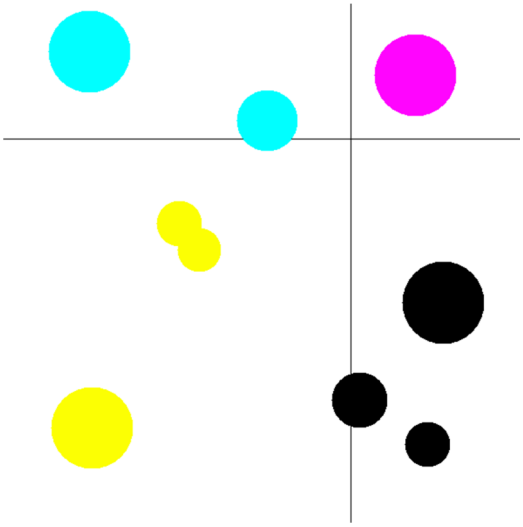
Programming Assignment 4 (100 Points)

Due: 11:59pm Thursday, October 20

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 cosmic spheres with cosmic 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 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)

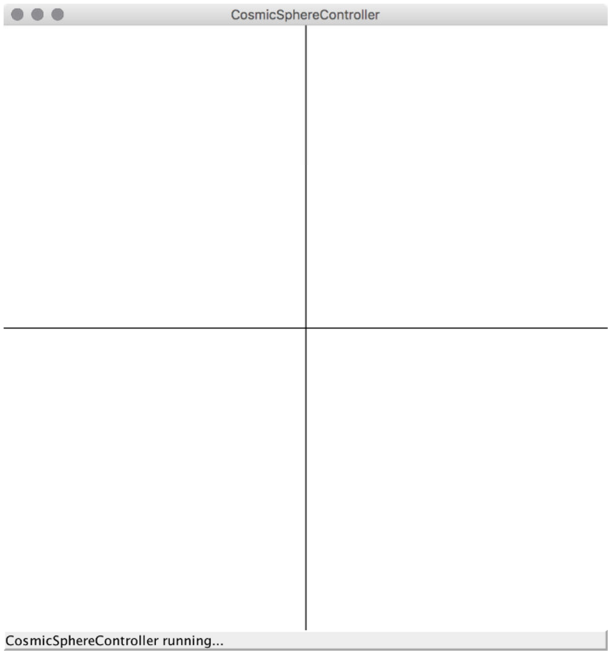
Setting up your pa4 directory:

Please see previous programming assignments.

You will need the **objectdraw** and **Acme** libraries for this assignment.

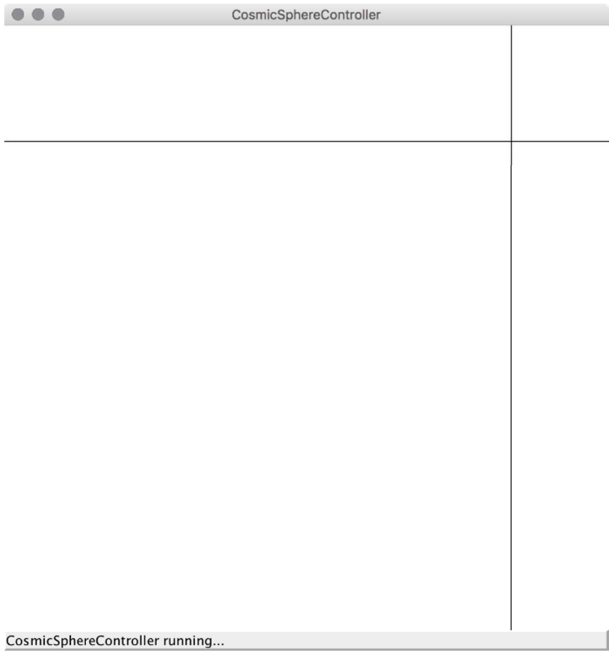
You will be creating two files: **CosmicSphereController.java** and **CosmicSphere.java**. As in previous assignments, the controller class must extend WindowController since it will be handling all the mouse movements and user interaction.

