Programming Assignment (PA4) - ReflectingPond

Due Date: Wednesday, October 24 @ 11:59 pm

<table>
<thead>
<tr>
<th>Assignment Overview</th>
<th>Grading</th>
<th>Gathering Starter Files</th>
<th>ReflectingPond</th>
</tr>
</thead>
<tbody>
<tr>
<td>README File</td>
<td>Extra Credit: Funky Pond</td>
<td>Turnin Summary</td>
<td></td>
</tr>
</tbody>
</table>

Assignment Overview

In this assignment you will continue exploring the graphical user interface (GUI), object oriented programming, and animation. You will create a reflecting pond on the canvas with four different quadrants divided up by a horizontal line and a vertical line. Each pond quadrant has its own color. When the user drags the lines, quadrants will be resized accordingly. When the user resizes the window, the lines will change position in order to preserve the proportion of each quadrant. When the user clicks on the pond, animated ripples will be created in the water. The ripples will have a different color depending on which quadrant they are in.

Grading

- **README: 10 points** - See README Requirements [here](http://cseweb.ucsd.edu/~ricko/CSE11READMEGuidelines.pdf) and questions below
  - [here](http://cseweb.ucsd.edu/~ricko/CSE11READMEGuidelines.pdf)
- **Style: 20 points** - See Style Requirements [here](http://cseweb.ucsd.edu/~ricko/CSE11StyleGuidelines.pdf)
  - [here](http://cseweb.ucsd.edu/~ricko/CSE11StyleGuidelines.pdf)
- **Correctness: 70 points**
- **Extra Credit: 5 points** - View Extra Credit section for more information.

NOTE: If what you turn in does not compile, you will receive 0 points for this assignment.

Gathering Starter Files

You will need to create a new directory named pa4 and go into that directory. The $ represents your command prompt. What you type in is in **bold**.

```
$ mkdir ~/pa4
$ cd ~/pa4
```

Copy the starter files from the public directory:

```
$ cp ~/../public/objectdraw.jar .
$ cp ~/../public/Acme.jar .
```
Starter files provided:

objectdraw.jar  Acme.jar

ReflectingPond

You will be creating three files: ReflectingPond.java (the main GUI controller), PondQuadrants.java and Ripples.java. As in previous assignments, the GUI controller class must extend WindowController since it will be handling all the mouse movements/events and user interaction. However, unlike previous assignments, this class does not directly manipulate any objectdraw GUI components. We recommend implementing the program in stages, in the order we have listed below.

Stage 1: Creating the Lines

Begin with the GUI controller class (ReflectingPond). Set up a 600 x 600 pixel canvas. Then create a new class (PondQuadrants) which is responsible for all of the creation and manipulation of the pond (and its 4 quadrants). In PondQuadrants, create two Line objects to divide the canvas into four quadrants of equal size (a horizontal line and a vertical line). The end points of the lines should be based on the current size of the canvas (width and height). Note that you will need to keep track of the proportions that the axis lines divide the canvas into. For example, at this point the width and height proportions will both be 0.5. Have ReflectingPond create a new PondQuadrants object in its begin() method. You should now be able to see basic axes on your canvas. Note that your PondQuadrants class needs a way to reference the GUI controller’s canvas object and your GUI controller needs a way to reference the PondQuadrants object.

Stage 2: Creating the Colored Quadrants

Modify PondQuadrants to color the background of each quadrant. To do so we need to create a FilledRect for each quadrant, and then set the color for it. We also need to change the color of the axis lines to white so we can still see them. We recommend creating a helper method to set the size/positions of the Lines and 4 FilledRect quadrants based on the current width and height proportions and current canvas size. This will help make the next stage easier. Your program should now look like the screenshot to the right.

```java
private static final Color UPPER_LEFT QUAD_COLOR = new Color(7,156,222);
private static final Color UPPER_RIGHT QUAD_COLOR = new Color(0,210,234);
private static final Color LOWER_LEFT QUAD_COLOR = new Color(27,67,93);
private static final Color LOWER_RIGHT QUAD_COLOR = new Color(9,0,137);
private static final Color LINE_COLOR = Color.WHITE;
```
Stage 3a: Manipulating the Quadrants -- Dragging the Axis Lines

The first manipulation you should implement is dragging the axis Lines. In ReflectingPond’s onMousePress() method, we need to call a method in PondQuadrants that takes in the position of the mouse and sets flags in PondQuadrants indicating whether or not each of the two axis lines were grabbed. Note that if you grab the intersection of the two lines, you should be able to drag both lines simultaneously.

You will need to update the position of the Lines as they are dragged. To do this, your ReflectingPond’s onMouseDrag() method should relay information and delegate to a method in PondQuadrants so that the lines (and quadrants) can be updated accordingly. **Do not let either line go beyond 6 pixels from the edge of the canvas in any direction (a 6 pixel margin)**—see the demo video for what this looks like. Note that you will need to update the width and height proportions whenever the lines are moved. This is where the helper function mentioned above (in Stage 2) can be useful for updating the sizes and positions of the lines and quadrants.

