area();
}

class Circle extends Shape {
    int radius;
    //*** let's take area() out, see if it still compiles:
    double area() { return 2 * Math.PI * radius * radius;
    //***
    double manhattanDistanceToOrigin() {
        return Math.abs(x) + Math.abs(y);
    }
}
class Tester {
    public static void main(String[] args) {
        Circle c = new Circle();
        c.radius = 10;
        c.x = -3; c.y = 5;
        System.out.println("Area is " + c.area() +
                           " and the distance is " + c.manhattanDistanceToOrigin());
    }
}

Javac Example.java
Example.java:6: class Circle must be declared abstract.
It does not define double area() from class Shape.
class Circle extends Shape {

   Example.java:18: class Circle is an abstract class. It can’t be instantiated.
   Circle c = new Circle();

2 errors

oldschool.cs.indiana.edu%

7. It doesn’t compile, and for two reasons, not just one.
8. But the two reasons are closely related.
9. I hope you enjoyed this tutorial.
10. I strongly hope the information presented in it was quite manageable.
11. Please let me know if you have any questions.
Threads

We start with this program:

```java
class One {
    public static void main(String[] args) {
        Two larry = new Two("Larry Bird", 7000);
        Two michael = new Two("Michael Jordan", 3000);
        // Thread one = new Thread(larry);
        // Thread two = new Thread(michael);
        System.out.println("Program started on " + new java.util.Date());
        larry.run();
        michael.run();
        // one.start();
        // two.start();
    }
}
class Two /*implements Runnable*/ {
    String name;
    long delay;
    Two (String name, long delay) {
        this.name = name;
        this.delay = delay;
    }
    public void run() {
        try {
            Thread.sleep(delay);
        } catch (Exception e) {
        }
        System.out.println(name + " at " + new java.util.Date());
    }
}
```

Run this a few times (here’s once):

```
frilled.cs.indiana.edu%javac One.java
frilled.cs.indiana.edu%java One
Program started on Tue Aug 07 19:02:27 EDT 2001
```
Larry Bird at Tue Aug 07 19:02:34 EDT 2001  
Michael Jordan at Tue Aug 07 19:02:37 EDT 2001  
frilled.cs.indiana.edu%

Then we make a few changes (in blue).

```java
class One {
  public static void main(String[] args) {
    Two larry = new Two("Larry Bird", 7000);
    Two michael = new Two("Michael Jordan", 3000);
    Thread one = new Thread(larry);
    Thread two = new Thread(michael);
    System.out.println("Program started on " + new java.util.Date());
    /* larry.run(); */
    /* michael.run(); */
    one.start();
    two.start();
  }
}
class Two implements Runnable {
  String name;
  long delay;
  Two (String name, long delay) {
    this.name = name;
    this.delay = delay;
  }
  public void run() {
    try {
      Thread.sleep(delay);
    } catch (Exception e) { }
    System.out.println(name + " at " + new java.util.Date());
  }
}
```

If you run it, here's what it looks like:

```
frilled.cs.indiana.edu%java One
Program started on Tue Aug 07 19:08:45 EDT 2001
Michael Jordan at Tue Aug 07 19:08:49 EDT 2001
Larry Bird at Tue Aug 07 19:08:53 EDT 2001
frilled.cs.indiana.edu%
```

Now time runs simultaneously for both.

So we make one last change.
class One {
    public static void main(String[] args) {
        Two larry = new Two("Larry Bird", 7000);
        Two michael = new Two("Michael Jordan", 3000);

        Thread one = new Thread(larry);
        Thread two = new Thread(michael);

        System.out.println("Program started on " + new java.util.Date());

        one.start();
        two.start();
    }
}

class Two implements Runnable {
    String name;
    long delay;
    Two (String name, long delay) {
        this.name = name;
        this.delay = delay;
    }
    public void run() {
        while (true) {
            try {
                Thread.sleep(delay);
            } catch (Exception e) {
            }
            System.out.println(name + " at " + new java.util.Date());
        }
    }
}

Here's how it runs.

```
frilled.cs.indiana.edu%java One
Program started on Tue Aug 07 19:13:58 EDT 2001
Michael Jordan at Tue Aug 07 19:14:01 EDT 2001
Michael Jordan at Tue Aug 07 19:14:04 EDT 2001
Larry Bird at Tue Aug 07 19:14:05 EDT 2001
Michael Jordan at Tue Aug 07 19:14:07 EDT 2001
Michael Jordan at Tue Aug 07 19:14:10 EDT 2001
Larry Bird at Tue Aug 07 19:14:12 EDT 2001
Michael Jordan at Tue Aug 07 19:14:13 EDT 2001
Michael Jordan at Tue Aug 07 19:14:16 EDT 2001
Larry Bird at Tue Aug 07 19:14:19 EDT 2001
Michael Jordan at Tue Aug 07 19:14:19 EDT 2001
Michael Jordan at Tue Aug 07 19:14:22 EDT 2001
Michael Jordan at Tue Aug 07 19:14:25 EDT 2001
Larry Bird at Tue Aug 07 19:14:26 EDT 2001
Michael Jordan at Tue Aug 07 19:14:28 EDT 2001
```
So that's another way you can implement *threads*.