Stage 1: Creating Quadrants



Begin with the controller class (CosmicSphereController) 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 5 pixels from the edge of the canvas in any direction (a 5 pixel margin).**

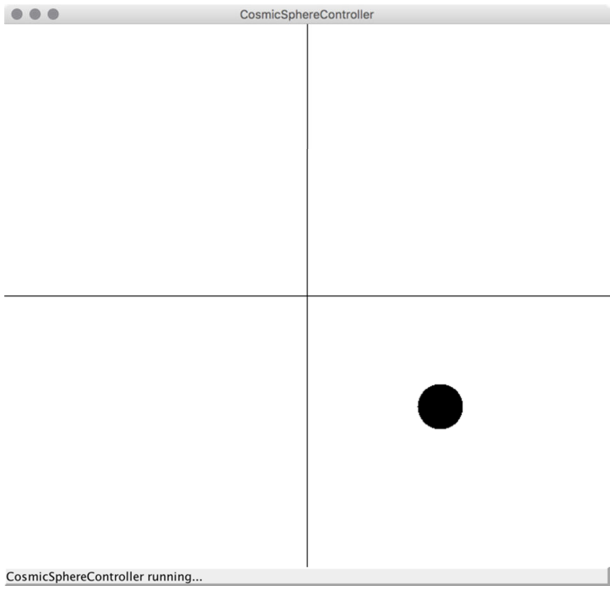
The second manipulation you should implement is keeping the Lines proportional when adjusting the size of the window. If you change the size of the window, the Lines should move to preserve their current proportions on the screen. To redraw the canvas, override the paint() method:

```
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 );
```

You can then add in your code for repositioning the Lines based on the proportions of where they were previously.



Stage 3: Creating the Cosmic Spheres

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

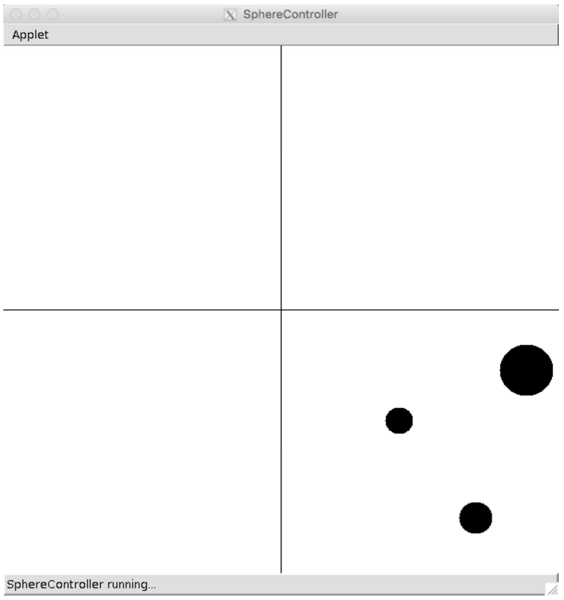
The constructor should look like this:

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

The first two parameters are the x and y coordinates of the <u>center</u> of the cosmic sphere (where the mouse was clicked). The third parameter is the starting size (diameter) of the cosmic sphere (50 pixels is a good starting size). The last two parameters are the horizontal and vertical Lines used to divide the canvas into different quadrants so this new CosmicSphere knows which quadrant it is being created in and can keep track of those Lines for later use.

For now, the constructor should just create and display the cosmic sphere. Remember: the FilledOval 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 CosmicSphere is an ActiveObject, so you need to call the start() method as the **<u>last</u>** line of code in this constructor.



Stage 4: Cosmic shrinking and growing

You should add a **run()** method with a forever loop to your CosmicSphere class to deal with the cosmic animation of the cosmic spheres. The diameter of the cosmic sphere should grow by 2 pixels each step to make it a smooth transition. You should pause for 50 milliseconds in each iteration so the spheres won't grow and shrink too quickly. Once the sphere grows to twice its starting size, it should start to shrink. Once the sphere reaches half its starting size, it should start to grow. **The center of the CosmicSphere needs to remain in its original location.** The screenshot shows multiple spheres shrinking and growing at the same time.

Note: because the CosmicSphere 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 cosmic sphere is created at the end of the constructor. (See Ch 9 and the lecture notes on Active Objects. We will go over all of this in class.)

Stage 5: Different Colors for Each Quadrant

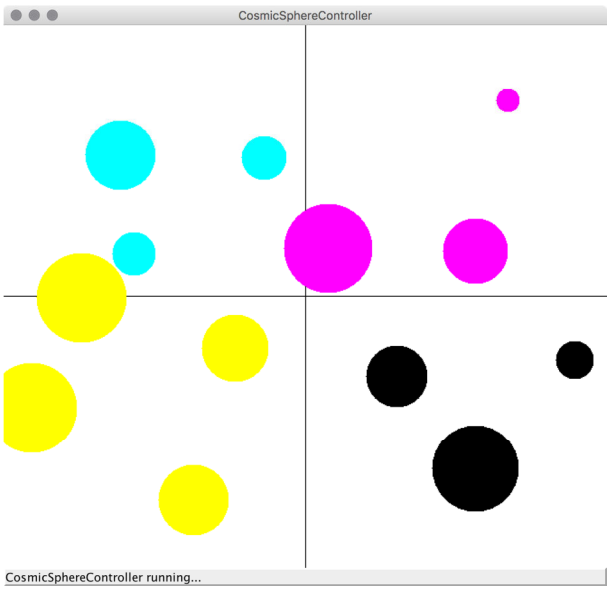
The cosmic sphere that is created in each quadrant (centered at the point of the mouse click) will be a particular color based on the quadrant: upper-left – **Cyan** (Cyan); upper-right – **Magenta** (Magenta); lower-left – **Yellow**(Yellow); lower-right – **Black** (Cyan/Magenta/Yellow/Black - CMYK - the printer colors).

