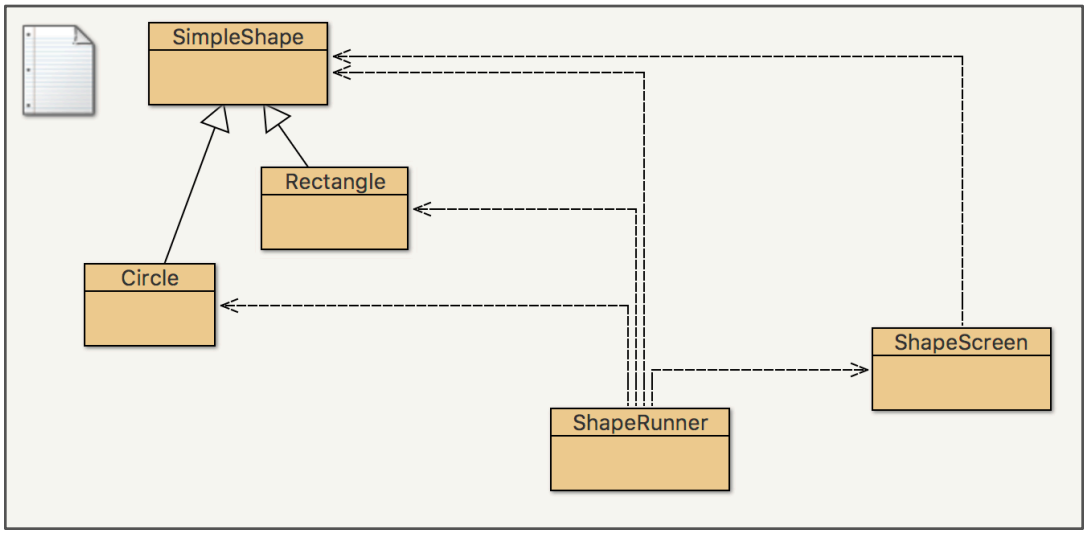
The University Of The West Indies
Five Islands Campus
Object Oriented Programming Concepts
Week6 Lab

In this lab, we will explore the polymorphic behaviour of subclass and superclass instances. This lab builds on the concepts of Inheritance, method overriding and replacement.

Part 1: Polymorphism, Method Binding, Principle of Substitutability



1. Create a new project in BlueJ called Lab 6.
2. Retrieve the following classes from the eLearning course website: **SimpleShape.java, Circle.java, Rectangle.java, ShapeRunner.java** and all 4 of the **.class** files for the **ShapeScreen.** Compile all java files. Run the **ShapeRunner** file and observe the output.
3. Create the following instances in the **ShapeRunner** class and invoke the **toString( )** method on them and print the output.

Object	Object Type	Features
s1	SimpleShape	
s2	Rectangle	Length = 50, Breadth = 100

4. Modify the **toString( )** method (inherited from the **SimpleShape** class) in the **Rectangle** class so that it prefixes the word "Rectangle" to the String produced in the parent **toString( )** method.

Is this an example of method refinement or method replacement?

TIP: Invoking a parent method from a child class

super.
methodName( )

5. Change the declaration of the instance **s2** in the **ShapeRunner** class to be of type **SimpleShape**. Observe what happens to the output when you execute the **ShapeRunner** class. Did anything change? Which **toString( )** method was selected for execution on **s2** the one in the parent or the child class?

Answer:

6. Modify the **toString( )** method (inherited from the **SimpleShape** class) in the **Circle** class so that it prefixes the word "Circle " to the String produced in the parent **toString( )** method.
7. Create the following instance in the **ShapeRunner** class but declare it to be of type **SimpleShape** and instantiate them as the respective Object type in the table.

TIP: Declaration vs Instantiation

DeclaredClass obj = new InstantiatedClass(..)

Object	Object Type	Features
s3	Circle	Radius = 50

8. Create the following instances in the **ShapeRunner** class but declare and instantiate them as the respective Object type in the table.

Object	Object Type	Features
s4	Circle	Radius = 30
s5	Rectangle	Length = 300, Breadth = 100

9. Invoke the **toString( )** method on the instances from Step 7 and print the output. Observe the outcome and identify which **toString( )** method (from the subclass or the superclass) is being called by each instance.

Answer:

10. Identify the **static** type and the **dynamic** type of each instance in the **ShapeRunner** class.

Answer:
s1:
s2:
s3:
s4:
s5:

11. Let's try to reduce the 5 print statements to run in a loop.
    (a) Create an array of 5 **SimpleShape** objects called **shapes**

```
SimpleShape[ ] shapes = new SimpleShape[5];
```

    (b) Insert the 5 objects (**s1..s5**) into the array. Did this work? Why?

```
/* e.g. */   shapes[0] = s1;
```

Answer:

    (c) Type the following code to iterate through the array and print the details of the objects in the array. This is a different way of writing a `for` loop in Java.

```
for (SimpleShape ss: shapes){
    System.out.println(ss.toString());
}
```

    Did this work? What is the **static** type of the objects in the **shapes** array? Why are we able to invoke **toString( )** like this?

