Creating a Sound Class

Time required: 45 minutes

Contents

To easily access the audioclips of our game we will create a Sound class. This class will have an enumeration with an audioclip for each of the sounds we use. These constants are public so that any object which have access to them and can play them. For example, in the **Ball** class we can play the sound of the bouncing of the ball using **Sound.BALL.play()** at the moment we know the ball changes its direction.

sound.java

```java
import java.io.*; // Read from file system
import java.net.URL; // Get and manage URL (file) path
import javax.sound.sampled.AudioInputStream; // Create Audio input stream object
import javax.sound.sampled.AudioSystem; // Gets the Audio stream
import javax.sound.sampled.Clip; // In memory source for playing audio
// Handle audio exceptions
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;

/**
 * This enumeration encapsulates all the sound effects of a game, so as to
 * separate the sound playing code from the game code.
 * 1. Define all your sound effect names and the associated wave files.
 * 2. To play a specific sound, invoke Sound.SOUND_NAME.play().
 * 3. Invoke the static method Sound.init() to
 *    pre-load all the sound files, so that the play is not paused
 *    while loading the file for the first time.
 * 4. Use the static variable Sound.volume to mute the sound.
 */
public enum Sound {
   BALL("ball.wav"), // Ping Pong ball strike
   GAMEOVER("gameover.wav"), // Game is over
   BACKGROUND("background.wav"); // Continous background music for the game

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

   // Initialize the volume to low
   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
   Sound(String soundFileName) {
      try {
         // Use URL to read the file path from the 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) {
```

```
52            e.printStackTrace();
53         }
54      }
55
56      // Play or Re-play the sound effect from the beginning, by rewinding
57      public void play() {
58         if (volume != Volume.MUTE) {
59            if (clip.isRunning()) // Is the audio clip still playing?
60               clip.stop(); // Stop the player if it is still running
61            clip.setFramePosition(0); // Rewind to the beginning
62            clip.start(); // Start playing
63         }
64      }
65
66      // Loop the sound effect continously
67      public void loop() {
68         if (volume != Volume.MUTE) {
69            if (clip.isRunning()) // Is the audio clip still playing?
70               clip.stop(); // Stop the player if it is still running
71            clip.setFramePosition(0); // Rewind to the beginning
72            clip.loop(Clip.LOOP_CONTINUOUSLY); // Start the loop
73         }
74      }
75
76      // Stop the sound
77      public void stop() {
78         clip.stop(); // Stop the player if it is still running
79      }
80
81      // Static method to pre-load all the sound files into memory
82      static void init() {
83         values(); // calls the constructor for all the elements
84      }
85   }
```

The audioclips objects will be created when the **Sound** class loads, which is the first time the program uses the Sound class. From this moment on, they will be re-used.

Let's look at the modifications in the **SimplePong** class:

```
37    // Construct the Game application
38    public SimplePong() {
39
40       Sound.init();                    // Load all sound files in memory
41       Sound.volume = Sound.Volume.LOW; // Set sound volume
42
43       // Add KeyListener to the application
44       addKeyListener(new KeyListener() {
45          @Override
46          public void keyTyped(KeyEvent e) {
47          }
48
49          @Override
50          public void keyReleased(KeyEvent e) {
51             player.keyReleased(e);
52          }
53
54          @Override
55          public void keyPressed(KeyEvent e) {
56             player.keyPressed(e);
57          }
58       });
59       setFocusable(true);      // Allow keyboard events to be captured from Frame
60       Sound.BACKGROUND.loop(); // Loop background sound
61    }
```

```
84    // Game Over called from Ball object
85    public void gameOver() {
86       Sound.BACKGROUND.stop();
87       Sound.GAMEOVER.play();
88       JOptionPane.showMessageDialog(this, "Game Over", "Game Over",
89       JOptionPane.YES_NO_OPTION);
90       System.exit(ABORT);
91    }
```

In the last line of the **SimplePong** class constructor, we add **Sound.BACKGROUND.loop()**, which will initiate the playing of our background music and will play repeatedly till it gets to the **gameOver()** method, where we stop the background music with **Sound.BACKGROUND.stop()**. After **Sound.BACKGROUND.stop()** and before the popup, we inform that the game is over playing "Game Over" **Sound.GAMEOVER.play()**.

