Lecture 20 – Multi-media

Java 1.1 Only supports `.gif`, `.jpg`, and `.au` media types. 1.2 adds support for `.png`, `.wav`, `.aiff`, `.rmf`, and MIDI types 0 and 1. The Java Media Framework (JMF) has complete support for MIDI, also recording, editing, mixing, and playback of sounds and video. (This is an optional package of Java SE, and must be downloaded and installed seperately. See also the open source alternative, *Freedom for Media in Java*, or *FMJ*.) **Review Smile2.java.** Java5 greatly simplified basic image handling, and added many new capabilities and enhancements (such as taking advantage of video card special features). **Discuss and demo `javax.imageIO` package (`ImageIO.read`)**.

Describe the role of an ***ImageObserver***. Point out cycle of loading media: Create Toolkit Image (creates empty image), `g.drawImage` (draws the part of the image is already loaded, starts loading next part of image). Notifies `ImageObserver` when the next part has been loaded (a background thread is started for this). The `ImageObserver` can then call `repaint()`.

Qu: Why such a complex scheme? Ans: Loading images is slow. If they were loaded when you created the Image in `init()`, the user would see nothing until the image were completely loaded, since `paint()` isn't called until after `init()` is done. This way, the rest of your UI (User Interface) is displayed right away, possibly including a "Please wait, loading images..." message.

This way of drawing images may be ugly (esp. for animations). The `ImageIO.read`(*URL or File*) method is *synchronous*: no background thread is started and the code blocks until image is fully loaded. This replaces the legacy class **`MediaTracker`** (which may be used when you don't want to use swing).

Toolkit Images are read-only, but `ImageIO` creates `BufferedImage`s instead, which are read-write (and *managed* in the sense that video card hardware acceleration is used of possible).

The media files can be put somewhere, possibly in a Jar file, relative to the ***codeBase*** (location of the class): (*Show* ***java.net.URL***)

```
URL url = getClass().getResource( "../images/pic.gif");
Image i = getImage( url );  // These steps are often combined.
```

Stand-alone programs don't have `Applet.getImage` so use:
`java.awt.Toolkit.getImage()` or
`javax.imageIO.ImageIO.read()`:

```
Toolkit tk = Toolkit.getDefaultToolkit();
Image i = tk.getImage(...);
BufferedImage i = ImageIO.read( URL or File);
```

(Point out other useful Toolkit methods, to center windows on the screen.)

> The location of media and other resource files can be confusing. Keep in mind that such files cannot be part of a package, so the class loader will find them the same way it finds classes in the default, nameless package. Thus, if your code is in a package, the resource files should be in the same directory as the package, not the class file in the package.

Show `Smile.java` (swing version showing `ImageIcon`s and `JLabel`s).

`AudioClip: .play(), .loop(), .stop(). Applet.getAudioClip`, static **`Applet.newAudioClip()`**.

(**Demo MediaDemo.java - multi-media stand-alone application.**)

Java Applets have always supported sound, but in 1.1 only in Sun's sound file format (.au files). 1.2 and newer support many different formats. However MIDI and other advanced sound support wasn't added to the JDK until 1.3, in the `javax.sound.*` packages. These classes are needed to work with advanced sound (including recording) from stand-alone programs. A URL to obtain a demo of these classes (unpack and double-click the jar) and a good sound bank for MIDI is http://java.sun.com/products/java-media/sound/.