1 Introduction

For the first lab you used graphics written by TAs to do the assignment. Now it's time for you to get your hands dirty and start coding things without support code.

1.1 Brief History of Swing

A long long time ago in a galaxy far far away, AWT was created for Java so that people could write user interfaces and not have to write their own 3rd party packages. AWT stands for Abstract Windowing Toolkit, and it was just that: way too abstract. Most people found it not very useful. So in response, the people at Sun created Swing. Swing came from the project's code name which was Swing Set. It's actually largely built off of AWT but provides a lot more concrete functionality. Swing is included with the Java Standard Development Kit available online from Sun.[1]

2 Getting Started

For this assignment you will write a simple GUI with a bouncing ball. The ball should bounce around the screen and be able to have its color changed by at least three radio buttons.

The code is available in `/course/cs032/pub/labs/02_swing/src`. Expect to have anywhere from 2-5 java files.

2.1 Coding Tips

This assignment should be done incrementally. First get a window to show up. Then try adding in the buttons. Try getting the buttons to respond to mouse clicks. Get a circle to show up in the panel. Create a timer to get it to move, etc. Doing it in steps will decrease the amount of debugging you have to do and make learning Swing go smoother.

3 Specifics

There are a few parts to Swing that aren't very intuitive to understand and may also be easily forgotten. For those of you who have never used Swing or need a refresher, the rest of this handout will describe the basic tools you'll need to use to write this program.

3.1 Let's start at the beginning - JFrames

The Swing and AWT packages are in java.awt.* and javax.swing.*. You can use either the 1.4 javadocs or the 1.5 docs for this lab (both are linked off the 32 website under the docs section). Open the docs up in a browser now.

Navigate to javax.swing.JFrame. This is the main component of most Swing programs. JFrames provide the window of the program itself. Look through the docs to see how to construct a JFrame.

JFrame's mostly contain one object of significance - the content pane. The content pane will be the panel that all your objects are contained in. To add a content pane to a JFrame you can use `setContentPane(Container contentPane)`. But just what do you want to add as a content pane? We'll get to that in a moment...

[1] http://java.sun.com/j2se/downloads/index.html

Two more important methods you should know about JFrames: `setVisible()` and `pack()`. You'll need to call `setVisible(true)` on your JFrame if you want your program to appear, which is usually important. `pack()` is a convenient method that resizes all the components within the JFrame, including buttons, panels, etc.

3.2 The next step - JPanels

Navigate the docs to `javax.swing.JPanel`. JPanels are used to further group and organize components within a JFrame. You'll want to create and add a JPanel as the JFrame's content pane.

You use JPanels to add components such as buttons or sliders to your GUI. To do so, simply call `add(java.awt.Component method)` on the desired JPanel. You can add pretty much anything to a JPanel, even other JPanels (which is actually done quite often). Other handy methods dealing with JPanels include:

- JPanel(LayoutManager layout, boolean isDoubleBuffered) - This constructor not only allows you to pass in a LayoutManager (see next topic) through the constructor, but it also allows you to turn on double buffering. Double buffering is a system in which graphics are first rendered off screen before they are displayed. Basically what this means is things look a lot smoother and nicer. We recommend experimenting with it for this lab.
- setBackground(java.awt.Color c) - sets the color of the background
- setSize(java.awt.Dimension d) - sets the dimension of the JPanel. setSize can also take two integers.
- setPreferredSize(java.awt.Dimension d) - sets the preferred dimension of the JPanel. Usually it's good to call both `setSize()` and `setPreferredSize()`

But now you ask: how do you use Swing to arrange all these components in a logical and visually appealing manner? Read on young pupil...

3.3 Organizing it all - Layout Managers

You could hard code your layout down to the last pixel, but after that you'd probably want to tear your eyes out. The creators of java knew this so they created `java.awt.LayoutManagers`. LayoutManagers arrange the components inside `javax.swing.JPanels` to look nice. Before you add a component to a JPanel, you'll want to make sure it has the LayoutManager you desire. To set the LayoutManager of a JPanel, call its `setLayout(LayoutManager layout)` method or you can pass it a LayoutManager in its constructor.