Stage 3b: Manipulating the Quadrants -- Resizing the Window

The second manipulation you should implement is keeping the PoolQuadrants proportional when resizing the window. If you change the size of the window, the Lines should move to preserve their current proportions on the screen. To redraw the canvas, override the paint() method in ReflectingPond:

```java
public void paint(java.awt.Graphics g) {}
```

You should **first** make a call to the superclass’s version of the method by adding the line:

```java
super.paint(g);
```

as the **first** line in your paint() method. You can then add code in paint() after the call to super.paint(g); that tells the PondQuadrants to redraw the Lines and the FilledRects based on the proportions of where they were previously (again, that helper method is starting to sound pretty helpful...). **Be sure to avoid any possible NullPointerExceptions in this method as paint() could possibly be called before begin().** Hint: what happens in begin()?
Stage 4: Creating Ripples

Create a new class called **Ripples**, which should be an ActiveObject (extends ActiveObject - see Ch. 9), to create the ripples in the pond. The GUI controller class should not store any references to any of the ripples created - **DO NOT** try to store all the ripples in a data structure. Instead, the GUI controller should simply create a new Ripples object **every time the mouse is clicked** in the canvas.

The constructor should look like this:

```java
public Ripples(Location center, DrawingCanvas canvas,
                      PondQuadrants quadrants)
```

The first parameter is the centerpoint of the ripples (where the mouse was clicked). The last parameter is the PondQuadrants object on the canvas so this new Ripples can ask the PondQuadrants what color it should be (more on this in Stage 6---for now you can just keep the FramedOval the default color of black).

For now, the constructor should just create and display one FramedOval as the first ripple. The FramedOval should be centered at the mouse location and should have a width and height of 0.

**Remember:** Ripples is an ActiveObject, so you need to call the start() method as the last line of code in this constructor. See Ch 9 in the textbook and the class Notes for Ch 9.

Stage 5: Ripple Animation

You should add a run() method with a forever loop to your Ripples class to handle the animation of the ripples. The diameter of each ripple should grow/shrink by 2 pixels each step to make it a smooth transition. You should pause for 75 milliseconds in each iteration so the ripples won’t grow and shrink too quickly. If it looks like your program isn’t doing anything, you probably aren’t pausing correctly.

You will only have four FramedOvals in total (per Ripples object). When the Ripples object is first created, it starts with only one ripple (FramedOval). After every 10 cycles of growing, a new ripple should be created (with a width and height of 0, like the FramedOval that was created in the constructor) and displayed until there are four FramedOvals on the canvas for this Ripples object.

Once a FramedOval ripple grows to a width of 80 pixels, it starts shrinking back down to a width of 2 pixels before growing again (see the demo video). **Note that the center of the Ripples object needs to remain in its original location as it shrinks/grows.**

Because Ripples is an ActiveObject and therefore a Thread, it can run on its own, independent of all other objects. The run() method contains all of the code that will run when the thread is started. You start the thread by calling the start() method at the end of the Ripples constructor. (See Ch. 9 and the lecture notes on Active Objects. We went over all of this in class.)
Stage 6: Different Colors for Ripples in Each Quadrant

You should set the color of each Ripples object based on which quadrant the center of the Ripples object is located in at any particular time. Write a method in PondQuadrants that takes in a Location and returns the ripple color corresponding to the quadrant the Location is in (see constants below). Now you can use the PondQuadrants passed into the Ripples constructor and the Ripples' center Location to determine what color it should be.

```java
private static final Color UPPER_LEFT RIPPLE_COLOR = new Color(0,234,212);
private static final Color UPPER_RIGHT RIPPLE_COLOR = new Color(0,50,250);
private static final Color LOWER_LEFT RIPPLE_COLOR = new Color(255,137,93);
private static final Color LOWER_RIGHT RIPPLE_COLOR = new Color(252,220,116);
```

When the user drags the lines, the quadrant areas are redefined. This means each Ripples object will need to re-ask PondQuadrants what color it should be with each iteration in the run() method since it may be in a new quadrant now. Note that if the canvas size is shrunk so that a Ripples is "off" the canvas, you do not need to do anything, the Ripples should stay in its location and should continue shrinking/growing as if it were still on the canvas. When the canvas is expanded again, the Ripples should still be in the same location.

Recap of What Each Class Should Contain:

**ReflectingPond.java:**
- Extends WindowController, meaning it creates the window, and handles all mouse events by delegating to the other two classes.
- Creates the PondQuadrants object in begin() (and needs to hold on to the reference to it).
- Creates Ripples objects in onMouseClick() (and does not hold on to any references to them!).

**PondQuadrants.java:**
- Needs to hold a reference to the canvas.
- Displays the quadrants on the canvas (Lines and FilledRects).
- Contains all logic for dragging the axis lines, including flags to indicate whether the lines are grabbed.
- Contains the logic for determining which ripple color should be used given the center location of the ripple.
**Ripples.java:**
- Needs to hold a reference to the canvas and to the PondQuadrants object.
- Extends ActiveObject, meaning it has a run() method to handle the ripple animation.
- Displays the FramedOvals on the canvas.