In the **Ball** class, we change the **move()** method so that it plays **Sound.BALL** when the ball bounces.

```
23    void move() {
24  // Move the ball by adding x, y integers to current location
25       BallX = BallX + MoveX;
26       BallY = BallY + MoveY;
27
28       // If the ball hits either paddle, reverse direction,
29       if (simplePong.player.getBounds().intersects(getBounds())
30                || simplePong.computer.getBounds().intersects(getBounds()))
31       {
32          Sound.BALL.play();
33          MoveX = -MoveX;   // Reverse horizontal direction
34       }
```

Complete Code

SimplePong.java

```java

/**
* Filename: SimplePong.java
* Written by:  William Loring
* Written on:  02-10-2018
* Revised:
* Add a Sound Class
*/

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class SimplePong extends JPanel {
   private static final long serialVersionUID = 1L;
   // Constants for the size of the JFrame
   final static int GAME_WIDTH = 800;
   final static int GAME_HEIGHT = 500;

   // Variable for the speed of the game
   static int gameSpeed = 17;

   // Paddle size for player and computer
   static int PADDLE_WIDTH = 10;
   static int PADDLE_HEIGHT = 100;

   // Create Ball and Paddle objects
   Ball ball = new Ball(this);
   Player player = new Player(this);
   Computer computer = new Computer(this);

   // Construct the Game application
   public SimplePong() {

      Sound.init(); // Load all sound files in memory
      Sound.volume = Sound.Volume.LOW; // Set sound volume

      // Add KeyListener to the application
      addKeyListener(new KeyListener() {
         @Override
         public void keyTyped(KeyEvent e) {
         }

         @Override
         public void keyReleased(KeyEvent e) {
```

```
52              player.keyReleased(e);
53           }
54
55           @Override
56           public void keyPressed(KeyEvent e) {
57              player.keyPressed(e);
58           }
59        });
60        setFocusable(true); // Allow keyboard events to be captured from Frame
61        Sound.BACKGROUND.loop(); // Loop background sound
62     }
63
64     // Move the Ball and paddles
65     private void move() {
66        ball.move();
67        player.move();
68        computer.move();
69     }
70
71     @Override // Override the default paint method
72     public void paint(Graphics g) {
73        super.paint(g); // Clear the JPanel
74        setBackground(Color.WHITE); // Set JPanel background to white
75        Graphics2D g2d = (Graphics2D) g;
76        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
77                             RenderingHints.VALUE_ANTIALIAS_ON);
78
79        // Override the game objects paint methods
80        ball.paint(g2d);
81        player.paint(g2d);
82        computer.paint(g2d);
83     }
84
85     // Game Over called from Ball object
86     public void gameOver() {
87        Sound.BACKGROUND.stop();
88        Sound.GAMEOVER.play();
89        JOptionPane.showMessageDialog(this, "Game Over", "Game Over",
90                                      JOptionPane.YES_NO_OPTION);
91        System.exit(ABORT);
92     }
93
94     public static void main(String[] args) throws InterruptedException {
95        JFrame frame = new JFrame("Simple Pong");
96        SimplePong simplePong = new SimplePong();
97        frame.add(simplePong);
98        frame.setSize(GAME_WIDTH, GAME_HEIGHT);
99        frame.setVisible(true);
100       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
101
```

```
        // Game loop, loops forever
        while (true) {
            simplePong.move(); // Call the move methods
            simplePong.repaint(); // Repaint the application screen
            Thread.sleep(gameSpeed); // Pause thread to let JFrame redraw
        }
    }
}
```

Player.java

```
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;

public class Player {

    // Create a reference to the game object
    private SimplePong simplePong;

    // Set horizontal position of paddle from left side of window
    private final int PADDLE_X = 5;
    // How many pixels at a time an object moves
    private final int MOVE = 3;

    // Create custom RGB color, Cougar Blue
    private Color cougarBlue = new Color(0, 58, 112);

    // Store vertical position
    int PaddleY = 0;
    // Store vertical movement
    int MoveY = MOVE;

    // Create object with Game reference
    public Player(SimplePong simplePong) {
        this.simplePong = simplePong;
    }

    public void move() {
        // If the paddle is not outside the top or bottom border, allow movement
        if (PaddleY + MoveY > 0 && PaddleY + MoveY <
            simplePong.getHeight() - SimplePong.PADDLE_HEIGHT) {
            PaddleY = PaddleY + MoveY;
        }
    }

