Collision Detection

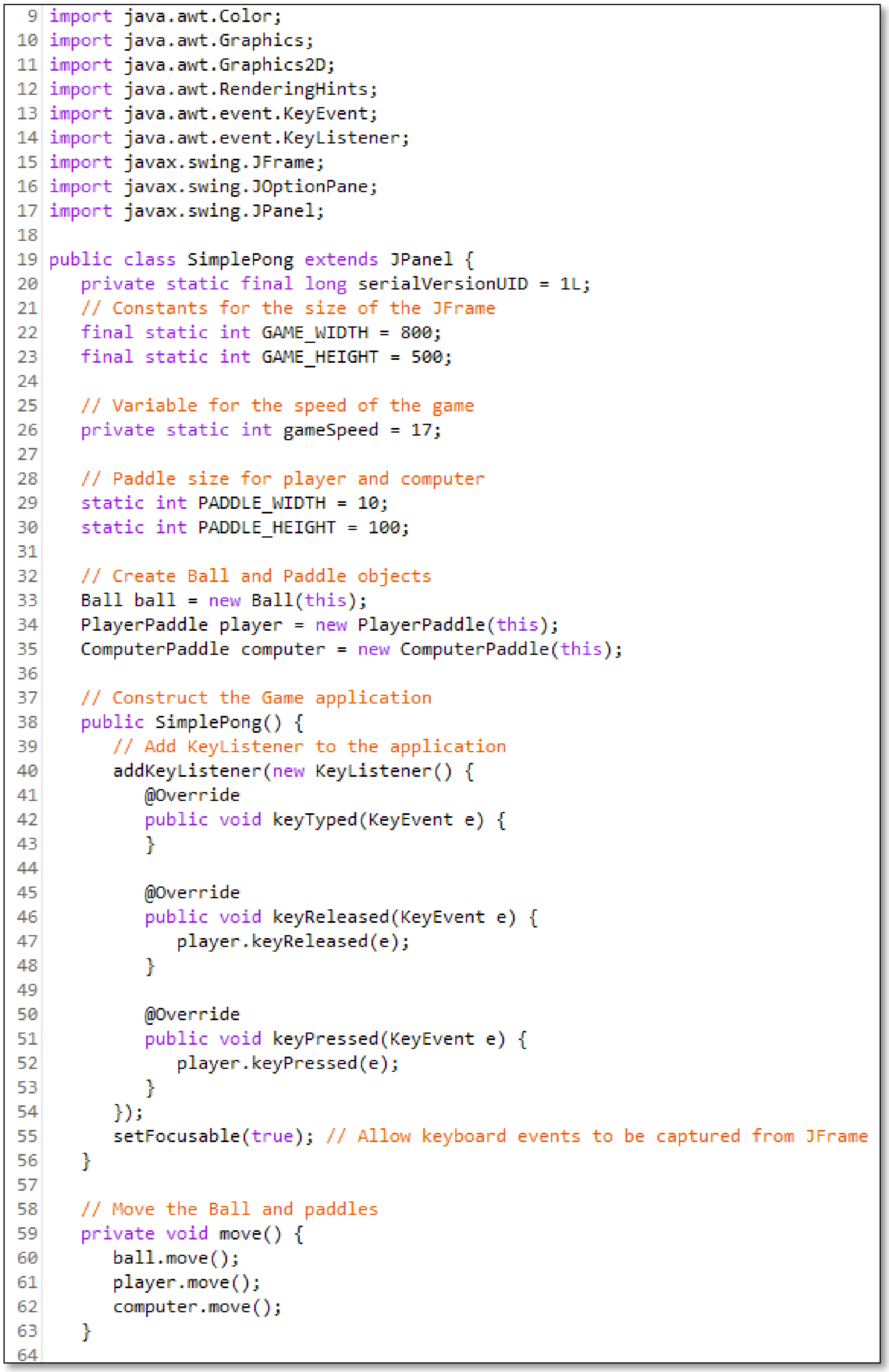
Time required: 45 minutes

Contents

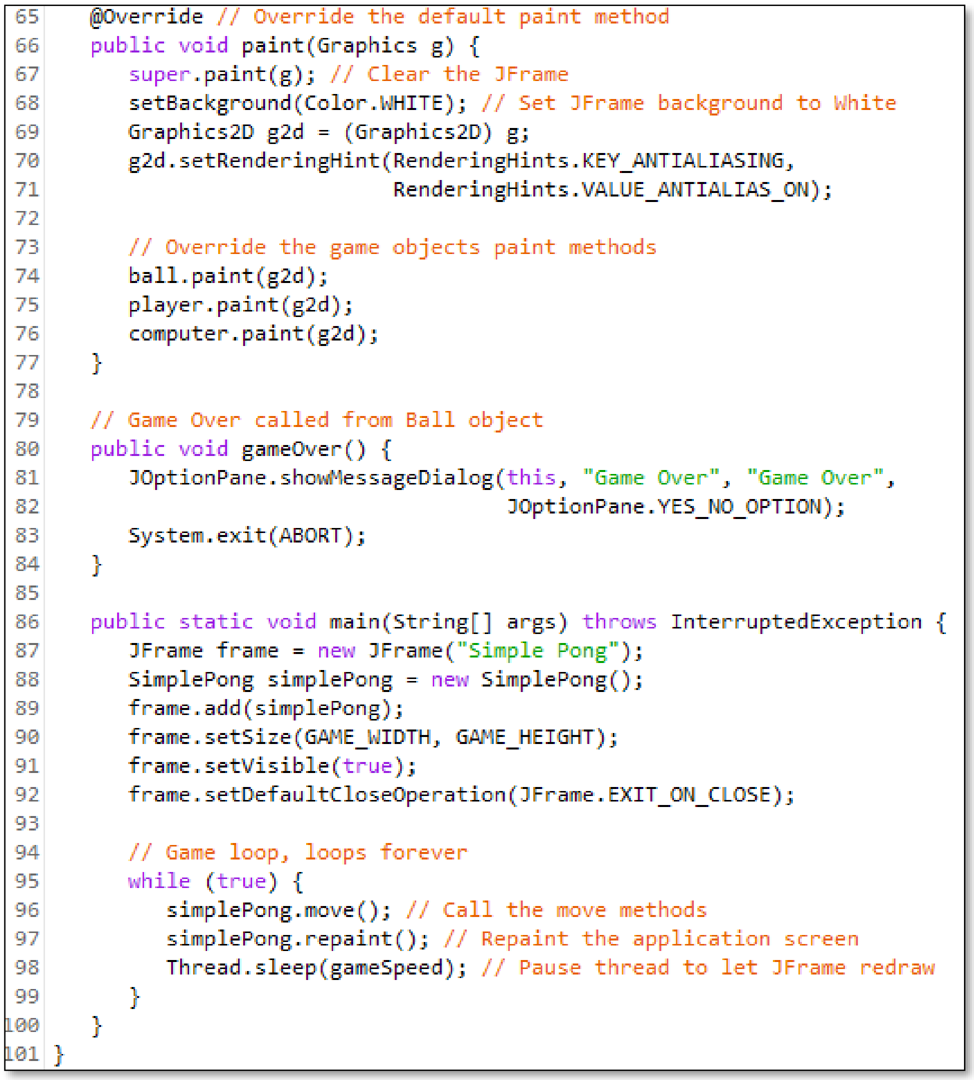
In this tutorial we will learn how to detect when one sprite collides with another. We will make the ball bounce on the paddle. We will also make the game finish if the ball gets to the left or right border of the canvas, showing a popup window with the message "Game Over".

