Chapter 3: Ball: A Class That Moves Itself Around

Ball: A Class That Moves Itself Around

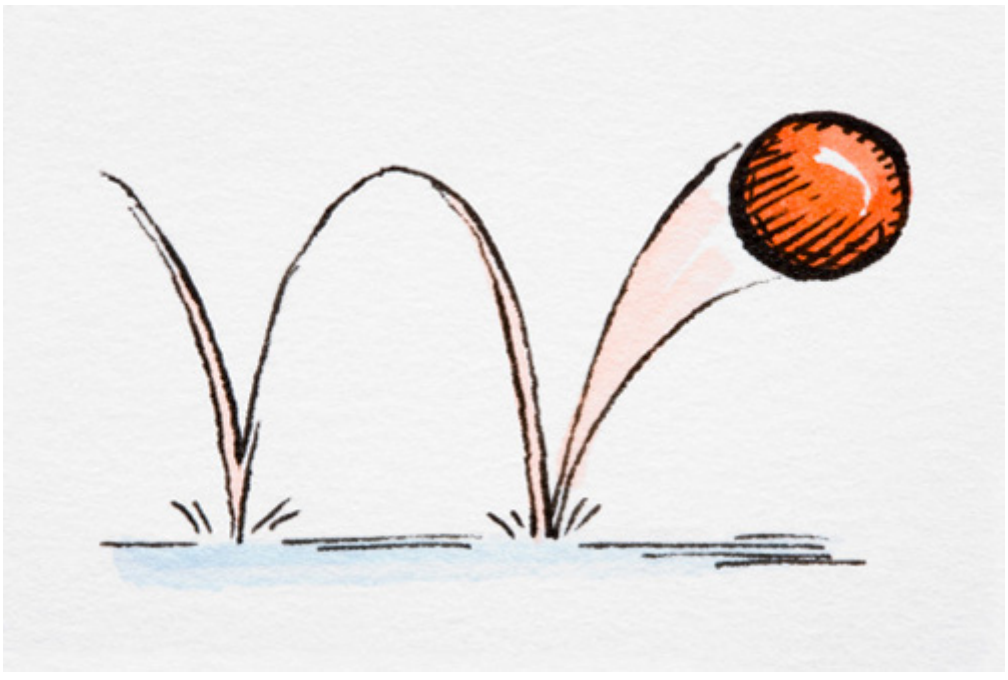
Here's what my Ball class looks like:

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.Graphics2D;

public class Ball {
    private Dimension screenSize;
    private Color color;
    private Point location;
    private int radius;
    private int diameter;
    private Point speed;

    public Ball(Dimension screenSize) {
        this.screenSize = screenSize;
        radius = ((int)(Math.random()*40))+10;
        diameter = radius * 2;
        location = new Point(screenSize.width/2,screenSize.height/2);
        color = new Color((float)Math.random(),(float)Math.random(),
(float)Math.random());
        speed = new Point(1 + ((int)(Math.random()*10)), 1 + ((int)
(Math.random()*10)));
    }

    public int getDiameter() {
        return diameter;
    }

    public Color getColor() {
        return color;
    }

    public Point getLocation() {
        return location;
    }

    public Point getSpeed() {
        return speed;
    }

    public void move() {
        location.setLocation(location.x+speed.x,location.y+speed.y);
```

```
        if ( (location.x < 0) || (location.x > screenSize.width-diameter) )
            speed.x = -speed.x;
        if ( (location.y < 0) || (location.y > screenSize.height-diameter) )
            speed.y = -speed.y;
    }

