

Game Programming: JFrame, JPanel, and Paint

Contents

Game Programming: JFrame, JPanel, and Paint	1
JFrame: The Window	1
What is a Thread in Java?	2
AWT Engine and Thread AWT-EventQueue	3
JPanel: The Canvas	3
Positioning in the Canvas Coordinate "x" and "y"	5
When does the AWT Engine call the paint Method?	5
The Position of the Circle	6
Game Loop	7
Analyzing the Paint Method	9
Analyzing the Concurrence and the Behavior of the Threads	11
Assignment Submission	13

Time required: 45 minutes

Many games rely on painting "sprites" on the canvas of the program. To paint something, we need a surface to paint on. The surface or canvas we are going to paint is a JPanel object. In the same way a canvas needs a frame to hold it, our JPanel will be framed in a window made by the JFrame class.

JFrame: The Window

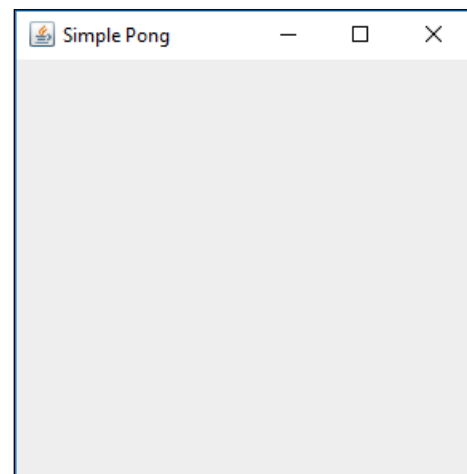
The following code creates a window "Simple Pong" of 300 pixels by 300 pixels. The window won't be visible until we call **setVisible(true)**. If we don't include the last line **frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)**, when we close the window the program won't finish and will continue running.

```

1 // Import Swing JFrame component for GUI
2 import javax.swing.JFrame;
3
4 public class Game1 {
5     public static void main(String[] args) {
6
7         // Create a JFrame object called frame
8         JFrame frame = new JFrame("Simple Pong");
9
10        // Set the size of the frame in pixels x, y
11        frame.setSize(300, 300);
12
13        // Make the frame visible
14        frame.setVisible(true);
15
16        // Exit the program when the frame is closed
17        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18    }
19 }

```

This code creates a window which can be maximized, minimized, change its size with the mouse, etc. When we create a JFrame object we start an engine which manages the user interface. This engine communicates with the operating system both to paint on the screen and receive information from the keyboard and from the mouse. We will call this engine "AWT Engine" or "Swing Engine" because it is created by these two libraries. In the first version of java, only AWT existed and then Swing was added. This engine uses several threads.



What is a Thread in Java?

A program is executed by just one processor, line by line. Threads allow a program to start several executions at the same time. It is as if there are several processors running at the same time with their own sequence of instructions.

Even though threads and concurrence are very powerful tools, there can be problems when two threads enter the same variables. It is interesting to think that two threads can be running the same code at the same time.