Game Over

Below we see our class **SimplePong** exactly the same as the previous one, with the only difference that this one has a **gameOver()** method.



```java
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
   private 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);
   PlayerPaddle player = new PlayerPaddle(this);
   ComputerPaddle computer = new ComputerPaddle(this);

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

         @Override
         public void keyReleased(KeyEvent e) {
            player.keyReleased(e);
         }

         @Override
         public void keyPressed(KeyEvent e) {
            player.keyPressed(e);
         }
      });
      setFocusable(true); // Allow keyboard events to be captured from JFrame
   }

   // Move the Ball and paddles
   private void move() {
      ball.move();
      player.move();
      computer.move();
   }

```

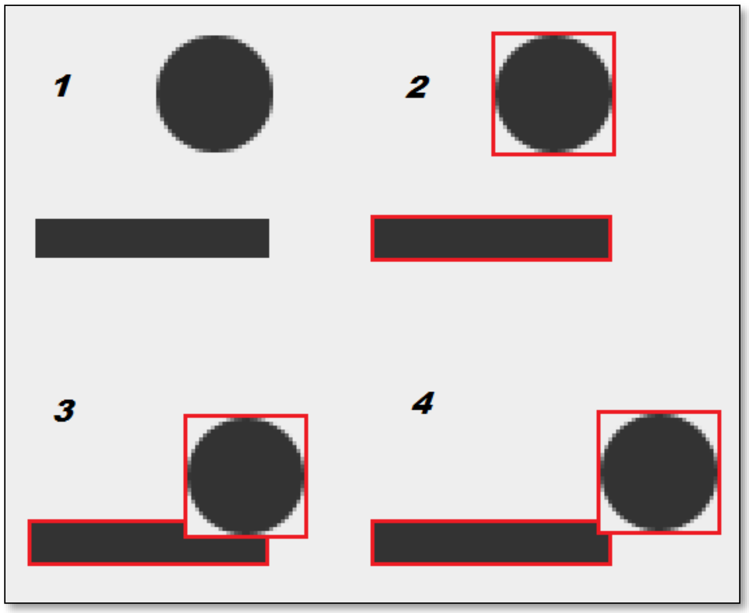


```java
    @Override // Override the default paint method
    public void paint(Graphics g) {
       super.paint(g); // Clear the JFrame
       setBackground(Color.WHITE); // Set JFrame background to White
       Graphics2D g2d = (Graphics2D) g;
       g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);

       // Override the game objects paint methods
       ball.paint(g2d);
       player.paint(g2d);
       computer.paint(g2d);
    }

    // Game Over called from Ball object
    public void gameOver() {
       JOptionPane.showMessageDialog(this, "Game Over", "Game Over",
                                     JOptionPane.YES_NO_OPTION);
       System.exit(ABORT);
    }

    public static void main(String[] args) throws InterruptedException {
       JFrame frame = new JFrame("Simple Pong");
       SimplePong simplePong = new SimplePong();
       frame.add(simplePong);
       frame.setSize(GAME_WIDTH, GAME_HEIGHT);
       frame.setVisible(true);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       // 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
       }
    }
}
```

The gameOver() method launches a popup using JOptionPane.showMessageDialog with the message "Game Over" and an "Accept" button. After the popup, System.exit(ABORT) makes the program finish. The gameOver() method is public, because it will be called from the sprite "Ball" when it detects that it has moved to the left or right border of the canvas.

