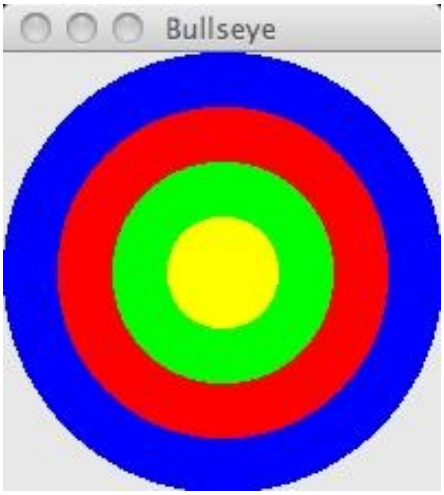
Lab #5 – Drawable/Scaleable Shapes using interfaces

Total Points: 50

Introduction

In this lab you will be working with a project that makes use of *Interfaces* and *polymorphism* to allow the creation of shapes that can be drawn on the screen. The final product of the lab will be an application that draws a picture or performs an animation on the screen. As a really simple example of what is possible, the image below is produced by the `Bullseye` program included with the lab.



Before you can get to the point of drawing pictures or creating animations though, you are going to need to implement and test several classes that are needed by the application.

Getting Started

1. Create your GitHub repository and import the code into Eclipse as described on the "How to…" webpage, available from the course homepage.
2. As in some previous labs, remove any compilation errors in the starter code by following the instructions for "Adding the JUnit4 library to a project in Eclipse" on the Moodle "How to…" webpage.

Design

At the heart of the design of the application created in this lab are two interfaces, `Drawable` and `Scaleable`. The definition of these interfaces is provided with the lab.

- The `Drawable` Interface:

This interface declares methods for drawing the object, getting/setting its color and getting/setting its visibility. Thus, we will know that every object that implements the `Drawable` interface will provide these methods. In this way it will be possible for the application to handle all `Drawable` objects in the same way. It will not have to worry about whether the object is a circle, a square, a line or a triangle. Further, any object will be able to declare to the application that it can be drawn on the screen by implementing the `Drawable` interface. The lab comes with one class `Circle` that implements the `Drawable` interface. You will be creating several additional classes that implement `Drawable`.

- The `Scaleable` Interface:

This interface declares one method for scaling an object. When an object is scaled it becomes larger or smaller by some factor. For example, if an object is scaled by 2.0 it will become twice as large. If it is scaled by 0.5, it will become half as large. Some objects will be able to be scaled and others will not. Those that can be scaled declare to the application that they can be scaled by implementing the `Scaleable` interface. The application is then able to handle all `Scaleable` objects in the same way, again without worrying about whether the object is a circle, a rectangle or a triangle. The `Circle` class provided with the lab also implements the `Scaleable` interface. Several of the classes that you create will also implement `Scaleable`.

These two interfaces are used by the `DrawableObjectList` class, which is a collection of objects that implement the `Drawable` interface. This class contains typical collection methods for adding/removing `Drawable` objects and finding out how many objects are in the collection. It also has methods for drawing and scaling all of the objects in the collection. These methods must rely on interfaces and polymorphism in order to accomplish their tasks. The `DrawableObjectList` will not know if the objects added are circles, rectangles, lines or triangles. However, it will know that they are all `Drawable` and thus may invoke any of the methods defined in the `Drawable` interface on any object in the collection. This includes the method necessary to draw each object. Similarly, any object that can be scaled will implement the `Scaleable` interface and thus the `DrawableObjectList` can scale any `Scaleable` object without knowing its true identity as a circle, square or triangle.

The Assignment

Your assignment for this lab can be divided into three tasks. First you will complete and test the implementation of the `DrawableObjectList` class. Second you will create and test several classes that implement `Drawable` and possibly the `Scaleable` interfaces. Finally, you will use the classes that you have implemented to create a picture or animation.

The `DrawableObjectList` Class

You need to implement and test the `DrawableObjectList` class. The steps below outline an incremental approach to implementing and testing this class.