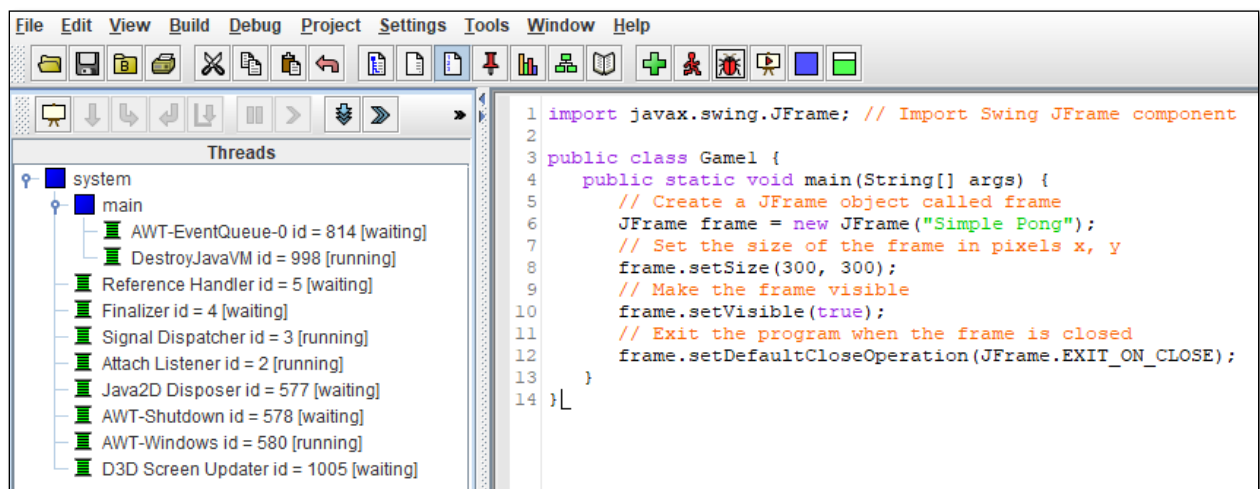
A thread is like a cook preparing a dish reading a recipe. Two concurrent threads would be like two cooks working in the same kitchen, preparing one dish with the same recipe or with

different recipes. The problems come when both try to use the same frying pan at the same time.

AWT Engine and Thread AWT-EventQueue

The AWT Engine starts several threads which can be seen if we start the application with debug and we go to the debug perspective. Each thread acts as if it was an independent program running at the same time as the other threads. The first thread we see in the debug view called "Thread [AWT-EventQueue-0]", this thread is the one in charge of painting the screen and receiving the mouse and keyboard events.

1. In jGrasp, Go to **Debug as Application**. You can see the threads.



JPanel: The Canvas

We extend the JPanel class to be able to overwrite the paint method which is the method called by the AWT Engine to paint what appears in the screen.

```

1 import java.awt.Color;
2 import java.awt.Graphics;
3 import java.awt.Graphics2D;
4 import java.awt.RenderingHints;
5 import java.awt.geom.Ellipse2D;
6 import javax.swing.JFrame;
7 import javax.swing.JPanel;
8
9 public class Game2 extends JPanel {
10     private static final long serialVersionUID = 1L;
11
12     // Override the default paint method of the program
13     @Override
14     public void paint(Graphics g) {
15
16         // Create a Graphics2D object to draw on screen
17         Graphics2D g2d = (Graphics2D) g;
18
19         // Set the color to RED, a Java constant
20         g2d.setColor(Color.RED);
21
22         // Create objects at (x, y, width, height)
23         g2d.fillOval(0, 0, 30, 35);
24         g2d.drawOval(0, 50, 30, 40);
25         g2d.fillRect(50, 0, 20, 30);
26         g2d.drawRect(50, 50, 30, 30);
27
28         // Draw a circle using the Ellipse object
29         g2d.draw(new Ellipse2D.Double(0, 100, 30, 35));
30     }
31
32     public static void main(String[] args) {
33         JFrame frame = new JFrame("Simple Pong");
34         frame.add(new Game2());
35         frame.setSize(300, 300);
36         frame.setVisible(true);
37         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
38     }
39 }

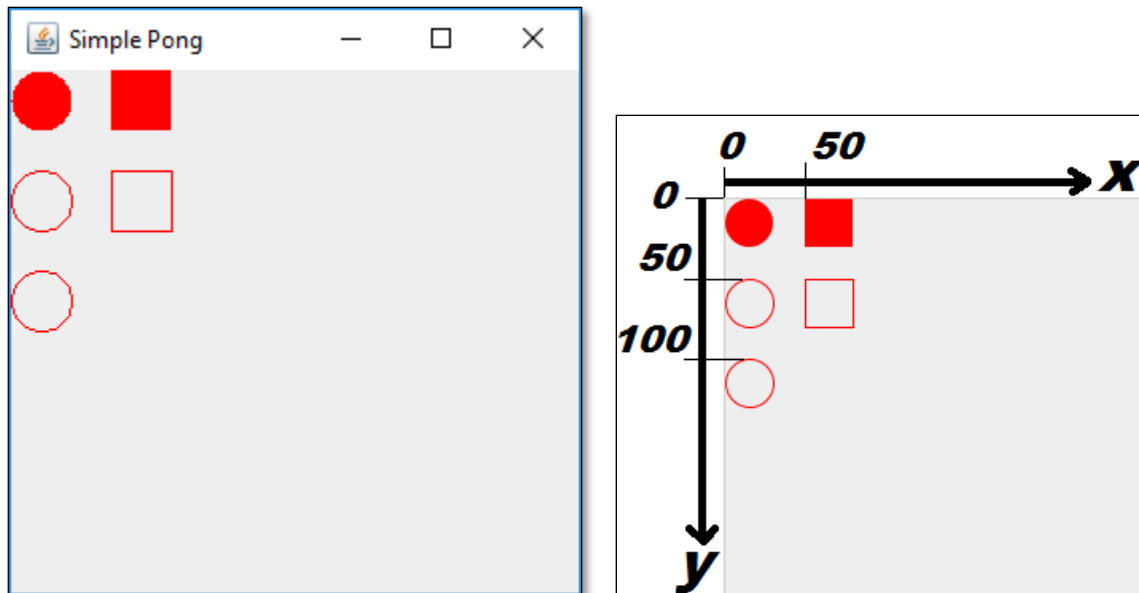
```

