Programming Assignment 3 (PA3) - GooglyEyes & RadioactivePinwheel

Due Date: Wednesday, October 17 @ 11:59 pm

<table>
<thead>
<tr>
<th>Assignment Overview</th>
<th>Grading</th>
<th>Gathering Starter Files</th>
<th>Program 1: GooglyEyes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 2: RadioactivePinwheel</td>
<td>README File</td>
<td>Extra Credit: Rolling Eyes</td>
<td>Turnin Summary</td>
</tr>
</tbody>
</table>

**Assignment Overview**

PA3 consists of two separate programs.

**GooglyEyes:**
Extend the DraggingEmoji GUI program from PA2 to be more Object-Oriented and add some additional functionality, such as making the eyes follow your mouse movements.

**RadioactivePinwheel:**
Write a standalone Java program that displays a Radioactive Pinwheel design on the console outputting individual characters one character at a time using nested loops.

**Grading**

- **README: 10 points** - See README Requirements [here](http://cseweb.ucsd.edu/~ricko/CSE11READMEGuidelines.pdf)
- **Style: 20 points** - See Style Requirements [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 pa3 and go into that directory. The $ represents your command prompt. What you type in is in **bold**.

```
$ mkdir ~/pa3
$ cd ~/pa3
```

Copy the starter files from the public directory:

```
$ cp ~/../public/objectdraw.jar .
$ cp ~/../public/Acme.jar .
$ cp ~/../public/PA3StarterCode/PA3Strings.java .
$ cp ~/../public/PA3StarterCode/Timer.java .
```

**Starter files provided:**

- objectdraw.jar
- Acme.jar
- PA3Strings.java
- Timer.java
Program 1: GooglyEyes

This program will build off of your DraggingEmoji program from PA2 (make sure your DraggingEmoji program is working properly before proceeding!). To make this program more Object-Oriented, we will separate the GUI controller and the Emoji object. We will also be adding a new feature where the eyes will look at the mouse as the mouse moves around (this means the eyes will now be open by default). The other feature we will be adding is allowing the user to long-click on the eyes to open and close them.

Note: In DraggingEmoji, the eyes were closed by default, and were only open while the mouse was holding the emoji. In GooglyEyes, the eyes are always open by default, and the eyes only switch between open and closed when the user long-clicks on them (more on this later—it'll be an eye-opening experience).

New Features to Add:

Eyes Look at the Mouse:
The first new feature to add is to make the eyes always look at the mouse as the mouse is moved around (onMouseMove()). To make this easier, we will define 5 different preset directions that the eyes can look. We create the effect of the eyes looking in a specific direction by moving the pupils to one of the preset locations shown below, depending on where the mouse is.

If the mouse is directly on the border between two (or more) sections, it does not matter which of the neighboring sections you choose to make the eyes look at.

Refer to the constants and diagrams provided in the PA2 writeup for positioning the pupils in each of the 5 configurations (no additional constants are needed). The "left, up" configuration is the same configuration that was used in PA2. The rest of the configurations (except the “middle” configuration) are mirror images of the “left, up” configuration. Very detailed diagrams are provided in the PA2 writeup for the “left, up” configuration. Drawing similar diagrams for the rest of the configurations is left as an exercise for the reader.
Long-Click to Open/Close Eyes:
The second new feature to add is to allow the user to open and close the eyes by “long-clicking” on the emoji. A “long click” is defined as a click (meaning onMouseClick() is called) that lasts longer than 500 milliseconds (1/2 second). Note that onMouseClick() is only called if there is no mouse movement between mouse press and mouse release. If the eyes are currently open and the user long-clicks on the emoji, close the eyes. If the eyes are currently closed and the user long-clicks on the emoji, open the eyes. More implementation details for this are provided below.

Sample Screenshots:
The following screenshots are ordered left to right, top to bottom. They should be looked at in that order to make the most sense out of them.

Click to display an eye emoji centered at the mouse click.
Mouse press in any part of the emoji and drag to move the emoji around.
Move the mouse around the canvas to make the eyes look at the mouse.

Close-up of the instructions displayed on startup (shown in the screenshot to the left). Note that there are 3 lines of instructions this time. These strings are provided in PA3Strings.java.

Your text must match the text in the screenshot *exactly* (character for character).
Mouse is moved to the “left, up” section. The pupils move to look at the mouse.

Mouse is moved to the “right, up” section. The pupils move to look at the mouse.

