VidWorks Entertainment New Developer Training Program
Tutorial #4

Advanced Swing Concepts

Layout Management

You have already seen two layout managers--Flow layout and Border layout--but I have yet to tell you what they are. With GUI applications, the user may resize the window, or perhaps the buttons will look different from platform to platform (although with Swing, the latter case is unlikely). Thus, specifying the absolute locations and sizes of a component can sometimes be problematic. A layout manager takes care of this for us by placing components in such a way that in most cases, it looks decent.

One concept to note is a component's "preferred size". This is the size the component would like itself to be. However, a layout manager may not honor this size and may set either the width or the height (or both) to suit the needs of the layout.

In this section, we will go over the Flow layout and the Border layout as examples. The rest of the layouts should then be simple to understand.

Flow Layout

This is the simplest layout. It always honors the preferred size of a component, and it simply places components from left to right, top to bottom (similar to how we read), centering all components in a row.

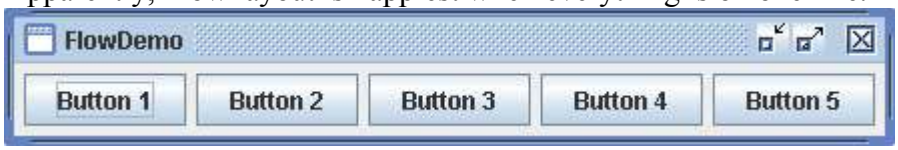
Since there are no special commands related to the Flow layout, we can jump right into an example:

```
/* FlowDemo.java */

import javax.swing.*;
import java.awt.*;

public class FlowDemo extends JFrame {
      public FlowDemo() {
            super("FlowDemo");
      }

      void createGUI() {
            setLayout(new FlowLayout());
            add(new JButton("Button 1"));
            add(new JButton("Button 2"));
            add(new JButton("Button 3"));
            add(new JButton("Button 4"));
            add(new JButton("Button 5"));
      }

      public static void main(String args[]) {
            JFrame.setDefaultLookAndFeelDecorated(true);
            FlowDemo fr = new FlowDemo();
            fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
            fr.createGUI();
            fr.setSize(300, 100);
            fr.setVisible(true);
        }
}
```

As a reminder, remember that since this is a *JFrame* and not an applet, we use "`java FlowDemo`". And here's our output:

When we try to resize it, the Flow layout automatically resizes it to the following. Apparently, Flow layout is happiest when everything is on one line.



Border Layout

This layout divides the frame into 5 portions: North, South, East, West, and Center. (Where they are should be self-explanatory, and if they aren't you'll understand after our example.)

Border layout does *not* honor preferred size. In the North and South regions, Border layout will stretch the components out horizontally. In the East and West regions, Border layout will stretch the components out vertically. In the Center region, the components will be stretched in all directions.

However, when we use Border layout, we have to use a special `add()` method:

`void add(Component comp, Object constraints)` - where constraints is `BorderLayout.NORTH`, `BorderLayout.SOUTH`, `BorderLayout.EAST`, `BorderLayout.WEST`, or `BorderLayout.CENTER`.

If you use the regular `add()` method, Border layout will add it to the center region by default.

If you add more than one component into the same region, only the last one added will be shown. It is not recommended that you do this, since the behavior of the hidden components may be unpredictable.

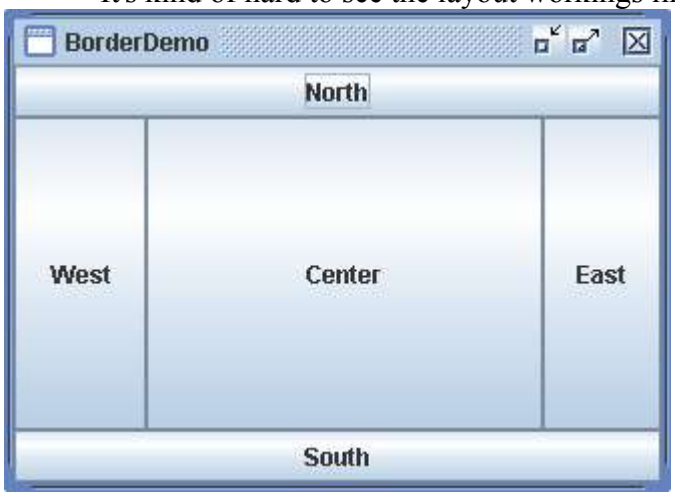
Note that for JApplets and JFrames, the default layout is Border layout, so we do not need to explicitly set the layout manager.

