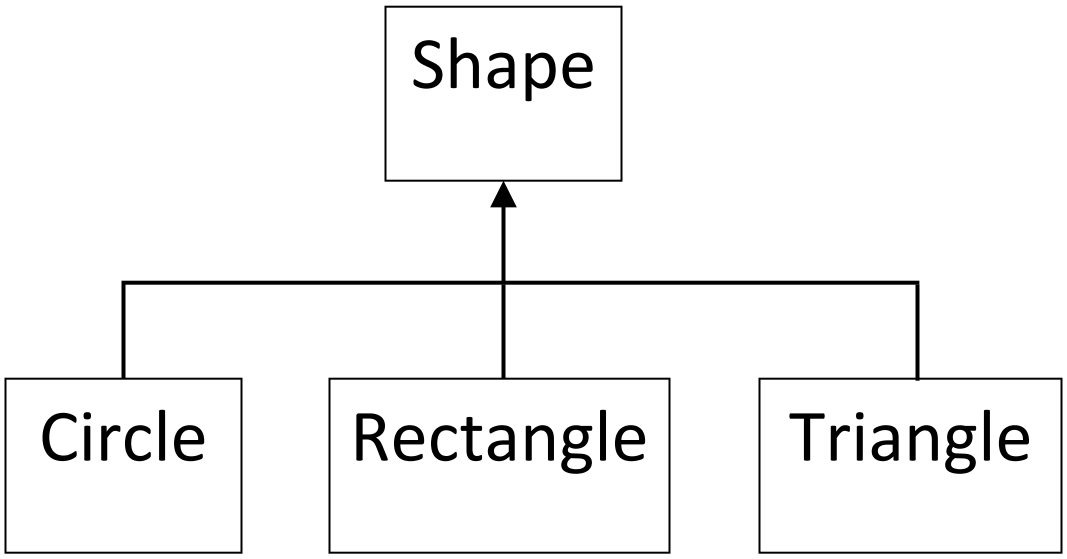
CMPSCI 111L Intro to Algorithms & Programming Lab

Programming Project #6 – Inheritance and Polymorphism

Due 12/6/18

Your mission, if you choose to accept it, will be to implement a classic Java inheritance relationship, will require you to use polymorphism, and will teach you to draw two dimensional shapes on a Java FX Canvas.

Java inheritance is a mechanism which allows a Java programmer to reuse the code written in an existing class when deriving new classes. In a Java inheritance relationship, the derived class is referred to as a **subclass** and the class from which the subclass is derived is referred to as the **superclass**. A subclass is said to **extend** its superclass, meaning the subclass inherits everything from its superclass including all variables and methods (but not the superclass constructors). Additionally, the subclass can define new variables and new methods to what was inherited from its superclass! The subclass can also redefine methods which were originally defined in its superclass (this is called a method override). Because the subclass can add more variables and more methods, it is often called a specialization of the superclass. In Phase I, you will implement this inheritance relationship:



Where Circle, Rectangle and Triangle classes all extend the Shape superclass.

Phase 1

Adding Shape and its derived classes

Extract the Project6FX NetBeans project from the attached ZIP file and open the project in NetBeans. In the NetBeans **Projects** window, right-click the **project6fx** package under **Source Packages** and add three new Java classes named **Shape**, **Circle**, **Rectangle** and **Triangle** to the package.

Change the class header, declare the variables, and write the methods for each class as described below:

Shape

- Change the class header to: `public abstract class Shape`
- Declare a private int-type *global variable* named **x**.
- Declare a private int-type *global variable* named **y**.
- Declare a private int-type *global constant* named **X_MAX_SIZE.** Initialize this constant to 800.
- Declare a private int-type *global constant* named **Y_MAX_SIZE**. Initialize this constant to 600.
- Write a no-argument constructor which sets **x** and **y** variables to 0.
- Write a constructor with **newX** and **newY** parameters which sets **x** and **y** variables to the value of the **newX** and **newY** parameters (you may want to invoke your **setX()** and **setY()** methods to do this).
- Write a **getX()** method which returns the value of the global **x** variable.
- Write a **getY()** method which returns the value of the global **y** variable.
- Write a **setX()** method which takes a parameter named **newX** and sets the value of the **x** variable to the value of **newX**, but only if **newX** is less than or equal to **X_MAX_SIZE**. This **setX()** method should return nothing.
- Write a **setY()** method which returns nothing, takes a parameter named **newY** and sets the value of the **y** variable to the to the value of **newY**, but only if **newY** is less than or equal to **Y_MAX_SIZE**. This **setY()** method should return nothing.

