JavaOne℠

Sun's 2002 Worldwide Java Developer Conference℠

Building a High Performance Multiplayer Internet Game—With Java™ Technology

John G. Miller, Jr.
President
Digital Gamers
www.digigamers.com

Overall Presentation Goal

Learn how Digital Gamers designed and built a client for the Java™ platform ("Java client") that mitigates the challenges inherent in building a multiplayer networked role playing game

Learning Objectives

- As a result of this presentation, you will be able to:
  - Understand and design for network bandwidth constraints
  - Use the J2SE™ 1.4 release VolatileImage and full screen mode to animate a user interface
  - Use MIDI sequences and sampled sound to add atmosphere to your application

Speaker's Qualifications

- Designed and built enterprise systems for 15 years
- Co-architected and helped build the first system in the U.S. to successfully integrate state justice agencies
- Principal/co-inventor of Sybase's Relational Beans and IQNavigator's resource matching system (US Patent Application Nos 09/855,767 and 60/180,421)

Is He MAD?

Java Technology for Games?

Who in their right mind would ever consider developing games using Java technology?

Java Technology for Games!

Before the current console shift from 32/64 bit technology to 128 bit technology is complete, entertainment software development companies will have lost hundreds of millions of dollars in revenues

Presentation Agenda

- How is Java technology being used for game development right now?
- What are the challenges of a multiplayer network game?
- Designing to accommodate network limitations
- Animating the client
- Adding sound and music

Gaming in Java Technology Today, in the Industry and at Digital Gamers

Java Technology in Game Development

- Sun participation:
  - Creation and continuing evolution of the Java 3D™ API
  - Performance-oriented additions to the J2SE™ 1.4 release (VolatileImage, BufferStrategy, hardware acceleration of BufferedImage, java.nio)

Java Technology in Game Development

- Industry movement toward Java technology:
  - Red Storm Entertainment titles
  - Electronic Arts' Majestic
  - Integration of Java platform into the Playstation 2 system

Java Technology in Game Development

- Community contributions:
  - Java™ Game Profile JSR 1340—defines a Games Profile for the J2ME platform
  - www.javagaming.org—an emerging community of game developers using Java technology
  - GL for Java™, Jsparrow—some of the efforts to provide Java technology bindings for OpenGL

Challenges in Multiplayer Network Gaming

- Network challenges:
  - Bandwidth: Ranges from dialup speeds to broadband
  - Reliability: Lost/garbled packets
  - Delivery: Disordered/delayed packets
  - Synchronization: Keeping multiple clients in synch, especially in “twitch-and-flex” games; we will not address this here
- … and then there is security

Challenges in Multiplayer Network Gaming

- Server-side challenges:
  - Client management: Separate or shared connections/threads for clients?
  - Object management: Locking in multi-threaded environment, “autonomous objects”
  - “Unpredictable” utilization patterns
  - Network bandwidth and spamming
  - Persistence
- We will not address server-side issues here

Challenges in Multiplayer Network Gaming

- Client-side challenges:
  - Client responsiveness: Does the UI continue to respond during intensive graphics/sound/network/disk operations
  - Unreliable network behavior: Does the client respond gracefully to lost/out-of-sequence packets?
  - High-speed rendering: Double buffering, page flipping, and making BufferedImage and VolatileImage play nicely together

Challenges in Multiplayer Network Gaming

- Social challenges:
  - Problem children: Nuisances, abusers, wreckers, and predators
  - Perceived value: Does the product offer enough functionality and content to justify the cost to the customer?
  - Customer support: Do support mechanisms leave the customer satisfied?
- Most social challenges cannot be directly addressed with technology

Addressing Network Challenges

- Bandwidth
  - What do you send?
  - How do you send it?
- Reliability
  - How do you handle lost/destroyed packets?
- Delivery
  - How do you handle out of sequence packets?

Addressing Network Challenges

- First guideline: Minimize what you send
  - Common sense rule broken by more than one commercial developer
  - Example: a role playing game takes place in a virtual world—store as much of the model for that world on the client as possible
  - Example: try to not push media to the client —store as much on the client as possible
- Ideally you will only send dynamic data

Addressing Network Challenges

- Second guideline: Minimize the size of what you send
  - Even if you are only sending information on dynamic world objects, a client may still be receiving messages for hundreds of objects
  - Example: instead of sending the complete new state for an object undergoing a state change, consider sending only that portion of the state which is changing