    public void paint(Graphics2D g2d) {
        g2d.setColor(color);
        g2d.fillOval(location.x, location.y, diameter, diameter);
    }
}
```

Again, let's go through it piece by piece. First let's look at its variable declarations.

- The Dimension variable screenSize holds the dimensions of the window that the ball can bounce around in. We need that size in order to calculate our bounces correctly. (Try giving it a size that doesn't match the size of your window and see what happens!)



- The Color variable color holds the color of the ball.

- The Point variable location holds the ball's current location. A Point object holds two numbers (an x-coordinate and a y-coordinate) that represent a point on the screen.

- The int variable radius holds the radius of the ball.

- The int variable diameter holds the diameter of the ball. It's not necessary to store both the radius and the diameter since you can easily calculate one based on the other, but I chose to do it this way since it simplifies the expressions in the class.

- The Point variable speed holds the ball's current speed. It shows another use for the two numbers in the Point class instead of representing a fixed location. The x-coordinate in this object is the ball's horizontal speed. A positive number moves right, and a negative number moves left. The y-coordinate is the ball's vertical speed. A positive number moves down, and a negative number moves up. The larger the number is, the faster the ball moves.

Ball's Constructor

Next let's look at the class constructor. It requires a Dimension argument from the caller that tells us how much room the ball has to bounce around in.

The first line in the constructor stores the value the caller gives us into our instance variable. You can see another use of the keyword "this" here, and it's a common way to use it.

The variable in the class and the parameter in the constructor have the same name. Since the parameter is a local variable in the constructor, if I use that name alone it refers to the parameter. If I put "this." in front of the name, it refers to the object's variable. So the first line copies the value from the parameter and stores it in the variable so we can use it later.

```
public Ball(Dimension screenSize) {
    this.screenSize = screenSize;
    radius = ((int)(Math.random()*40))+10;
    diameter = radius * 2;
    location = new Point(screenSize.width/2,screenSize.height/2);
    color = new Color((float)Math.random(),(float)Math.random(),
                      (float)Math.random());
    speed = new Point(1 + ((int)(Math.random()*10)),
                      1 + ((int)(Math.random()*10)));
}
```

The second line sets the ball's radius. It uses a new (to us) method from the Math class named random(). Math's random() method is a random number generator that returns a random double value between zero and one. Calling it gets us a number whose value we can't predict, which is often useful in programming. In this case I'm using it to randomize the size of the ball.

```
public Ball(Dimension screenSize) {
    this.screenSize = screenSize;
    radius = ((int)(Math.random()*40))+10;
    diameter = radius * 2;
    location = new Point(screenSize.width/2,screenSize.height/2);
    color = new Color((float)Math.random(),(float)Math.random(),
                      (float)Math.random());
    speed = new Point(1 + ((int)(Math.random()*10)),
                      1 + ((int)(Math.random()*10)));
}
```

If you run the bouncing ball project several times, you should notice that the ball's size and color change each time. In this case, once I get the random number, I multiply it by 40 to get a random value between 0 and 40. Then I add 10 to that result to get a random number between 10 and 50.

```
public Ball(Dimension screenSize) {
    this.screenSize = screenSize;
    radius = ((int)(Math.random()*40))+10;
    diameter = radius * 2;
    location = new Point(screenSize.width/2,screenSize.height/2);
    color = new Color((float)Math.random(),(float)Math.random(),
                      (float)Math.random());
    speed = new Point(1 + ((int)(Math.random()*10)),
                      1 + ((int)(Math.random()*10)));
}
```

Finally the radius value gets cast from a double to an int. The result of the calculation is a ball with a random radius between 10 and 50 pixels.

```
public Ball(Dimension screenSize) {
    this.screenSize = screenSize;
    radius = ((int)(Math.random()*40))+10;
    diameter = radius * 2;
    location = new Point(screenSize.width/2,screenSize.height/2);
    color = new Color((float)Math.random(),(float)Math.random(),
                      (float)Math.random());
    speed = new Point(1 + ((int)(Math.random()*10)),
                      1 + ((int)(Math.random()*10)));
}
```

The third line calculates and stores the diameter from the radius. As I said earlier, that's not necessary, but it simplifies the expressions we'll use later.

```
public Ball(Dimension screenSize) {
    this.screenSize = screenSize;
    radius = ((int)(Math.random()*40))+10;
    diameter = radius * 2;
    location = new Point(screenSize.width/2,screenSize.height/2);
    color = new Color((float)Math.random(),(float)Math.random(),
                      (float)Math.random());
    speed = new Point(1 + ((int)(Math.random()*10)),
                      1 + ((int)(Math.random()*10)));
}
```

The fourth line sets the ball's starting location. For simplicity's sake I decided to always start the ball at the middle of the window, so I used an x-coordinate that's half the screen's width and a y-coordinate that's half the screen's height. To do those calculations I used the height and width fields from the Dimension object screenSize.

