Crossroads
The ACM Student Magazine

Make Your GUI Swing

by ***Matt Tucker***

Early Java programs suffered from lackluster user interfaces. Even worse, creating a program that appeared consistent on all platforms was often very hard. However, the Swing toolkit can change all of that: it will make your programs look great and work well in Windows, Unix, and any other Java platform.

Fundamentals of the Swing GUI Library

Swing is a toolkit for creating rich graphical user interfaces (GUIs) in Java applications and applets. It is available as a separate download for JDK 1.1, and is included in Java 2 (JDK 1.2 and on). Swing has many advantages over the older AWT classes. We will examine four features in depth:

- A rich set of interface primitives.
- Pluggable look and feel.
- Separate model and view architecture.
- Built-in HTML support.

At the end of the column, we'll look at a simple Swing program to try to gain a practical understanding of how to use the toolkit.

Windows, Buttons, Tooltips, Tables, and More!

Creating a complex user interface using Java's AWT classes is nearly impossible because there are no tables, trees, and other critical interface primitives. Swing provides these components and more. The toolkit also makes interface creation easier with a rich set of customizable borders and additional layout managers such as BoxLayout (more information about layout managers is available in a previous Objective Viewpoint column).

Swing components always begin with a J, such as JFrame, JTable, and JMenu. The meaning of most components should be clear to those that are familiar with the AWT. For example, just as the Frame class is used to create top-level windows in the AWT, JFrame is used to create top-level windows in Swing. A brief description of some of the important Swing classes that don't have a corresponding AWT class is given below:

JInternalFrame	A window that exists inside of another top-level window, such as a JFrame.
JProgressBar	A bar that displays the progress of an event, such as the progress of a download.
JSlider	A bar that allows a user to choose a range of values.
JTable	A component that represents tabular data, such as in a spreadsheet.
JTree	A component that represents hierarchical data, such as a filesystem.

An Interface That Adapts With Your Platform

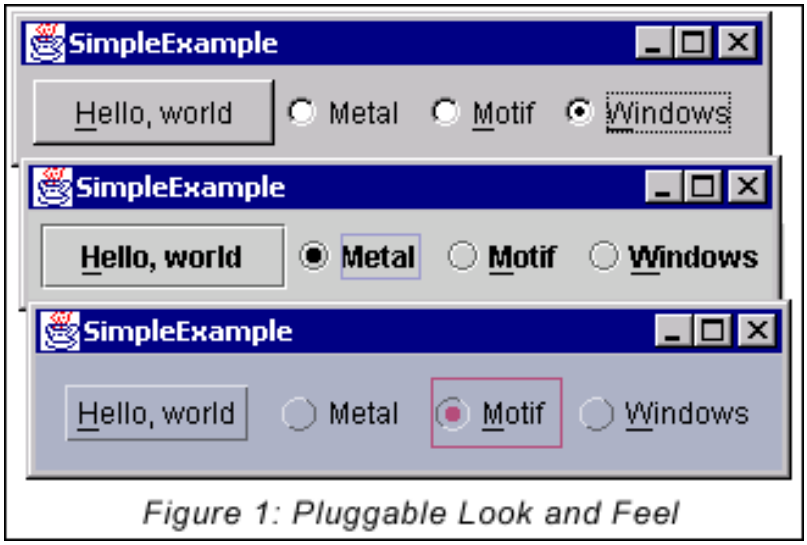

Figure 1: Pluggable Look and Feel

A second major advantage of Swing is its pluggable look and feel. This simply means that the interface can dynamically change. For example, it can look like a Windows, Unix, or Macintosh program, or use the Java look and feel. If it's important that your program look like the other programs on a user's computer, choosing the system look and feel makes the most sense. However, many seem to prefer the cross-platform Java Look and Feel since it provides a consistent and nice-looking interface among all platforms.

Figure 1 shows three screen shots of the same program under different look and feels. The

"Metal" look and feel is the codename for the Java Look and Feel. Changing the look and feel in your own program is simple. The following code snippet demonstrates how:

```
public static void main(String [] args) {
   //Set the look and feel to the Java Look and Feel (default)
   try {
      UIManager.setLookAndFeel(
         UIManager.getCrossPlatformLookAndFeelClassName());
   }
   catch (Exception e) { }
   //create rest of program here
}
```

Other arguments that can be passed to the setLookAndFeel() method:

- `UIManager.getSystemLookAndFeelClassName()` -- the look and feel of the platform the program is running on.
- `"com.sun.java.swing.plaf.windows.WindowsLookAndFeel"` -- the Win32 look and feel (only available in Windows).
- `"com.sun.java.swing.plaf.motif.MotifLookAndFeel"` -- the Motif (Unix) look and feel.
- `"javax.swing.plaf.mac.MacLookAndFeel"` -- the Macintosh look and feel (only available on Macs).