Addressing Network Challenges

- Third Guideline: Send everything to the client
  - Send both the before image and the after image for whatever state is changing
  - Allow for initial transmission of entire object state, and retransmission on demand
- Counterpoint: differential transmission
  - Only delta for state change is sent
  - Automatic periodic retransmission of entire object

Addressing Network Challenges

- Fourth guideline: Create your own sequenced protocol on top of UDP
  - Object id to tell the client what the message applies to
  - Sequence number that is tracked by client to detect earlier messages
  - Sequence number may be used to detect and ignore superceded messages

Addressing Network Challenges

- Fifth guideline: Create separate command and data network channels
  - TCP for command channel
  - Two-way communication on command channel (client command/optional server response)
  - UDP for data channel
  - One-way communication on data channel (server transmits/multicasts state changes to interested clients)

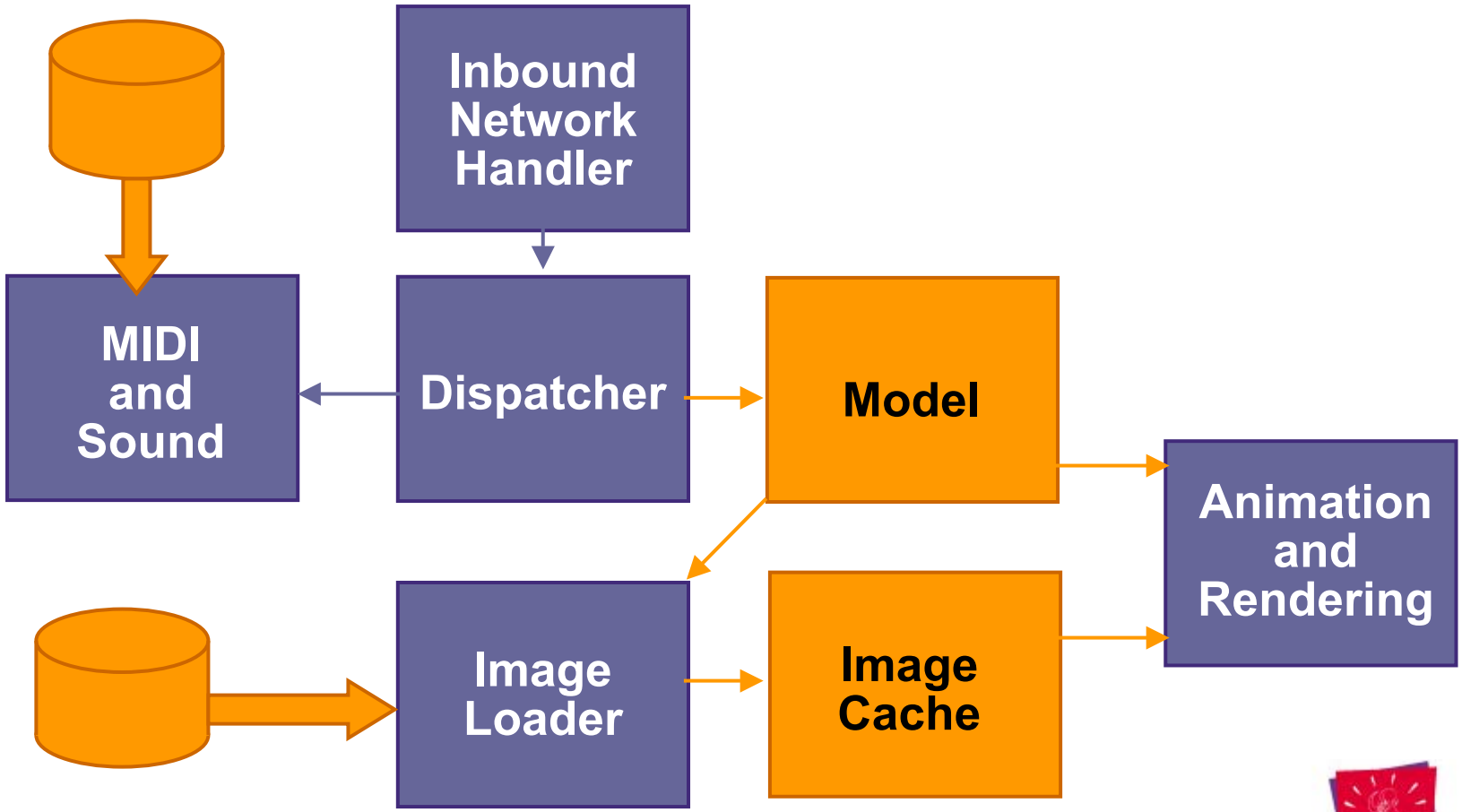
Addressing Client Challenges

- UI responsiveness
  - Multiple potential UI blockers: Disk access, network io, animation, sound
  - Place blockers on separate threads from UI
  - Perform animation with a `java.util.TimerTask` or a `javax.swing.ActionListener`
  - Use `wait()/notify()` in audio threads to wait for midi or sound samples to be queued up