This particular way is quite convenient because

- one cannot extend more than one class, but
- one can implement any number of interfaces, and
- you sometimes want your applet to be its own broker, while
- also providing the `run()` method of a related *thread*.

Let's see an example below.

What is a sprite?

```bash
tw9frilled.cs.indiana.edu%webster sprite
sprite \'sprai-t n
[ME sprit, fr. MF esprit, fr. L spiritus spirit -- more at SPIRIT]
(14c)
1a archaic: SOUL
1b: a disembodied spirit: GHOST
2a: [ELF], FAIRY
2b: an elfish person
```

A sprite is an elf.

```bash
tw9frilled.cs.indiana.edu%webster elf
elf \'elf n, pl elves \'elvz\n[ME, fr. OE aelf; akin to ON alfr elf & prob. to L albus white -- more at ALB]
(bef. 12c)
1: a small often mischievous fairy
2a: a small lively creature; esp: a mischievous child
2b: a usu. lively mischievous or malicious person
```
-- elf-ish el-fish adj
-- elf-ish-ly adv
frilled.cs.indiana.edu%

Sprites are

• figures or elements on the screen that
• have the capability of moving independently of one another.

These elements could be

• text
• graphics
• or bitmaps

Bitmaps are preformed images that can be pasted on the screen.
Here are ten bitmaps that have been designed with a clear goal in mind.


You’d have to agree that the artist did a very good job.
Let’s describe a Sprite. Here’s a very general description.

```java
abstract class Sprite {
    protected boolean visible;
    public boolean isVisible() {
```
return this.visible;
}
public void setVisible(boolean visibility) {
    this.visible = visibility;
}
protected boolean active;
boolean isActive() {
    return this.active;
}
void setActive(boolean howActive) {
    this.active = howActive;
}
abstract void paint(Graphics g);
abstract void update();
public void suspend() {
    this.setVisible(false);
    this.setActive(false);
}

public void restore() {
    this.setVisible(true);
    this.setActive(true);
}
}

Now the questions are:

- what is an abstract class?
- what is protected?
- what is an abstract method?

An abstract class is an unfinished class.
It can't be instantiated, and can leave methods unimplemented.
For example our old inheritance example could be reworked as follows.

class Example {
    public static void main(String[] args) {
        Shape[] shapes = new Shape[2];
        shapes[0] = new Circle(50, 50, 1);
        shapes[1] = new Rectangle(100, 100, 2, 3);
        for (int i = 0; i < shapes.length; i++) {
            System.out.println(shapes[i].area());
        }
    }
}
abstract class Shape {
    int x, y;
    Shape(int x, int y) {

```java
    this.x = x; this.y = y;
}
abstract double area();
}
class Circle extends Shape {
    int radius;
    Circle(int x, int y, int radius) {
        super(x, y);
        this.radius = radius;
    }
    double area() {
        return Math.PI * this.radius * this.radius;
    }
}
class Rectangle extends Shape {
    int width, height;
    Rectangle(int x, int y, int width, int height) {
        super(x, y);
        this.width = width;
        this.height = height;
    }
    double area() {
        return width * height;
    }
}
```

If you want to store circles and rectangles in an array you need to be able to look at them uniformly, so you need to use a class that abstracts the notion of shape (or, if you want, geometrical figure).

Then if you want to be able to simply compute the areas by invoking a method with that name (area) without any casting you need to have a method with that name in class Shape.

Otherwise you’d need to cast or you would get an error.

Note that since you’re overriding the area() method in the subclasses (Circle and Rectangle) the method defined in the class of the object (as opposed to the one defined in the class of the reference) will be executed when invoked. So areas are computed properly.

But what should the area() method in class Shape look like?

It should look like "not enough information to compute" and that’s what it looks like in our example.

(If don’t feel comfortable with the example, please refer to Lecture Notes Twenty-Eight54.)