The paint method receives by parameter a Graphics2D object which extends from Graphics. Graphics is an old class used by AWT which has been replaced with Graphics2D which has more and better functionality. The parameter is still a Graphics type due to compatibility but we will use Graphics2D, so we need to create a variable g2d "Graphics2D g2d = (Graphics2D) g;". Once we have g2d we can use all the Graphics2D methods to draw.

The first thing we do is choose the color we use to draw: "g2d.setColor(Color.RED);". After, we draw circles and squares.

Positioning in the Canvas Coordinate "x" and "y"

To draw something inside the canvas we should indicate in which position we are going to start painting. For this, each of the points in the canvas has an associated position (x,y) being (0,0) the point of the top-left corner.



The first red circle is painted with `"g2d.fillOval(0, 0, 30, 30)"`: the first two parameters are the position (x,y) and after comes the width and the height. As a result, we have a circle with 30 pixels of diameter in the position (0,0).

The empty circle is drawn with `"g2d.drawOval(0, 50, 30, 30)"`: which draws a circle in the position x=0 (left margin) and y=50 (50 pixels below the top margin) with a height of 30 pixels and a width of 30 pixels.

Rectangles are drawn with `"g2d.fillRect(50, 0, 30, 30)"` and `"g2d.drawRect(50, 50, 30, 30)"` in a similar way to the circles.