**No data structures! No Arrays, ArrayLists, Trees, Priority Queues, HashMaps, HashBrowns, etc. Only use concepts that have been covered in lecture.**

**Sample Screenshots:** (For this assignment, the demo video will be more helpful than the screenshots since it is difficult to capture the ripple animation with screenshots.)

- **On startup**
- **After first click in lower left quadrant**
- **Dragging horizontal line changes which quadrant the ripples object is in**
- **Dragging vertical line changes which quadrant the ripples object is in again**
- **Adding more ripples. Note that their colors reflect the current quadrant they are in**
- **After dragging the horizontal and vertical lines together. Now all the ripples are in the lower left quadrant and all of their colors have updated**

**No exceptions should be thrown!**
**README File**

Remember to follow all of the guidelines outlined in the README Guidelines. If you did the extra credit, write a program description for it in the README file as well.

**Questions to Answer in your README:**

1. From your current directory how do you copy over a java file named zumba from a folder four directories above? Write the full command required to perform this action. Please provide the exact command as we will not be lenient if any part of the command is incorrect or missing.
2. Say you get a compiler error or runtime exception telling you there is an error on line 42 of PondQuadrants.java. How can you open the file in vim/gvim from the Linux command line so that you are taken directly to the line where the error is?
3. While in command mode in vim, how do you create a new line below and enter insert mode by pressing just a single key on the keyboard?
4. What does the command “man diff” do (with no quotes)?
5. Why is sharing solution code with other students an academic integrity violation?

**Extra Credit: Funky Pond**

- **5 Points:** Create a funky color changing mode for the ripples.

**Getting Started:**

Create copies of the following files in your directory to do the extra credit in.

```bash
$ cd ~/pa4
$ cp ReflectingPond.java EC_ReflectingPond.java
$ cp PondQuadrants.java EC_PondQuadrants.java
$ cp Ripples.java EC_Ripples.java
```

Make sure to change all instances of Ripples, PondQuadrants and ReflectingPond to their corresponding EC names so your code can compile and run properly.

**Important:** Your original files must remain unchanged. You need all six files for turnin (the three original files and the three extra credit files).

You will need a cycling colors toggle/flag/mode that you can switch on and off (note that this should only change the colors of the ripples, and not of the background quadrant colors). Before any Ripples are created, the mode should be off and cannot be toggled on, no matter what other mouse events may happen. After at least one Ripples object is created, the cycling colors mode is switched/toggled on/off every time the mouse (re)enters the canvas. If the cycling colors mode is off, then on mouse enter the cycling colors mode is turned on. If the cycling colors mode is on, then on mouse enter the cycling colors mode is turned off.

You will need **FIVE** ripple colors to cycle through. These can include the original four ripple colors, or you may choose your own colors. The important thing is that you must add a fifth color. With the cycling colors mode on, each FramedOval in the Ripples objects will cycle through these five colors once that FramedOval grows to its maximum size. The same 5 colors should be cycled through in each quadrant (as opposed to each quadrant having its own 5 colors).

After the ripples change color, they stay with that color shrinking and growing like regular ripples. When you create a new Ripples when the cycling colors mode is on, the Ripples will start with color of a regular Ripples in that quadrant, then each ripple changes color individually once it reaches its maximum size.
For example: 5 colors: red, orange, yellow, green, purple
Let's say the cycle color mode is toggled on and a new Ripples object is created. The 4 ovals have been created and are all the same original color (original being that quadrant’s ripple color). Once oval #1 (the first oval that was created) reaches the maximum size, its color will change to red. This oval will remain red until it shrinks to the minimum size and then grows to the maximum size again. Once it reaches the max size again, it will turn orange. This pattern continues as it cycles through the colors. Once it turns purple, it will cycle back around to red. This should happen for each individual oval within the Ripples object. Note that the ovals don’t all change color at the same time. Also note that in this example, oval #1 will first start cycling with red; oval #2 will start cycling with orange; oval #3 will start cycling with yellow; oval #4 should start cycling with green.

When the color cycling mode is on, you can still grab and move the horizontal and vertical Line around to define new quadrants, but the Ripples continue with their color cycling.

When the cycling color mode is on, and the mouse exits and re-enters the canvas, the mode should be turned off, and all Ripples should change color in the next run() animation cycle back to its regular color in each quadrant, defined by the vertical and horizontal Lines in PondQuadrants.

See the demo video for what this looks like.

Turnin Summary

See the turnin instructions here. Your file names must match the below *exactly*.

Due Date: Wednesday night, October 24th @ 11:59 pm

Files Required for Turnin:

- PondQuadrants.java
- ReflectingPond.java
- Ripples.java
- Acme.jar
- objectdraw.jar
- README

Extra Credit Files:

- EC_PondQuadrants.java
- EC_ReflectingPond.java
- EC_Ripples.java

If there is anything in these procedures which needs clarifying, please feel free to ask any tutor, the instructor, or post on the Piazza Discussion Board.

NO EXCUSES!
NO EXTENSIONS!
NO EXCEPTIONS!
NO LATE ASSIGNMENTS ACCEPTED!
DO NOT EMAIL US YOUR ASSIGNMENT!

Start Early, Finish Early, and Have Fun!