There are two types of LayoutManagers we sugguest you use for this assignment. The first is `java.awt.FlowLayout`. If you construct a JPanel with no parameters, it will have a FlowLayout by default. It arranges the contents of the JPanel from left to right, top to bottom (like writing).

The other basic LayoutManager we sugguest you use for this assignment is `java.awt.BorderLayout`. BorderLayouts have 5 regions, NORTH, SOUTH, EAST, WEST, CENTER. The regions will rezise to fit their contents automatically. The default size of each region is (0,0). You do not have to fill each region in order to use a BorderLayout. To add a component to a JPanel with a BorderLayout, you call the `add` method on the JPanel, passing it the component and the region name you would like to add it to.

Example:

```
add(quitButton, java.awt.BorderLayout.SOUTH);
```

To allow for absolute positioning, you may set a JPanel's LayoutManager to null. This is usually not what you want.

You have now learned enough to get a simple window to show up. This is CHECKPOINT 1.

3.4 Swing Event Model

The Swing event model is a callback system. You first need to associate a listener with a component. The component will add that listener to its list of listeners. Then when an event occurs (i.e. a mouse click) the component will call the approprate method on its listener.

For buttons (`javax.swing.JButton`) and timers (`javax.swing.Timer`) you will use `java.awt.event.ActionListener`s. The ActionListener interface has only one method you must define which is:

`actionPerformed(java.awt.event.ActionEvent e )`.

To add the listener to a component, simply call the `addActionListener( ActionListener )` method on the component. For this assignment you'll only need them for `javax.swing.JButton`s (or `javax.swing.RadioButton`s) and `javax.swing.Timer`. It's customary for most ActionListeners to be inner classes within the class that will contain the component (i.e. the top level panel).

Example - A complete program that's just a simple frame with a quit button inside. Just copy/paste this into a file named Quitter.java and try to compile and run it.

```
public class Quitter extends javax.swing.JFrame {
   public Quitter() {
      super("Title message");
      javax.swing.JPanel cp = new javax.swing.JPanel();
      javax.swing.JButton quit = new javax.swing.JButton("Quit");
      quit.addActionListener(new QuitListener());

      cp.add(quit);
      this.setContentPane(cp);

      this.pack();
      this.setVisible(true); //aka this.show();
   }

   private class QuitListener implements java.awt.event.ActionListener {
      public void actionPerformed(java.awt.event.ActionEvent _e) {
         System.exit(0);
      }
   }

   public static void main(String [] args){
      new Quitter();
   }
}
```

You now have enough skillz to get four buttons to show up that respond to mouse clicks. This is CHECK-POINT 2.

3.5 Simple 2D Graphics in Swing

This is probably one of the most confusing parts of Swing. In order to draw shapes in Swing you need a `java.awt.Graphics2D`. It is in charge of painting things to the screen and has a lot of methods for drawing shapes and images. The methods you'll be interested in are:

- `setColor( java.awt.Color )` sets what color the Graphics2D will draw with
- `drawOval( int x, int y, int w, int h )` draws an oval outline to the screen using the color last specified by setColor
- `fillOval( int x, int y, int w, int h )` fills an oval on the screen using the color last specified by setColor

- `draw( java.awt.Shape )` draws any Object which implements the Shape interface using the color last specified by setColor
- `fill( java.awt.Shape )` fills any Object which implements the Shape interface using the color last specified by setColor

It is sugguested that you instantiate or subclass the `java.awt.geom.Ellipse2D.Double` (which implements the Shape interface) to use for your ball, but you may also simply use the drawOval and fillOval methods as they do the same thing. Also keep in mind that to resize an Ellipse2D.Double you have to call `setFrame( double x, double y, double w, double h )`. There are no setSize() or setLocation() methods like there ought to be.