Mouse is moved to the “right, down” section. The pupils move to look at the mouse.

Mouse is moved to the “middle” section. The pupils move to look at the mouse.

After a long click, the eyes close.

On mouse press. Note that the eyes do NOT open.

The emoji can still be dragged normally with the eyes closed.

After another long click, the eyes open, and the eyes are looking at the mouse (the pupils should still “look” at the mouse while the eyes are closed).

After yet another long click, the eyes close again.
The emoji can be dragged off the canvas. When the mouse exits the canvas, the emoji is removed from the canvas.

When the mouse enters the canvas, the instructions reappear.

On mouse click. The eyes should always be open when the emoji is created on a mouse click.

No exceptions should be thrown!

Files Required for this Program:

GooglyEyes.java  Emoji.java  Timer.java  PA3Strings.java

GooglyEyes.java:
This is the main GUI controller class that handles all mouse events and controls what we see on the canvas. This class (and only this class) performs all the event handling on the canvas. To make that possible, it needs to extend WindowController. Methods defined in this class include: main(), begin(), onMouseClick(), onMousePress(), onMouseRelease(), onMouseDrag(), onMouseMove(), onMouseExit(), onMouseEnter(). Where appropriate, code in this class will create an instance of an Emoji and send messages to this Emoji object (tell the Emoji figure to move, ask the Emoji figure to check if the current mouse Location is contained in the eyes, tell the Emoji figure to move its pupils, tell the Emoji to remove itself from the canvas, etc.).

Emoji.java:
This class defines what an Emoji figure is (the 4 filled ovals and 2 framed ovals) and what a Emoji can do (what messages it can respond to). This class will define a constructor to initialize a new Emoji object and place it on the canvas as part of the GooglyEyes. The canvas from the GUI controller and a Location will be passed as actual arguments to the Emoji constructor so the constructor knows where to place the Emoji. This class will also have methods to determine if the mouse pointer is contained in the Emoji, move the Emoji some delta, remove the Emoji from the canvas, and move the pupils to look at the mouse as it moves around the canvas.

Timer.java:
This class calculates timing between events. This code is copied from Figure 6.5 in the textbook, and was provided as one of the starter files.

General Info to Get You Started:
It is important to note the difference between the GooglyEyes class and the Emoji class. The Emoji class should have all relevant variables and methods specific to each Emoji created on the canvas (in this case we only create one at a time). The GUI controller class (GooglyEyes) handles all the activities that occur on the canvas--this includes the mouse interactions/events with the canvas and Emoji object on the canvas. To reiterate: the Emoji class knows what an Emoji object is made of (the 6 ovals) and the main GUI controller knows about the mouse events and the drawing canvas.
Since both classes interact with each other, the GooglyEyes (GUI) needs to have a reference to an Emoji object and the Emoji object needs to be passed a reference to the GUI canvas in GooglyEyes so it knows where to draw itself. This is done by passing the canvas from the GUI as an actual argument to the Emoji constructor when the code in onMouseClick() in GooglyEyes creates an Emoji object. Examples in Chapter 6 of the textbook show how to do this (sections 6.2 and 6.3 will be especially useful).

The program should still have the same functionality as was specified in PA2 (able to create an Emoji on the canvas and drag it around). Note that the eyes will now default to being open, as mentioned at the beginning of this section of the writeup (right under the heading “Program 1: GooglyEyes”).

Use calls to the Timer class methods in class GooglyEyes to determine how long the mouse button was pressed down during a mouse click (remember - anything dealing with event handlers is done in the GUI controller - GooglyEyes in this case):

- Create a Timer object in begin().
- When the mouse button is pressed, the timer should be reset.
- When the mouse button is clicked, get the elapsed time in milliseconds. If the mouse click was in the Emoji figure and the elapsed time was longer than 500 milliseconds (1/2 second), then the Emoji eyes should close if they were previously open, or open if they were previously closed.

A mouse click will fire these events: onMousePress(), onMouseRelease(), onMouseClick()
  - There is NO mouse movement between mouse press and mouse release!

A mouse drag will fire these events: onMousePress(), onMouseDrag(), onMouseRelease()
  - There IS mouse movement between mouse press and mouse release!