```

```
37    // Draw paddle rectangle
38    public void paint(Graphics2D g) {
39       g.setColor(cougarBlue); // Use a custom RGB color, Cougar Blue
40       g.fillRect(PADDLE_X,
41                  PaddleY,
42                  SimplePong.PADDLE_WIDTH,
43                  SimplePong.PADDLE_HEIGHT);
44    }
45
46    // Stop movement when key is released
47    public void keyReleased(KeyEvent e) {
48       MoveY = 0;
49    }
50
51    // Get which cursor key is pressed, change vertical movement variable
52    public void keyPressed(KeyEvent e) {
53       if (e.getKeyCode() == KeyEvent.VK_UP)
54          MoveY = -MOVE;
55       if (e.getKeyCode() == KeyEvent.VK_DOWN)
56          MoveY = MOVE;
57    }
58
59    // Used by Ball to get location of racquet
60    public Rectangle getBounds() {
61       return new Rectangle(PADDLE_X,
62                            PaddleY,
63                            SimplePong.PADDLE_WIDTH,
64                            SimplePong.PADDLE_HEIGHT);
65    }
66
67    // Allows the Game object to get right hand side of the racquet
68    public int getRightX() {
69       return PADDLE_X + SimplePong.PADDLE_WIDTH;
70    }
71 }
```

Computer.java

```java
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.Rectangle;

public class Computer {

   // Create a reference to the game object
   private SimplePong simplePong;

   // Set horizontal position of racquet from right side of window
   private final int PADDLE_X = SimplePong.GAME_WIDTH - 30;

   // Create custom RGB color, Cougar Gold
   private Color cougarGold = new Color(249, 190, 0);

   // Store vertical position
   private int PaddleY = 0;
   // Set Computer paddle speed
   private int MoveY = 3;

   // Create object with Game reference
   public Computer(SimplePong simplePong) {
      this.simplePong = simplePong;
   }

   public void move() {
      // If the paddle is not outside the top or bottom border, allow movement
      if (PaddleY + MoveY > 0 && PaddleY + MoveY <
          simplePong.getHeight() - SimplePong.PADDLE_HEIGHT) {
         PaddleY = PaddleY + MoveY;
      } else {
         MoveY = -MoveY;
      }
   }

   // Draw paddle rectangle
   public void paint(Graphics2D g) {
      // Use custom RGB color, Cougar Gold
      g.setColor(cougarGold);
      g.fillRect(PADDLE_X,
                 PaddleY,
                 SimplePong.PADDLE_WIDTH,
                 SimplePong.PADDLE_HEIGHT);
   }

   // Used by Ball to get location of paddle
   public Rectangle getBounds() {
      return new Rectangle(PADDLE_X,
                           PaddleY,
                           SimplePong.PADDLE_WIDTH,
                           SimplePong.PADDLE_HEIGHT);
```

```
53
54      // Allows the Game object to get left hand side of the paddle
55      public int getLeftX() {
56          return PADDLE_X - SimplePong.PADDLE_WIDTH;
57      }
58  }
```

Ball.java

```
1  import java.awt.Graphics2D;
2  import java.awt.Rectangle;
3
4  public class Ball {
5      private final int BALL_DIAMETER = 30;
6
7      // Store the ball's x, y location
8      private int BallX = 400;
9      private int BallY = 250;
10
11     // Store the balls x & y movement
12     private int MoveX = -3;
13     private int MoveY = 3;
14
15     // Create Game variable
16     private SimplePong simplePong;
17
18     // Create a ball object with a reference to the game board
19     public Ball(SimplePong simplePong) {
20         this.simplePong = simplePong;
21     }
22
23     void move() {
24         // Move the ball by adding x, y integers to current location
25         BallX = BallX + MoveX;
26         BallY = BallY + MoveY;
27
```

```
28        // If the ball hits either paddle, reverse direction,
29        if (simplePong.player.getBounds().intersects(getBounds())
30              || simplePong.computer.getBounds().intersects(getBounds())) {
31           Sound.BALL.play();
32           MoveX = -MoveX; // Reverse horizontal direction
33        }
34
35        // If the ball runs into the top or botton border, reverse direction
36        if (BallY < 0 || BallY + BALL_DIAMETER > simplePong.getHeight()) {
37           MoveY = -MoveY; // Reverse the vertical direction of the ball
38        }
39
40        // If the ball runs into the left border, Computer wins
41        if (BallX + MoveX < 0) {
42           simplePong.gameOver();
43        }
44
45        // If the ball runs into the right border, Player wins
46        if (BallX + BALL_DIAMETER > simplePong.getWidth()) {
47           simplePong.gameOver();
48        }
49     }
50
51     // Paint the ball/circle
52     public void paint(Graphics2D g) {
53        g.fillOval(BallX,
54                   BallY,
55                   BALL_DIAMETER,
56                   BALL_DIAMETER);
57     }
58
59     public Rectangle getBounds() {
60        return new Rectangle(BallX,
61                             BallY,
62                             BALL_DIAMETER,
63                             BALL_DIAMETER);
64     }
65  }
```

Assignment Submission

Attach the .java files to the assignment in Blackboard.