Now let’s become more concrete in our definitions.

```java
    class BitmapSprite extends Sprite {
        protected int locx;
        protected int locy;
        protected Image image;
        protected Applet applet;
        public BitmapSprite(int x, int y, Image i, Applet a) {
```
lo cx = x;
lo cy = y;
image = i;
applet = a;
this.restore();
}
protected int width, height;
public void setSize(int w, int h) {
    width = w;
    height = h;
}
public void update() {
    /* do nothing */
}
pull public void paint(Graphics g) {
    if (this.visible) {
        if (image != null)
            g.drawImage(image, locx, locy, applet);
    }
}
}

Now let's define an interface.

interface Moveable {
    public abstract void setPosition(int x, int y);
    public abstract void setVelocity(int x, int y);
    public abstract void updatePosition();
}

And let's put all these definitions to work in our new gizmo.

class BitmapLoop extends BitmapSprite implements Moveable {
    protected Image images[];
    protected int currentImage;
    protected boolean foreground;
    public BitmapLoop (int x, int y,
        Image background,
        Image foreground[],
        Applet applet)
    {
        super(x, y, background, applet);
        this.width = applet.getWidth();
        this.height = applet.getHeight();
        this.images = foreground;
        this.currentImage = 0;
        if (this.images.length == 0) {
            this.foreground = false;
        } else {
            this.foreground = true;
        }
    }
}
} 
public void update() {
    if (this.active && this.foreground) {
        this.currentImage = (this.currentImage + 1) % this.images.length;
    }
    this.updatePosition();
}
public void paint(Graphics g) {
    if (this.visible) {
        if (this.image != null)
            g.drawImage(this.image, this.locx, this.locy, this.applet);
        if (this.foreground) {
            g.drawImage(this.images[currentImage],
                this.locx,
                this.locy,
                this.applet // the ImageObserver
            );
        }
    }
}
public void setPosition(int x, int y) {
    this.locx = x;
    this.locy = y;
}
protected int vx, vy;
public void setVelocity(int x, int y) {
    this.vx = x;
    this.vy = y;
}
public void updatePosition() {
    this.locx += vx;
    this.locy += vy;
    this.vx = 0;
    this.vy = 0;
}

Now let's put this to use.

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class UFOControl extends Applet implements Runnable, KeyListener {
    Graphics offscreen;
    Image offscreenImage;
    static final int NUM_SPRITES = 1;
    static final int REFRESH_RATE = 80; // in milliseconds...
    Sprite sprites[];
    int width, height;
    public void init() {
        System.out.println("*** init *** ");
        }
this.setBackgroundColor(Color.white);
this.width = this.getBounds().width;
this.height = this.getBounds().height;
this.initSprites();
this.offscreenImage = this.createImage(width, height);
this.offscreen = this.offscreenImage.getGraphics();
this.addKeyListener(this);
}
public void keyPressed (KeyEvent e) {
    System.out.println("Key Pressed");
    switch (e.getKeyCode()) {
        case KeyEvent.VK_UP:
            System.out.println("Up!");
            (Moveable)this.sprites[0]).setVelocity(0, -3);
            break;
        case KeyEvent.VK_RIGHT:
            System.out.println("Right!");
            (Moveable)this.sprites[0]).setVelocity(3, 0);
            break;
        case KeyEvent.VK_DOWN:
            System.out.println("Down!");
            (Moveable)this.sprites[0]).setVelocity(0, 3);
            break;
        case KeyEvent.VK_LEFT:
            System.out.println("Left!");
            (Moveable)this.sprites[0]).setVelocity(-3, 0);
            break;
    }
}
public void keyReleased(KeyEvent e) {
    /* do nothing */
}
public void keyTyped (KeyEvent e) {
    /* do nothing */
}
public void initSprites() {
    sprites = new Sprite[NUM_SPRITES];
    Image backImage = null;
    Image foreImage[] = new Image[10]; // ten Duke bitmaps
    MediaTracker t = new MediaTracker(this); // need tutorial on this
    try {
        System.out.println("Getting started.");
        for (int i = 1; i <= 10; i++) {
            String imageName = "images/T" + i + ".gif";
            foreImage[i - 1] = getImage(getCodeBase(), imageName);
            t.addImage(foreImage[i - 1], 0);
        }
        System.out.println("Loading images.");
        t.waitForAll();
    } catch (Exception e) {
System.out.println(e);
return;
}
if (t.isErrorAny()) {
    System.out.println("error");
} else if (t.checkAll()) {
    System.out.println("successfully loaded.");
}
sprites[0] = new BitmapLoop(13, 17, backImage, foreImage, this);
}
Thread animation;
public void start() {
    System.out.println("*** start ***");
    this.animation = new Thread(this);
    if (this.animation != null) {
        this.animation.start();
    }
}
public void run() {
    while (true) {
        this.repaint();
        this.updateSprites();
        try {
            Thread.sleep(REFRESH_RATE);
        } catch (Exception e) {
        }
    }
}
public void stop() {
    System.out.println("*** stop ***");
    if (this.animation != null) {
        this.animation.stop(); // this approach is now deprecated...
        this.animation = null;
    }
}
public void updateSprites() {
    for (int i = 0; i < sprites.length; i++) {
        this.sprites[i].update();
    }
}
public void update(Graphics g) {
    this.paint(g);
}
public void paint(Graphics g) {
    offscreen.setColor(Color.white); // why?
    offscreen.fillRect(0, 0, this.width, this.height);
    for (int i = 0; i < sprites.length; i++) {
        this.sprites[i].paint(offscreen);
    }
    g.drawImage(this.offscreenImage, 0, 0, this);
Now let’s put together the HTML.

```html
<html>
<head>
    <title>Bitmap Loop</title>
</head>
<body bgcolor:white>
    <applet code="UF0Control.class" width=300 height=300>
    </applet>
</body>
</html>
```