Now you're probably thinking you can just instantiate a Graphics2D and call methods on it, right? That would be way too easy. In this lab, we'll be drawing graphics on a JPanel. You'll need to PARTIALLY override the `paintComponent(java.awt.Graphics g)` method of the JPanel you're trying to draw the ball in. If you don't partially override paintComponent, the JPanel itself won't paint properly. Make sure you call `super.paintComponent(g)` BEFORE you start painting your stuff. If you do this later, the panel will just paint over what you painted.

There are still a few more quirks to work out. You may be thinking, "Shouldn't `paintComponent()` take a `Graphics2D`?" Actually, the Graphics class was originally created for AWT, and thus still remains for backward compatibility. The Graphics you get from paintComponent is actuallly a Graphics2D and you can safely cast it to Graphics2D whenever you need.

Next to last note. Since you can never call paintComponent directly you'll need some way to update the screen whenever a change has happened (e.g. the ball moves). Just call `repaint()` (no parameters) on the JPanel and it will schedule a call to paintComponent to take place.

Last note. For really smooth and snazzy graphics (anti-aliasing for you CS123ers), you need to turn on some flags of the Graphics2D. The code below accomplishes this feat of wonder:

```
myGraphics2D.setRenderingHint(java.awt.RenderingHints.KEY_ANTIALIASING,
                              java.awt.RenderingHints.VALUE_ANTIALIAS_ON);
```

You now should be able to get a circle to appear in a panel, which is CHECKPOINT 3.

3.6 Timing is everything

The last item your program needs is animation. This can be accomplished with the use of a `javax.swing.Timer`. Look at the docs for more info.

We've provided simple code for the bouncing ball below. Feel free to come up with your own formula if you think of something cooler. In this example ball is an instance of java.awt.geom.Ellipse2D.Double and v_x and v_y are its components of x and y velocity, respectively.

```
//get the width and height of the panel we're in
int panel_width = myContainingPanel.getSize().width;
int panel_height = myContainingPanel.getSize().height;

//check for collisions and update the velocities
if (ball.y <= 0 || (ball.y >= panel_height - ball.height) )
   v_y = -v_y;
else if (ball.x <= 0 || (ball.x >= panel_width - ball.width) )
   v_x = -v_x;

\\set the new location of the ball
ball.setFrame((ball.x+v_x)%panel_width, (ball.y+v_y)%panel_height, ball.height, ball.width);
```

If you've got a bouncing ball and buttons that change its color and quit the program, then you've reached CHECKPOINT 4.

4 Good News, Everyone!

Swing is a major pain in the ass, because you can't tell what your GUI will look like until you compile and run it. Additionally, this lab only covers 0.4% of Swing, and it will take you forever to learn and then use anything beyond what we covered here, like callbacks and GUI threads and what not. For more information, tutorials are available online[2] and books are available in the back of the SunLab.

Usually your TAs are on hand to answer questions that go beyond the scope of the course, but because Swing and AWT are so frustrating, we are going to introduce you to the wonderful IDE known as Netbeans[3], which has a built-in GUI Designer. If you've ever used Visual Basic or Visual C#, it will look very familiar to you. It looks like an ordinary drag-and-drop GUI designer, but this is no ordinary GUI designer! If you want, you have access to all of the underlying Swing and AWT code, in case you want your GUI to do some extra sexiness behind the scenes. It's pretty legit and it's available on all the Sunlab computers. Ch-check it out, y'all.[4]

[2] See Sun's own Swing Tutorial at: http://java.sun.com/docs/books/tutorial/uiswing/

[3] http://www.netbeans.org/

[4] This section was written by Josh Dawidowicz, who prefers C to Java and has hated Swing since he was 15 years old.

5 Checkpoints

1. 25 points TA Initials: _____
   - Get a panel to show up
2. 25 points TA Initials: _____
   - Add RadioButtons and a Quit button
3. 25 points TA Initials: _____
   - Get a circle to show up in the panel
4. 25 points TA Initials: _____
   - Make the circle move with a timer