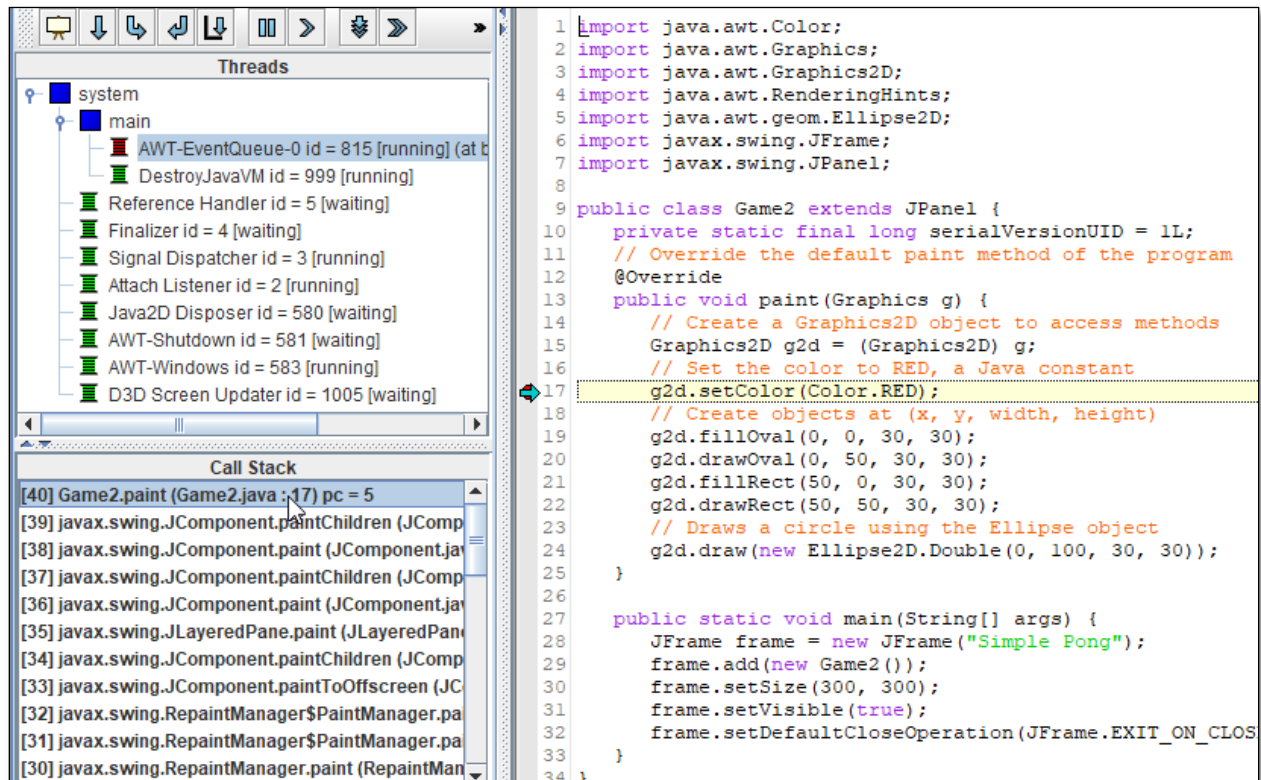
The line `"g2d.draw(new Ellipse2D.Double(0, 100, 30, 30))"` draws the last circle using an `Ellipse2D.Double` object.

When does the AWT Engine call the paint Method?

The AWT engine calls the paint method every time the operating system reports that the canvas is to be painted. When the window is created for the first-time paint is called. The paint

method is also called if we minimize and after we maximize the window and if we change the size of the window with the mouse.

We can watch this behavior if we put a breakpoint in the first line of the paint method and we run the program in the debug mode.



The paint method is run by the **Thread AWT-EventQueue**. This is in charge of painting the screen.

The Position of the Circle

Every time we paint something we have to define its position (x, y). To make the circle move, we have to modify the position (x, y) each time and paint the circle in the new position.

In our example, we keep the current position of our circle in two properties called "x" and "y". We also create a method called moveBall() which will increase in 1 both "x" and "y", each time we call it. In the paint method we draw a circle with a diameter of 30 pixels in the position (x, y) given by the properties before described; "g2d.fillOval(x, y, 30, 30);".

Game Loop

At the end of the main method we start an infinite loop "while (true)" where we repeatedly call **moveBall()** to change the position of the circle. We call **repaint()**, which forces the AWT engine to call the paint method to paint again the canvas.