Addressing Client Challenges

- UI responsiveness (cont.)
  - Use the new `java.nio` packages for file and network reads
  - `java.nio` provides a clean mechanism for interrupting blocking socket operations
  - Use blocking reads on the SocketChannel in the thread managing the socket channel
  - Consider placing file io under control of a thread and using `wait()/notify()` to queue and service read-ahead requests

Client Architecture Block Diagram



Addressing Client Challenges

- Handling network unreliability
  - Network guidelines are used to create a “self-correcting” network protocol
  - Objects broken into substates; substate changes are “all-or-nothing”
  - Incremental substate change with before and after image allows updates to model; before image provides sanity check
  - Sequence number ensures earlier message does not overwrite substate set by later message

Addressing Client Challenges

- Handling network unreliability (cont.)
  - Lost messages are not a concern as full substate is sent in in each new message for substate
  - Issue: It is possible to lose a substate message for an infrequently transmitted substate
  - Server checks client commands for indications of possible substate corruption and refreshes object state as needed

The BIG Challenge: Rendering

The BIG Challenge: Rendering

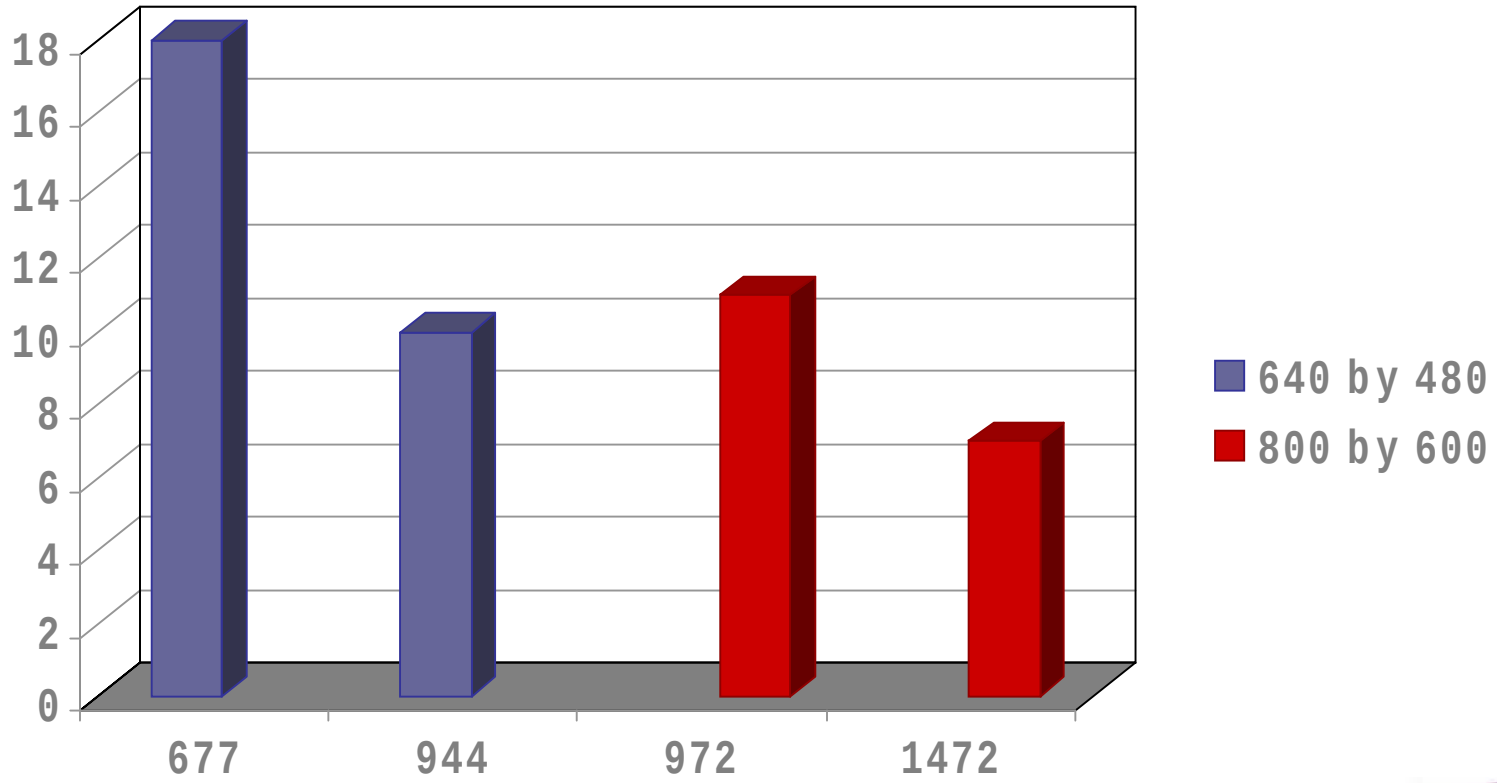
- High speed rendering
  - In almost any game, fast graphics are a must
  - We blit partially transparent images (sprites) on top of each other to build the display
  - BufferedImage is great for image manipulation (such as alpha compositing), but horrible for high speed rendering (such as blitting)