The Swing Architecture is Extensible

Many Swing components are built using a modified version of the Model-View-Controller (MVC) design pattern. This entails a separation between the data (model) of a component and the way a user sees and interacts with it (view). For example, in an MVC spreadsheet application, the data is completely independent of the user interface. The user interface can dynamically change by first connecting the data to a rows and columns layout view and then a pie or line graph view. If the data was actually part of a rows and columns or a graph view (instead of being separated), it wouldn't be possible to switch interfaces as in the MVC application.

Separating the model and view in Swing has many advantages. One is Swing's pluggable look and feel that is discussed above. Another advantage is the ability to override the default model for a component and implement your own. For example, you could create a JTable view that dynamically pulls column and row data out of a database.

Built-in HTML Support

Swing's built-in support for HTML allows easy customization of the look of a component. A

normal JButton declaration might look like:

```
JButton myButton = new JButton
("OK");
```

If you want to make the "OK" message larger and red, you can simply use the appropriate HTML tags:

```
JButton myButton = new JButton
("<html><b><font color="red">OK</font></b></html>");
```

All text based components support HTML since Swing 1.1.1 (or JDK1.2.2). While basic HTML support is pretty good, it is not at the level of a full-fledged web browser such as Netscape or Internet Explorer. Still, it is good enough to view simple web pages, which means that it's possible to create a web browser in just a few hundred lines of code. Other applications might include a small help browser that uses HTML, or a mixed HTML and Swing component interface.

A First Swing Program

Figure 2: CurrentIP.java

Even a simple Swing program demonstrates many important concepts in using the toolkit. We'll look at a utility that displays your current IP address:

Code Listing: CurrentIP.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;

public class CurrentIP {
   public static void main(String [] args) {
      JFrame frame = new JFrame("Current IP");
      frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
```

```
                System.exit(0);
            }
        });
      String IP = "";
      try {
         IP = InetAddress.getLocalHost().getHostAddress();
      }
      catch (Exception e) {
         IP = "Error finding IP";
      }
      //Create a panel that will hold the IP information
      JPanel panel = new JPanel();
      //Add a border to the panel
      panel.setBorder(BorderFactory.createTitledBorder("Current IP Address"));
      panel.add(new JLabel("          " + IP + >"          "));
      //Add the panel to the frame
      frame.getContentPane().add(panel);
      //Pack tells Swing to adjust the size of components so they fit correctly.
      //Another option is to call setSize(int width, int height).
      frame.pack();
      //You must call setVisible(true) to actually display the frame
      frame.setVisible(true);
   }
}
```

Every Swing application must use a top-level window. In most cases, it will be a JFrame as our application uses. The code after the JFrame construction demonstrates the use of *anonymous inner classes*. We need a WindowListener or WindowAdapter to listen for when the user tries to close the program by clicking on the close icon of the JFrame. We could write a separate class that extendedd WindowAdapter or implemented the WindowListener interface to accomplish the task, but that would be a lot of extra work for such as small segment of programming logic. A better solution would be to create an internal class that extended WindowAdapter or implemented WindowListener.

However, the easiest solution is what we have chosen: we create an anonymous inner class that extends WindowAdapter and overrides the windowClosing method. If you look at the code, you'll notice that we simply define the body of an unnamed class within the addWindowListener method call. Using anonymous inner classes is an easy and quick solution to handle many event listening tasks in your applications.

The rest of the application is mostly trivial. We use the InetAddress class to find the user's IP, and then write that data to a JPanel. After adding the JPanel to our JFrame, we call pack() and

setVisible(true). You should only call those methods after you are finished adding components to your JFrame.

Conclusion

We've looked at four advantages of the Swing GUI toolkit and examined a small Swing program. Of course, there is much more to learn about Swing, but its elegance and functionality should motivate you to explore its use further.

References

1. Sun Microsystems, The Java Tutorial -- **http://thejavatutorial.com**.
2. Sun Microsystems, Java Platform 1.2 API Specification, 1999.
3. Sun Microsystems, The Swing Connection -- **http://theswingconnection.com**.

Image by Bill Lynch.

Biography

Matt Tucker is a senior majoring in Computer Science at the University of Iowa. He held a summer internship in the Java Division of Sun Microsystems and in his spare time runs **CoolServlets.com**, a site that gives away open-source Java servlets. Matt thinks that Java is pretty neat and finds it strange to write in third-person.