CS 201
Advanced Object-Oriented Programming
Lab 2 - Polymorphism
Due: February 10/11, 11:30 PM

Objectives

The objectives of this assignment are:

- to gain experience with interfaces and polymorphism
- to gain experience with Swing

Introduction to the Assignment

In this lab, you will create a tool that can display several types of shapes. By clicking on a shape, you can change its color.

Here are some snapshots from this program:

<table>
<thead>
<tr>
<th>Image</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Original Display" /></td>
<td>This is the original display. Some shapes are drawn in a window. There are two buttons that will be used to change the color of the shapes.</td>
</tr>
<tr>
<td><img src="image2" alt="Selected Shape" /></td>
<td>When the user clicks on a shape, a black border is drawn around the shape to indicate that it is selected.</td>
</tr>
<tr>
<td><img src="image3" alt="Color Change" /></td>
<td>When the user clicks the Darken button, the color used by the selected shape is made darker.</td>
</tr>
</tbody>
</table>
When the user clicks the Brighten button, the color of the selected shape gets brighter.

The Program Design

This project will have 4 classes and 1 interface:

- **ShapeExplorer** is the main class. It creates the JFrame and adds the buttons and DrawingPanel to the window.
- **Drawing Panel** is the JPanel that the shapes are drawn on. It remembers the shapes, is responsible for determining which shape is selected, and redraws the shapes when the selection or color changes.
- **Shape** is the interface that defines the contract that all implementations of Shape must satisfy.
- **Rectangle** is an implementation of Shape.
- **Circle** is a second implementation of Shape.

Writing the Program

Create a new Eclipse project called ShapeExplorer.

**Step 1: Create the window and buttons**

For the first step, create a class called ShapeExplorer. In the main method, create the JFrame as you did in last week’s lab. Add a method createButtonPanel that creates the two buttons at the top of the window, placing them in a panel. Add that panel to the NORTH of the content pane. Refer to the PizzaOrder example from class to see how to do this. Run your program. You should see the buttons, but nothing should happen when you click on them.

Create a second class called DrawingPanel. Declare that class to “extend JPanel”. Define a constructor in DrawingPanel that contains one line:

```java
setBackground(Color.WHITE);
```

Add drawing panel to the CENTER of the content pane.

Run your program. At this point, your window should look like the picture at the right. Nothing should happen yet when the user clicks the buttons.

**Step 2: Add a shape to the window**

Next, you should define a Shape interface and the Rectangle class. The Shape interface should declare the following method in its contract:
public void draw(Graphics g);

When you define the Rectangle class, you should declare that it “implements Shape”. You will need to define the draw method in Rectangle to satisfy the Shape contract. You will also need to define a constructor in Rectangle that takes parameters for the location, size and color of the rectangle being constructed. Your constructor will need to save these parameter values in instance variables so that the draw method can refer to them.

Then, add a method to your DrawingPanel class called addShapes. This method should create one rectangle and save it in an instance variable whose type should be Shape. You should call this method in your DrawingPanel constructor.

You will also need to define a paintComponent method in DrawingPanel, similar to what you did in last week’s lab, calling the shape’s draw method to draw your shape on the screen.

Now when you run your program, you should see something like the picture at the right. (The exact size, location and color of the rectangle is not important.)

**Step 3: Implement the button behavior**

Next we will add the action listeners for the Brighten and Darken buttons. As the first step, add a method to DrawingPanel called getSelectedShape. For now, it should return the one shape that has been added to the panel. (Determining which shape is selected will get more interesting in Step 4.)

Now, add the action listeners to the Brighten and Darken buttons. They should each get the selected shape from the drawing panel. Then, they should call brighten (or darken) on that shape. This will require you to add these two methods to your Shape contract and your Rectangle implementation. The implementation of the brighten or darken method itself is quite simple. Java’s Color class defines the following methods, which you can just call from your new methods to change the value of your color instance variable.

```java
public Color brighter()
Returns:
    a new Color object that is a brighter version of this Color.

public Color darker()
Returns:
    a new Color object that is a darker version of this Color.
```

After changing the color of the shape, call the repaint method on your drawing panel. This will result in the paintComponent method that you defined in Step 2 to be called automatically, displaying your rectangle with its new color. For example, if your drawing panel is in a variable named drawingPanel, you should call:

```java
drawingPanel.repaint();
```

**Step 4: Select a shape when it is clicked on**

Next, you should add code to allow the user to select a shape by clicking on it. The color of the shape should only change if it is selected. To get this to work, you will need to know when the user clicks the mouse in the drawing panel and use the information about the mouse location and the shape location to determine if the user clicked on the shape.
To be notified of mouse clicks in the drawing panel, you need to add a mouse listener to the
drawing panel. To do that you need to call the method:

    public void addMouseListener(MouseListener l)

Note that this method requires a parameter of type MouseListener. MouseListener is an in-
terface that defines the following contract:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mouseClicked</td>
<td>Invoked when the mouse button has been clicked (pressed and released) on a</td>
</tr>
<tr>
<td></td>
<td>component.</td>
</tr>
<tr>
<td>mouseEntered</td>
<td>Invoked when the mouse enters a component.</td>
</tr>
<tr>
<td>mouseExited</td>
<td>Invoked when the mouse exits a component.</td>
</tr>
<tr>
<td>mousePressed</td>
<td>Invoked when a mouse button has been pressed on a component.</td>
</tr>
<tr>
<td>mouseReleased</td>
<td>Invoked when a mouse button has been released on a component.</td>
</tr>
</tbody>
</table>

For your class to be notified of mouse events, your DrawingPanel class needs to “implement
MouseListener” and define the five methods listed above. Most of these methods can have
an empty body, like this:

    public void mouseEntered (MouseEvent e) {
    }

The one method you need to provide a body for is the mouseClicked method.