Use the constants defined in java's Color class. (Look at the javadocs).

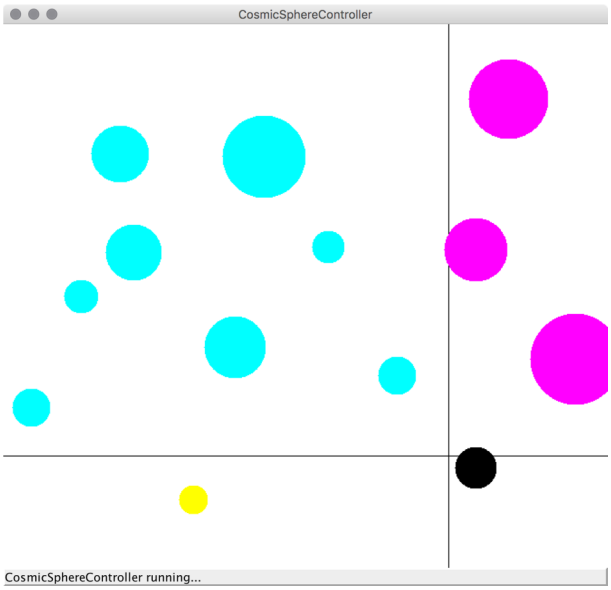
You should set the color of each cosmic sphere based on which quadrant the center of the cosmic sphere is located in at any particular time. Use the two Lines passed into the constructor and the cosmic sphere's center to determine which quadrant the cosmic sphere is in.

If the center of the sphere is exactly on the **vertical** line, consider it to be to the **right** of the line (use < logic). If the center of the sphere is exactly on the **horizontal** line, consider it to be **below** the line(use < logic). If the center of the sphere is exactly at the intersection of both lines, consider it to be bottom right quadrant (use < 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 cosmic sphere needs to check which quadrant it is in and update its color if necessary. Note: the cosmic spheres do not move when the applet resizes.



Before dragging the lines.



After dragging the lines.

EXTRA CREDIT (5 points): Psychedelic Cosmic Spheres

Getting Started:

For extra credit, you will be adding some features to your Cosmic Spheres, so before you start, make sure your Cosmic Spheres are working perfectly.

Once your Cosmic Spheres are working perfectly, make a copy of your source files like so:

```
$ cd ~/pa4
$ cp CosmicSphere.java EC_CosmicSphere.java
$ cp CosmicSphereController.java EC_CosmicSphereController.java
```

By using the previous unix commands, you are creating a copy of CosmicSphere.java and CosmicSphereController.java and naming those copies EC_CosmicSphere.java and EC_CosmicSphereController.java. ALL 4 FILES ARE NEEDED IF YOU ATTEMPT THE EXTRA CREDIT.

After this, make sure to change the class definition in EC_CosmicSphere.java and EC_CosmicSphereController.java to the new class name. For example:

```
public class EC_CosmicSphere extends ActiveObject {
  ...
}
```

Main Idea:

You will need a psychedelic toggle that you can switch on and off. When the first sphere is created, the toggle must be off (no matter what). Once at least one sphere is created, the toggle is switched every time the mouse enters the canvas. The following chart shows what needs to happen when the toggle is switched on and off.

Psychedelic Toggle Off	Psychedelic Toggle On
● Cosmic Spheres are FilledOvals ● Quadrants correspond to these colors: ○ Upper left - Color.CYAN ○ Upper right - Color.MAGENTA ○ Lower left - Color.YELLOW ○ Lower right - Color.BLACK ● Background color of canvas is **white** ● Horizontal/vertical lines are **black**	● Cosmic Spheres are FramedOvals ● Quadrants correspond to these colors: ○ Upper left - Color.RED ○ Upper right - Color.GREEN ○ Lower left - Color.BLUE ○ Lower right - Color.PINK ● Background color of canvas is **black** ● Horizontal/vertical lines are **white**

There are multiple approaches to this problem, so feel free to be creative. One way you could accomplish this is to change your `FilledOval sphere` variable to be a `Resizable2DInterface sphere`. This way at runtime you can initialize the sphere as either a new FilledOval or a new FramedOval.