The BIG Challenge: Rendering

- High speed rendering
  - BufferedImage (prior to the J2SE 1.4 release) did not take advantage of accelerated graphics
  - Frame rate as a function of blitted elements suffers greatly

BufferedImage Performance

Frames per second vs. Blits per Frame



The BIG Challenge: Rendering

- The J2SE 1.4 release to the rescue—sort of
  - VolatileImage takes advantage of accelerated graphics; useful for accelerated area copies, rectangle fills
  - However…if VolatileImage is used in any complex image manipulation, the benefits of acceleration are lost
- Alpha compositing is a complex image manipulation!
- Consequence: we have an accelerated surface, but we cannot blit sprites to it!

Question

So how do we overlay partially transparent BufferedImage instances onto a VolatileImage?

Answer

We use BITMASK transparency instead of TRANSLUCENT transparency.

The BIG Challenge: Rendering

- Obtain your BufferedImage from createCompatibleImage()
  - Get a `java.awt.GraphicsConfiguration` from a class extending `java.awt.Window`
  - Call `createCompatibleImage(width,height, Transparency.BITMASK)` on the `GraphicsConfiguration`
  - The returned object is a BufferedImage with BITMASK transparency

The BIG Challenge: Rendering

- Where is the bitmask?
  - Examining the ColorModel for a BufferedImage obtained from createCompatibleImage reveals a one-bit alpha channel
  - This bit is the mask bit in the BufferedImage
  - Use your image manipulation of choice to set the mask bit to the desired value

The BIG Challenge: Rendering

- Code fragment from somewhere within an instance of java.awt.Window

```
GraphicsConfiguration gc =
   this.getGraphicsConfiguration;
BufferedImage sprite =
   gc.createCompatibleImage(width,height,
   Transparency.BITMASK);
// some code to set BufferedImage here
for (int x = 0; x < sprite.getWidth(); x++)
   for (int y = 0; y < sprite.getHeight(); y++)
   {
      int color = sprite.getRGB(x,y);
      sprite.setRGB(x,y,
         color == 0 ? 0 : color | 0x1000000);
   }
```

The BIG Challenge: Rendering

- The resulting Image has an associated VolatileImage
  - The Java 2D™ API uses the VolatileImage for accelerated operations where possible (including blits using bitmask transparency)
  - If the VolatileImage is lost, or the BufferedImage is altered, Java 2D technology uses the contents of the BufferedImage to restore the VolatileImage

The BIG Challenge: Rendering

- Next step: Fullscreen mode
  - We take over the entire display for our application
  - Once in full screen mode, we may utilize page flipping
  - Page flipping allows us to build up our display in video memory, and then simply reset a pointer to show our result
  - We save a blit operation

The BIG Challenge: Rendering

- To get full screen mode:
  - Get the local graphics environment
  - From the local graphics environment, get the screen device
  - Call setFullScreenWindow() on the screen device, passing in the window to be made a full screen window
  - Call setDisplayMode() on the screen device, specifying the desired display mode

The BIG Challenge: Rendering

- Code fragment:

```
GraphicsEnvironment ge =GraphicsEnvironment.
   getLocalGraphicsEnvironment();
GraphicsDevice gd =
   ge.getDefaultScreenDevice();
try
{
   gd.setFullScreenWindow(myWindow);
   gd.setDisplayMode(new DisplayMode(width,
      height,32,60));
}
```