```
public Ball(Dimension screenSize) {
    this.screenSize = screenSize;
    radius = ((int)(Math.random()*40))+10;
    diameter = radius * 2;
    location = new Point(screenSize.width/2,screenSize.height/2);
    color = new Color((float)Math.random(),(float)Math.random(),
                      (float)Math.random());
    speed = new Point(1 + ((int)(Math.random()*10)),
                      1 + ((int)(Math.random()*10)));
}
```

The fifth line sets the ball's color value. It uses the random() method again for each of the three RGB values so that I get a different color each time I run the program.

```
public Ball(Dimension screenSize) {
    this.screenSize = screenSize;
    radius = ((int)(Math.random()*40))+10;
    diameter = radius * 2;
    location = new Point(screenSize.width/2,screenSize.height/2);
    color = new Color((float)Math.random(),(float)Math.random(),
                      (float)Math.random());
    speed = new Point(1 + ((int)(Math.random()*10)),
                      1 + ((int)(Math.random()*10)));
}
```

The sixth (and last) line of the constructor sets the ball's speed. I used random() in the calculations again to get two random numbers between 0 and 10, and then I added 1 to them so the speed will never be zero. Our beginning speeds are always positive, so the ball will always start out moving down and to the right, but the angle and speed will vary.

```
public Ball(Dimension screenSize) {
    this.screenSize = screenSize;
    radius = ((int)(Math.random()*40))+10;
    diameter = radius * 2;
    location = new Point(screenSize.width/2,screenSize.height/2);
    color = new Color((float)Math.random(),(float)Math.random(),
                      (float)Math.random());
    speed = new Point(1 + ((int)(Math.random()*10)),
                      1 + ((int)(Math.random()*10)));
}
```

Ball's Other Methods

The next methods in Ball are simple get methods that allow a user to see what the ball's settings are. I haven't used those methods in this project, but they're there if you want to use them. For example, you could use them to display the ball's settings in the window where it's bouncing around if you wanted to.

The last two methods are the ones I want to describe in detail. They're the move() method, which "moves" the ball, and the paint() method, which draws the ball onto the panel.

Let's see how the move() method works first. It moves the ball by calling the Point method setLocation() to set new coordinates. The new location is the previous one with the ball's speed values added to it. So the ball will move from 1 to 10 pixels up or down and from 1 to 10 pixels left or right to its new location.

Once the new location is set, the method checks to see if we've reached one of the boundaries of our window. It checks the x-coordinate first to see if we've moved off the left or right edge.

Remember, the location in our Point is for the top-left corner of an imaginary box around the ball. So to check the left edge, we compare our x-coordinate with 0, the left boundary. To check the right edge, though, we compare the x-coordinate with the width of our window minus the ball's diameter. That way we'll change direction as soon as the right edge of our ball (our x-coordinate plus the ball's diameter) touches the right edge of the screen.

If our x-coordinate is outside those values, we change our speed's sign so the ball will start moving in the opposite direction. After checking the x-coordinate, it also checks the y-coordinate against the screen's height using the same technique.

Last let's look at the paint() method. After all we've gone over so far, it's pretty simple. It takes the Graphics2D object that our caller passes to us and uses it to draw on. The first line of the method sets the drawing color to the color of the ball, and the second line draws an oval on the panel at the ball's location using the ball's diameter.

And with that, our Ball class is complete. I know that some of the calculations here are new and not very straightforward, so if you have questions about how any of it works, please let me know in the Discussion Area.

Time to Play

Before we get into Chapter 4, take a break and play with the project for a bit. Try changing some of the settings like the window size, the ball's color or size, or its speed calculations, and see how it all works together.

In the next chapter I'll show you how to generate a larger number of balls and get them all running at once.