In order to change the background color of the canvas, remember you need to cast it as a JDrawingCanvas (as in the extra credit for PA2) like so:

```
((JDrawingCanvas) canvas).setBackground( Color.BLACK );
```

NOTE:

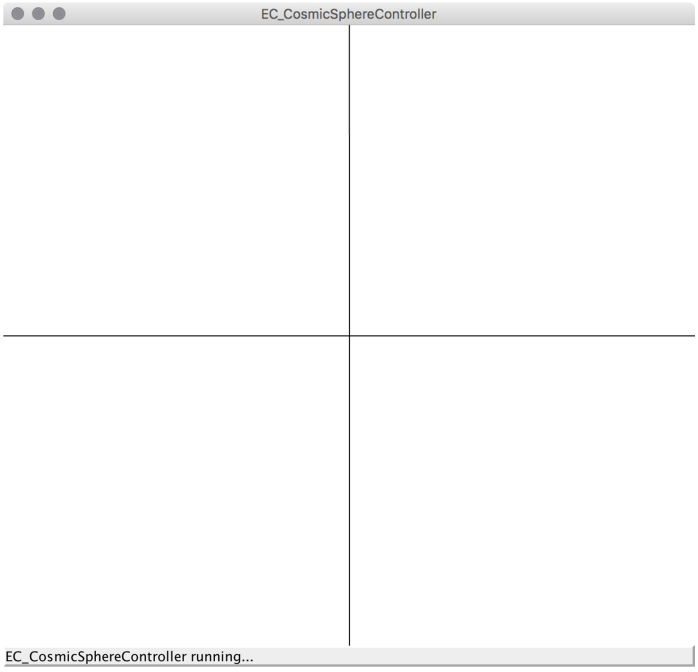
Just like the non-extra credit part, when the lines move:

- The lines redefine the quadrant areas
- The psychedelic cosmic spheres need to possibly change color (no matter if the spheres are currently FilledOvals or FramedOvals)

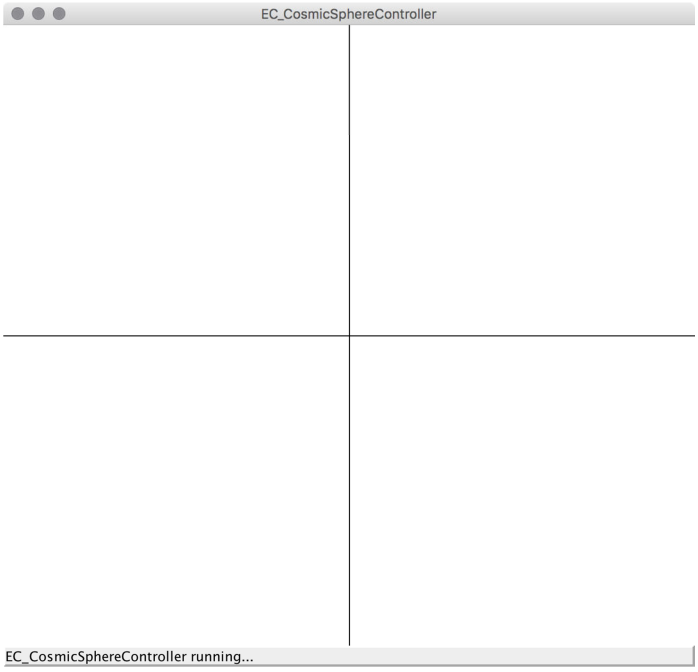
When adjusting the window size:

- 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 cosmic spheres stay where they are when the applet resizes.

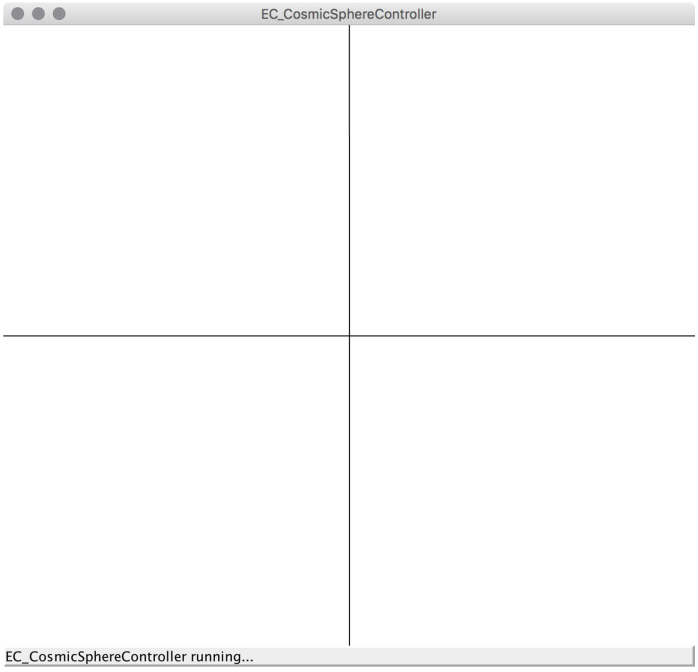
Sample Screenshots:
(Showing a sequence of actions going from left to right, top to bottom)