This cycle is known as a "Game loop" and carries out two operations:

1. **Update:** updates of the physics of our world. In our case the update is given by the **moveBall()** method, which increases the "x" and "y" in 1.
2. **Render:** painting of the current state of our world including the changes made before. In our example, it is carried out by the call to the method **repaint()** and the following call to the "paint" method carried out by the AWT engine, and more specifically by the "event queue thread".

```

1 /**
2  * Filename: MovingBall.java
3  * Written by: William Loring
4  * Written on: 02-15-2016
5  * Revised:
6  * Move a ball across the frame
7  */
8 import java.awt.Graphics;
9 import java.awt.Graphics2D;
10 import java.awt.RenderingHints;
11 import javax.swing.JFrame;
12 import javax.swing.JPanel;
13
14 public class MovingBall extends JPanel {
15     private static final long serialVersionUID = 1L;
16
17     // Diameter of the ball
18     private final int BALL_DIAMETER = 30;
19     private final int MOVE = 3;
20     // Starting coordinates of the circle
21     // Upper left hand side of the JPanel
22     int BallX = 0;
23     int BallY = 0;
24
25     // Method to move the circle 1 pixel down and to the right
26     // Move the circle diagonally
27     private void moveBall() {
28         BallX = BallX + MOVE;
29         BallY = BallY + MOVE;
30     }
31
32     // Override the default object paint method.
33     @Override
34     public void paint(Graphics g) {
35         // Clear JPanel
36         super.paint(g);
37         // Create Graphics2D object for better drawing to screen
38         Graphics2D g2d = (Graphics2D) g;
39         // Turn on anti aliasing, makes the circle bitmap smoother
40         g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
41                             RenderingHints.VALUE_ANTIALIAS_ON);
42         // Paint new position of the ball
43         g2d.fillOval(BallX, BallY, BALL_DIAMETER, BALL_DIAMETER);
44     }
45

```

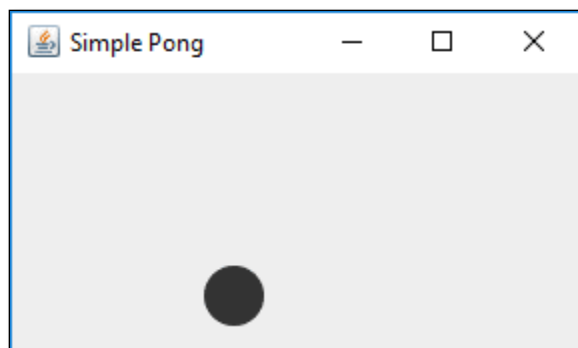


```

46     public static void main(String[] args) throws InterruptedException {
47         JFrame frame = new JFrame("Simple Pong");
48         MovingBall game = new MovingBall();
49         frame.add(game);
50         frame.setSize(600, 400);
51         frame.setVisible(true);
52         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
53
54         // An infinite "Game Loop"
55         while (true) {
56             // Move the ball
57             game.moveBall();
58             // Repaint the JPanel
59             game.repaint();
60             // Sleep this thread for 17 ms (Approximately 60 FPS)
61             // Other threads can process, redrawing the screen
62             Thread.sleep(17);
63         }
64     }
65 }

```

Example run:



Analyzing the Paint Method

The paint method is run each time the operating system tells the AWT engine that it is necessary to paint the canvas. If we run the **repaint()** method of a **JPanel** object, what we are doing is telling the AWT engine to execute the paint method as soon as possible. Calling **repaint()**, the canvas is painted again and we can see the changes in the position of the circle.

```

@Override
public void paint(Graphics g) {

    // Clear JPanel
    super.paint(g);

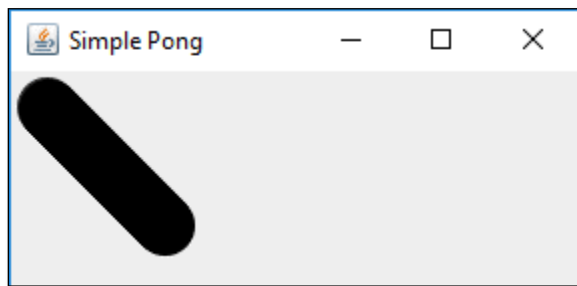
    // Create Graphics2D object to for better drawing to screen
    Graphics2D g2d = (Graphics2D) g;

    // Turn on anti aliasing, makes the circle bitmap smoother
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);

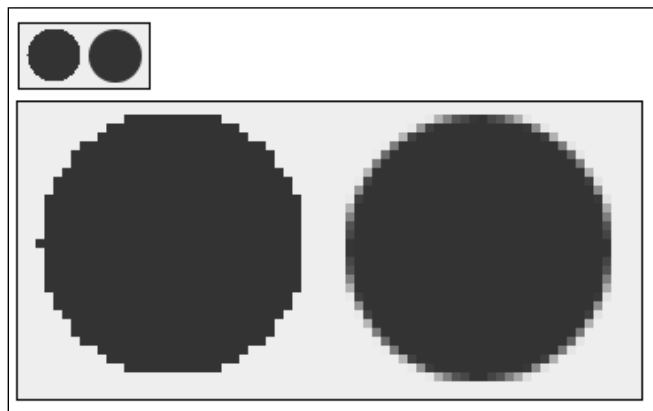
    // Paint new position of the ball
    g2d.fillOval(BallX, BallY, BALL_DIAMETER, BALL_DIAMETER);
}

```

The call to **super.paint(g)**, clears the screen. If we comment out this line we see the following effect:

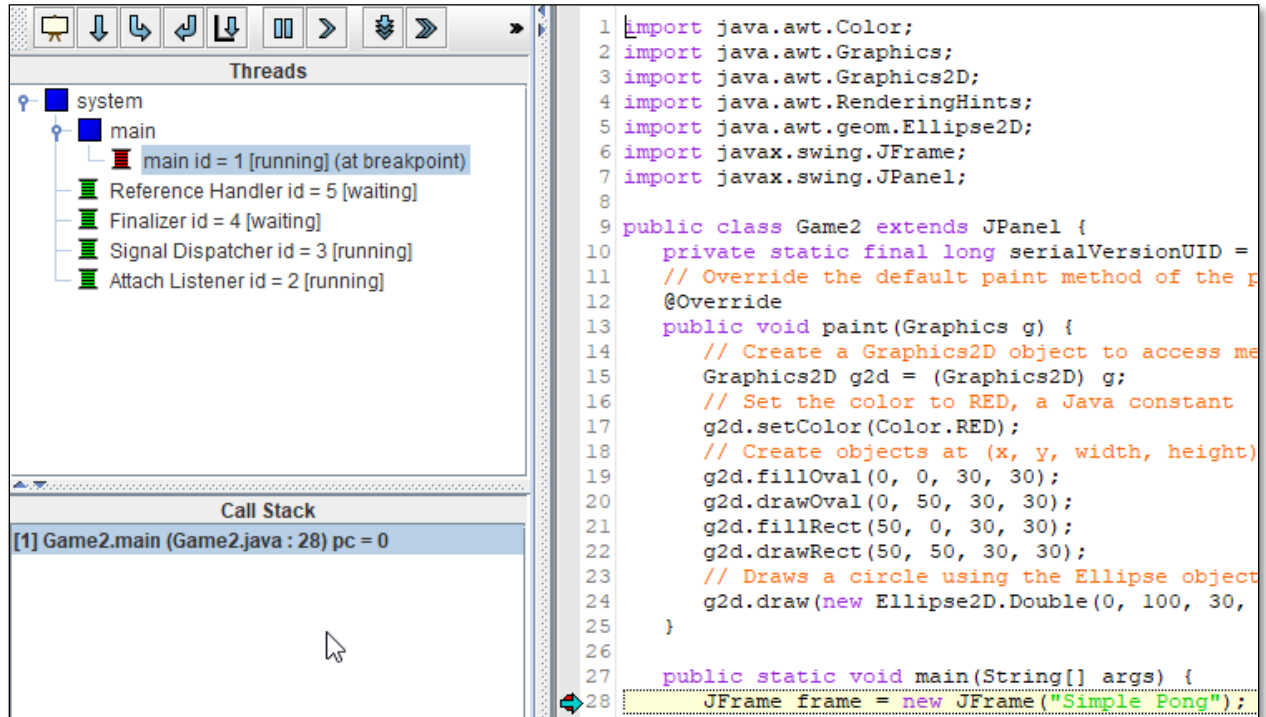


The instruction; **g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON)** makes the borders of the figure smoother, as you can see in the following graphic. The circle on the left is without applying ANTIALIAS and the one on the right is applying ANTIALIAS.



Analyzing the Concurrency and the Behavior of the Threads

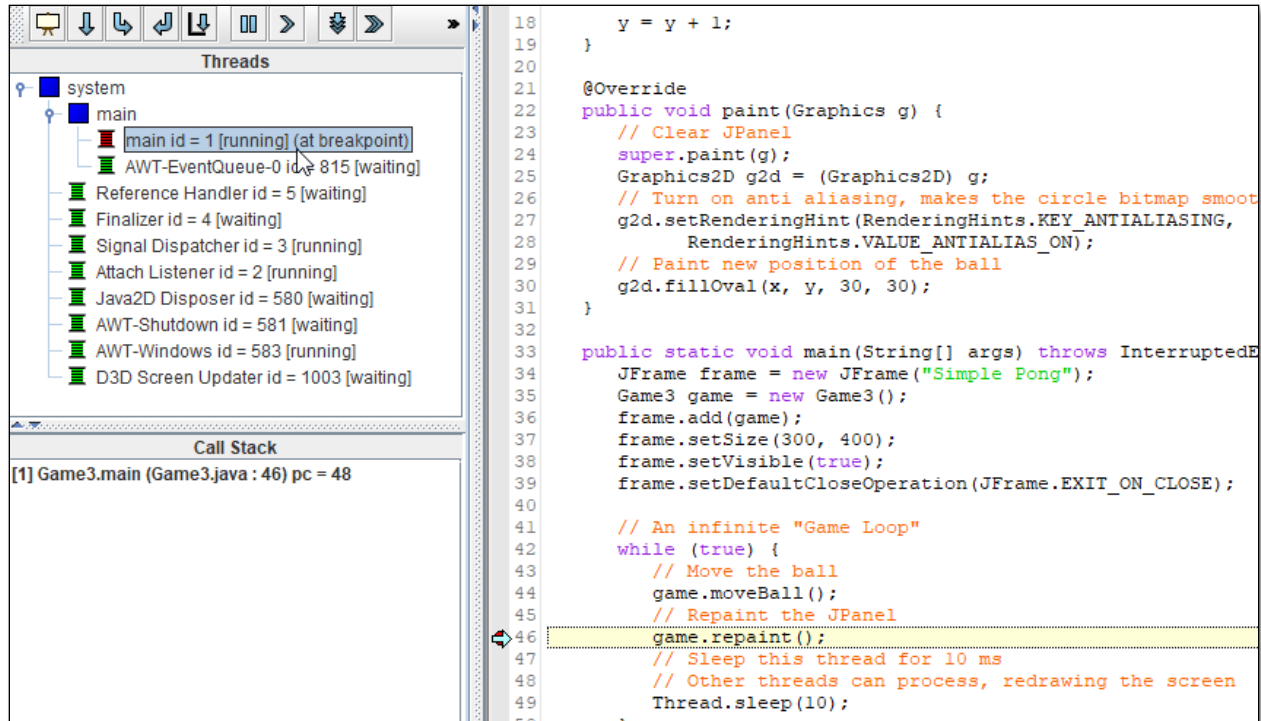
At the beginning of the execution of the main method, there is only one thread. We can see this, putting a breakpoint in the first line of the main method.