Answer:

12. Override the **calculateArea( )** methods in the **Rectangle** and **Circle** classes so that the **toString( )** method works more correctly.
13. Invoke the **calculateArea( )** method on the instances within the loop from 11(c). Observe what happens to the output. Why doesn't **s1** have an area? What is the area of a Shape?

TIP: Use the Math class in Java to get the value of PI

Math.PI

Answer:

14. Type the following line of code in the **ShapeRunner**:

```
Rectangle s6 = new Shape( );
```

    Did this compile? Explain why the compilation error occurs.

Answer:

Part 2: Reverse Polymorphism

The **ShapeScreen** class has a method that will render the shapes specified in the array on the Applet window. However, the method requires that all **SimpleShape** objects provide a **draw( )** method that returns a **java.awt.Shape** object.

1. Type the following line of code in the **ShapeRunner**:

   ```
   ShapeScreen screen = new ShapeScreen(shapes); //pass array as param
   ```

   Observe the Applet window displayed. No shapes are displayed. Why not?

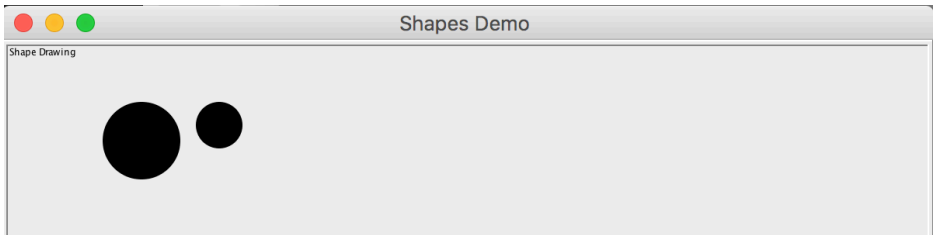
   Answer:

2. Override the **draw( )** method in the **Circle** class so that it returns an **Ellipse2D.Double** object with the appropriate dimensions. The constructor of the **Ellipse2D.Double** class, **Ellipse2D.Double**(double x, double y, double w, double h) constructs and initialises an **Ellipse2D** object from the specified coordinates.

   TIP: Visit the API of any Java class to learn more about a method

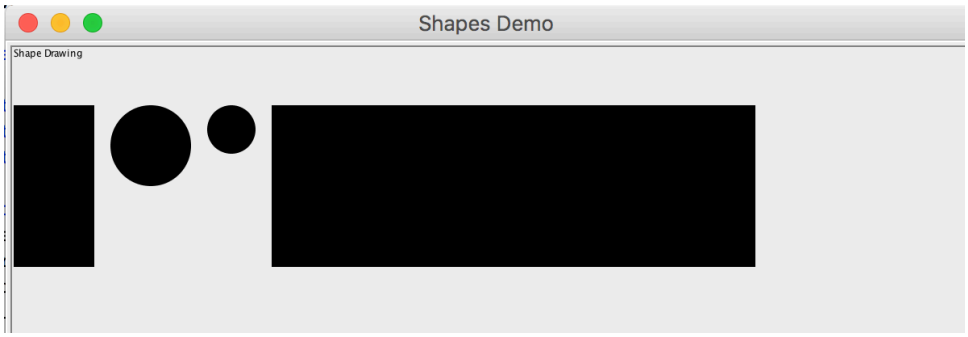
   Google: Java + className

3. Run the **ShapeRunner** class. You should see the following output if your **draw( )** method works properly in the **Circle** class.



4. Override the **draw( )** method in the **Rectangle** class so that it returns an **RoundRectangle2D.Double** object with the appropriate dimensions. The constructor of the **RoundRectangle2D.Double** class, constructor RoundRectangle2D.Double(double x, double y, double w, double h, double arcw, double arch) constructs and initialises a **RoundRectangle2D** object from the specified `double` coordinates. Set the last two parameters, **arcs** and **arch**, to 0.

5. Run the **ShapeRunner** class. You should see the following output if your **draw( )** method works properly in the **Rectangle** class.