Copy these abstract method headers into your Shape class:

```
public abstract void display();
public abstract double getArea();
```

Circle

- Change the class header to: `public class Circle extends Shape`
- Declare a private int-type *global variable* named **radius**.
- Write a no-argument constructor which invokes the **super** class no-argument constructor, then sets the **radius** variable to 1.
- Write a constructor with **newX**, **newY**, and **newRadius** parameters, in that order, which invokes the **super** class constructor and passes **newX** and **newY** as arguments, then sets the radius variable to the value of **newRadius** (you may want to invoke your **setRadius()** method to do this).
- Write a **getRadius()** method which returns the value of the global **radius** variable.
- Write a **setRadius()** method which takes a parameter named **newRadius** and sets the value of the **radius** variable to the value of **newRadius**, but only if **newRadius** is greater than 0. This **setRadius()** method should return nothing.
- Write a **getArea()** method which computes and returns a double-type value representing the area of this **Circle** class object.
- Write a **display()** method which uses a **System.out.println()** statement to output a String containing all of the data in this **Circle** class object. The String should contain the word “Circle”, the **x** and **y** coordinates of the object, as well as the **radius** and **area** of the object. This **display()** method should return nothing.

Rectangle

- Change the class header to: `public class Rectangle extends Shape`
- Declare a private int-type *global variable* named **width**.
- Declare a private int-type *global variable* named **height**.
- Write a no-argument constructor which invokes the **super** class no-argument constructor, then sets **width** and **height** variables to 1.
- Write a constructor with **newX**, **newY**, **newWidth** and **newHeight** parameters, in that order, which invokes the **super** class constructor and passes **newX** and **newY** as arguments, then sets the **width** and **height** variables to the values of **newWidth** and **newHeight** (you may want to invoke your **setWidth()** and **setHeight()** methods to do this).
- Write a **getWidth()** method which returns the value of the global **width** variable.
- Write a **getHeight()** method which returns the value of the global **height** variable.
- Write a **setWidth()** method which takes a parameter named **newWidth** and sets the value of the **width** variable to the value of **newWidth**, but only if **newWidth** is greater than 0. This **setWidth()** method should return nothing.
- Write a **setHeight()** method which takes a parameter named **newHeight** and sets the **height** variable to the value of **newHeight**, but only if **newHeight** is greater than 0. This **setHeight()** method should return nothing.
- Write a **getArea()** method which computes and returns a double-type value representing the area of this **Rectangle** class object.
- Write a **display()** method which uses a **System.out.println()** statement to output a String containing all of the data in this **Rectangle** class object. The String should contain the word “Rectangle”, the **x** and **y** coordinates of the object, as well as the **width, height,** and **area** of the object. This **display()** method should return nothing.

Triangle

- Change the class header to: `public class Triangle extends Shape`
- Declare a private int-type *global variable* named **base**.
- Declare a private int-type *global variable* named **height**.
- Write a no-argument constructor which invokes the **super** class no-argument constructor, then sets **base** and **height** variables to 1.
- Write a constructor with **newX**, **newY**, **newHeight**, and **newBase** parameters, in that order, which invokes the **super** class constructor and passes **newX** and **newY** as arguments, then sets the **height** and **base** variables to the values of **newHeight** and **newBase** (you may want to invoke your **setHeight()** and **setBase()** methods to do this).
- Write a **getHeight()** method which returns the value of the global **height** variable.
- Write a **getBase()** method which returns the value of the global **base** variable.
- Write a **setHeight()** method which takes a parameter named **newHeight** and sets the **height** variable to the value of **newHeight**, but only if **newHeight** is greater than 0. This **setHeight()** method should return nothing.
- Write a **setBase()** method which takes a parameter named **newBase** and sets the **base** variable to the value of **newBase**, but only if **newBase** is greater than 0. This **setBase()** method should return nothing.
- Write a **getArea()** method which computes and returns a double-type value representing the area of this **Triangle** class object.

- Write a **display()** method which uses a **System.out.println()** statement to output a String containing all of the data in this **Triangle** class object. The String should contain the word "Triangle", the **x** and **y** coordinates of the object, as well as the **height, base,** and **area** of the object. This **display()** method should return nothing.

