Lab 02: Bouncing Spheres

In this lab, you will be implementing a (somewhat simplified) 2-dimension physics simulator.

First, copy and paste the following code into `Bouncing.java`:

```
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Dimension;

public class Bouncing extends JPanel{
    public static final int WIDTH = 1024;
    public static final int HEIGHT = 768;
    public static final int FPS = 60;
    public static final int RADIUS = 50;
    double positionX;
    double positionY;

    //Note: The following are not used yet, you should use them in writing your code.
    double velocityX;
    double velocityY;

    double accelerationX;
    double accelerationY;

    class Runner implements Runnable{
        public Runner()
        {
            //Feel free to change these default values
            positionX = 275;
            positionY = HEIGHT - 275;
            velocityX = 100;
            velocityY = -100;
            accelerationY = 98;
            //your code here for adding the second sphere

        }
        public void run()
```

```java
        {
            while(true){
                //your code here
                //Implement Movement  here
                positionX += 20 / (double)FPS; //delete this line
                positionY += 40 / (double)FPS; //delete this
                //(Use velocityX and velocityY rather than fixed constants)

                //Implement bouncing here

                //Implement gravity here (Bonus)

                //don't mess too much with the rest of this method
                repaint();
                try{
                    Thread.sleep(1000/FPS);
                }
                catch(InterruptedException e){}
            }
        }
    }

    public Bouncing(){
        this.setPreferredSize(new Dimension(WIDTH, HEIGHT));

        Thread mainThread = new Thread(new Runner());
        mainThread.start();
    }
    public static void main(String[] args){
        JFrame frame = new JFrame("Physics!!!");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Bouncing world = new Bouncing();
        frame.setContentPane(world);
        frame.pack();
        frame.setVisible(true);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        g.setColor(Color.BLACK);
        g.fillRect(0, 0, WIDTH, HEIGHT);

        //The cannon you see is actually *not* a photograph of a real cannon.
        //It's drawn by the following.
        g.setColor(Color.ORANGE);
        int xpts[] = {75, 275, 275, 350, 325, 150};
        int ypts[] = {HEIGHT-50, HEIGHT-250, HEIGHT-275, HEIGHT- 175, HEIGHT-175, HEIGHT-
25};
        g.fillPolygon(xpts, ypts, 6);

        g.setColor(Color.BLUE);
        g.fillOval(150, HEIGHT-200, 200, 200);

        //this is where the sphere is drawn. As a bonus make it draw something else
        // (e.g., your object from the previous lab).
        g.setColor(Color.WHITE);
        g.drawOval((int)positionX, (int)positionY,  RADIUS,  RADIUS);
        //your code here for drawing the second sphere
    }
}
```

There are two classes in this file: `Bouncing`, and `Runner`.
`Bouncing` is the main class which does the following:

a. The nitty-gritty of setting up a `JFrame` to let us draw to the screen.
b. Drawing the cannon and sphere.
c. Creating and starting the `Thread` (`mainThread`) which handles the sphere physics.

We'll learn more about `Thread`s soon, but the important thing to note is that `Runner`'s `run` method runs in parallel (at the same time as) the drawing code. Therefore updating the position of the sphere in the `run` method (in the `while(true)` loop) causes it to move on the screen.

To complete this lab, you should:

1. Implement edge detection so that the sphere stays in the window and bounces around.
   If the sphere moves passed the left, right, top, or bottom edge of the window (defined by `HEIGHT` and `WIDTH`), you should:
   a. Move the sphere to back within the window (at the end).
   b. Make it bounce (Hint: What should happen to `velocityX` and `velocityY`?)
2. Implement a second sphere. You may place it wherever you want initially, with some initial velocity. It should respond to gravity (if you decide to implement gravity) and bounce the same as the first sphere, but be a different color.

If you feel like a challenge:

1. Implement gravity. Note that in `Runner`'s constructor, `accelerationY` is set to 98. You don't need to change this, but you are welcome to play around with it. What you need to do is to update the position of the ball such that that acceleration is applied. (Hint: use `velocityX` and `velocityY` to keep track of the sphere's velocity.)
   You don't need to be super accurate in your physics simulation. A simple formula to use when updating the sphere's velocity for one loop iteration is:
   $V^{\text{new}} = V^{\text{old}} + At$
   where A is the acceleration and t is the time it takes to run the loop. In the code, t is one over the framerate (`1.0 / FPS`).
2. Change the sphere to be a more intricate object. For example, you could make it be whatever object you created in Lab 01. (Feel free to copy and paste code from that)
3. Turn one of the spheres into a vortex (that is, have it pull in a gravity-like fashion on the other sphere).

What you should submit:

`Bouncing.java`

In addition, fill out the form here:
https://cosc112s22.page.link/Lab02Survey