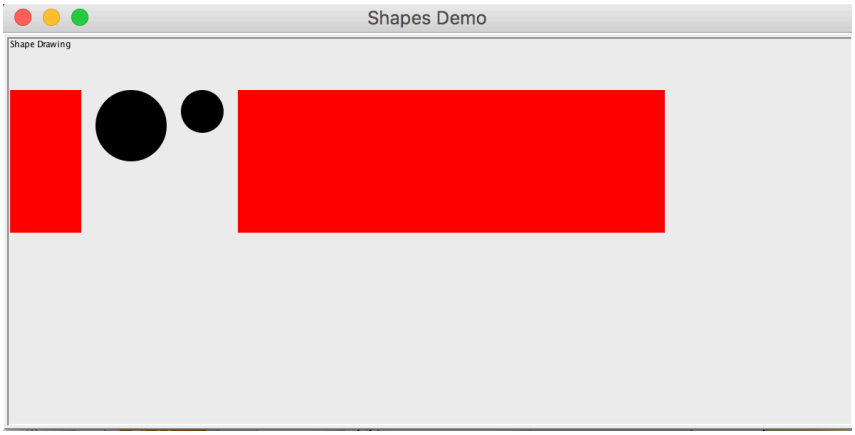


6. Let's try to change the colours of the **SimpleShape** objects. In a `for` loop, change the colour of the **SimpleShape** objects to red. Use the mutator to set the colour using Color.red as the parameter. Try some other colours for fun.

TIP: For more colour codes

Google: Java + Color

7. Try to get your code to generate this colour pattern:



TIP: To find out the dynamic type of an object

if (objectname instanceof className)

Here, all of the **Rectangle** objects are red and all **Circle** objects are black. How can this be done? Why can't we just cast the objects?

Answer:

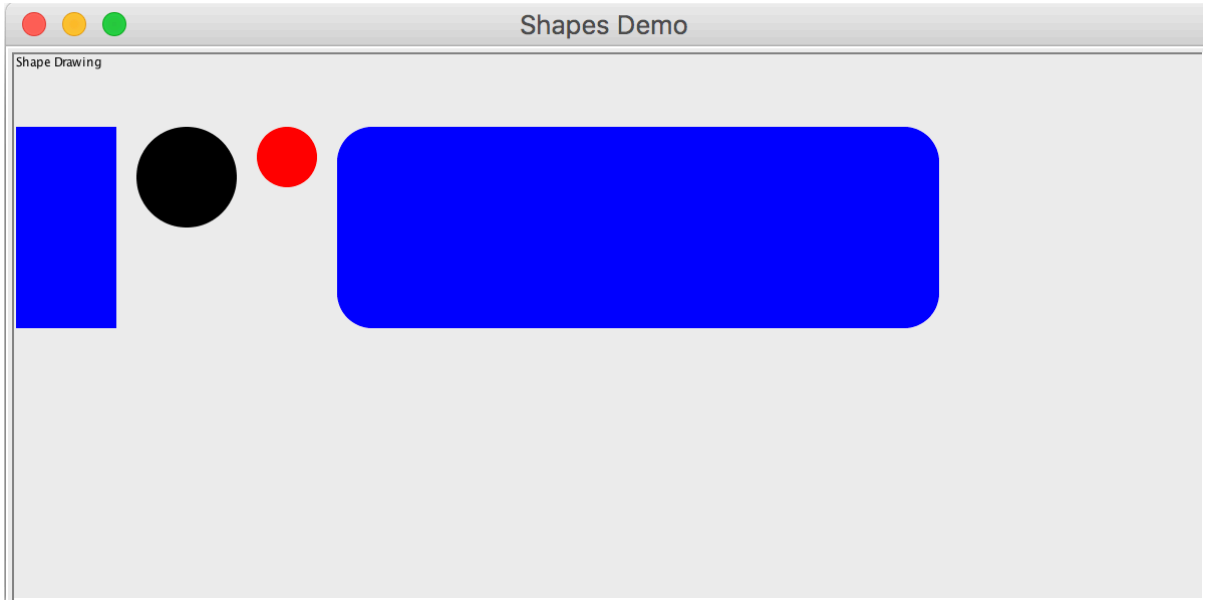
8. Let's enrich the **Rectangle** class just a bit more. Modify the draw( ) method in the Rectangle class such that the last two parameters of the constructor of the **RoundRectangle2D** are the **edgeRoundness** variable. This means that all **Rectangles** will have edges set to the value of **edgeRoundness.** In the **Rectangle** class constructor, the default is 0 which means straight edges.
9. Write a method in the **Rectangle** class called **roundEdge(int curve)** that sets the **edgeRoundness** variable to the incoming value. This would allow us to be change a Rectangle object's edges to rounded.
10. Test your **roundEdge( )** method by invoking it on the instances **s2** and **s5** in the **ShapeRunner** class with a curve of **35**. Did it work for both objects? Explain what is happening.

Answer:

11. How can you get your **roundEdge( )** method to work on the **Rectangle** objects in the **shapes** array using a loop? Why do you need to cast here?

Answer:

12. Try to get your code to generate this colour pattern in the **for** loop for the various shapes, and **Rectangle** roundness (curve of 35) :



TIP:

Use getArea( ) instanceOf