Programming Assignment (PA6) - ReflectingPond 2.0

Due Date: Wednesday, November 7 @ 11:59 pm

<table>
<thead>
<tr>
<th>Assignment Overview</th>
<th>Grading</th>
<th>Gathering Starter Files</th>
<th>ReflectingPond 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Screenshots</td>
<td>README File</td>
<td>Extra Credit: Keyboard Controls</td>
<td>Turnin Summary</td>
</tr>
</tbody>
</table>

Assignment Overview

For this assignment, you will extend your PA4 ReflectingPond program to include standard Java GUI components (JButton, JLabel, JSlider, JPanel) and standard Java Event Handling. You will want to reference Chapter 11 and the notes for Chapter 11!

This ReflectingPond will function the same as it did in PA4 with the following additional features: play/pause the ripples animation, adjust the speed of the ripples animation, clear the ripples from the canvas, click and drag the ripples around the canvas, and keep the location of the ripples proportional as the window is resized.

Grading

- **README: 10 points** - See README Requirements [here](http://cseweb.ucsd.edu/~ricko/CSE11READMEGuidelines.pdf) and questions below
  - [sec](http://cseweb.ucsd.edu/~ricko/CSE11READMEGuidelines.pdf)
- **Style: 20 points** - See Style Requirements [here](http://cseweb.ucsd.edu/~ricko/CSE11StyleGuidelines.pdf)
  - [sec](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 pa6 and go into that directory. The $ represents your command prompt. What you type in is in **bold**.

```bash
$ mkdir ~/pa6
$ cd ~/pa6
```
Copy the starter files from the public directory:

```bash
$ cp ~/../public/objectdraw.jar .
$ cp ~/../public/Acme.jar .
```

You will also need to copy over your files from PA4:

```bash
$ cp ~/pa4/*.java .
```

**Starter files provided:**

- objectdraw.jar
- Acme.jar

---

**ReflectingPond 2.0**

**NOTE:** Your PA4 program must work perfectly according to the PA4 specifications before continuing. There WILL be deductions if your program has errors from PA4!

The implementation details have been split up into 4 different stages, to help you split up the assignment into byte-sized (hah) chunks. Stages 1-3 focus on GUI event handling, and Stage 4 focuses on some additional features (dragging and resizing the canvas). If you prefer to work on something more familiar before jumping into Java GUI-land, you are free to start with Stage 4 and then proceed to Stages 1-3 if you wish. Also note that there are some general implementation notes listed after Stage 4.

**Stage 1: Creating the layout**

The basic layout of quadrants and lines are the same as in your PA4. You do not have to modify your `PondQuadrants.java` for PA6.

**Java GUI Components to Add:**

1. The **top control panel** at the top with a centered label and three buttons.
   - **Label:** Reflecting Pond Control Panel
   - **Buttons:** Run, Pause, Clear ripples

2. The **bottom speed panel** to control the speed of the Ripples.
   - **Label:** indicating the current speed
   - **Slider:** to control the speed (minimum speed is 1, maximum speed is 100, default speed is 50)

---

The text on your buttons and labels must match what is in the writeup *exactly*, character-for-character!
**Note:** When creating this layout, create your labels, buttons, panels, and sliders first (see the diagram on the next page for some help). Don’t forget to call `this.validate()` to set the GUI, and then create your PondQuadrants object after. Do not worry about the event handling on the GUI components yet.

![Diagram of a GUI layout with JPanel components in BorderLayout positions](image)

The textbook and the class notes are your best friends for this assignment!

**Stage 2: Converting to Standard Java Event Handling**
Once you get the layout looking good, it’s time to start using some standard Java event handling. You will need to change from using the objectdraw helper event handlers to the standard Java event handlers.

For example, change from using the objectdraw event handler

```java
public void onMouseClick( Location point )
```

to using (and defining) the standard Java event handler

```java
public void mouseClicked( MouseEvent evt )
```

You will also need to adjust your class declarations so both `ReflectingPond` and `Ripples` implement the following interfaces:

- `MouseListener`
- `MouseMotionListener`
- `ActionListener`
- `ChangeListener`

Modify any code that used a `Location` object to use a `MouseEvent` object. Again, **Chapter 11 and the notes should be excellent guides**. You will need to provide an implementation for each abstract method declaration in the interfaces you implement. You will need to do this in both `ReflectingPond.java` and `Ripples.java` (since both classes will need to respond independently to the buttons, slider, and mouse--more on this later).