In any case, it's time for an example:

```
/* BorderDemo.java */

import javax.swing.*;
import java.awt.*;

public class BorderDemo extends JFrame {
      public BorderDemo() {
            super("BorderDemo");
      }

      void createGUI() {
            add(new JButton("North"), BorderLayout.NORTH);
            add(new JButton("South"), BorderLayout.SOUTH);
            add(new JButton("East"), BorderLayout.EAST);
            add(new JButton("West"), BorderLayout.WEST);
            add(new JButton("Center"), BorderLayout.CENTER);
      }

      public static void main(String args[]) {
            JFrame.setDefaultLookAndFeelDecorated(true);
            BorderDemo fr = new BorderDemo();
            fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            fr.createGUI();
            fr.pack();
            fr.setVisible(true);
      }
}
```

And this is what we get:

It's kind of hard to see the layout workings like this, so let's resize the thing:



Yes, those are buttons. Yes, the Center button is a little bloated. That's Border layout for you.

JPanels, again

Is there a way to put more than one component in a Border layout region? And is there a way to make those buttons not look so bloated? Yes. It's called JPanel. Last time, we used them to paint on, but there's a much better use for them.

The important thing to note is that a JPanel is both a component and a container. That means you can add components to it, and in turn, add it to a larger component, be it a JFrame, a JApplet, or another JPanel.

Oh, and the JPanel's default layout is Flow layout.

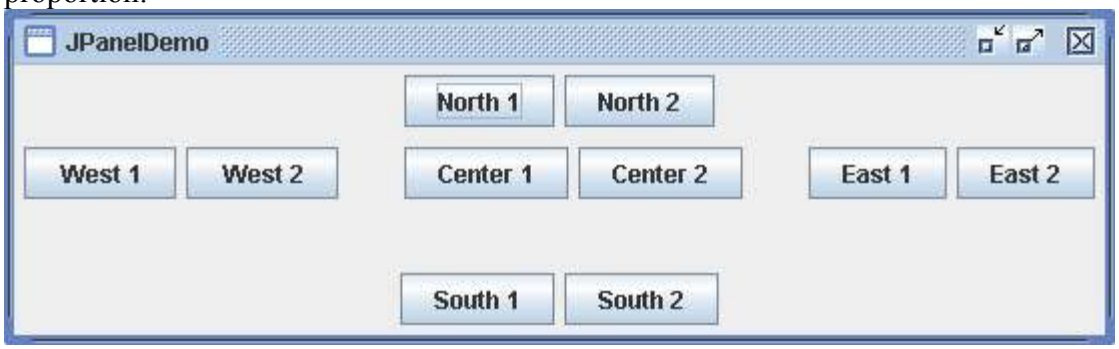
Now, for an example combining Border layout with JPanels:

```
/* JPanelDemo.java */

import javax.swing.*;
import java.awt.*;

public class JPanelDemo extends JFrame {
      public JPanelDemo() {
            super("JPanelDemo");
      }

      void createGUI() {
            JPanel nPanel = new JPanel();
            nPanel.add(new JButton("North 1"));
            nPanel.add(new JButton("North 2"));
            add(nPanel, BorderLayout.NORTH);

            JPanel sPanel = new JPanel();
            sPanel.add(new JButton("South 1"));
            sPanel.add(new JButton("South 2"));
            add(sPanel, BorderLayout.SOUTH);
```

```
            JPanel ePanel = new JPanel();
            ePanel.add(new JButton("East 1"));
            ePanel.add(new JButton("East 2"));
            add(ePanel, BorderLayout.EAST);

            JPanel wPanel = new JPanel();
            wPanel.add(new JButton("West 1"));
            wPanel.add(new JButton("West 2"));
            add(wPanel, BorderLayout.WEST);

            JPanel cPanel = new JPanel();
            cPanel.add(new JButton("Center 1"));
            cPanel.add(new JButton("Center 2"));
            add(cPanel, BorderLayout.CENTER);
      }

      public static void main(String args[]) {
            JFrame.setDefaultLookAndFeelDecorated(true);
            JPanelDemo fr = new JPanelDemo();
            fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            fr.createGUI();
            fr.pack();
            fr.setVisible(true);
      }
}
```

And here is our output:

Much better, yes? Now, let's resize it and see if anything gets bloated out of proportion:



Note that this also solves our JTabbedPane problem from tutorial #3, as suggested by the hint.

Other Layout Options

Here are some other layout managers defined by Java:

`GridLayout` - arranges components into a grid
`CardLayout` - arranges components like a stack of cards, only one being visible at a time
`GridBagLayout` - a more powerful version of GridLayout, where the grid sizes need not be uniform, and components can take up multiple grid spaces
`BoxLayout` - places all components into a row or a column, but components never wrap; this is similar to, but more powerful than, Flow Layout
`SpringLayout` - one of the newer layouts (introduced in 1.4); this powerful, but complicated layout works by defining the relationships between the edges of different components
And many others...

In addition, you can create your own layout by implementing the interfaces `LayoutManager` and/or `LayoutManager2`. However, this is an advanced topic and will not be covered here.