Also because of `initSprites()` make sure you have the following:

```
frilled.cs.indiana.edu%pwd
/nfs/moose/home/user3/dgerman/lect28
frilled.cs.indiana.edu%ls -ld *
drwx-------  2 dgerman    512 Aug 5 23:50 images
frilled.cs.indiana.edu%du -a
 2 ./images/T1.gif
 2 ./images/T2.gif
 2 ./images/T3.gif
 2 ./images/T4.gif
 2 ./images/T5.gif
 2 ./images/T6.gif
 2 ./images/T7.gif
 2 ./images/T8.gif
 2 ./images/T9.gif
 2 ./images/T10.gif
21 ./images
22 .
frilled.cs.indiana.edu%
```

So now you’re ready to go.

Create a file `UF0Control.java` and put everything in it.

Then compile and run the appletviewer on the HTML file.

![NOTE](https://example.com) **NOTE**: To get the above to compile and run you only need to `import` the right packages.

(See, do it with care and good luck).
You’ve reached the end of this course. A202 is up next for you. I have a few more documents on the web site that I would like to bring to your attention, but other than that, if you understood all that we have done thus far—you’re ready for A202, and more. Here’s what couldn’t fit in this book:

1. Sample Midterm Exam from Summer 2001\textsuperscript{55} (with solutions)
2. Practice Quizzes One\textsuperscript{56}, Two\textsuperscript{57}, Three\textsuperscript{58}, Four\textsuperscript{59}, Five\textsuperscript{60}, Six\textsuperscript{61}, Seven\textsuperscript{62}, Eight\textsuperscript{63}, Nine\textsuperscript{64}, Ten\textsuperscript{65}
3. Homework Assignment Five\textsuperscript{66}
4. Lecture Notes Twenty-Nine\textsuperscript{67} (A Summary, Then Threads).
5. Lecture Notes Thirty\textsuperscript{68} (Exceptions and Basic I/O)
6. Lecture Notes Thirty-One\textsuperscript{69} (A video Game in Java. The Alien Landing Game)
7. Lab Fourteen\textsuperscript{70} (IceBlocx)
8. An Older Lab Ten\textsuperscript{71} (Classes and Objects Revisited)
9. Solutions\textsuperscript{72} to the Older Lab Ten problems.

\begin{itemize}
\item \textsuperscript{55}http://www.cs.indiana.edu/classes/a201-dger/sum2001/exams/midterm.html
\item \textsuperscript{56}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceOne.html
\item \textsuperscript{57}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceTwo.html
\item \textsuperscript{58}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceThree.html
\item \textsuperscript{59}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceFour.html
\item \textsuperscript{60}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceFive.html
\item \textsuperscript{61}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceSix.html
\item \textsuperscript{62}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceSeven.html
\item \textsuperscript{63}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceEight.html
\item \textsuperscript{64}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceNine.html
\item \textsuperscript{65}http://www.cs.indiana.edu/classes/a201-dger/sum2002/practiceTen.html
\item \textsuperscript{66}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/h5Five.html
\item \textsuperscript{67}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/TwentyNine.html
\item \textsuperscript{68}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/Thirty.html
\item \textsuperscript{69}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/ThirtyOne.html
\item \textsuperscript{70}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/labFourteen.html
\item \textsuperscript{71}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/review.html
\item \textsuperscript{72}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/reviewSol.html
\end{itemize}
10. First Warmups\textsuperscript{73} with Solutions\textsuperscript{74}