Sprite Collision

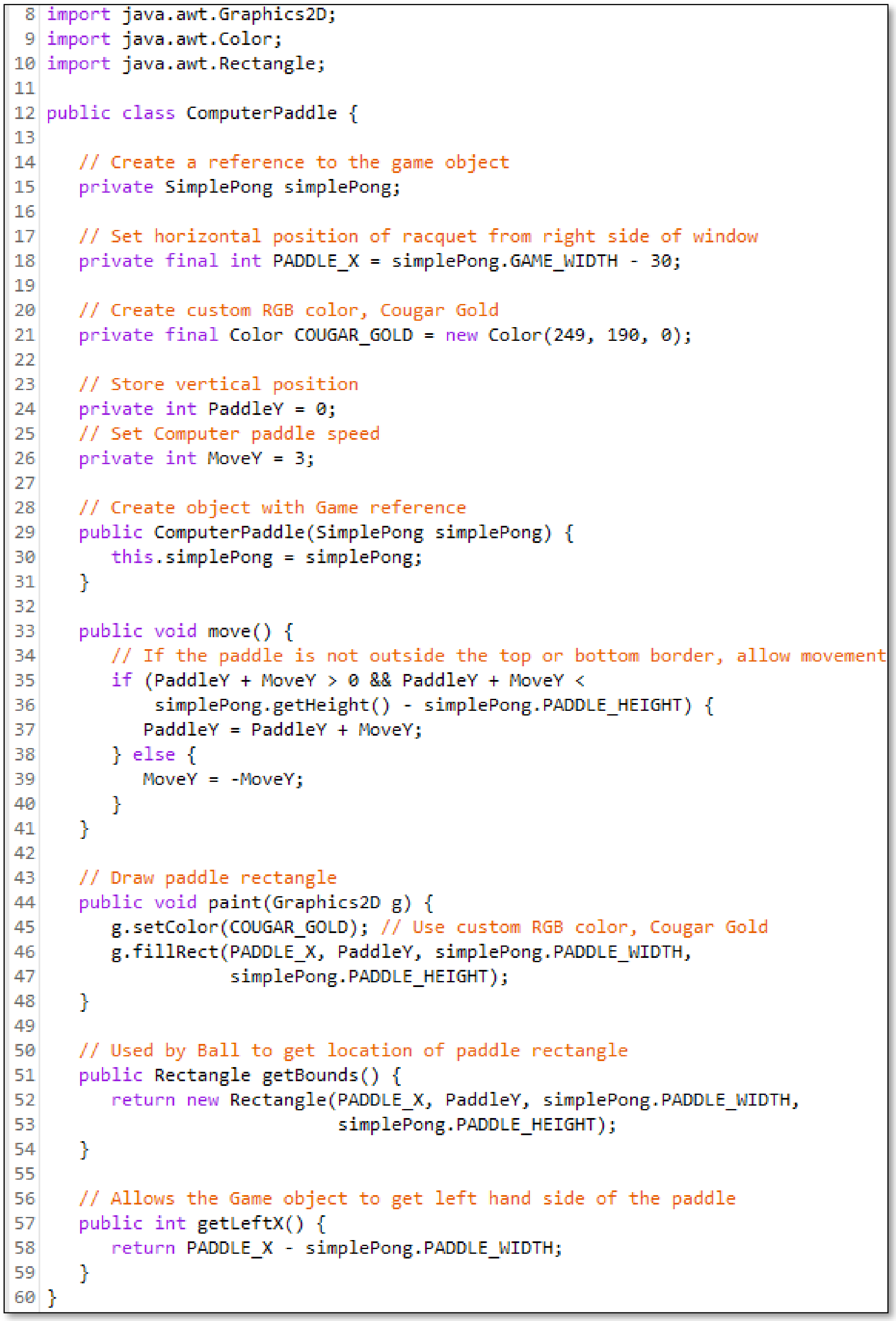
To detect the collision between the ball and the racquet we will use rectangles. In the case of the ball we will use a square around the ball as you can see in the figure 2.



The class java.awt.Rectangle has an intersects method(Rectangle r) which returns true when two rectangles occupy the same space, like in the case of the figure 3 or 4. This method is not quite exact, because as you can see in the figure 4, the ball doesn't touch the paddle. For our program it is close than enough.

Computer Paddle

The **ComputerPaddle** and the **PlayerPaddle** class add a **getBounds()** method, which returns a rectangle type of object, indicating the position of the paddle. This method will be used by the sprite **Ball**, to know the position of the paddle and to detect the collision.