The BIG Challenge: Rendering

- To use page flipping:
  - Call createBufferStrategy() on your window, specifying the number of buffers you desire, and optional buffer capabilities
  - Each time you wish to render graphics to the window, call getBufferStrategy() on your window to get the BufferStrategy, then call getDrawGraphics() on the BufferStrategy to get a Graphics object for rendering
  - As of J2SE 1.4 beta 3 getDrawGraphics() returns a Graphics2D-compatible object

The BIG Challenge: Rendering

- Create the BufferStrategy; with accelerated images and true page flipping

```
try
{
   myWindow.createBufferStrategy(2,
      new BufferCapabilities(
      new ImageCapabilites(true),
      new ImageCapabilities(true),
      BufferCapabilities.FlipContents.PRIOR));
}
catch (AWTException awte)
{
   // exception processing
}
```

The BIG Challenge: Rendering

- Get the BufferStrategy, and from it the Graphics object to be used for rendering

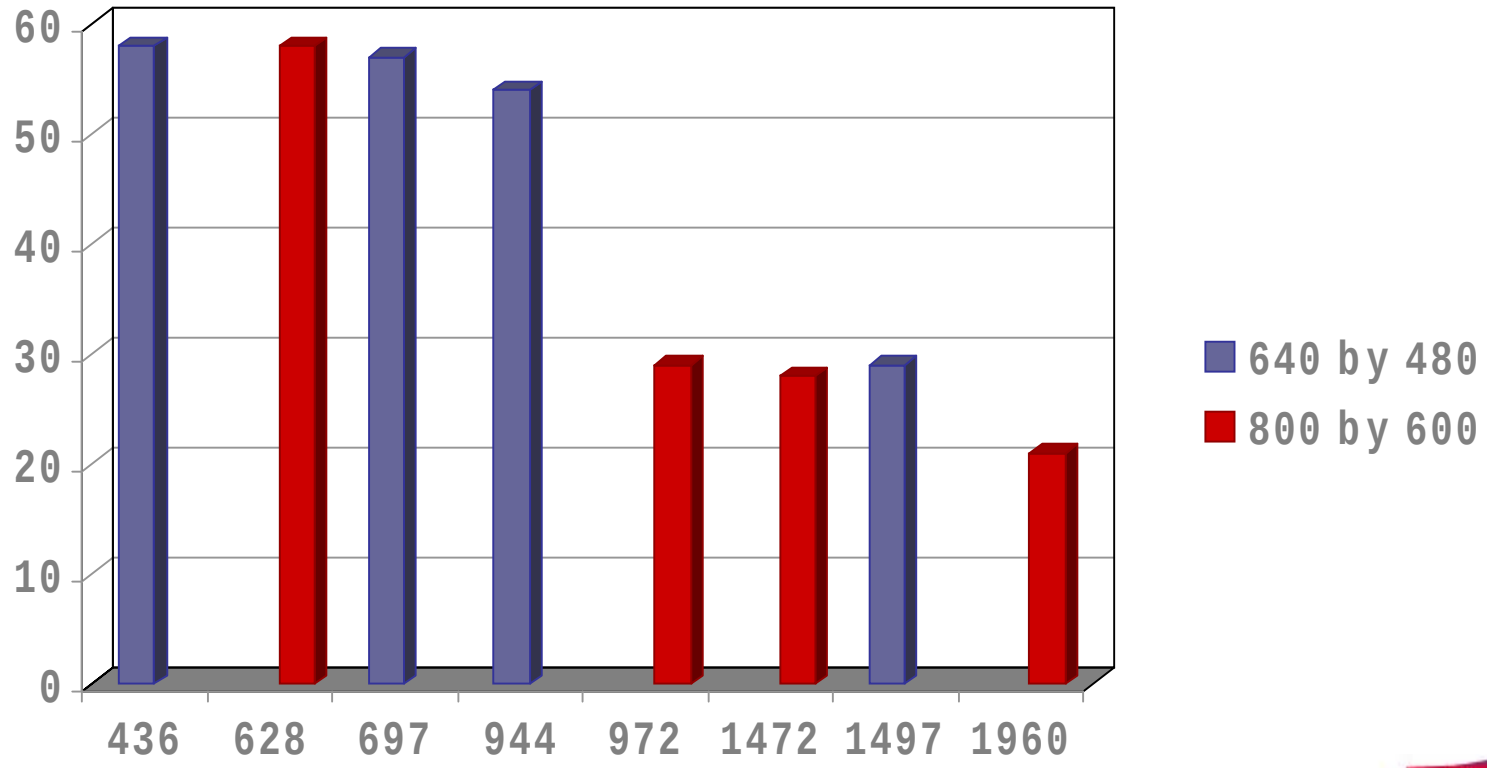
```
BufferStrategy renderStrategy =
   myWindow.getBufferStrategy();
Graphics renderGraphics =
   renderStrategy.getDrawGraphics();

// perform rendering as desired
```