Skeleton Example of GooglyEyes.java:
Note that you are responsible for writing class and method headers for all of the skeleton code provided below.

```java
import Acme.*;
import objectdraw.*;

public class GooglyEyes extends WindowController {

private static final int WINDOW_SIZE = 750;

// Position of instructions
private static final int INSTR1_X = 50;
private static final int INSTR1_Y = 50;
private static final int INSTR2_X = INSTR1_X;
private static final int INSTR2_Y = INSTR1_Y;
private static final int INSTR3_X = INSTR2_X;
private static final int INSTR3_Y = INSTR2_Y;

// Length of long mouse click (in milliseconds) to open/close the eyes
private static final int EYE_CLICK_OPEN_CLOSE_THRESHOLD = 500;

private Timer timer;
private Emoji emoji;
...

// Additional variables you might need and ...

public static void main(String[] args) {
    new Acme.MainFrame(new GooglyEyes(), args, WINDOW_SIZE, WINDOW_SIZE);
}

public void begin() {}

public void onMouseClick(Location point) {}

public void onMousePress(Location point) {}
```
import objectdraw.*;
import java.awt.Color;

public class Emoji {

  // Eye size and positioning
  private static final int EYE_WIDTH = 100;
  private static final int EYE_HEIGHT = 150;
  private static final int HALF_EYE_WIDTH = EYE_WIDTH / 2;
  private static final int HALF_EYE_HEIGHT = EYE_HEIGHT / 2;

  // Pupil size and positioning
  private static final int PUPIL_WIDTH = 50;
  private static final int PUPIL_HEIGHT = 60;
  private static final int HALF_PUPIL_WIDTH = PUPIL_WIDTH / 2;
  private static final int HALF_PUPIL_HEIGHT = PUPIL_HEIGHT / 2;
  private static final int PUPIL_X_OFFSET = 23;
  private static final int PUPIL_Y_OFFSET = 20;

  // Any additional constants you might need ...

  // Left eye
  private FilledOval leftEye, leftPupil;
  private FramedOval leftEyeFrame;

  // Right eye
  private FilledOval rightEye, rightPupil;
  private FramedOval rightEyeFrame;

  ... // Any additional variables you might need ...

  public Emoji(Location point, DrawingCanvas canvas) {} // Constructor
  public boolean contains(Location point) {} // Constructor
  public void move(double dx, double dy) {} // Constructor
  public void lookAt(Location point) {} // Constructor
  public void lookLeftUp() {} // Constructor
  public void lookLeftDown() {} // Constructor
  public void lookRightUp() {} // Constructor
  public void lookRightDown() {} // Constructor
  public void lookMiddle() {} // Constructor

  public void onMouseRelease(Location point) {} // Constructor
  public void onMouseDrag(Location point) {} // Constructor
  public void onMouseMove(Location point) {} // Constructor
  public void onMouseExit(Location point) {} // Constructor
  public void onMouseEnter(Location point) {} // Constructor

  // Any additional methods you might need ...

  Skeleton Example of Emoji.java:
  Note that you are responsible for writing class and method headers for all of the skeleton code provided below.
public void removeFromCanvas() {} 

Note: All of the skeleton code provided is *skeleton* code, meaning it is the bare minimal code to get you started. You should definitely be adding constants, variables, and methods to both of the above classes.

Compiling:
To compile your code, use the following:

```
$ javac -cp ./Acme.jar:./objectdraw.jar:. GooglyEyes.java
```

To compile all three files, you only need to specify the source file that is the main driver that references these other types. Since GooglyEyes.java uses the types Emoji and Timer, the compiler looks for Emoji.java/Emoji.class and Timer.java/Timer.class in the classpath specified and you have . in the classpath. The compiler will automatically (re)compile Emoji.java and Timer.java if needed.

Running:
To run your program, use the following:

```
$ java -cp ./Acme.jar:./objectdraw.jar:. GooglyEyes
```

Program 2: RadioactivePinwheel

Write a Java application (RadioactivePinwheel.java) that displays the following radioactive pinwheel design to the terminal using nested loops. The user will be required to enter the size of the pinwheel and can optionally specify the characters to use to draw the pinwheel. Like ShortLongWords from PA2, this is a standalone application with no graphical elements (you do not need objectdraw or Acme.MainFrame for this program).

