Lab 04: Object Oriented Bouncing Shapes

In this lab we will be exploring the use of creating our own classes. We'll be creating a program very similar to what we had in an earlier lab, but this time we'll refactor the structure with additional use several additional user defined classes.

To begin, copy and paste the following code into `OOBouncing.java`
(Note, it's oh-ohBouncing.java. It's not zero-zeroBouncing.java.)

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

class Pair{
      //HACKED BY Pr0HaX0r
      //HAHAHAHA
}

abstract class Shape{
    Pair position;
    Pair velocity;
    Pair acceleration;
    double width;
    double halfwidth;
    double dampening;
    Color color;
    public Shape()
    {
        Random rand = new Random();
        position = new Pair(500.0, 500.0);
        velocity = new Pair((double)(rand.nextInt(1000) - 500), (double)(rand.nextInt(1000) - 500));
        acceleration = new Pair(0.0, 200.0);
        width = 50;
        halfwidth = width / 2.0;
        dampening = 1.1;
        color = new Color(rand.nextFloat(), rand.nextFloat(), rand.nextFloat());
    }
```

```
32
33      public void update(World w, double time){
34          position = position.add(velocity.times(time));
35          velocity = velocity.add(acceleration.times(time));
36          bounce(w);
37      }
38
39      public void setPosition(Pair p){
40          position = p;
41      }
42
43      public void setVelocity(Pair v){
44          velocity = v;
45      }
46
47      public void setAcceleration(Pair a){
48          acceleration = a;
49      }
50
51      abstract public void draw(Graphics g);
52
53      private void bounce(World w){
54          Boolean bounced = false;
55          if (position.x - halfwidth < 0){
56              velocity.flipX();
57              position.x = width;
58              bounced = true;
59          }
60          else if (position.x + halfwidth > w.width){
61              velocity.flipX();
62              position.x = w.width - halfwidth;
63              bounced = true;
64          }
65          if (position.y - halfwidth < 0){
66              velocity.flipY();
67              position.y = halfwidth;
68              bounced = true;
69          }
70          else if(position.y + halfwidth >  w.height){
71              velocity.flipY();
```

```
            position.y = w.height - halfwidth;
            bounced = true;
        }
        if (bounced){
            velocity = velocity.divide(dampening);
        }
    }
}

class Sphere extends Shape{
    public void draw(Graphics g){
        Color c = g.getColor();

        g.setColor(color);
        g.drawOval((int)(position.x - halfwidth), (int)(position.y - halfwidth), (int)(width), (int)(width));
        g.setColor(c);
    }
}

class Square extends Shape{
    public void draw(Graphics g){
        Color c = g.getColor();

        g.setColor(color);
        g.drawRect((int)(position.x - width/2), (int)(position.y - width/2), (int)(width), (int)(width));
        g.setColor(c);
    }
}

class World{
    int height;
    int width;

    int numShapes;
    Shape shapes[];

    public World(int initWidth, int initHeight, int initNumShapes){
        width = initWidth;
        height = initHeight;
```

```
111
112         numShapes = initNumShapes;
113         shapes  = new Shape[numShapes];
114 
115         for (int i = 0; i < numShapes; i ++)
116             {
117                 if (i % 2 == 0)
118                     shapes[i] = new Sphere();
119                 else
120                     shapes[i] = new Square();
121             }
122     }
123 
124     public void drawShapes(Graphics g){
125         for (int i = 0; i < numShapes; i++){
126             shapes[i].draw(g);
127         }
128     }
129 
130     public void updateShapes(double time){
131         for (int i = 0; i < numShapes; i ++)
132             {
133                 shapes[i].update(this, time);
134             }
135     }
136 
137 }
138 
139 public class OOBouncing extends JPanel{
140     public static final int WIDTH = 1024;
141     public static final int HEIGHT = 768;
142     public static final int FPS = 60;
143     World world;
144 
145     public void run()
146     {
147         while(true){
148             world.updateShapes(1.0 / (double)FPS);
149             repaint();
150             try{
```