The BIG Challenge: Rendering

- Recap
  - Call createCompatibleImage() on your GraphicsConfiguration to get BufferedImages that will be accelerated
  - Call setFullScreenMode() on your screen device to take over the display
  - Call createBufferStrategy() to attempt to enable page flipping
  - Get the current graphics object for drawing from the BufferStrategy

VolatileImage Performance with Page Flipping

Frames per second vs. Blits per Frame



The BIG Challenge: Rendering

- So if VolatileImage is so good, let's create a few more VolatileImages as buffers
  - Create a back buffer into which constant imagery is rendered
  - Create a work area into which certain intermediate images are rendered
  - Increase the number of buffers used by our BufferStrategy to perform earlier rendering of animations
- Run a test: 5 frames per second?

Question

What Happened?

Answer

There is only so much video memory available to us. If we attempt to allocate too much, we will wind up swapping some images into and out of video memory (as we were doing earlier with the buffered images).
Very expensive!

Sound and Music

Finishing Touches: Sound and Music

- Simple APIs make sound and music easy to add to an application
  - javax.sound.sampled provides classes allowing playback of sampled sounds
  - javax.sound.midi provides classes allowing playback of MIDI sequences
- Although we could stream data from the server to the client, we will assume we are reading local data
  - Remember the first network guideline minimize what you sned

Adding Sound

- Begin by defining the format of the sound sample
  - `new AudioFormat(encoding, sampleRate, sampleSizeInBits, channels, frameSize, frameRate, bigEndian)` to specify a-law, signed linear pcm, unsigned linear pcm, or u-law data
  - `new AudioFormat(sampleRate, sampleSizeInBits, channels, signed, bigEndian)` to specify linear pcm data

Adding Sound

- Describe the Line to which data will be written
  - A DataLine.Info will describe our data line
  - `new DataLine.Info(lineClass,audioFormat)`
  - Use a SourceDataLine for lineClass and the AudioFormat defined previously for audioFormat
- Get the line to which we will write data
  - Call AudioSystem.getLine(lineInfo), passing in the DataLine.Info obtained above

Adding Sound

- We may play a sample by writing it to the line obtained from AudioSystem.getLine()
- For better performance, wrap the line in a class extending Thread or implementing Runnable
  - Maintain a queue of sounds to be played
  - In run(), call wait() to put the thread to sleep whenever the queue is empty
  - Provide a method to add a sound to the queue; call notify() when a sound is added

Adding Music

- javax.sound.midi provides an almost idiot-proof mechanism for playing sequences without deep knowledge of MIDI
  - Begin by getting the default sequencer from the MIDI system
  - Open the sequencer
  - Set the sequence on the sequencer using an InputStream (a Sequence may also be used)
  - Start the sequencer
  - When desired, stop the sequencer

Adding Music

- For better performance, wrap the sequencer in a class extending Thread or implementing Runnable
  - Code the class to handle all sequence events of interest, as well as sequence changes
  - Register the class as a listener for meta events with `addMetaEventListener()`
  - Implement method meta() to receive meta messages

Adding Music

- To loop a sequence, look for message type 47 (end of sequence) in `meta()`, and restart the sequencer
- The same `wait()/notify()` metaphor used in the class wrapping the SourceDataLine for sampled sounds may be used to wrap the sequencer; the details will be a little more complex as the sequencer itself must notify the thread on completion of a sequence

Summary

- Java™ technology can and is being used to implement games
- Multiplayer network games can be written using Java technology, but present challenges not faced by non-networked games
- The Java 2 Platform, Standard Edition (J2SE™) 1.4 provides significant new capabilities for high-performance games
- Java technology provides simple mechanisms for enhancing games with sounds and music

If You Only Remember One Thing…

Remember how to obtain a BufferedImage that may be used to render into a VolatileImage with transparency. Few other questions seem to create such confusion, consternation, and flame wars as the question of how to use transparency with VolatileImage

Acknowledgements

- Daniel Larson
  - Client side artwork
- Clever Media
  - Midi sequences

Q&A

JavaOne℠

Sun's 2002 Worldwide Java Developer Conference™

BEYOND BOUNDARIES

Session 3199