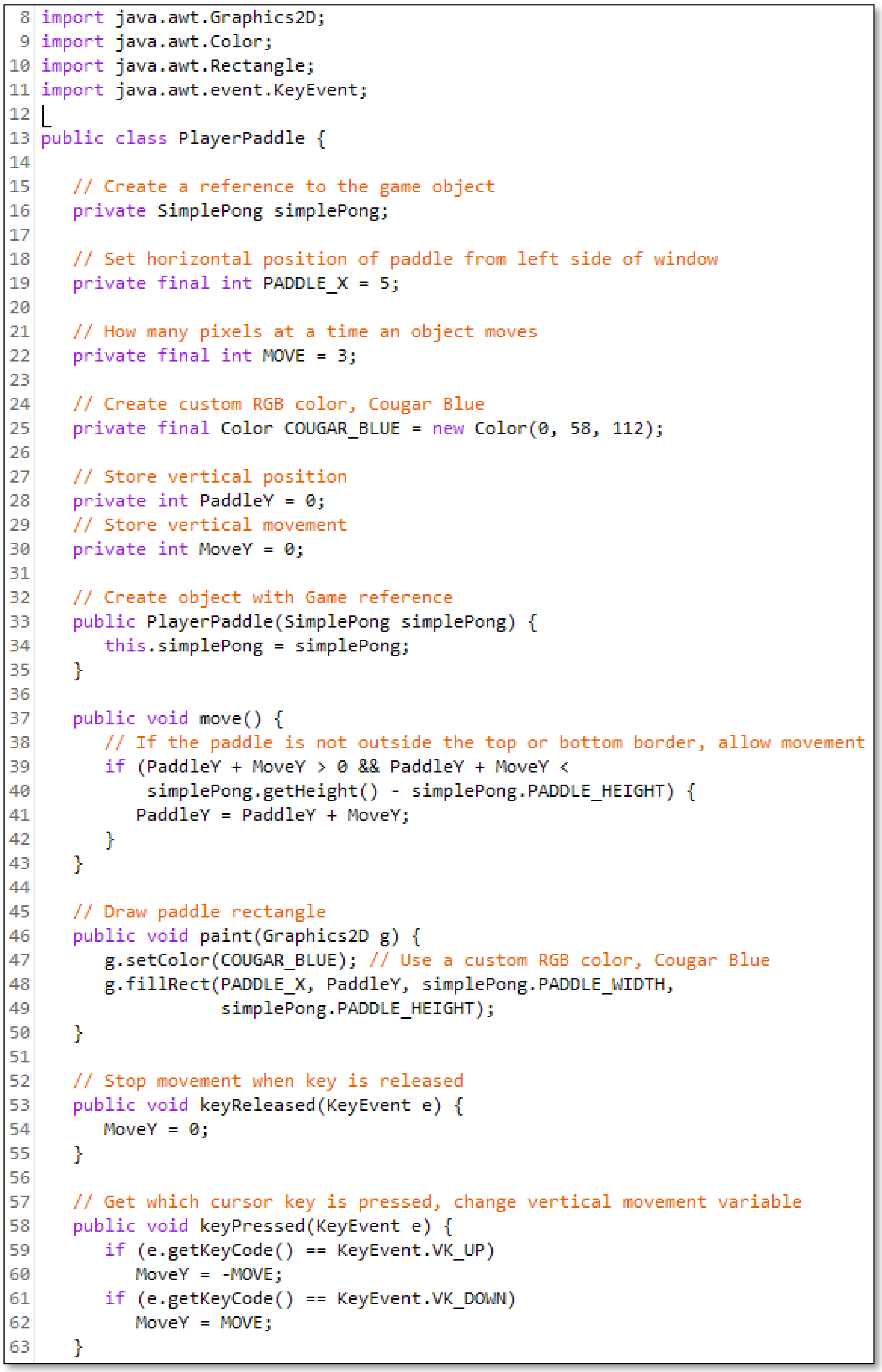


```java
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.Rectangle;

public class ComputerPaddle {

   // 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 final Color COUGAR_GOLD = 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 ComputerPaddle(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) {
      g.setColor(COUGAR_GOLD); // Use custom RGB color, Cougar Gold
      g.fillRect(PADDLE_X, PaddleY, simplePong.PADDLE_WIDTH,
                 simplePong.PADDLE_HEIGHT);
   }

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

   // Allows the Game object to get left hand side of the paddle
   public int getLeftX() {
      return PADDLE_X - simplePong.PADDLE_WIDTH;
   }
}
```

