Java Game Programming

Playing Sound

JavaSE, via Java Sound API (in packages `javax.sound`), supports two types of audio:

- Sampled Audio: Sampled audio is represented as a sequence of time-sampled data of the amplitude of sound wave. It is supported in package `javax.sound.sampled`. The supported file formats are: "wav", "au" and "aiff". The samples can be either 8-bit or 16-bit, with sampling rate from 8 kHz to 48 kHz.
- Musical Instrument Digital Interface (MIDI): MIDI music is *synthesized* from musical notes and special sound effects, instead of time-sampled, like a *recipe* for creating musical sound. MIDI is supported in package `javax.sound.midi`.

Java Sound API also include a software sound mixer that supports up to 64 channels for sound effect and background music.

Java Media Framework (JMF), which is not part of JavaSE, is needed to support MP3 and advanced features. JOAL (Java Bindings on OpenAL) supports 3D sound effect.

Sampled Audio

To play sampled audio, you create an instance of a `SourceDataLine` or a `Clip`, which acts as a *source* to the software audio *mixer*. Audio samples are then loaded into it, and delivered to the mixer. The mixer may mix the samples with those from other sources and then deliver the mix to a target (usually an audio output device on a sound card).

`javax.sound.Clip`

Code Example:

```
import java.io.*;
import java.net.URL;
import javax.sound.sampled.*;
import javax.swing.*;

// To play sound using Clip, the process need to be alive.
// Hence, we use a Swing application.
public class SoundClipTest extends JFrame {

   // Constructor
   public SoundClipTest() {
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      this.setTitle("Test Sound Clip");
      this.setSize(300, 200);
      this.setVisible(true);

      try {
         // Open an audio input stream.
         URL url = this.getClass().getClassLoader().getResource("gameover.wav");
         AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
         // Get a sound clip resource.
         Clip clip = AudioSystem.getClip();
         // Open audio clip and load samples from the audio input stream.
         clip.open(audioIn);
         clip.start();
      } catch (UnsupportedAudioFileException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      } catch (LineUnavailableException e) {
         e.printStackTrace();
      }
   }

   public static void main(String[] args) {
      new SoundClipTest();
   }
```

```
}
```

The steps of playing sounds via `Clip` are:

1. Allocate a `AudioInputStream` piped from a file or URL, e.g.,

```
// from a wave File
File soundFile = new File("eatfood.wav");
AudioInputStream audioIn = AudioSystem.getAudioInputStream(soundFile);
// from a URL
URL url = new URL("http://www.zzz.com/eatfood.wav");
AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
// can read from a disk file and also a file contained inside a JAR (used for distribution)
// recommended
URL url = this.getClass().getClassLoader().getResource("eatfood.wav");
AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
```

2. Allocate a sound `Clip` resource via the static method `AudioSystem.getClip()`:

```
Clip clip = AudioSystem.getClip();
```

3. Open the clip to load sound samples from the audio input stream opened earlier:

```
clip.open(audioIn);
// For small-size file only. Do not use this to open a large file over slow network, as it blocks.
```

4. You can now play the clip by invoking either the `start()` or `loop()` method

```
// start()
clip.start();  // play once
// Loop()
clip.loop(0);  // repeat none (play once), can be used in place of start().
clip.loop(5);  // repeat 5 times (play 6 times)
clip.loop(Clip.LOOP_CONTINUOUSLY);  // repeat forever
```

5. You can stop the player by invoking `stop()`, which may be useful to stop a continuous `loop()`.

```
if (clip.isRunning()) clip.stop();
```

Playing Sound Effects for Java Games

A typical game requires various sound effects, e.g., move, eat, shoot, kill, gameover, and etc. The following enumeration encapsulates all the sound effects in a single class, to simplify game programming.

```
import java.io.*;
import java.net.URL;
import javax.sound.sampled.*;

/**
 * This enum encapsulates all the sound effects of a game, so as to separate the sound playing
 * codes from the game codes.
 * 1. Define all your sound effect names and the associated wave file.
 * 2. To play a specific sound, simply invoke SoundEffect.SOUND_NAME.play().
 * 3. You might optionally invoke the static method SoundEffect.init() to pre-load all the
 *    sound files, so that the play is not paused while loading the file for the first time.
 * 4. You can use the static variable SoundEffect.volume to mute the sound.
 */
public enum SoundEffect {
   EXPLODE("explode.wav"),    // explosion
   GONG("gong.wav"),          // gong
   SHOOT("shoot.wav");        // bullet

   // Nested class for specifying volume
   public static enum Volume {
      MUTE, LOW, MEDIUM, HIGH
   }

   public static Volume volume = Volume.LOW;

   // Each sound effect has its own clip, loaded with its own sound file.
   private Clip clip;

   // Constructor to construct each element of the enum with its own sound file.
   SoundEffect(String soundFileName) {
      try {
         // Use URL (instead of File) to read from disk and JAR.
         URL url = this.getClass().getClassLoader().getResource(soundFileName);
         // Set up an audio input stream piped from the sound file.
```

