Lab 11.5 :: Introduction to Graphics
The basics of the Object Draw Package.

Pre-amble:
ObjectDraw, like Karel J. Robot, is intended to be used as a strategy to for teaching new programmers how to program in Java. Since you already know the basics, during these labs you should focus your attention on how the Graphics components work. You will also get good practice reading JavaDoc-style documentation. The examples are taken mostly from a text on ObjectDraw and are chosen for their graphical elements, not their programming concepts.

Much of the work you will do here is exploratory and experimental. Most of the answers to questions can be found either through common sense, or reading the documentation. You and your partner need to be persistent and systematic about how you figure things out.

The dual goals are to:
1. Learn something about graphics
2. Learn how to research answers to your java-related questions using the online documentation. (As we move further, Franke will provide answers to questions about Computer Science, and you will be expected to answer your own questions about Java).

Operating Notes
these apply (I think) to all graphics programs you will write with ObjectDraw:

1.) You may be directed to copy example code that exists on the web for ObjectDraw. IN ALL INSTANCES when you see a class header that’s something like:

    public class SomeClassName extends WindowController

you should modify it to read:

    public class SomeClassName extends FrameWindowController

This will allow the programs to work in BlueJ.

The sample programs can be found at:
http://eventfuljava.cs.williams.edu/sampleProgs.html
(**Editor’s note: BOOOOOO Williams.**)

2.) To get the programs to work in BlueJ you have two options:
   1.) Create a new instance of a class that extends FrameWindowController on the blueJ object bench. When you construct an instance of FrameWindowController a graphical window will automatically pop up. However, it will open to a default size. If you want to change the size of the window you need to...
   2.) Write a main method like the following (using the SomeClassName example from above):
public static void main (String[] args){
    (new SomeClassName()).startController(640, 480);
    ...
}

Even this is something of a workaround. You will see two windows pop up, one the default window, and other the size which you set when you run startController(…).

3.) DO NOT try to run the program as an applet. BlueJ does not give applets access to imported libraries in some instances. So, you’re stuff won’t work if you do.
PROBLEM 1 (Getting started):

Copy and paste the code for TouchyWindow from the ObjectDraw sample programs website (listed above). Also make the changes specified in the “Operating Notes” above.

1.) Create an instance of TouchyWindow on the Object Bench to make sure the program works as designed. (Or run the main method if you wrote one).

2.) Modify the coordinates of the text that appears. Through experimentation, try to get the text to appear in the top left, bottom right, and center of the window. Anything funny about the coordinate system?

3.) The on MousePress method has a parameter of type Location, but the code, as yet, does nothing with it. In fact, the argument passed to the “point” parameter represents the point where the user clicked the mouse relative the window. Modify your code so that the text will appear at the point where the user clicked the mouse. You’ll have to look at the online documentation for the Location class to find the methods that allow you to get the coordinates out of the Location object.

4.) on MousePress and on MouseRelease are methods inherited from FrameWindowController which inherits them from WindowController. What other mouse-related methods do we inherit from WindowController? Note the difference between onMouseDrag and onMouseMove. Change on MousePress/Release to other mouse event methods to make the program do something different.

5.) Read the documentation for the Line class. Compare it with the documentation for the Text class. Modify your code so that when the text is drawn it is underlined with an instance of the Line class.

6.) There are several classes of basic shapes provided by ObjectDraw. You can draw lines, rectangles, rounded rectangles, ovals, and arcs. Each class (with the exception of Line) has a “framed” version, meaning it just draws an outline of the shape, and a “filled” version which makes the shape into a solid color. Thus, FilledRect is the class that will draw a solid rectangle on the canvas.

Experiment with a few Framed and Filled Objects (I know you want to see different colors, but hold off for now, that’s coming soon) and see if you can modify your code to get your line of text to appear on a sign post like the one shown.
7.) Color. Each shape has a method that allows you to set its color after you construct it. In Java, Color, like everything else is represented as a class. Your computer screen is an array of lighted pixels, with each pixel containing a certain saturation of Red, Green and Blue light. R,G and B can be combined to make almost any color. Thus, the \texttt{Color} class allows you construct a color with the level of R,G, and B as its construction parameters. The amount of each of R,G, and B must range, in decimal, between 0 and 255. Some examples are below:

\begin{verbatim}
new Color(255,0,0): //red
new Color(255,255,255); //white
new Color(0,0,0); //black
\end{verbatim}

You can look up the Color class in the Java API for more information. NOTE: making color by mixing light is the OPPOSITE of making color by mixing paint.

Modify your sign-post so that there's some color in it somewhere.

8.) Finally, you should know that \texttt{FrameWindowController} has a method called \texttt{void begin()}. The body of the begin method should contain things you want to appear in the Window before any action or events occur (it’s sort of like a constructor…sort of). The \texttt{FrameWindowController} constructor, does a lot of behind the scenes setup to create the window object and then it calls the begin method. Modify your program so that the sign post appears by default at the beginning and the text appears in it when the user clicks the mouse. If you have a more creative way to use the begin method you may do that instead.

End of Problem I.
Problem II :: Persistent behavior.

To keep objects around for a while you will need to declare private instances of the object – that way you can refer to them over time.