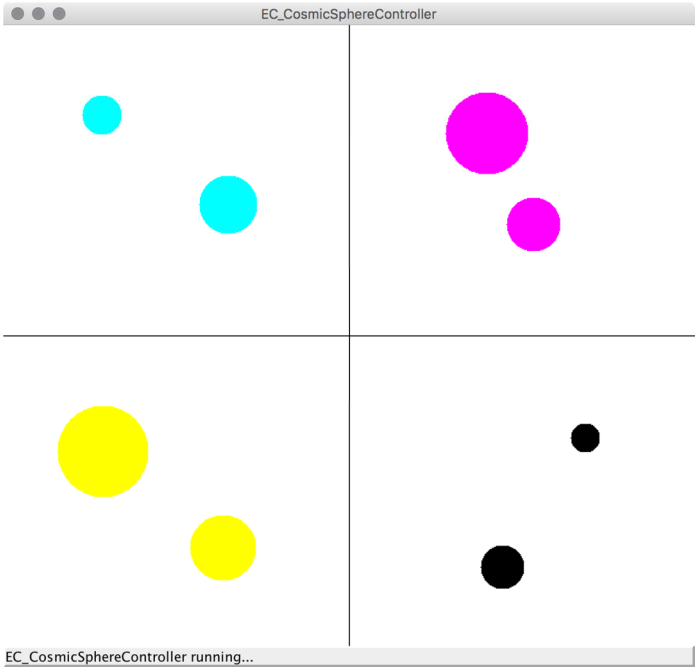
Upon startup.



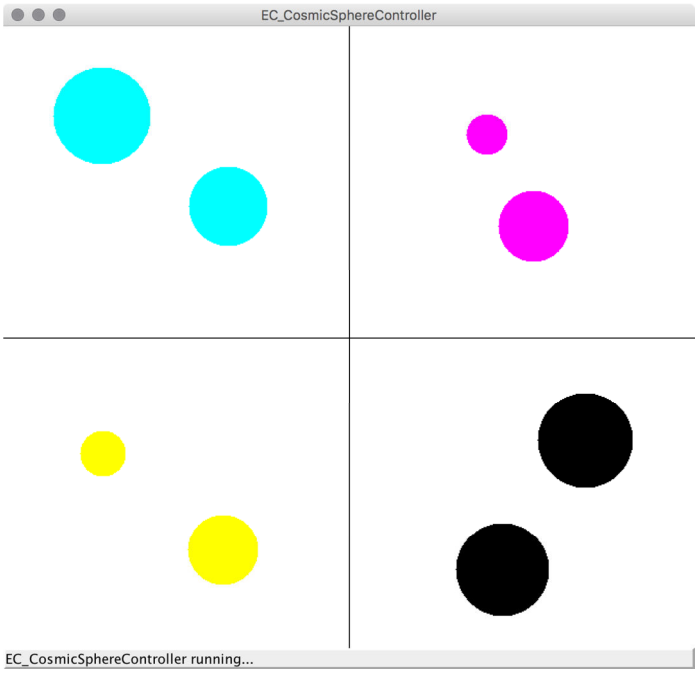
Mouse exit.



