Chapter 4: Animating Multiple Objects

Animating Multiple Objects

We're down to our last major topic in our last lesson! In this chapter, I'll go over some modifications to our bouncing ball project that animates multiple balls in our window.

We'll start with the same three classes we used for the single bouncing ball. When I went to add more, I used BlueJ's **Save As** option to save the whole project under a new name, *Bouncing Balls*.

The good news is that the Ball class doesn't need to change at all for this to work. We can create and animate as many Ball objects as we like using the exact same Ball class.

The only changes to the top-level class are class name changes and a window title change. I changed my class names from BouncingBallWindow to BouncingBallsWindow and from BouncingBall to BouncingBalls. I also changed my window title to *Ten Bouncing Balls*. I don't need to go through that code with you again, so I won't show it here. If you want to look at it, it's behind this link:

Hide answer

```
import javax.swing.*;
/**
 * BouncingBallsWindow is the top level of three that produce a window with
 * 10 animated bouncing balls in it.
 *
 * @author Merrill Hall
 * @version 1
 */
public class BouncingBallsWindow extends JFrame {
    public BouncingBallsWindow() { // Create window
        setTitle("Ten Bouncing Balls");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        add(new BouncingBalls()); // Add bouncing balls panel

        pack();
        setLocationRelativeTo(null); // Center window in display screen
    }

    public static void main(String[] args) {
        // Run GUI in the Event Dispatcher Thread (EDT)
        SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    // Set up main window (using Swing's Jframe)
                    BouncingBallsWindow bbw = new BouncingBallsWindow();
                    bbw.setVisible(true);
                }
            });
    }
}
```

That means that most of our changes are in the middle class, which I renamed BouncingBalls. So one more time, here we go! I'll show you the code first, and then we'll discuss it:

```
import java.awt.*;
import java.util.Formatter;
import javax.swing.*;
/**
 * BouncingBalls
 *
 */
public class BouncingBalls extends JPanel {
    private final Dimension SIZE = new Dimension(600,450); // Panel size
    private final int UPDATE_RATE = 30; // Number of refreshes per second
    private Ball[] balls; // An array of 10 balls
    private Thread ballThread; // Animation thread

    /**
     * Constructor sets up the panel and starts its animation thread
     */
    public BouncingBalls() {
        int index = 0;
        setPreferredSize(SIZE);
        balls = new Ball[10];
        while (index < balls.length) {
            balls[index] = new Ball(SIZE);
            index++;
        }
        // Start the balls bouncing (in their own thread)
        ballThread = new Thread() {
                public void run() {
                    while (true) { // Execute one update step
                        // Calculate the balls' new positions
                        int index = 0;
                        while (index < balls.length) {
                            balls[index].move();
                            index++;
                        }
                        // Refresh the display
                        repaint(); // Callback paintComponent()
                        // Delay so the animation's not too fast
                        try {
                            Thread.sleep(1000/UPDATE_RATE);  // milliseconds
                        } catch (InterruptedException ex) { }
                    }
                }
```

```
        };
    ballThread.start();  // Callback run()
}

/**
 * Override the JPanel paintComponent method with our drawing of 10 balls
 */
public void paintComponent(Graphics g) {
    super.paintComponent(g);    // Paint background
    Graphics2D g2d= (Graphics2D) g;
    // Draw the box
    g2d.setColor(Color.BLACK);
    g2d.fillRect(0, 0, getWidth(), getHeight());

    // Draw the balls
    int index = 0;
    while (index < balls.length) {
        balls[index].paint(g2d);
        index++;
    }
  }
}
```

Since this class is similar to BouncingBall, I'm going to focus on their differences rather than go through this one line by line. The first difference is the third declaration. In BouncingBall, it's this line:

```
private Ball ball; // The beautiful bouncing ball
```

In BouncingBalls, it looks like this:

```
private Ball[] balls; // An array of 10 balls
```

I'm going to explain just a little about arrays—just enough for this lesson. I'll talk more about arrays in my *Intermediate Java Programming* course, in case you decide to take it. (Just thought I'd throw in a little plug for it here.)

What's an Array?

An *array* is a structure that appears in many programming languages, dating back to Fortran in the 1950s. It's a data structure that holds a group of items of the same type. Once you've built the array, you can reference any item in it by creating an *index* containing its position in the array.

> It's like having a shopping list and being able to check off the items one at a time in any order. I can check off the fourth item first if I find it at the store before the other items in the list, then the second item, the seventh item, the first item, and so on.

In Java, square brackets indicate arrays. In the declaration above, Ball[] tells Java that the variable balls refers to an array of Ball items instead of a single Ball object. Once I finish building the array, I can use the variable name followed by an index in square brackets to get a single Ball object out of the array. So balls[2] refers to a single Ball object, as does balls[5] or balls[7].

One important note about Java's array indexes, though: Java uses *zero-based indexing*. That means the first element of an array uses an index of 0, not 1. The reasons have to do with some fairly complicated formulas used to calculate memory addresses, and I'd rather not get into that here. But quite a few programming languages (including C, C++, C#, and others) use the same indexing scheme.

So in our array, balls[0] is the first object, balls[1] is the second one, and balls[9] is the last one. Be sure to tell me in the Discussion Area if any of this is confusing—I expect it might be.

The next difference between the classes is the first local variable in the constructor:

int index = 0;

We'll use this variable as the index when we're referring to our array.

Next, instead of one line that declares a single Ball object, we have these lines:

```
    balls = new Ball[10];
    while (index < balls.length) {
      balls[index] = new Ball(SIZE);
      index++;
    }
```

The first line actually creates the array and tells Java how many items we want it to hold. In this case it's 10, but it can be any number as long as you've got enough memory to hold the array. (My computer was okay with a hundred million entries, but it wouldn't allow a billion. I didn't try to pin it down any closer than that.)

The while loop that starts in the next line starts with a value of zero in index, and the loop repeats until index reaches the end of the array. Every Java array has a field named length that tells us the size of the array. So balls.length in this case contains the value 10, and the loop repeats 10 times. Each time through the loop, a new Ball object gets created and stored in the next array location. The next line increments index, both so the loop will end and so we'll update the next array location.

So at the point we set up our animation thread, we have 10 Ball objects in our array ready to go, each with its own color, size, location, speed, and direction.

The only difference in the animation thread is that instead of moving one ball before repainting the window, I have another small loop that pulls each of the 10 balls out of the array and calls its move() method, so that we move all 10 balls simultaneously.

In just the same way, in the paintComponent() method for this class, instead of calling one ball's paint() method, I have a third loop set up that gets all 10 balls and calls each of their paint() methods so that we repaint all 10 balls.

Time to Play

If you recompile the project and rerun it now, you should see 10 different-colored balls starting from the middle of your window and diverging in 10 directions with 10 different speeds. And if you want to play around with the number of balls, you can set up any number by just changing the size of the array in the third line of the BouncingBalls constructor from 10 to some other number (as long as it's a positive number).

Let me know in the Discussion Area if you have any trouble getting the recompiled project to work.

© 2022 Cengage Learning, Inc. All Rights Reserved