To help determine if the user clicked on a shape, add a contains method to your Shape con-
tact and Rectangle class with this signature:

    public boolean contains(int x, int y)

It should return true if the (x, y) coordinate is within the boundary of the rectangle. You
will need to call this in your mouseClicked method, passing in the parameters of where the
mouse is. You will get this location information from the mouseEvent that is passed to
mouseClicked. MouseEvent has methods:

    public int getX()
    public int getY()

These methods return the location of the mouse when it is clicked.

You will need a new instance variable in DrawingPanel called selectedShape. You should set
this to the shape if the user clicked on it. If the user clicks in the background where there is
no shape, set this variable to the value “null”.

Now that you can determine if the user clicked on the shape, modify getSelectedShape to
return the selected shape. When you use the Brighten and Darken buttons, only the color
of the selected shape should change. You will also need to change Brighten and Darken so
that if there is no selected shape, the methods do nothing. To determine if there is a se-
lected shape, make sure that the value returned by getSelectedShape is not null, like this:

    if (drawingPanel.getSelectedShape() != null)
Finally, it is helpful to the user if the selected shape is highlighted somehow. The easiest way to do that is to draw a black rectangular frame on top of the selected rectangle. You can do this by calling drawRect. This works the same as fillRect, except that only the border is drawn. To accomplish this, add a drawHighlight method to your Shape contract and Rectangle class. In your paintComponent method, add a call to draw the highlight for the selected shape.

**Step 5: Create multiple shapes**

Having just one shape is boring! In your DrawingPanel class, change the variable that holds a Shape to instead be an array of Shapes. Reminder: You declare an array of Shapes as Shape[]. Also, remember to construct the array with code similar to this:

```java
private Shape[] shapes = new Shape[10];
```

You will now need to change your method that adds a shape to add enough shapes to fill your array. You will need to change your paintComponent method to walk the array, painting each shape that is in the array. You will also need to change your mouseClicked method to walk through the array of shapes to determine which the user clicked on. Think about what the correct behavior should be if the user clicks in a location where multiple shapes overlap. Which shape would the user expect to be selected? How do you write your code to return the correct shape?

**Step 6: Add a Circle class**

Having only rectangle is also boring! Define a Circle class. It should implement Shape. It is somewhat trickier to determine if a point is within the boundary of a Circle than a Rectangle. To determine if the user clicked on the circle, calculate the distance from the point to the center of the circle. (Remember the Pythagorean theorem?) If it is less than or equal to the circle’s radius, then the point is inside the circle.

Change the addShapes method so that some of the shapes are circles. Notice that you can assign both Rectangles and Circles to elements of your Shapes array. When you draw the shapes in paintComponent, or determine which shape the user clicked on, it is not necessary for you to figure out which shapes are rectangles and which are circles. Java automatically calls the right methods. That is the power of polymorphism!

**Extra credit ideas**

Possible ideas for extra credit:

- Add buttons named “Bring Forward” and “Send Backward” that change the order of the shapes in your array so that the layering looks differently, similar to what you might see in a drawing program.

- Change the way that your addShapes method works so that it uses randomization when creating the shapes. Randomly choose between Rectangle and Circle, give them random locations, sizes and colors. Look at the Java API for the Random class at [http://docs.oracle.com/javase/6/docs/api/](http://docs.oracle.com/javase/6/docs/api/) to learn how to work with random numbers in Java.
Grading

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Step 1: Laying out the display</td>
</tr>
<tr>
<td>10</td>
<td>Step 2: Adding one shape</td>
</tr>
<tr>
<td>10</td>
<td>Step 3: Brighten and darken</td>
</tr>
<tr>
<td>10</td>
<td>Step 4: Select a shape</td>
</tr>
<tr>
<td>10</td>
<td>Step 5: Multiple shapes</td>
</tr>
<tr>
<td>10</td>
<td>Step 6: Circles</td>
</tr>
<tr>
<td>20</td>
<td>Comments (including javadoc comments)</td>
</tr>
<tr>
<td>20</td>
<td>Style</td>
</tr>
<tr>
<td>max 10</td>
<td>Extra credit</td>
</tr>
</tbody>
</table>

**Turning in Your Work**

To turn in your work, create a jar file that contains your source code. From Eclipse, you can create a jar file as follows. Select your project. Open the File menu and select Export. Select Jar file and click Next. **Check the box that says export Java source files.** Select a name and destination for the jar file. Click Next twice. Enter ShapeExplorer as the Main Class. Then click Finish. You should be able to run your Java program by double-clicking on the jar file you just created. Submit your jar file on Ella.