1. Define the field(s) that will be needed by the `DrawableObjectList`.
2. Implement the constructor and the `getSize()` method.
3. Create a JUnit Test Case for the `DrawableObjectList` class and test the constructor and the `getSize()` method.
4. Implement and test the `addDrawable()` method.
5. Implement and test the `removeDrawable()` method.
6. Implement the `drawAll()` method. **You are not required to create a JUnit test method for the `drawAll` method**.
7. Implement and test the `scaleAll()` method. Hint: You can create and use `Circle` objects to test `scaleAll()`.

Once you have correctly implemented and tested the `DrawableObjectList` class you can run the sample programs provided with the lab. The sample programs are `Bullseye`, which draws the figure shown earlier in the lab, and `Annihilation`, which performs a simple animation.

Creating `Drawable` and `Scaleable` Objects

Create a **minimum of two classes** that implement the `Drawable` interface. **At least one of these classes** must also implement the `Scaleable` interface. You may wish to study the `Circle` class before attempting your own classes. You may also wish to look at the methods available in the `java.awt.Graphics` class to see what it is capable of drawing. You can find the `java.awt.Graphics` class by following the link to the full Java API Documentation on the course home page or via a web search engine. Finally, you may want to confer with other students so that you each produce different classes. You can later share these classes to help create more interesting pictures and animations.

For each of the classes you choose to create:

8. Decide on a suitable class name and create a new class. Be sure to write a Javadoc comment above the class describing it and listing yourself as the author of the class. You must also include Javadoc comments for all of the methods defined in your class.
9. Define the fields that objects will need.

10. Implement at least one constructor for your class.
11. Add accessors for the fields of your class.
12. Create a JUnit Test Case for your class and test the constructor(s).
13. Add and test any other methods that make sense for your class (e.g. `move()`, `translate()` etc...).
14. If your class implements `Scaleable`, add that to its declaration then implement and test the `scale()` method.
15. Declare that your class implements `Drawable` and add stubs for the required methods.
16. Implement and test the `get/setColor()` and `get/setVisibility()` methods required by the `Drawable` interface.
17. Implement the `draw` method. **You are not required to create a JUnit test for your `draw` method.**

Creating a Picture or Animation

Create a picture or animation using the `Drawable` and `Scaleable` classes that you have created. You should also feel free to share/borrow classes from others in our COMP132 class. If you do, please be sure that the `author` information at the top of each file accurately reflects who produced each source file.

The basic approach to creating a picture or animation is to create `Drawable` objects, then add those to a `DrawableObjectList` that is part of a `DrawingTablet`. The `DrawingTablet` takes care of displaying the window and invoking the `drawAll()` method to display the objects. Below is the `RedLight` example included with the lab. It draws a green circle at the center of the window and after 3 seconds turns the circle yellow and then after another second turns the circle red.

```
public static void main(String[] args) {

/** Create a DrawableObject list and use it to create
  * a DrawingTablet
  */

DrawableObjectList objList = new DrawableObjectList();

DrawingTablet tablet = new DrawingTablet("Red Light", 200, 200, objList);

// Create a green circle at the center of the screen.

Circle light = new Circle(100, 100, 75, Color.green);

objList.addDrawable(light);
/*
 * Each time a change is made to the DrawableObjectList or to one of
 * the Drawable objects that it contains you need to invoke repaint()
 * on the DrawingTablet. This causes the Drawing tablet to repaint
 * the screen reflecting the changes.
*/

tablet.repaint();
```

```
/* sleep is a static method in the AnimationTimer class that causes
 * the program to pause for a specified number of milliseconds when
 * it is invoked.  This line sleeps for 3 seconds.
 */
AnimationTimer.sleep(3000);

// Change the color and repaint!
light.setColor(Color.yellow);
tablet.repaint();

// Sleep for 1 second.
AnimationTimer.sleep(1000);

// Change the color and repaint.
light.setColor(Color.red);
tablet.repaint();

}
```

You may also want to study the `Bullseye` and `Annihilation` programs for other examples of how to create and draw objects and perform animations. You may also want to look at the `java.awt.Color` class to see what colors are defined and also how you can create your own custom colors. You can find the `java.awt.Color` class by following the link to the full Java API Documentation on the course home page or via a web search engine.

Submitting your solution

As usual, push your code to GitHub regularly for backup purposes and push your final version to submit the assignment.