**There should be no objectdraw onMouse*() methods.** However, we are still using objectdraw library objects like `Line`, `FilledRect`, `FramedOval` and their methods like `contains()` which take `Location` objects. So when necessary, you will need to create `Location` objects from the `MouseEvent`'s x and y - for example, `evt.getX()` and `evt.getY()` - see the [Java API docs for MouseEvent](https://docs.oracle.com/javase/8/docs/api/java/event/MouseEvent.html) and the [objectdraw docs for Location](https://objectdraw.com/doc/api/com/objectdraw/Location.html).
Pro Tip: Bring your textbook and class notes to the lab. They have pretty much everything you'll need to implement this assignment. You will probably still need to use some online docs for the objectdraw library and the standard Java API libraries.

Stage 3: Adding Standard Java Event Handling for the Buttons and Slider
We recommend using the event handling model discussed in class and in Chapter 11 in the textbook. You can use inner classes (private member inner classes or anonymous inner classes) for this assignment, but this is optional.

Top Control Panel:

<table>
<thead>
<tr>
<th>ReflectingPond</th>
<th>Ripples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Button</td>
<td>New Ripples should start growing/shrinking upon creation</td>
</tr>
<tr>
<td>Pause Button</td>
<td>New Ripples should NOT start growing/shrinking upon creation</td>
</tr>
<tr>
<td>Clear ripples Button</td>
<td></td>
</tr>
</tbody>
</table>

Note on the Clear ripples Button: After clicking “Clear ripples”, whichever mode (“Run” or “Pause”) you were in before clicking “Clear ripples” is the mode you will stay in after clearing the Ripples. That is, if you click the pause button, then the clear button, then create a new Ripples object, it should appear as just a tiny dot and should not be growing. Similarly, if you click the clear button, then the pause button, then create a new Ripples object, it should also just be a dot and not be growing.

Bottom Speed Panel:

<table>
<thead>
<tr>
<th>ReflectingPond</th>
<th>Ripples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Label</td>
<td>Update the label displayed when the slider is adjusted and when the Ripples are paused and un-paused (see screenshots below)</td>
</tr>
<tr>
<td>Speed Slider</td>
<td>New Ripples should start with a growing/shrinking speed according to the current value of the slider</td>
</tr>
</tbody>
</table>

Note on the speed slider: The smaller the speed value, the slower the Ripples shrink and grow; the higher the speed value, the faster the Ripples shrink and grow. You will have to do some calculation with the speed value extracted from the speed slider in order to have the speed of the Ripples work correctly.

For example: a speed of 1 should correspond to a pause time of 100 milliseconds; a speed of 100 should correspond to a pause time of 1 millisecond; a speed of 70 should correspond to a pause time of 31 milliseconds; and so on...
Implementation Notes:

- You will need to modify the Ripples constructor to deal with new information that needs to be sent to each Ripples object so that each Ripples object can operate independently (hint: see the boxes above that say “New Ripples ...”)
- Each Ripples object should not have any references to the buttons or slider, and the GUI controller should not keep track of the Ripples (no data structures!). This means you will need to register each new Ripples object with the buttons and slider right when you create it.
- When a Ripples is cleared from the canvas, it's run() method running in its own thread should end (change the forever loop in run() to have a condition that will fail when the Ripples is removed from the canvas). This will end the thread.
- Be careful to not accidentally end the while loop in the run() method when pausing the Ripples.

Stage 4: Adding New Features

Add the following features, remembering to use the Java event handler methods instead of objectdraw where appropriate.

Changing the size of the canvas:

- Quadrant Lines should stay proportional to where they were before adjusting the canvas size (same as PA4—you shouldn’t need to change anything here).
- Ripples should stay proportional to where they were before adjusting the canvas size (think about how you did this for the PondQuadrants and apply this to the Ripples).

Dragging objects on the canvas:

- Use the same 6 pixel margins as in PA4 to prevent the quadrant lines from being dragged off the canvas.
- Ripples can now also be grabbed and dragged around the canvas.
  - Ripples should change colors as they are dragged from one quadrant to another (based on which quadrant the center of the Ripples resides).
  - If several Ripples objects overlap each other, you can grab multiple Ripples in the overlapping area and drag them together around the canvas. You should not have to do anything special for this.
  - If a Ripples overlaps one or both lines you can grab the Ripples and the line(s) at the same time and move them all together. You should not have to do anything special for this.
  - Do not allow the center of a Ripples to be dragged past the 6 pixel margin.