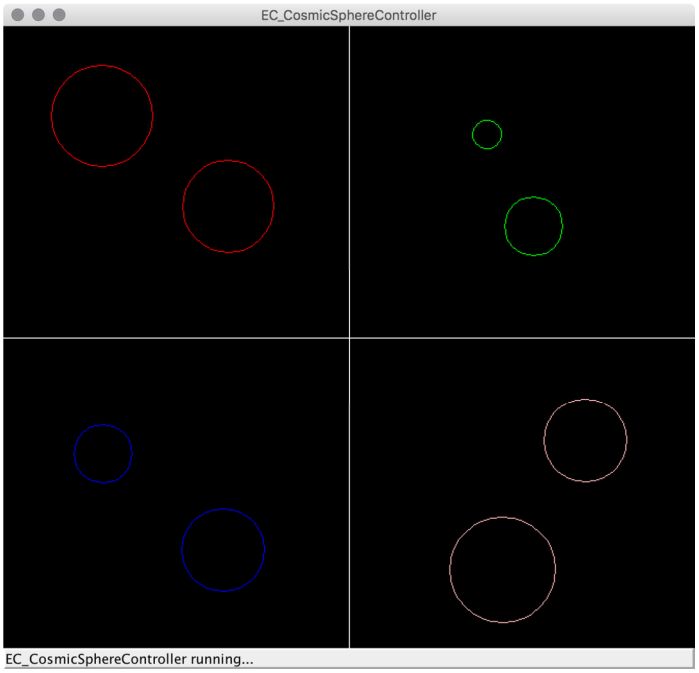
Mouse enter → psychedelic mode is still turned off because no spheres have been created yet.



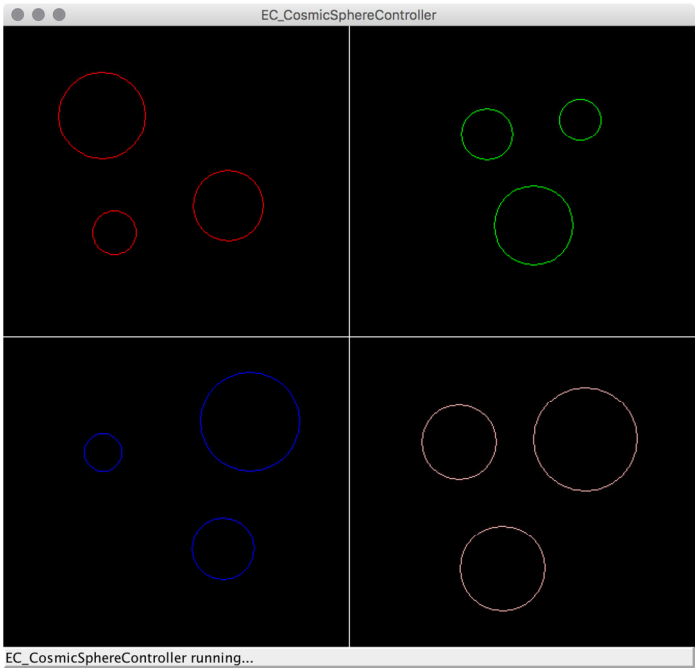
Initial group of spheres are created without the mouse exiting the canvas.



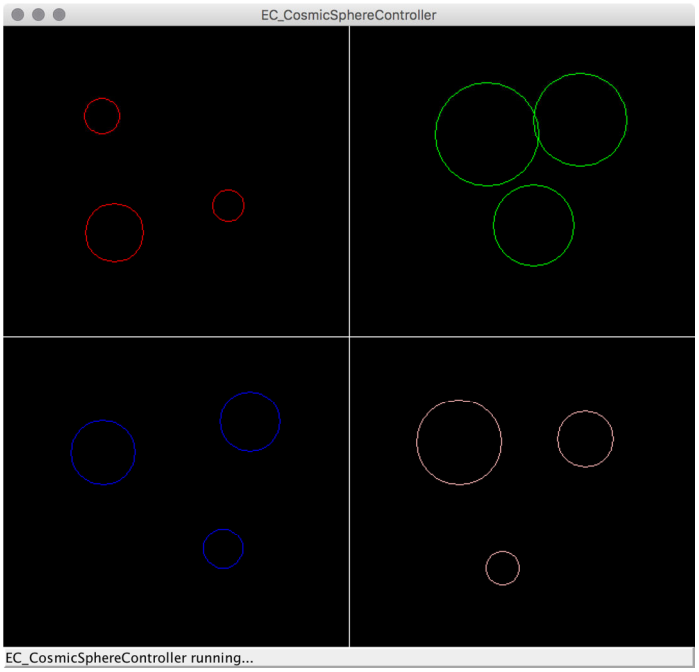
Mouse exit.