```
                Thread.sleep(1000/FPS);
            }
            catch(InterruptedException e){}
        }

    }

    public OOBouncing(){
        world = new World(WIDTH, HEIGHT, 50);
        this.setPreferredSize(new Dimension(WIDTH, HEIGHT));
    }

    public static void main(String[] args){
        JFrame frame = new JFrame("Physics!!!");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        OOBouncing mainInstance = new OOBouncing();
        frame.setContentPane(mainInstance);
        frame.pack();
        frame.setVisible(true);
        mainInstance.run();
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.BLACK);
        g.fillRect(0, 0, WIDTH, HEIGHT);
        world.drawShapes(g);
    }
}
```

Oh no! It looks like the nefarious Pr0Hax0r has struck. They may have tried to delete the entire file, but luckily they only managed to delete the contents of the `Pair` class.
Unfortunately, as part of their attack, they also deleted the code from my own copy of the file. And my local backups. And my person offsite backups. And the backups on Dropbox. And the backups that Amherst College maintains. And the files in the git repository. And my memory of how I implemented it in the first place ("Pr0" isn't in this hacker's name for no reason, after all).

> *Note: As a researcher working in security I feel compelled to tell you: this is not how hacking works. The story about `Pr0Hax0r` is a slightly exaggerated telling of when I intentionally deleted the code in the `Pair` class. For the purposes of narrative for the assignment, however, it's my story. If you're interested in how hacking actually works, you should take our COSC 383 (Security) course.*

Your task is to figure out what methods and fields went into the `Pair` class, and how they worked. The code initially provided won't compile. But if you try, the error messages may be helpful in determining what went into the `Pair` class. By using these error messages, and reading through the code, you should reverse engineer what was in the `Pair` class.

To complete the lab:

To complete this lab, you should implement the fields, constructor, and methods in the `Pair` class. The **access control**, **return type**, **name**, and **parameter types** for the constructor and methods can be inferred from how `Pair` is used throughout the rest of the program. To figure out what's in the body of each, you'll need to look at the code for `World` and `Sphere` and see how `Pairs` are used and remember how the physics from the last lab works. As a hint, if you're writing more than 2 or 3 lines for any single method, you're probably doing too much.

Tips:

1. Get out a piece of paper and pencil when you're going through the code. Take notes on what gets called where, and what the high level idea of the logic behind each method is.
2. When reading a method, remember which class it's in. For example the `bounce` method is in `Sphere`, which is an indication that it only handles bouncing for one sphere (if it was in `World`, then you might expect it to loop over all the `Sphere`s in the world).
3. Some classes are reasonably complicated, while others are very simple. `Sphere`, for example, has a lot going on in it. `Pair` should end up being much simpler than `Sphere` or `World`. You shouldn't need more than a few fields for the class and a few lines in each method.

To go beyond

If you're looking for a challenge, here are some ideas:

1. Right now, the `Sphere` constructor takes no arguments, and sets properties of the sphere randomly. Add another constructor (we'll cover having multiple constructors in the next lecture — if you don't feel comfortable adding one, just change the existing one) that takes in desired quantities (color, initial location, etc.) and then create spheres in a pattern more pleasing than random.
2. Right now, `World` just takes a width, height, and number of spheres in its constructor. Add other properties to the world, like how bouncy each wall is, how strong and in what direction gravity pulls, etc..
3. Create a new class: `Spring` which has two `Sphere` objects as fields. In `Spring`, have a method which pulls the spheres together if they are more than a certain distance apart, and pushes them apart if they are less that distance apart. If you want to be physically accurate, use Hooke's law (https://en.wikipedia.org/wiki/Hooke%27s_law).
   Don't forget to create a `Spring` instance and call the method as appropriate.
4. Make the spheres collide with each other. (Note: you can do this by treating the spheres as spheres, in which case you need to do a bit of geometry to figure out how they should bounce off of each other, or you can treat them as cubes for a less realistic, but easier to code/debug collision detection).

What you should submit:

`OOBouncing.java`
Remember to also fill out the survey:

https://cosc112s22.page.link/Lab04Survey