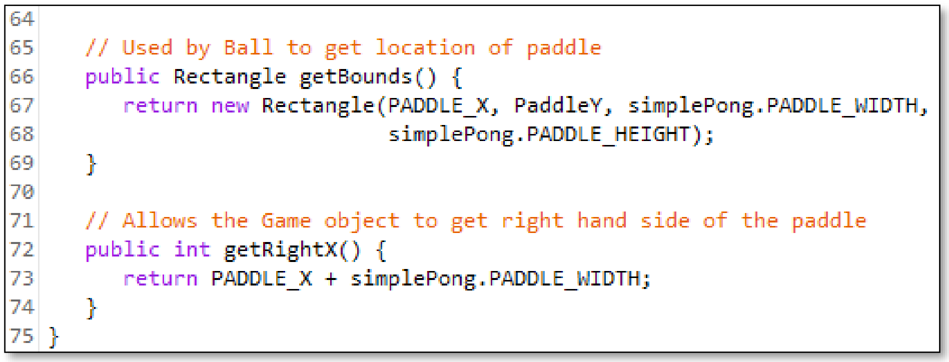
The value of the Racquet "y" position is fixed to GAME_HEIGHT - 70. This value is used both in the paint method as in **getBounds()**. When we create a constant, the good thing is that if we want to change the value, we change it in only one place. We avoid the possible error of changing it in one place and not changing it in another.

The way of defining a constant is declaring a "static final" property and writing it in upper case. The compiler allows us to use lower case, but the standard says we use upper case for the constants.

Player Paddle

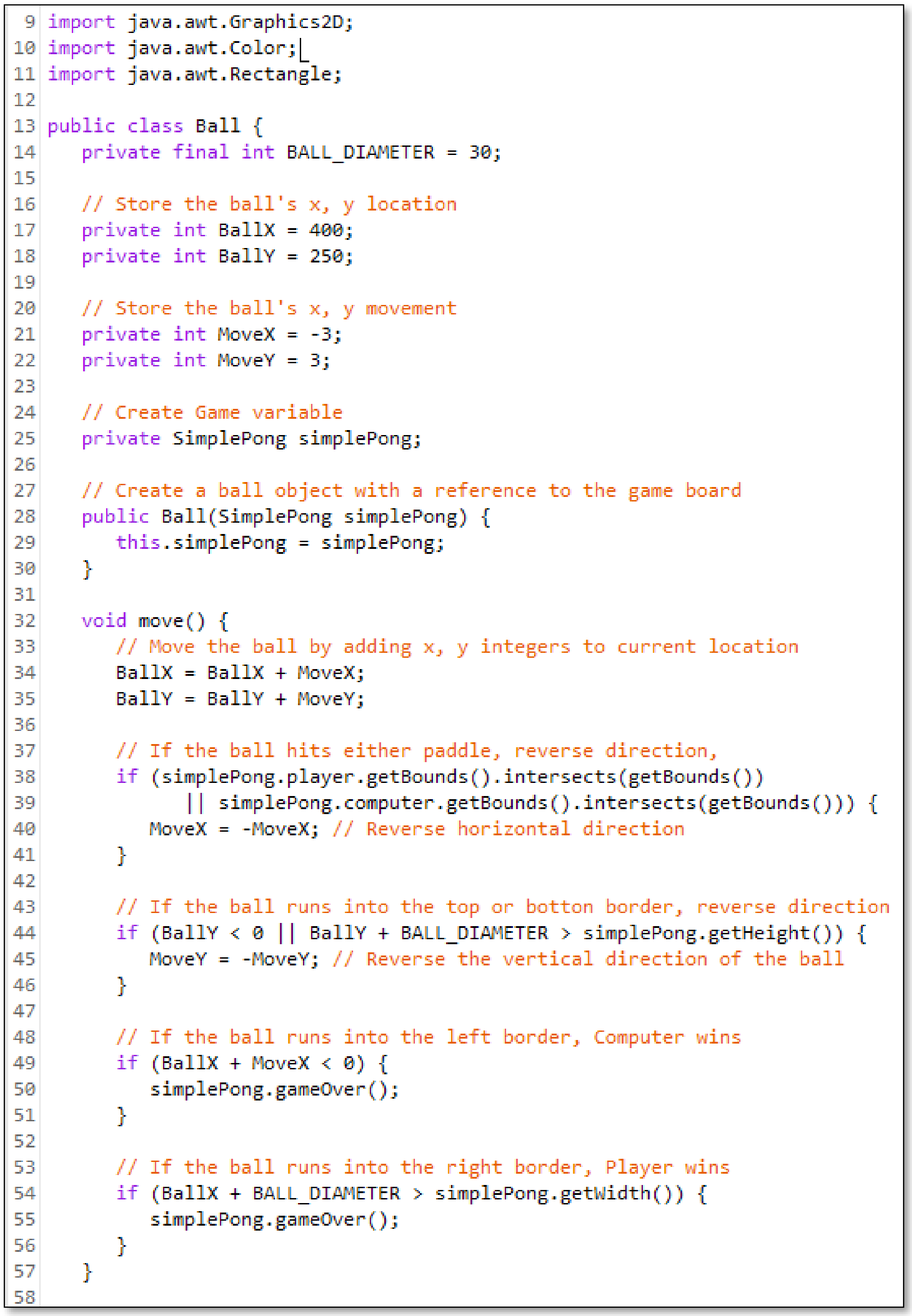


```java
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;

public class PlayerPaddle {

   // 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 final Color COUGAR_BLUE = new Color(0, 58, 112);

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

   // Create object with Game reference
   public PlayerPaddle(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;
      }
   }

   // Draw paddle rectangle
   public void paint(Graphics2D g) {
      g.setColor(COUGAR_BLUE); // Use a custom RGB color, Cougar Blue
      g.fillRect(PADDLE_X, PaddleY, simplePong.PADDLE_WIDTH,
                 simplePong.PADDLE_HEIGHT);
   }

   // Stop movement when key is released
   public void keyReleased(KeyEvent e) {
      MoveY = 0;
   }

   // Get which cursor key is pressed, change vertical movement variable
   public void keyPressed(KeyEvent e) {
      if (e.getKeyCode() == KeyEvent.VK_UP)
         MoveY = -MOVE;
      if (e.getKeyCode() == KeyEvent.VK_DOWN)
         MoveY = MOVE;
   }
```