Example 1: Rising Sun
1. Within the same BlueJ project that you did your last work in, create a new class called RisingSun and start it out like this:

   ```java
   import objectdraw.*;
   import java.awt.*;
   public class RisingSun extends FrameWindowController{
     private FilledOval sun;
     public void begin(){
       //...method body not shown...
     }
     //...other methods not shown...
     public static void main(String[] args){
       (new RisingSun()).startController(200,300);
     }
   }
   ``

   Write code into the begin method that:
   • initializes sun as a 100x100 FilledOval that’s yellow, and sits halfway off the bottom of the window (i.e. the top half of the oval is visible).
   • Prints some text that instructs the user to click the mouse repeatedly.

   Test this to make sure it works.

2. In ObjectDraw all of the things that can be drawn onto the screen implement DrawableInterface. If you look to the documentation you’ll see that all drawable objects have a number of helpful methods. Modify your code so that when the user clicks the mouse:
   • The text “hides” (to do this, you will need to make the Text object a private member variable as well).
   • The sun “moves” up by 5 pixels.
   When you’re done, the user should be able to repeatedly click the mouse and watch the “sun” move up in the window.

3. Modify your program once more so that the when the mouse “exits” the screen, the sun resets to it’s original location, and the instructions reappear.
4. Add another private FilledRect that is the size of the canvas (if you look at the
*DrawingCanvas* interface you’ll see it provides methods for getting the current size of the
window) so that it will serve as a background color. Make it red, and create it so that the
“sun” appears on top of it. Now, when the user clicks the mouse, as the sun goes up,
change the color of the background rectangle so that it adds a little bit of green. (The
idea is to light up the sky as the sun rises. If the color starts as red (255,0,0) you can add
a little bit of green until you get to yellow (255,255,0)).
Problem III – a small challenge

Within the same BlueJ project you’ve been working in, write a program that draws a line between the point where the user clicked the mouse and the point where they released the mouse. You should annotate the drawing with text “pressed” and “released” to indicate which point is which. See diagram to right. Hint: you’ll need to need to keep track of a Location privately.

Problem IV – a bigger challenge (Slightly)

Write a program that allows the user to “scribble” on the canvas. HINT: You do not need to do anything particularly fancy to do this: no arrays of points or clearing the canvas repeatedly. Utilize onMouseDrag, and keep track of one previous location. You can make it fancier by allowing the user to select a color to draw with, and/or allowing the user to “draw” with different thicknesses.
**Part II – Animation**

**Notes on using Threads with ObjectDraw**

To make an object that runs in a Thread:
1. Extend ActiveObject
2. Set up the properties of the object in the constructor and be sure to make a call to `start();`
3. implement `public void run()` and remember the following:
   - `run()` will execute once (but you can’t be exactly sure when)
   - typically the run method contains a while-loop that keeps the thread going
   - Be sure that this while-loop has a stopping condition
   - If your threads don’t stop you should reset the JVM.
4. If you’re doing animation you will need to invoke the `pause(...) method within the run method in order to put the thread to “sleep” long enough for the screen to redraw itself.
   If you pause too long, however, your animation can look jerky. You will have to experiment to find a happy medium – the industry's

Explanation:
When you tell an Active Object to “start()” you’re giving control of the execution over to the JVM. The JVM will manage when and how each thread is executed and it will try to execute each thread’s run() method until completion. However, what the JVM really does, is execute a little bit of the `run()` method in each thread repeatedly. How this is done is hidden from you. Just make sure that your run method will stop SOMEHOW.

**Onto making stuff...**

**Problem 1: Starting out**
Use the Falling ball example we developed in class to guide your work.

- Make a new class called NaturalBall that extends active object. It’s constructor should be very similar to the FallingBall example except it should start at different location: the location should be the X-coordinate of the user’s mouse click and just above the bottom of the canvas.
- You should also declare a private int member variable called `yChange` and initialize it to -25.
- The run method of NaturalBall should have a loop that executes as long as the ball is above the bottom of the canvas.
- The body of the while loop should move the ball by 0 in the x-direction and `yChange` in the y-direction. After moving the ball, increment `yChange`.
- When you run this, the effect should be a ball looking like it naturally rises and falls. Get this to work before moving on.

**Problem 2: Make the ball bounce**
Making the ball look like it’s bouncing off the floor is tricky because you’ll have to change the looping condition since the ball will not fall below the screen.
To make the ball bounce, do the following 3 things:
1. Inside your loop add an if-statement that detects the ball is at the bottom of the canvas. Inside the if-statement, reset yChange back to -25, and re-position the ball just above the bottom of the screen (you need to reposition the ball because it won’t come down exactly where it started).

2. add a new member variable called bounceCount and increment it every time the ball hits the floor.

3. Change your looping condition so that the ball bounces 10 times before the loop ends. Now the balls will come to a rest at the bottom of the canvas.

**Problem 3: Add movement in the x-direction**
Make a new member var called xChange and initialize it to 10.

Modify your ball movement so that it will move in the x-direction by xChange.

Add an if-statement inside the while loop that detects if the ball has hit the left or right wall (remember to account for the width of the ball when figuring out the right wall). If the ball touches the wall, “flip” xChange by multiplying it by -1. (i.e. xChange *= -1).

If you want to get fancier, initialize xChange to a random number between some negative value and some positive value.

**Problem 4: Making it more natural**
A ball really doesn’t bounce to the same height every time. Devise a way for the ball to bounce a little less high every time it bounces – it will require the use of more variables.

After you’ve done that, change your looping condition so that you quit the loop when the ball actually stops bouncing.

**Problem 5:**
By this point you have your own ideas for what the ball can do. Try to execute one of them. Ideas: colored balls, different size balls, balls growing and shrinking as they rise and fall, balls compressing when they hit the ground…