General Implementation Notes:

- No static variables!
- No objectdraw mouse event handling methods!
- No data structures!
- There should be no mechanism in the GUI controller object that knows anything about how many Ripples there are or where they are or if they have been cleared from the canvas, etc.
- Each Ripples object must handle the various events (mouse, button, slider) on its own and make its own decisions, based on things like where its center is located and how big it is in its resizing cycle.
- Each Ripples object should not have any references to GUI components, and should simply respond to the events triggered by the GUI components.
Sample Screenshots

After clicking, ripples are created.

The ripples can be dragged around.

Overlapping ripples can be dragged at the same time.

Lines can be dragged at the same time as ripples.
The lines can also be dragged around on their own.

After clicking the “Clear ripples” button, the ripples should be removed and the lines should stay in place.

After clicking the “Pause” button, ripples can still be created, but they do not resize. **Note the speed label says paused with the speed value in parens.**

Lines and ripples can still be dragged in “Pause” mode. New ripples can be created, but they are paused. Can move speed slider with value changed in parens.
Clicking the “Run” button starts the ripples growing/shrinking again.

Changing speed with the speed slider. Note the speed label changes with the slider value.

When the size of the window is changed, both the lines and the center of the Ripples should stay proportional to their previous positions. The two screenshots above show before adjusting window size (left) and after adjusting window size (right), the positions of the Ripples and the lines stay proportional.
Questions to Answer in your README:

1. In vim, how do you move the cursor to the end of a line with a single command? To the beginning of a line with a single command?
2. How do you select a line in vim with a single command?
3. What does the keyword static mean in regards to variables?
4. What does the keyword static mean in regards to methods? Provide an example use of a static method.
5. If you have these two objectdraw shapes, properly created in this order and displayed on the canvas, provide one line of code that would tell you if the FramedArc is on top of the FilledOval?
   ```java
   FilledOval oval = new FilledOval(...);
   FramedArc arch = new FramedArc(...);
   ```
6. You are writing a program and when you run it, you get this NullPointerException:
   ```java
   Exception in thread "main" java.lang.NullPointerException
   at Test2.maybeThisOne(Test2.java:370)
   at Test.orIsItThisOne(Test.java:27)
   at Test.howAboutThisOne(Test.java:22)
   at Test.isItWorking(Test.java:18)
   at Test.main(Test.java:14)
   ```
   What file, method, and line should you look at to try to solve this exception?
7. How did you demonstrate a commitment to academic integrity while completing this assignment?

Extra Credit: Keyboard Controls

- [5 Points] Add keyboard controls to the 3 buttons and the speed slider.

Getting Started:
Make copies of the following files to do the extra credit in.
```
$ cd ~/pa6
$ cp ReflectingPond.java EC_ReflectingPond.java
$ cp Ripples.java EC_Ripples.java
```

**Important:** Your original ReflectingPond.java and Ripples.java files must remain unchanged. You need both the regular and the EC versions of these two files for turnin.

EC Requirements:
Add the following keyboard controls to the GUI buttons and slider:

- **Left Arrow Key**
  - Move the speed slider one unit to the left (i.e., if the speed is 50 and you press the left arrow key once, the speed should change to 49). If you press and hold the left arrow key, the speed slider should continuously decrease until you let go.

- **Right Arrow Key**
  - Move the speed slider one unit to the right (this should work just like the left arrow key, except it increases the speed slider's value instead of decreasing it).
- **Spacebar**
  - Toggle between paused and unpaused (i.e., if the Ripples are paused, pressing the spacebar is equivalent to pressing the Run button; if the Ripples are running, pressing the spacebar is equivalent to pressing the Pause button).

- **Backspace**
  - Clear all the Ripples (pressing backspace is equivalent to pressing the “Clear ripples” button).

The key to accomplishing this extra credit is listening during class and looking back at the textbook and your class notes. The rest of the functionality of your program should be unchanged from the regular PA6 assignment.

### Turnin Summary

See the turnin instructions [here](#). Your file names must match the below *exactly*.

**Due Date:** Wednesday night, November 7 @ 11:59 pm

<table>
<thead>
<tr>
<th>Files Required for Turnin:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PondQuadrants.java</td>
<td>Acme.jar</td>
<td>README</td>
</tr>
<tr>
<td>ReflectingPond.java</td>
<td>objectdraw.jar</td>
<td></td>
</tr>
<tr>
<td>Ripples.java</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extra Credit Files:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_ReflectingPond.java</td>
<td>EC_Ripples.java</td>
<td></td>
</tr>
</tbody>
</table>

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!**