Conceptual Explanation:

```
  
  size = number of rows
  
  = upward triangle
  
  = downward triangle
```

Example:

```
  *0000000*
  ****00000****
  *****000*****
  ********0********
  0000000000000000
  000000000000
  00000000000
  0000000000
  000000000
```

```
size: 8
upward triangle char: *
downward triangle char: 0
```

Requirements for this Program:

- You may only print one character at a time. Do not use a string with the pattern. You are not allowed to use a String or StringBuilder or StringBuffer or any data structure to build each line to be output. You must calculate how many times to loop printing a single upward triangle character at a time, how many times to loop printing a single downward triangle character at a time, and how many times to loop printing a single space character at a time.
- You must create at least one helper method. Try to modularize your design so you don’t have all of your code in one giant main() method. Think about how you could split up the different parts of the program (hint -- the way the writeup splits up the different parts of the program is a good place to start that train of thought).
**Understanding Format Strings:**

**General Idea:**
In this program, we’ll be using format strings to print some (not all) of the output and error messages. A format string is like a regular string, except it contains format specifiers (such as %d, %c, %s, etc). Format specifiers allow us to fill in values in the string at the moment we are using the string (in our case, at the time we are printing the string), rather than needing to fill in these values when we define the string.

This is helpful for cases where we don’t know the values when we define the string, or for cases where we want to define several similar strings where only one or a few values are different. There are many other scenarios where format strings are useful as well. All of the strings you will need to print in this program are provided in PA3Strings.java. Go ahead and take a look at that file to see some of the places we use format specifiers.

**Printing Format Strings:**

To print a string with format specifiers in Java, we can use `System.out.format()` where the format string is passed in as the first argument, and the values to fill in the format specifiers are passed as additional arguments.

For example:

```java
String drinkStr = "I drank %d cups of %s today.\n";
System.out.format(drinkStr, 4, "water");
System.out.format(drinkStr, 2, "coffee");
System.out.format(drinkStr, 0, "soda");
```

Output:

```
I drank 4 cups of water today.
I drank 2 cups of coffee today.
I drank 0 cups of soda today.
```

Use these concepts when printing the strings with format specifiers in PA3Strings.java.

**Part 1 -- Gathering User Input:**

First print the instructions (string provided in PA3Strings.java), making sure to fill in all the format specifiers. Then use a Scanner like in ShortLongWords from PA2 to read user input.

**Steps for reading and checking each input:**

1. Prompt the user for a specific input (size, upward triangle char, downward triangle char)
2. Check if there is user input and read it (Hint: hasNextLine() and nextLine() in Scanner will be helpful).
3. Check if the input is valid (See “Checking user input” section below for input requirements)

Note: After getting the next line of user input from the scanner, there might be leading and trailing whitespace. You can use the trim() method in String to remove them. Be careful to not use this when reading the triangle chars because space is a valid char.

**Checking user input:**

Each time after you read a value from the user, you need to check that it is valid. Use the constants provided after the example output section to check the validity of each input.

Errors with each input must be checked in the following order! If any error listed below occurs, print the correct error message from PA3Strings.java and exit right away with exit code 1 (`System.exit(1);`).

* `System.exit(1);` can be called from any method or constructor to immediately end the program. The exit code “1” indicates that the program ended because of an error.

**Size:**

- Can’t be converted to an int. Catch the NumberFormatException that is thrown and then handle the error as noted above. To convert the size to an integer, use `Integer.parseInt()`.
- Not within [4, 50] (inclusive)
- Not an even number
By now we know the size is a valid number and we can begin to check if the user entered values for one or both of the possible pinwheel characters. Only complete the following checks if there is user input. If the user did not give the character, be sure to use the default values in the constants provided.

Triangle Chars (Upward char and Downward char):
- Not a single character
- Not within printable ASCII range [32, 126] (inclusive)

Note: if the user doesn’t enter anything and just presses the enter key, use the appropriate default character for that triangle character.

After all of the above checks have been completed (and no errors were encountered), make sure the two triangle characters are different. If they are the same character, handle the error as it says to handle all errors above.

Part 2 -- Print the Radioactive Pinwheel:
If you haven’t already, read the requirements for this program under the “Program 2: Radioactive Pinwheel” heading. The main idea here is that you must print each character one at a time in a loop. To print the entire pinwheel, you will need nested loops. We suggest splitting this up into two parts (one for printing out the top half of the pinwheel, one for printing out the bottom half of the pinwheel). To get you started thinking about how to structure your loops, consider the fact that you need to print the pinwheel left to right, top to bottom. In other words you need to print one character at a time, and print the characters row by row.

