
Java
Object-oriented
programming

Objective

The primary objectives for this lab are as follows:

- To create and use references.
- To consolidate on passing by value and passing by reference.

Part 1 – Creating and using reference types

Step by step

1. Back in the **labs** project, add a new class called **Program** with a **main**() method in a package called **exercise2**.
2. Create a class called **Account** in the **exercise2** package with the following fields.
   ```
   private int id;
   private String owner;
   private double balance;
   ```
   and methods:
   ```
   void deposit(double amount) { }
   void withdraw(double amount) { }
   ```
3. Make sure no one can deposit a negative amount of money!
4. Also make sure no one can withdraw money they don't have.
5. Create a constructor for the **Account** class to set its fields (id, owner and balance).
6. Create a method called **getDetails**() to return details of the account as a **String** made up of the id + owner + balance (put a ',' between these)
7. In the `main()` method create an instance of the Account class and then invoke its deposit()and withdraw() methods.
8. Create an account in the exercise2's main() method and then call the account instance's getDetails() method. Print the result to make sure your code works.

Part 2: Create a graphical application

Objective

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

Step by step instructions

You'll soon be creating a new graphical application. You'll copy and paste many lines of code and you are not required to understand how the graphics code works but it gives you the opportunity to try OO concepts in a fun way.

1. Back in the **exercise2** package create a class called **Game**.
2. Create an instance of Game in the **main()** method. This will kick start the **Game's** constructor.
3. 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.

Please copy the following code (below the **package** statement) in the Game class:

```
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 called balls
	// tip:
	//	Ball[] balls = {
	//		new Ball(30,30,20,20,1,2),
	//		new Ball(30,50,20,15,3,2),
	//		new Ball(40,60,30,20,2,3)
	// };


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() {
		this.repaint();
	}

	public void paint(Graphics g) {
		g.drawRect(1, 1, 300, 300);
		// Use an enhanced for loop to call the move() method of
```

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

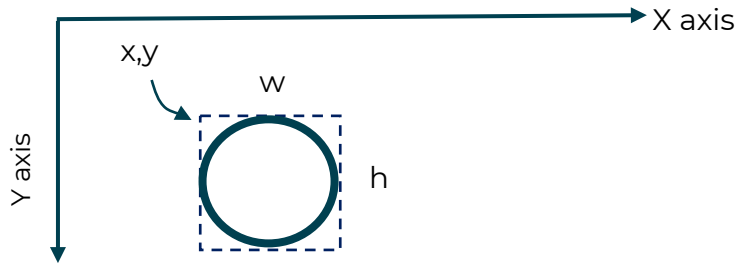
```
                // each ball in the balls array and then draw the ball
                //
                // Tip: use an enhanced for loop to pick
                //      each ball in the balls array.
                //      see below for more instructions
        }
}
```

5. Run the application to see if your code works as expected.
   You should see a rectangle drawn by the line g.drawRect(0, 0, 300, 300);

6. Create a class called **Ball** in the same package 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 coordinate 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 the x, y, w, h values but not the values for dirX and dirY.

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

   Tip:
```
public Ball(int x, int y, int w, int h) {
    this(x, y, w, h, 1, 1);
}
```

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

if (x < 1) then set x=0 and change the sign of dirX
Tip: **dirX = -dirX**

Do the same for the y coordinate (y < 1) but this time changing **dirY**.

if (x+w > 300) then set x=300-w and then reverse the sign of dirX.

Do the same for the y coordinate.
if (y+h > 300) then set y=300-h and then reverse the sign of dirY.

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

    Tip: in the **paint** method you can draw a circle using this code:

    ```
    // use an enhanced for loop to pick each ball in the Balls array
    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 3 (optional) – Passing reference types to a method

In this part you'll examine passing reference types between methods. This is an important topic which has security implications.

Step by step

1. In the Account class, create a new method as:
   public void **addInterest**()

   Write code in this method to add 2.5% interest to the balance.

2. Add code in main() to create an account *instance called ***myAccount***, with a starting balance of £100.

3. Call the addInterest() method of the myAccount instance..

4. After the call, display the details of myAccount (using its getDetails()).
   Did the balance change?

5. Create another Account reference called **partnerAccount** and set it to the myAccount instance like:
   Account partnerAccount = myAccount;

6. Call the **partnerAccount.addInterest()** method

7. Call **myAccount.getDetails()** method. Did the balance change? Why?

Let's do another experiment

8. Create a method in the **Program** class (just under the main())
   static void **processAccount**(**Account acc**){
   acc.addInterest();
   }

9. Back in Main(),call the **processAccount()** method, passing **myAccount** as parameter.

10. Call myAccount.getDetails() method. Did the balance change? Why?

11. Let's do another important experiment.

12. Create another method under the main() method as:
```
private static void incInt(int x) {
      x++;
}
```

13. Add code to the main() method to create an integer like
```
int k=100;
```

14. Call **incInt** and pass it **k**

15. After the call, print the value of k.

    Did k change after the call?

Explanation

All primitive data types (built into the language) are **Passed by Copy** (also known as **passed by value**). Therefore, changing **x** did not change **k** and changing the name of **k** to **x** will not alter anything!

All Reference types (like myAccount) are also passed by copy but what is passed is their address in memory. Therefore, when you call a method like **processAccount**(**Account acc**) and pass it an account reference (such as myAccount) you are actually passing its address! The **acc** reference will be pointing to (referencing) myAccount. Therefore, any change made by the **acc** reference will directly affect myAccount.