After you have finished writing the Shape, Circle, Rectangle and Triangle classes, you should be able to run the Project. **Please note, that since this is a JavaFX application, you will not be able to compile and run Project6.java, because it contains no main method**. You should be able to click the "green play button" in the NetBeans button bar, or choose **Run => Run Project (Project6FX)** or press **F6** on your Windows keyboard to run the project. If you've written the classes correctly, you should see the following output in the NetBeans output window:

```
Circle: (250, 120), radius: 150
Triangle: (350, 30), base: 100 height: 100
Rectangle: (550, 250), width: 40 height: 40
Rectangle: (210, 250), width: 40 height: 40
Circle: (300, 200), radius: 30
Circle: (440, 200), radius: 30
Circle: (320, 210), radius: 5
Circle: (470, 240), radius: 5
Rectangle: (330, 315), width: 50 height: 150
Rectangle: (330, 315), width: 25 height: 25
Rectangle: (355, 315), width: 25 height: 25
Rectangle: (380, 315), width: 25 height: 25
Rectangle: (405, 315), width: 25 height: 25
Rectangle: (430, 315), width: 25 height: 25
Rectangle: (455, 315), width: 25 height: 25
Rectangle: (330, 340), width: 25 height: 25
Rectangle: (355, 340), width: 25 height: 25
Rectangle: (380, 340), width: 25 height: 25
Rectangle: (405, 340), width: 25 height: 25
Rectangle: (430, 340), width: 25 height: 25
Rectangle: (455, 340), width: 25 height: 25
Triangle: (380, 220), base: 50 height: 40
22 shapes with total area  100697.78
```

Additionally, an empty Java FX frame (window) should open up. Close this Java FX frame to stop the program. If you see any red error messages in the NetBeans output window, carefully analyze those messages to determine where they occurred and see if you can fix them. However, if you saw no errors then proceed to Phase 2.

Phase 2

Adding the Square class

Again, in the NetBeans **Projects** window, right-click the **project6fx** package and add a new Java Class named **Square**.

Change the Square class header and write the methods described below:

<u>Square</u>

- Change the class header to: `public class Square extends Rectangle`
- Write a no-argument constructor which invokes the **super** class no-argument constructor.
- Write a constructor with **newX**, **newY**, **newHeight**, and **newWidth** parameters, in that order, then invokes the super class constructor and passes **newX**, **newY**, **newHeight**, and **newHeight**. Note that **newHeight** is <u>intentionally</u> substituted for **width** here because the **height** and **width** of a Square are always the same.
- Write a method called **setHeight()** with a parameter named **newHeight** which invokes the **setHeight()** and **setWidth()** method in the **super** class, passing **newHeight** to both methods. This **setHeight()** method should return nothing.
- Write a method named **setWidth()** with a parameter named **newWidth** which invokes the **setHeight()** and **setWidth()** method in the **super** class, passing **newWidth** to both methods. This **setWidth()** method should return nothing.
- Write a **display()** method override which uses a **System.out.println()** statement to output a String containing <u>all</u> of the data in this **Square** class object. The String should contain the word "Square", the **x** and **y** coordinates of the object, as well as the **height, width,** and **area** of the object. This **display()** method should return nothing.

Once you have finished writing the Square class, open the **Project6.java** source code file and, on lines 33-44, change the name "Rectangle" to "Square". Run the Project again and make sure no red error messages are shown in the NetBeans output window. If there are no red error messages in the NetBeans output window, proceed to Phase 3.

Phase 3

Let's get GUI with it!

Add the following abstract method header to your **Shape** class:

```
public abstract void display(GraphicsContext g);
```

It will be necessary to **import** the **GraphicsContext** class from the **javafx.scene.canvas** package.

Next, write a **display(GraphicsContext g)** method overload in each one of the derived Shape classes; Circle, Rectangle, Triangle and Square. This display method overload should invoke a method (or three) on the **GraphicsContext** object, **g**, to draw the respective shape on the **GraphicsContext** object. For example, the **display(GraphicsContext g)** method in the **Circle** class should draw (stroke) an oval on the **GraphicsContext** object, **g**. The following Oracle web page should provide more detailed information on drawing in Java FX:

https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm

Once you have added the **display(GraphicsContext g)** method overloads to your Shape classes, open the **Project6.java** source code file again and uncomment the statements on lines 52-54, which set the fill and stroke colors for the graphics context, and then invokes the **display(GraphicsContext g)** methods you just wrote in your Shape classes. After uncommenting the statements, run the Project again. If all of your display methods are written correctly, the program should draw a funny face in the Java FX frame that opens up on screen. Email me the **Project6FX.jar** (executable JAR) file which you will find in the **Project6FX/dist** folder to receive full credit for this project.