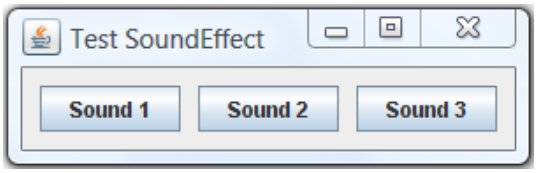
```
            AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(url);
            // Get a clip resource.
            clip = AudioSystem.getClip();
            // Open audio clip and load samples from the audio input stream.
            clip.open(audioInputStream);
         } catch (UnsupportedAudioFileException e) {
            e.printStackTrace();
         } catch (IOException e) {
            e.printStackTrace();
         } catch (LineUnavailableException e) {
            e.printStackTrace();
         }
      }

      // Play or Re-play the sound effect from the beginning, by rewinding.
      public void play() {
         if (volume != Volume.MUTE) {
            if (clip.isRunning())
               clip.stop();   // Stop the player if it is still running
            clip.setFramePosition(0); // rewind to the beginning
            clip.start();     // Start playing
         }
      }

      // Optional static method to pre-load all the sound files.
      static void init() {
         values(); // calls the constructor for all the elements
      }
   }
```

Dissecting the Program

[PENDING]




```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

// Testing the SoundEffect enum in a Swing application
public class SoundEffectDemo extends JFrame {

   // Constructor
   public SoundEffectDemo() {
      // Pre-load all the sound files
      SoundEffect.init();
      SoundEffect.volume = SoundEffect.Volume.LOW;  // un-mute

      // Set up UI components
      Container cp = this.getContentPane();
      cp.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
      JButton btnSound1 = new JButton("Sound 1");
      btnSound1.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            SoundEffect.EXPLODE.play();
         }
      });
      cp.add(btnSound1);
      JButton btnSound2 = new JButton("Sound 2");
      btnSound2.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            SoundEffect.GONG.play();
         }
      });
      cp.add(btnSound2);
      JButton btnSound3 = new JButton("Sound 3");
      btnSound3.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            SoundEffect.SHOOT.play();
```

```
            }
        });
        cp.add(btnSound3);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("Test SoundEffct");
        this.pack();
        this.setVisible(true);
    }

    public static void main(String[] args) {
        new SoundEffectDemo();
    }
}
```

Dissecting the Program

[PENDING]

(optional) javax.sound.SourceDataLine

A source data line acts as a source to the audio mixer. Unlike Clip (which pre-load the audio samples), an application writes audio samples to a source data line, which handles the buffering of the bytes and delivers them to the mixer, in a *streaming* manner.

```
import java.io.*;
import javax.sound.sampled.*;
/**
 * Use SourceDataLine to read line-by-line from the external sound file.
 * For computer game, use Clip to pre-load short-duration sound files.
 */
public class SoundLineTest {
    public static void main(String[] args) {
        SourceDataLine soundLine = null;
        int BUFFER_SIZE = 64*1024;  // 64 KB

        // Set up an audio input stream piped from the sound file.
        try {
            File soundFile = new File("gameover.wav");
            AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(soundFile);
            AudioFormat audioFormat = audioInputStream.getFormat();
            DataLine.Info info = new DataLine.Info(SourceDataLine.class, audioFormat);
            soundLine = (SourceDataLine) AudioSystem.getLine(info);
            soundLine.open(audioFormat);
            soundLine.start();
            int nBytesRead = 0;
            byte[] sampledData = new byte[BUFFER_SIZE];
            while (nBytesRead != -1) {
                nBytesRead = audioInputStream.read(sampledData, 0, sampledData.length);
                if (nBytesRead >= 0) {
                    // Writes audio data to the mixer via this source data line.
                    soundLine.write(sampledData, 0, nBytesRead);
                }
            }
        } catch (UnsupportedAudioFileException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (LineUnavailableException ex) {
            ex.printStackTrace();
        } finally {
            soundLine.drain();
            soundLine.close();
        }
    }
}
```

MIDI Synthesized Sound

In a computer game, MIDI can be used for providing the background music, as it save you quite a bit of bandwidth if you are downloading the file from a slow connection. Java Sound API supports three types of MIDI: MIDI Type 1, MIDI Type 2 and Rich Music Format (RMF).