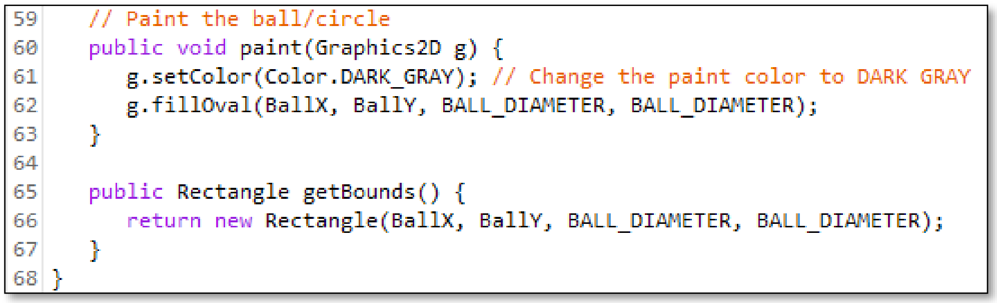


```java

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

   // Allows the Game object to get right hand side of the paddle
   public int getRightX() {
      return PADDLE_X + simplePong.PADDLE_WIDTH;
   }
}
```

Ball



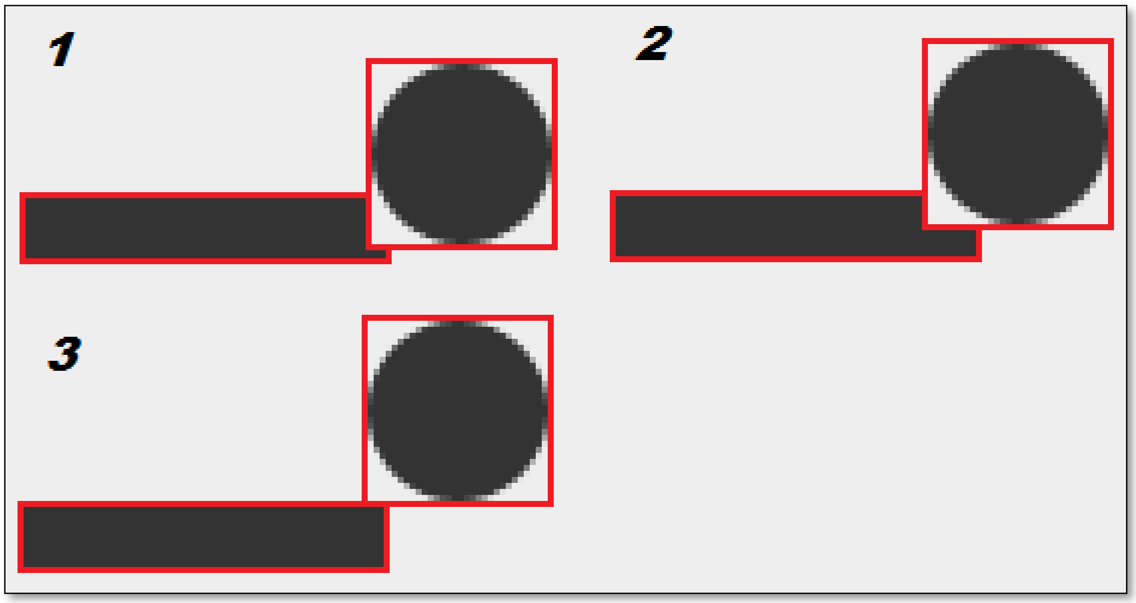
```java
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.Rectangle;

public class Ball {
   private final int BALL_DIAMETER = 30;

   // Store the ball's x, y location
   private int BallX = 400;
   private int BallY = 250;

   // Store the ball's x, y movement
   private int MoveX = -3;
   private int MoveY = 3;

   // Create Game variable
   private SimplePong simplePong;

   // Create a ball object with a reference to the game board
   public Ball(SimplePong simplePong) {
      this.simplePong = simplePong;
   }

   void move() {
      // Move the ball by adding x, y integers to current location
      BallX = BallX + MoveX;
      BallY = BallY + MoveY;

      // If the ball hits either paddle, reverse direction,
      if (simplePong.player.getBounds().intersects(getBounds())
            || simplePong.computer.getBounds().intersects(getBounds())) {
         MoveX = -MoveX; // Reverse horizontal direction
      }

      // If the ball runs into the top or botton border, reverse direction
      if (BallY < 0 || BallY + BALL_DIAMETER > simplePong.getHeight()) {
         MoveY = -MoveY; // Reverse the vertical direction of the ball
      }

      // If the ball runs into the left border, Computer wins
      if (BallX + MoveX < 0) {
         simplePong.gameOver();
      }

      // If the ball runs into the right border, Player wins
      if (BallX + BALL_DIAMETER > simplePong.getWidth()) {
         simplePong.gameOver();
      }
   }

```



```
   // Paint the ball/circle
   public void paint(Graphics2D g) {
      g.setColor(Color.DARK_GRAY); // Change the paint color to DARK GRAY
      g.fillOval(BallX, BallY, BALL_DIAMETER, BALL_DIAMETER);
   }

   public Rectangle getBounds() {
      return new Rectangle(BallX, BallY, BALL_DIAMETER, BALL_DIAMETER);
   }
}
```

In a similar way, we have included the **getBounds()** method constant to the **PlayerPaddle** and **ComputerPaddle** class.

If the collision takes place, we will change the direction and the position of the ball. If the collision occurs on the side (figure 1), the ball could be several pixels below the upper side of the racquet. In the following game loop, even if the ball moved upwards (figure 2), it could still be in collision with the racquet.



It is the **move()** method of the **Ball** class which uses the new methods **collision()** and **gameOver()** of the **Game** class. The bounce when it gets to the lower border is replaced by a call to **game.gameOver()**.

Assignment Submission

Attach the .java files to the assignment in Blackboard.