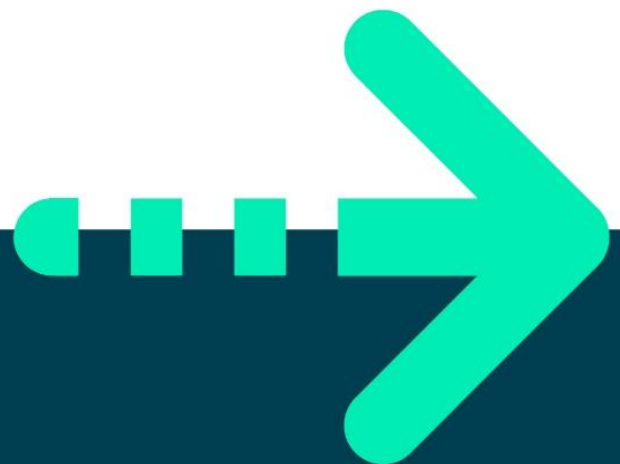




Types II – Fields and Methods

Java Fundamentals





Lab 9: Types II – Fields and Methods

Objective

You will further your understanding of object oriented programming. In this lab you'll create a graphical application instead of a Console app.

Part 1 - Step by step

You'll soon be creating a graphical application which is new. You'll copy and paste many lines of code. You are not required to understand how the graphics code works but it is not too complex to investigate by yourself.

1. In the **labs** project which you created in Lab1, add a new package called **lab09**.

Please refer to Lab1's instructions if you need help.

2. Create a class called **Game** in this package.
3. Create an instance of Game in the **main()** method. This will kickstart the game using its constructor.
4. You need a canvas upon which you can draw and show your animation. In this case you'll create a number of ball objects bouncing around a rectangle (the world).

You can get creative later and do something else but for now please follow the steps. The aim is to investigate Object Oriented, not concentrate on animation and graphics.

Delete any code in the **Game** class and copy the following code.

There are a few comments in this code which are placeholders for code which you'll write later on.



```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.util.Timer;
import javax.swing.*;

public class Game extends Canvas {
    // create an array of 3 balls
    // call this array balls

    Game() {
        JFrame frame = new JFrame();
        this.setSize(400, 400);
        frame.add(this);
        frame.pack();
        frame.setVisible(true);

        Timer t = new Timer();
        TimerTask tt = new TimerTask() {
            @Override
            public void run() {
                draw();
            }
        };

        t.schedule(tt, 0, 50);

        frame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                t.cancel();
                tt.cancel();
            }
        });
    }

    public void draw() {
        // call the move() method of each balls
        // Tip: use an enhanced for loop to pick
        // each ball in the balls array.
        this.repaint();
    }

    public void paint(Graphics g) {
        g.drawRect(0, 0, 300, 300);
        // move and draw each ball in balls array
        // Tip: use an enhanced for loop to pick
        // each ball in the balls array.
    }
}
```

Creates a form

Creates a timer to call
the draw() method every
50ms (1/20 sec)

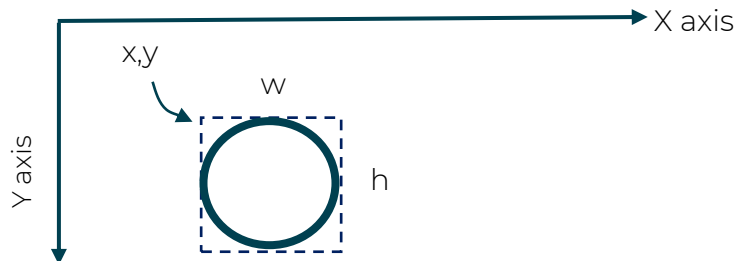
Runs when
the
window is
closed

Invokes a
repaint which
will run the
paint() method

5. Run the application to see if your code works as expected.

6. Create a class called **Ball** with the following fields:

```
public int x, y, w, h;
private int dirX, dirY;
```



A ball's dimension is represented as a rectangle. X and y are the top coordinates and w and h are the width and height of the ball's rectangle. dirX and dirY are the amount by which each ball moves in the x and y direction.

The coordinate system is perhaps unlike what you're familiar with as it is upside-down! X increases to the right and Y increases towards the bottom.

7. Create a constructor to set every field (x,y,w,h,dirX,dirY).
8. To practise using constructor chaining, create another constructor to set just x,y,w,h but not the dirX and dirY.

Use constructor chaining to call the first constructor with dirX=1 and dirY=1.

9. Create a method called **move()** in the **Ball** class.
In this method, increase x by dirX and y by dirY and then follow a series of tests to make sure each ball stays within the world which is set in the **paint** event as 300px wide, 300px high, and starts at x=0,y=0.

if (x < 0) then set x=0 and change the sign of dirX

Tip: **dirX = -dirX**

do the same for the y coordinate (when it goes below zero) but this time changing dirY.

if (x > 300-width of the ball) then set

x=300-width of the ball and reverse the sign of dirX.

do the same for the y coordinate (taking the height of the ball into account).

10. Back in the Game class, view the comments and perform those tasks (create three balls, call move()... and draw each ball).

Tip: Paint method can draw a circle using this code:

```
g.drawOval( b.x, b.y, b.w, b.h);
// where b is a ball reference
```

11. If everything works, you should see three objects bouncing on your screen. If



you have a need for speed then increase the dirX and dirY values!

Part 2 - Step by step

In this section you'll make use of the static keyword. Clearly, setting the world dimension to 300px (height and width) is not a good idea.

All balls share the same world coordinates. This is a perfect place to use the **static** keyword.

1. Create a static int field in Ball called **worldW** and another called **worldH** to represent the width and height of the world. You should see three balls bouncing!
2. Create static method as **public static void setWorld(int w, int h)** to set the static *worldW* and *worldH* values.
3. In the Game() constructor just before the line `Timer t = new Timer();` invoke the setWorld() method of the Ball class to set the world's width and height for every Ball.
4. Change the move() method in the Ball class to use the Ball.worldH and Ball.worldW values instead of 300px. You'll also need to change the paint() method to use the new static values (to draw the world's rectangle).
5. Run and test your code.

*** End ***