11. An Older Set of Lecture Notes Sixteen\textsuperscript{75} (Anticipating the Java Fandango Notes)

12. Spring 2001 Final Exam\textsuperscript{76}

13. Summer 2001 Final Exam\textsuperscript{77}

14. Spring 2002 Final Exam\textsuperscript{78}

15. Appendix A: Miscellaneous Exercises\textsuperscript{79}

16. What’s New\textsuperscript{80} Page from Spring 2003

Here’s a program:

```java
class StrawDispenser {
    final static CAPACITY = 100;
    int balance = StrawDispenser.CAPACITY;

    Straw dispense() {
        this.balance -= 1;
        return new Straw();
    }

    int getBalance() {
        return this.balance;
    }

    void refill() {
        this.balance = StrawDispenser.CAPACITY;
    }
}
```

Admittedly, this looks more like a Bubble Machine (if it makes the Straws as it goes).

\textsuperscript{73}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/warmOne.html
\textsuperscript{74}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/warmOneS.html
\textsuperscript{75}http://www.cs.indiana.edu/classes/a201-dger/sum2000/lectures/Sixteen.html
\textsuperscript{76}http://www.cs.indiana.edu/classes/a201-dger/spr2001/finsol.html
\textsuperscript{77}http://www.cs.indiana.edu/classes/a201-dger/sum2001/scme.html
\textsuperscript{78}http://www.cs.indiana.edu/classes/a201-dger/spr2002/exams/One.html
\textsuperscript{79}http://www.cs.indiana.edu/classes/a201-dger/sum2002/notes/misc.html
\textsuperscript{80}http://www.cs.indiana.edu/classes/a201-dger/spr2003/whatsnew.html
Here’s another one:

class NumberTwo {
    Sandwich a; // Larry Bird is from Freedom Lick, IN...
    Fries f;
    Drink d;
    NumberTwo(int size, String drink) {
        a = new Sandwich("cheeseburger");
        f = new Fries(size);
        d = new Drink(drink, size);
    }
}

Last two, of which the very last is a challenge.

class Point {
    double x;
    double y;
    Point(double a, double b) {
        this.x = a;
        this.y = b;
    }
    double dist(Point other) {
        double dX = this.x - other.x,
        dY = this.y - other.y;
        return Math.sqrt(dX * dX + dY * dY);
    }
}

class Triangle {
    Point a, b, c;
    Triangle (double x1, double y1,
                double x2, double y2,
                double x3, double y3) {
        this.a = new Point(x1, y1);
        this.b = new Point(x2, y2);
        this.c = new Point(x3, y3);
    }
    double area() {
        double m = a.dist(b), n = b.dist(c), q = a.dist(c),
                  s = (m + n + q) / 2;
        return Math.sqrt(s * (s - m) * (s - n) * (s - q));
    }
}

class Experiment {
    public static void main(String[] args) {
        Triangle a = new Triangle(0, 0, 0, 3, 4, 0);
        System.out.println(a.area()); // prints 3 * 4 / 2 (that is: 6 (six))
        System.out.println((new Point(0, 3)).dist(new Point(4, 0))); // (5.0)
class R {
    public static void main(String[] args) {
        String b = " System.out.println("class R {";\n" + 
        " System.out.println(" public static void main(String[] args) {";\n"
        " System.out.println(" String b = ";\n"
        " for (int i = 0; i < b.length(); i++) {\n" + 
        " char c = b.charAt(i);\n"
        " if (c == '\n') System.out.print("\n\n\n" + \n        " else { if (c == '\\\n' || c == '\\\n')\n" + 
        " System.out.print('\\n');\n" + 
        " System.out.print(c);\n" + 
        " }\n" + 
        " }\n" + 
        " System.out.println("\n\n\n";\n"
        " System.out.println(b + "\n}\n\";\n"
        " System.out.println("class R {";\n        System.out.println(" public static void main(String[] args) {";\n        System.out.println(" String b = ";\n        for (int i = 0; i < b.length(); i++) {
            char c = b.charAt(i);
            if (c == '\n') System.out.print("\n\n\n";\n"
            else { if (c == '\\\n' || c == '\\\n')\n                System.out.print('\\n');\n                System.out.print(c);\n            }\n        }
        System.out.println("\n\n";\n"
        System.out.println(b + "\n}\n\";\n"
    }
}

What's this program doing?

Can you write a shorter one?