The steps are:

1. Allocate a Sequence piped from a MIDI file:

```
Sequence song = MidiSystem.getSequence(new File("song.mid"));
```

2. Allocate a Sequencer to play a MIDI sequence:

```
Sequencer player = MidiSystem.getSequencer();
```

3. Set the current sequence for the sequencer to play:

```
player.setSequence(song);
```

4. Open the sequencer to load the sequence:

```
player.open();
```

5. You could set the loop count via `setLoopCount()` and start playing the music by invoking `start()`:

```
player.setLoopCount(0);  // repeat 0 times (play once)
player.setLoopCount(Sequencer.LOOP_CONTINUOUSLY);  // repeat forever
player.start();
```

6. You can set the tempo (rate of play) via `setTempoFactor()`, which is useful in a computer game when the difficulty level is increased.

```
player.setTempoFactor(1.0F);  // takes a factor (in float) - normal tempo
player.setTempoFactor(1.5F);  // faster by 50%
```

7. You can stop the play via `stop()`:

```
if (player.isRunning()) player.stop();
```

Code Example

The method is declared `static`, as we only need one global copy, and no instance variable involved in the operation.

```
1  import java.io.*;
2  import javax.sound.midi.*;
3
4  public class MidiSoundTest {
5     private static Sequencer midiPlayer;
6
7     // testing main method
8     public static void main(String[] args) {
9        startMidi("song1.mid");     // start the midi player
10       try {
11          Thread.sleep(10000);   // delay
12       } catch (InterruptedException e) { }
13       System.out.println("faster");
14       midiPlayer.setTempoFactor(2.0F);   // >1 to speed up the tempo
15       try {
16          Thread.sleep(10000);   // delay
17       } catch (InterruptedException e) { }
18
19       // Do this on every move step, you could change to another song
20       if (!midiPlayer.isRunning()) {  // previous song finished
21          // reset midi player and start a new song
22          midiPlayer.stop();
23          midiPlayer.close();
24          startMidi("song2.mid");
25       }
26    }
27
28    public static void startMidi(String midFilename) {
29       try {
30          File midiFile = new File(midFilename);
31          Sequence song = MidiSystem.getSequence(midiFile);
32          midiPlayer = MidiSystem.getSequencer();
33          midiPlayer.open();
34          midiPlayer.setSequence(song);
35          midiPlayer.setLoopCount(0); // repeat 0 times (play once)
36          midiPlayer.start();
37       } catch (MidiUnavailableException e) {
38          e.printStackTrace();
39       } catch (InvalidMidiDataException e) {
40          e.printStackTrace();
41       } catch (IOException e) {
42          e.printStackTrace();
43       }
44    }
45 }
```

Notes: Windows Vista seems to have lot of problems with Midi.

MP3 & Java Media Framework (JMF)

Java Media Framework (JMF) is not part of JavaSE, but can be downloaded from http://java.sun.com/javase/technologies/desktop/media/jmf. JMF, among other things, provides support for playing MP3, AAC music. Download JMF and run the installation. Check to ensure that "`jmf.jar`" is installed into "`$JAVA_HOME\jre\lib\ext`".

Example: Playing MP3 Music

```
import javax.media.*;
import java.net.URL;

public class Mp3PlayerDemo extends Thread {

   private String filename;
   Player player;

   public Mp3PlayerDemo(String mp3Filename) {
      this.filename = mp3Filename;
   }

   public void run() {
      try {
         URL url = this.getClass().getClassLoader().getResource(filename);
         MediaLocator locator = new MediaLocator(url);
         player = Manager.createPlayer(locator);
         player.addControllerListener(new ControllerListener() {
            public void controllerUpdate(ControllerEvent event) {
               if (event instanceof EndOfMediaEvent) {
                  player.stop();
                  player.close();
               }
            }
         });
         player.realize();
         player.start();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }

   public static void main(String[] args) {
      new Mp3PlayerDemo("song.mp3").start();
   }
}
```

[TODO] To be continued...

REFERENCES & RESOURCES

- TODO

Latest version tested: JDK 1.6
Last modified: September 4, 2008