You must print space characters on both the left and right sides of the pinwheel where there is empty space. If you do not print the space characters on the right side of the pinwheel, you will lose points!

Example:

```
size: 4
upward triangle char: @
downward triangle char: -
```

You must print space characters in the empty boxes.

```
+-----+-----+-----+-----+
|     | @    |    | @   |
| @    | @    |    | @   |
| @    | @    | @  |    |
|     | @    |    |     |
+-----+-----+-----+-----+
```

Example Output:
The following examples below should help you gain an understanding of what needs to be printed in different cases. User input is shown in bold. Note that when you see <TAB>, this is the user pressing the tab key. Your output must match this output exactly, character-for-character, newline-for-newline.

No size entered: (in this case, the user is just pressing enter when prompted to enter the size)

```
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
    -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
    -- must be a single character within the ASCII range [32, 126]
    -- must be different than downward triangle char
    -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
    -- must be a single character within the ASCII range [32, 126]
```
Enter the size of the Radioactive Pinwheel to display:

Error: Radioactive Pinwheel size must be a single integer
$
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: aaaaahhhhh

Error: Radioactive Pinwheel size must be a single integer
$
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 1 2 3 4 5

Error: Radioactive Pinwheel size must be a single integer
$
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')
Enter the size of the Radioactive Pinwheel to display: 17

Error: Radioactive Pinwheel size must be an even integer
$

Size is out of bounds:
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 52

Error: Radioactive Pinwheel size must be within the range [4, 50]
$

Size is both odd and out of bounds:
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 51

Error: Radioactive Pinwheel size must be within the range [4, 50]
$

Upwards triangle character is more than one character:
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 4
Enter the character to use for the upward triangles: .*.*.*

Error: The upward triangle char must be a single character
$

Upwards triangle character is out of bounds:
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 4
Enter the character to use for the upward triangles: <TAB>

Error: The upward triangle char must be within ASCII range [32, 126]
$

Downwards triangle character is more than one character:
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 4
Enter the character to use for the upward triangles: a
Enter the character to use for the downward triangles: potato

Error: The downward triangle char must be a single character
$

Downwards triangle character is out of bounds:
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
   -- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than downward triangle char
   -- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
   -- must be a single character within the ASCII range [32, 126]
   -- must be different than upward triangle char
   -- enter nothing to use default character (')
Enter the size of the Radioactive Pinwheel to display: 4
Enter the character to use for the upward triangles: a
Enter the character to use for the downward triangles: <TAB>

Error: The downward triangle char must be within ASCII range [32, 126]
$

Downwards and upwards triangle characters are the same:
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
-- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
-- must be a single character within the ASCII range [32, 126]
-- must be different than downward triangle char
-- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
-- must be a single character within the ASCII range [32, 126]
-- must be different than upward triangle char
-- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 4
Enter the character to use for the upward triangles: a
Enter the character to use for the downward triangles: a

Error: Upward and downward triangle chars cannot be the same char
$

Default characters with size of 4: (in this case the user is just pressing enter when prompted to enter each triangle character)
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
-- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
-- must be a single character within the ASCII range [32, 126]
-- must be different than downward triangle char
-- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
-- must be a single character within the ASCII range [32, 126]
-- must be different than upward triangle char
-- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 4
Enter the character to use for the upward triangles:
Enter the character to use for the downward triangles:

    @''@'
    @@@@@@
    ''''@'''
    '@@@'

$

Default characters with size of 8:(in this case the user is just pressing enter when prompted to enter each triangle character)
$ java RadioactivePinwheel
Size: equal to the number of rows in the Radioactive Pinwheel
-- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
-- must be a single character within the ASCII range [32, 126]
-- must be different than downward triangle char
-- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
-- must be a single character within the ASCII range [32, 126]
-- must be different than upward triangle char
-- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 8
Enter the character to use for the upward triangles:
Enter the character to use for the downward triangles:

```
@''''''''
@@''''''''
@''''''''''
''''''''''
''''''''''
''''''''''
''''''''''
''''''''''
```

Different characters with size of 10:

```bash
$ java RadioactivePinwheel
```

Size: equal to the number of rows in the Radioactive Pinwheel
-- must be an even integer in the range [4, 50]
Upward triangle char: used for drawing the upward triangles
-- must be a single character within the ASCII range [32, 126]
-- must be different than downward triangle char
-- enter nothing to use default character (@)
Downward triangle char: used for drawing the downward triangles
-- must be a single character within the ASCII range [32, 126]
-- must be different than upward triangle char
-- enter nothing to use default character (')