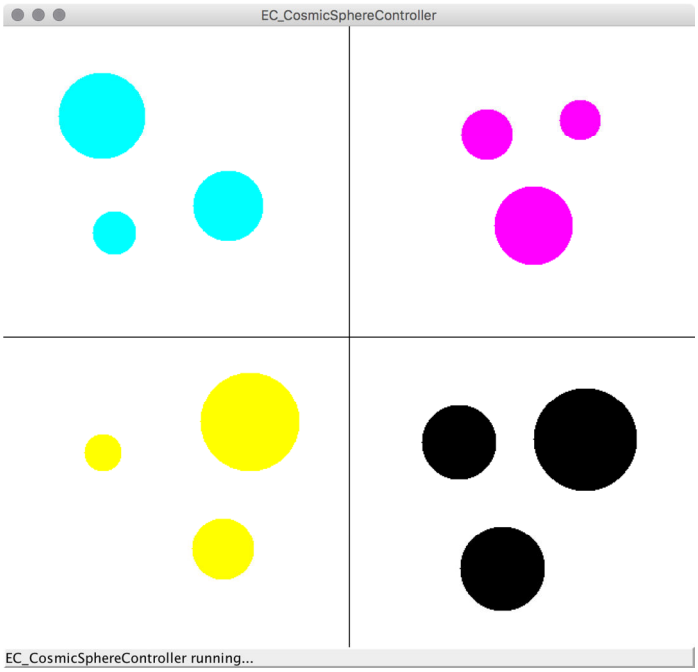
Mouse enters → psychedelic mode turned on.



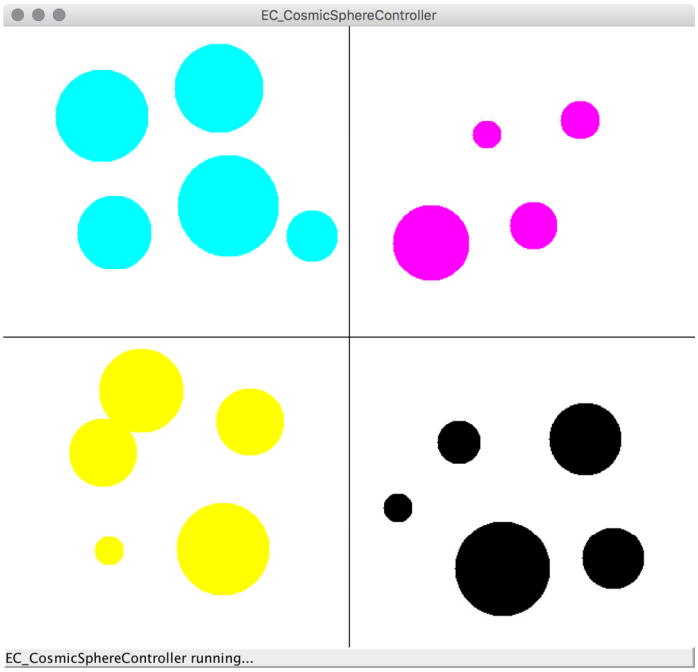
Creating more cosmic spheres in psychedelic mode.



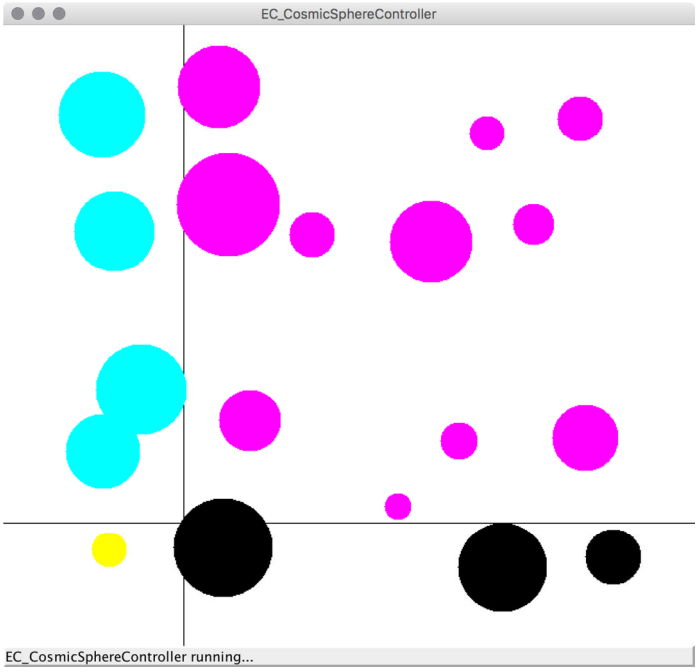
Another mouse exit.