The screenshot displays an IDE interface with three main components: a toolbar at the top, a 'Threads' panel on the left, and a 'Call Stack' panel below it. The 'Threads' panel shows a tree structure starting with 'system', then 'main'. Under 'main', several threads are listed: 'main id = 1 [running] (at breakpoint)' (highlighted with a red bar), 'Reference Handler id = 5 [waiting]', 'Finalizer id = 4 [waiting]', 'Signal Dispatcher id = 3 [running]', and 'Attach Listener id = 2 [running]'. The 'Call Stack' panel shows a single entry: '[1] Game2.main (Game2.java : 28) pc = 0'. On the right, the source code for 'Game2.java' is visible. It includes imports for 'java.awt.*' and 'javax.swing.*', followed by a class definition 'Game2 extends JPanel'. The code contains a 'paint' method with various drawing calls and a 'main' method that creates a 'JFrame' titled 'Simple Pong'. A breakpoint is set at line 28, which is the first line of the 'main' method: 'JFrame frame = new JFrame("Simple Pong");'.

```
1 import java.awt.Color;
2 import java.awt.Graphics;
3 import java.awt.Graphics2D;
4 import java.awt.RenderingHints;
5 import java.awt.geom.Ellipse2D;
6 import javax.swing.JFrame;
7 import javax.swing.JPanel;
8
9 public class Game2 extends JPanel {
10     private static final long serialVersionUID =
11     // Override the default paint method of the p
12     @Override
13     public void paint(Graphics g) {
14         // Create a Graphics2D object to access me
15         Graphics2D g2d = (Graphics2D) g;
16         // Set the color to RED, a Java constant
17         g2d.setColor(Color.RED);
18         // Create objects at (x, y, width, height)
19         g2d.fillOval(0, 0, 30, 30);
20         g2d.drawOval(0, 50, 30, 30);
21         g2d.fillRect(50, 0, 30, 30);
22         g2d.drawRect(50, 50, 30, 30);
23         // Draws a circle using the Ellipse object
24         g2d.draw(new Ellipse2D.Double(0, 100, 30,
25     }
26
27     public static void main(String[] args) {
28         JFrame frame = new JFrame("Simple Pong");
```

If we add a breakpoint in the line with; **game.repaint()** and in the first line of the paint method and we then press F8 (Resume continues the execution to the end till it gets to the following breakpoint), we obtain:



In the left hand side we can see that four threads have been created and two of them are stopped in breakpoints. The main Thread is stopped in line 40 in the **game.repaint()** instruction. The **AWT-EventQueue** thread is stopped in the paint method in line 22.

If we select the **AWT-EventQueue** thread in the Debug view and we press F8 twice, we will see that it no longer stops in the paint method. This is because the operating system doesn't have a reason to ask for the canvas to repaint itself, once it is initialized.

If we press F6 (the thread execution moves only one line); (this time over the main thread) we will see that the paint method is called again by the **AWT-EventQueue** thread. We now take out the breakpoint of the paint method, we press F8 and we once again have only the main thread stopped.

Each call to **moveBall()** increases the position (x, y) of the circle and the call to repaint() tells the **AWT-EventQueue** thread to again paint the canvas.

The line "**Thread.sleep(10)**" (the last instruction inside the "Game loop"). To see what this does, comment the line with // and execute without debug. The result is that the circle is not painted in the canvas. Why does this happen? This is because the main thread takes over the processor and does not share it with the **AWT-EventQueue** thread, which cannot then call the paint method.

"Thread.sleep(17)" tells the processor that the thread which is being run must sleep for 17 milliseconds, which allows the processor to execute other threads and the **AWT-EventQueue** thread which calls the paint method. 17 milliseconds is approximately 60 frames per second.

This example illustrates the concepts of "game loop", threads and concurrence. There are better ways to manage the game loop and the concurrence in a game, coming up next.

Assignment Submission

Attach the .java files to the assignment in Blackboard.