Enter the size of the Radioactive Pinwheel to display: 10
Enter the character to use for the upward triangles: x
Enter the character to use for the downward triangles: -

```
x---------x
xxxx--------
xxxxxx------xxxxx
xxxxxxxx------xxxxxxx
xxxxxxxxxxxx-xxxxxxxxxxx
-------------x----------
-------------xxxx------
-------------xxxxxxx----
-------------xxxxxxxxxx--
```

$
Constants to Use for this Program:
private static final int MIN_SIZE = 4;
private static final int MAX_SIZE = 50;

private static final char DEFAULT_UPWARD_CHAR = '@';
private static final char DEFAULT_DOWNWARD_CHAR = '\';
private static final char SPACE_CHAR = ' ';
private static final int CHAR_ASCII_MIN = 32;
private static final int CHAR_ASCII_MAX = 126;

Compiling:
To compile your code, use the following:
$ javac RadioactivePinwheel.java

Running:
To run your program, use the following:
$ java RadioactivePinwheel

README File

Remember to follow all of the guidelines outlined in the README Guidelines. Don’t forget to do a program description. 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. What happens when you type ":vs Foo.java" in command mode in vim?
2. What do the Vim commands :q and :q! do? How are they similar? How are they different?
3. From your current directory, how do you remove all jar files in your home directory?
4. What is the command to clear a Unix terminal screen?
5. Your friend told you that they found solution code online and are planning to use it in their PA. How can you encourage your friend to excel with integrity?

Extra Credit: I bet you’ll roll your eyes at this one

- [5 Points] Make the Emoji roll its eyes when the user long-clicks on the eyes

Getting Started:
Make a copy of your GooglyEyes files to do the extra credit in.
$ cd ~/pa3
$ cp GooglyEyes.java EC_GooglyEyes.java
$ cp Emoji.java EC_Emoji.java

After copying, be sure to change all instances of Emoji to EC_Emoji and GooglyEyes to EC_GooglyEyes so your code can compile and run properly.

Important: Your original GooglyEyes.java and Emoji.java files must remain unchanged. You need all four files for turnin (the two original files and the two extra credit files).

EC Requirements:
- The eyes should no longer toggle between open and closed after a long click (the eyes should always be open).
- If the Emoji is long clicked in the **center region**, make the pupils look outwards instead of inwards. If the pupils are already looking outwards, make them look back inwards towards the middle again.
- If the Emoji is long clicked on the **left eye** (outside the center region), both pupils will roll **clockwise** to the next quadrant.
- If the Emoji is long clicked on the **right eye** (outside the center region), both pupils will roll **counterclockwise** to the next quadrant.

**Sample Screenshots:**

Emoji has already been previously created. Pupils are looking down, right at the mouse.

After a long click on the right eye. The pupils rolled counterclockwise.

After a long click on the right eye. The pupils rolled counterclockwise.

After moving the mouse over the left eye.
After a long click on the left eye. The pupils rolled clockwise.

After a long click on the left eye. The pupils rolled clockwise.

After a long click on the left eye. The pupils rolled clockwise.

After moving the mouse over the center region. Pupils moved to look outwards.

After a long click on the center region. Pupils moved back to look inwards.

After a long click on the center region. Pupils moved to look outwards.

After moving the mouse. Pupils go back to normal behavior.
Compiling:
You can compile the extra credit program using the following command.

$ javac -cp ./Acme.jar:./objectdraw.jar:. EC_GooglyEyes.java

Running:
You can run the extra credit program using the following command.

$ java -cp ./Acme.jar:./objectdraw.jar:. EC_GooglyEyes

Turnin Summary

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

Due Date: Wednesday night, October 17 @ 11:59 pm

Files required for the Turn-in:

<table>
<thead>
<tr>
<th>Emoji.java</th>
<th>Acme.jar</th>
<th>README</th>
</tr>
</thead>
<tbody>
<tr>
<td>GooglyEyes.java</td>
<td>objectdraw.jar</td>
<td></td>
</tr>
<tr>
<td>PA3Strings.java</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RadioactivePinwheel.java</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer.java</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Extra Credit Files:

<table>
<thead>
<tr>
<th>EC_Emoji.java</th>
<th>EC_GooglyEyes.java</th>
</tr>
</thead>
</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, Start Often, Finish Early, and Have Fun!