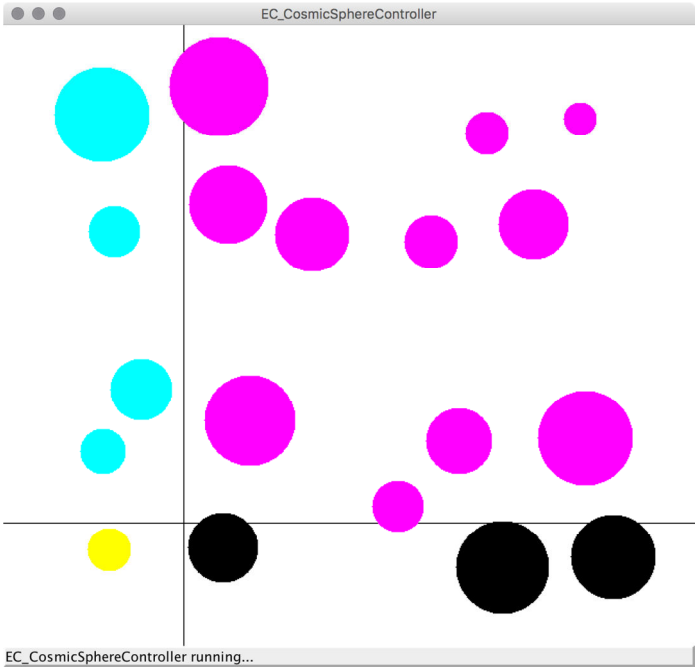
Mouse enter → psychedelic mode turned off.



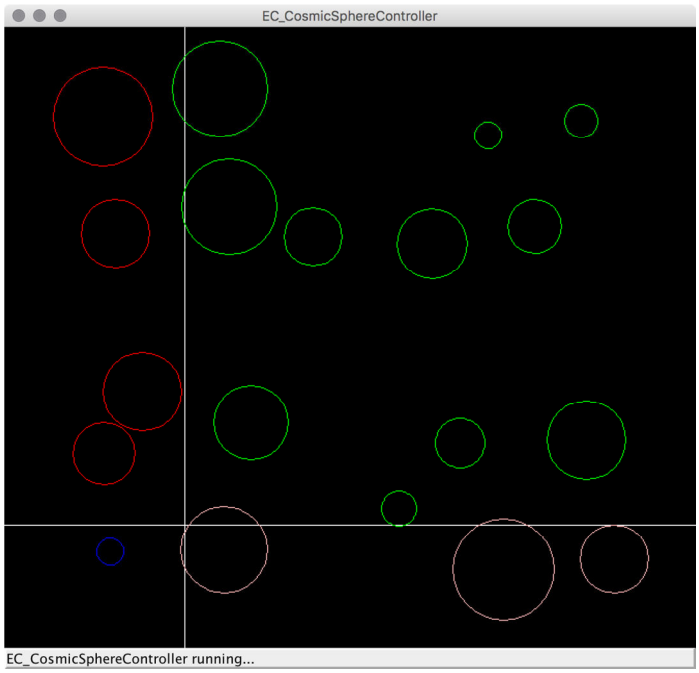
Creating more cosmic spheres in normal mode.



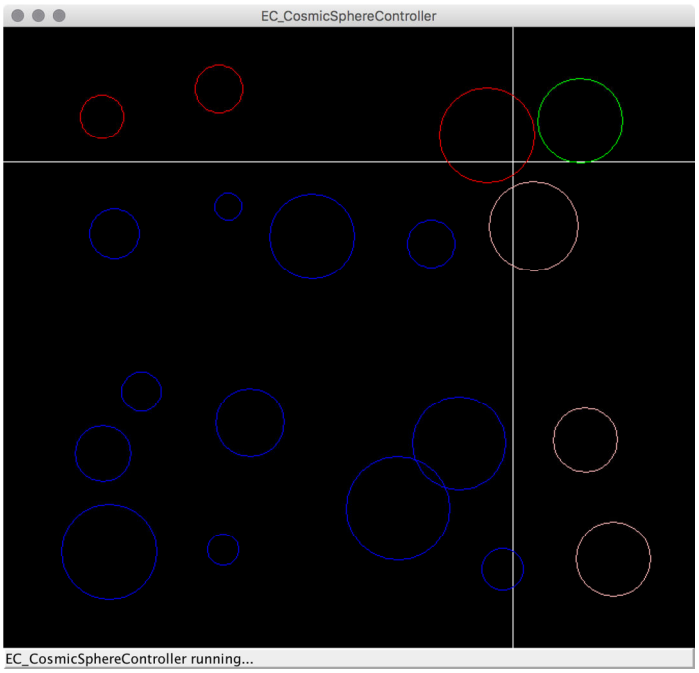
Dragging lines.



Mouse exit.



Mouse enter → psychedelic mode turned on.



Dragging lines.

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
- CosmicSphere.java
- CosmicSphereController.java
- README

Additional files to be collected for Extra Credit:

- EC_CosmicSphere.java
- EC_CosmicSphereController.java

NO LATE ASSIGNMENTS ACCEPTED!

START EARLY!

And above all … # HAVE FUN!