Programming Assignment 4  ( 100 Points )
Due: 11:59pm Thursday, October 26
START EARLY!!

In PA4 you will continue exploring the graphical user interface (GUI) and object oriented programming. You will be creating a GUI that responds to mouse events to create colorful crazy orbs with shrinking and growing animation.

README ( 10 points )
You are required to provide a text file named README, NOT Readme.txt, README.pdf, or README.doc, with your assignment in your pa4 directory. There should be no file extension after the file name "README". Your README should include the following sections:

Program Description ( 3 points ) : Provide a high level description of what your program does and how you can interact with it. Make this explanation such that your grandmother or uncle or someone you know who has no programming experience can understand what this program does and how to use it. Write your READMEs as if it was intended for a 5 year old. Do not assume your reader is a computer science major.

Short Response ( 7 points ) : Answer the following questions:
1. (AI) What was your process for completing this assignment with integrity?
2. (Unix) From your current directory how do you copy over a java file named Zumba.java 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.
3. (Unix) What does the command “man diff” do (with no quotes)?
4. (Unix) From your current directory, how do you remove all jar files in your home directory?
5. (Vim) What is the difference between :q and :q! when using the commands to close a vim file?
6. (Vim) What does the command “gg” do (with no quotes)?
7. (Java) What is method overloading?

STYLE ( 20 points )
Please see PA2 for the style guidelines. Note: Some of the guidelines have changed since PA1. In terms of grading, we will be using the style guidelines in PA2 for the rest of the PAs.

CORRECTNESS ( 70 points )

Setting up your pa4 directory:
Please see previous programming assignments.
You will need the objectdraw and Acme libraries for this assignment.
You will be creating two files: CrazyOrbGUI.java and CrazyOrb.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.

Stage 1: Creating Quadrants

Begin with the GUI controller class (CrazyOrbGUI) by creating 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 (width and height) of the canvas. The initial size of the canvas should be 600 x 600 pixels.

Stage 2: Manipulating Quadrants

The first manipulation you should implement is dragging the Lines. Check in the onMousePress() method to determine whether one or both of the Lines have been grabbed or not. You can set boolean flags in this method to indicate which line(s) have been selected. Note that if you grab the intersection of the two Lines, you should be able to drag both lines simultaneously. Refer to the objectdraw documentation (linked from the Useful Links page) for other mouse event methods that may be useful.

You will need to update the position of the Lines as they are dragged [onMouseDrag()]. You should include logic to make sure that the Lines are not dragged off the visible canvas/applet area. Do not let either line go beyond 6 pixels from the edge of the canvas in any direction (a 6 pixel margin). See screenshot at bottom of page 4.

The second manipulation you should implement is keeping the Lines proportional when adjusting the size of 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:

```
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:

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

as the first line in your paint(). You can then add in your code for repositioning the Lines based on the proportions of where they were previously.
Stage 3: Creating the Crazy Orbs

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

The constructor should look like this:

```java
public CrazyOrb( double xLoc, double yLoc, double size,
                  DrawingCanvas canvas, Line hLine, Line vLine )
```

The first two parameters are the x and y coordinates of the center of the crazy orb (where the mouse was clicked). The third parameter is the starting size (diameter) of the crazy orb (must be 60 pixels for this assignment). The last two parameters are the horizontal and vertical Lines used to divide the canvas into different quadrants so this new CrazyOrb knows which quadrant it is being created in and can keep track of those Lines for later use.

A crazy orb is made up of both a FilledOval and a FramedOval. Both have the same starting size.

For now, the constructor should just create and display the crazy orb. Remember: the Filled/FramedOval object shapes use the upper-left corner of an invisible bounding box as the x and y location to draw their shapes. So translate accordingly.

**Remember:** the CrazyOrb 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 4: Crazy Orb shrinking and growing

You should add a **run()** method with a forever loop to your CrazyOrb class to deal with the animation of the crazy orbs. The diameter of the crazy orb 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 orbs won’t grow and shrink too quickly. Once one of the orb parts grows by greater than half its starting size (starting size + half starting size), it should start to shrink. Once one of the orb parts shrinks to less than half its starting size, it should start to grow. **The center of the CrazyOrb parts need to remain in their original location.** The screenshot shows multiple orbs shrinking and growing at the same time.
The FilledOval part of a CrazyOrb starts off growing. The FramedOval part of a CrazyOrb starts off shrinking. These two parts of a CrazyOrb should be in sync. When the FilledOval part reaches its max size and starts shrinking, the FramedOval part should reach its minimum size and start growing, and vice versa.

Note: Because the CrazyOrb 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 when the crazy orb is created at the end of the constructor. (See Ch 9 and the lecture notes on Active Objects. We went over all of this in class.)

Stage 5: Different Colors for Each Quadrant

The FilledOval part of a crazy orb that is created in each quadrant (centered at the point of the mouse click) will be a particular color based on the quadrant:
- upper-left – Cyan (Cyan);
- upper-right – Magenta (Magenta);
- lower-left – Yellow (Yellow);
- lower-right – Black (Black).

Likewise, the FramedOval part of a crazy orb will be a particular color based on the quadrant:
- upper-left – Blue (Blue);
- upper-right – Pink (Pink);
- lower-left – Green (Green);
- lower-right – Gray (Gray).

Use the constants defined in Java’s Color class. (Look at the javadocs).

You should set the color of each crazy orb part based on which quadrant the center of the crazy orb is located in at any particular time. Use the two Lines passed into the constructor and the crazy orb’s center to determine which quadrant the crazy orb is in.

If the center of the orb is exactly on the vertical line, consider it to be to the right of the line (use < logic). If the center of the orb is exactly on the horizontal line, consider it to be below the line (use < logic). If the center of the orb is exactly at the intersection of both lines, consider it to be lower right quadrant (in general, use < logic vs. <= logic).

When the user drags the lines, the quadrant areas are redefined. The lines also move when adjusting the window size, meaning that the Lines stay proportional to how they were before the canvas was resized. Whenever the lines move, each crazy orb does not move but needs to check which quadrant it is in and update its color if necessary. Note: the crazy orbs do not move when the window resizes.

Image showing 6 pixel margin from edge of canvas. Do NOT allow the lines to be dragged closer than 6 pixels from the edge of the current canvas width and height.
Before dragging the lines.

After dragging the lines.

Resizing the window. Lines stay proportional. Crazy orbs do not move, but may change color based on which quadrant each crazy orb center now lies in.
EXTRA CREDIT (5 points): Crazy Orbs with Cycling Colors

Getting Started:
For extra credit, you will be adding some features to your Crazy Orbs, so before you start, make sure your Crazy Orbs are working perfectly.
Once your Crazy Orbs are working perfectly, make a copy of your source files like so:

```bash
$ cd ~/pa4
$ cp CrazyOrb.java EC_CrazyOrb.java
$ cp CrazyOrbGUI.java EC_CrazyOrbGUI.java
```

By using the previous Unix commands, you are making a copy of CrazyOrb.java and CrazyOrbGUI.java and naming those copies EC_CrazyOrb.java and EC_CrazyOrbGUI.java. ALL 4 FILES ARE NEEDED IF YOU ATTEMPT THE EXTRA CREDIT.

After this, make sure to change **all** instances of CrazyOrb to EC_CrazyOrb and **all** instances of CrazyOrbGUI to EC_CrazyOrbGUI in both EC_CrazyOrb.java and EC_CrazyOrbGUI.java. For example:

```java
public class EC_CrazyOrb extends ActiveObject {
    ...
}
```

Main Idea:

You will need a cycling colors toggle/flag/mode that you can switch on and off. Up to and including when the first crazy orb is created, the toggle must be off (no matter what other mouse events may happen). Once at least one crazy orb is created, the cycling colors mode is switched/toggled on/off every time the mouse 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.

With the cycling colors mode on, the crazy orbs cycle through the next CMYK color each time the orb reaches the point to change from growing to shrinking and from shrinking to growing.

For example, if a crazy orb is in the upper left quadrant (FilledOval part is Cyan and the FramedOval part is Blue), in cycling colors mode it will cycle to the next color scheme (Magenta and Pink) when that crazy orb switches from growing to shrinking (or vice versa). At the next point in time when that crazy orb switches from growing to shrinking (or vice versa), it will cycle to the next color scheme (Yellow and Green). And the next switch from growing to shrinking (or vice versa), it will cycle to the next color scheme (Black and Gray). Then the next switch back to Cyan and Blue, and so on.

The color cycling will continue until the next mouse enter event which will toggle the cycling colors mode to off and each orb will **immediately** (the next run() loop iteration) go back to whatever color scheme it is supposed to be based on which quadrant its center is currently in. The next mouse enter event will toggle the cycling colors mode to on again. Perform all mouse exit/enter operations slowly. One way to make sure the system is recognizing your mouse enter is look for the cursor to change to a bi-directional (or something similar) arrow at the edge of the canvas when you slowly bring the mouse into the canvas.
During color cycling mode, any new crazy orb that is created will always start with the color scheme of the quadrant it is created in, and then with the next switch from growing to shrinking (or vice versa) that orb will start cycling through the next CMYK color scheme depending on what color scheme it was.

A (new) crazy orb in upper-left cycles through Cyan/Blue, Magenta/Pink, Yellow/Green, Black/Gray, and back to Cyan/Blue, and so on.

A (new) crazy orb in upper-right cycles through Magenta/Pink, Yellow/Green, Black/Gray, Cyan/Blue, and back to Magenta/Pink, and so on.

A (new) crazy orb in lower-left cycles through Yellow/Green, Black/Gray, Cyan/Blue, Magenta/Pink, and back to Yellow/Green, and so on.

A (new) crazy orb in lower-right cycles through Black/Gray, Cyan/Blue, Magenta/Pink, Yellow/Green, and back to Black/Gray, and so on.

In color cycling mode, you can still grab and move the horizontal and vertical Lines around to define new quadrant areas, but the crazy orbs continue with their color cycling. When the current color cycling mode is switched off, all of the crazy orbs take on the color scheme based on the quadrant they are currently in based on any moving of the horizontal and vertical Lines while they were in color cycling mode.

Remember - you can move the mouse in and out of the canvas and expand the size of the window, etc. but none of those events affect the color cycling mode until the first crazy orb is created with a mouse click. Once the first crazy orb is created, then any subsequent mouse enter will start toggling cycling colors mode.

The growing/shrinking of the crazy orb parts never reset. They keep growing/shrinking with each iteration in the run() method loop. Only the color scheme changes with subsequent mouse enters.

**NOTE:**

Just like the non-extra credit part, when the lines move:

- The lines redefine the quadrant areas for the orb color scheme in non-cycling color mode.

When adjusting the window size:

- The lines also move and redefine the quadrant areas, meaning that the lines stay proportional to how they were before the canvas was resized.
- The crazy orbs stay where they are when the window resizes.

**Sample Screenshots:**

(Showing a sequence of actions going from left to right, top to bottom)
Upon startup.

Mouse exit.

Mouse enter → cycling color mode is still turned off because no crazy orbs have been created yet.

Initial group of orbs are created without the mouse exiting the canvas.
Mouse exit and then enter to toggle cycle color mode. Each crazy orb is cycling through its color schemes.

Move Lines that define quadrants. Still in cycle color mode. No change to what each crazy orb is doing.

Creating another crazy orb in cycle color mode in far upper-left corner. Note how it starts in the color scheme for that quadrant and then starts cycling through the color schemes once it stops growing/shrinking and starts shrinking/growing.

Another mouse exit and enter to toggle cycle color mode to off. Each crazy orb takes on the color scheme of the quadrant its center is in. The growing/shrinking is not affected and continues wherever it was before. Only the color scheme changes.
**TURNIN**

**Turnin**

To turnin your code, navigate to your home directory and run the following command:

```
$ cse11turnin pa4
```

You may turn in your programming assignment as many times as you like. The last submission you turn in before the deadline is the one that we will collect.

**Verify**

To verify a previously turned in assignment,

```
$ cse11verify pa4
```

If you are unsure your program has been turned in, use the verify command. We will not take any late files you forgot to turn in. Verify will help you check which files you have successfully submitted. It is your responsibility to make sure you properly turned in your assignment.

**Files to be collected:**
- objectdraw.jar
- Acme.jar
- CrazyOrb.java
- CrazyOrbGUI.java
- README

**Additional files to be collected for Extra Credit:**
- EC_CrazyOrb.java
- EC_CrazyOrbGUI.java

**NO LATE ASSIGNMENTS ACCEPTED!**

**START EARLY!**

**HAVE FUN!**

And above all ...