The last option... what if you don't want to use a layout manager? Well, then, set the layout managed to `null`, and you'll be forced to set the location and the sizes of all your components by hand. For those brave adventurers who would walk this path, here are some useful methods that you can call on your components:

```
void setBounds(int x, int y, int width, int height)
Dimension getPreferredSize()
Dimension getMinimumSize()
```

We leave these other layout options up to your own experimentation.

Look & Feel

You ever noticed why the windows and buttons in the screenshots I made look different from any windows or buttons on any known platform? That's because they use a special Java Look & Feel. You can actually change the Look & Feel so that the windows will mimic those from a different system. To do so, we can use a class called UIManager, which has the following static methods:

`static void setLookAndFeel(LookAndFeel lookandfeel) throws UnsupportedLookAndFeelException` - generally not used, since most look & feels are labeled by classname, as per the next method

`static void setLookAndFeel(String classname) throws ClassNotFoundException, InstantiationException, IllegalAccessException, UnsupportedLookAndFeelException` - the argument denotes a fully qualified class name, ie "`javax.swing.plaf.metal.MetalLookAndFeel`"

`static String getSystemLookAndFeelClassName()` - gets the native system look & feel class name; for example, if called on a Windows machine, returns the Windows look and feel class name

`static String getCrossPlatformLookAndFeelClassName()` - returns the class name for the Java look & feel

And in case you wanted, here are some class names for look & feels:

`javax.swing.plaf.metal.MetalLookAndFeel` - the Java look & feel (it was code-named *metal* in development)

`com.sun.java.swing.plaf.gtk.GTKLookAndFeel` - the GTK+ look & feel

`com.sun.java.swing.plaf.motif.MotifLookAndFeel` - the CDE/Motif look & feel

`com.sun.java.swing.plaf.windows.WindowsLookAndFeel` - the Microsoft Windows look & feel; it only works on Microsoft Windows machines

Note that you usually want to set the look & feel at the first line of your program, or else there's a chance that Java will automatically load the Metal look & feel.

In any case, for our example, we'll take our JFrameDemo.java from last time and add the following lines of code to the top of our `main()` method:

```
try {
	UIManager.setLookAndFeel("com.sun.java.swing.plaf.gtk.GTKLookAndFeel");
} catch (Exception e) {
	System.out.println("Error loading look & feel!");
}
```

We're also going to vary that class name so that we can see the various look & feels available for our programs.

For reference, this was the original JFrameDemo in Metal:

Here it is again in Motif:

Here it is in Windows XP:

And for some reason, GTK didn't work. Oh, well.

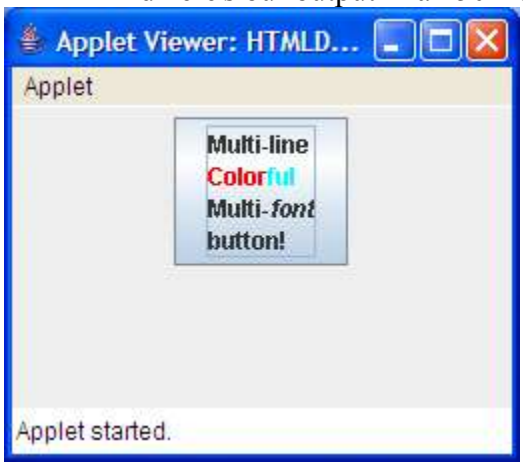
HTML Formatting

Remember how before, we were setting fonts and colors for our components using `setFont()` and `setForeground()`? Of course, that meant that our components can only have one font and one color. How would you like to mix it up?

It turns out that all Swing components that have text can use HTML formatting on that text. If you want to use HTML formatting, just put `<html>` and `</html>` in your String, and put all your text and HTML tags between those two tags. Here's an example:

```
/* HTMLDemo.java */

import java.awt.*;
import javax.swing.*;

public class HTMLDemo extends JApplet {
      public void init() {
            setLayout(new FlowLayout());
            add(new JButton("<html>Multi-line<br><font
                              color=#ff0000>Color</font><font
                              color=#00ffff>ful</font><br><b>Multi</b>-
                              <i>font</i><br>button!</html>"));
      }
}
```

And here's our output in a 250x150 applet:



Nifty, huh? And remember, HTML can be used on *all* Swing components.

Conclusion

Now, you've learned an arsenal of concepts that will allow you to make any kind of GUI you please. You can make buttons and checkboxes. You can place them on the application yourself or have a layout manager automate it for you. You can emulate different platform's look & feels. You can make fancy looking buttons with HTML.

Too bad there are still more tutorials and a vastly larger knowledge pool to dive into.

Next time: Images & Animation!

Homework Assignment!

Take your assignment from last time and make it even better by applying a layout manager to it. Bonus points if you use a layout manager other than Flow and Border layouts!