Swing from A to Z

Some Simple Components

By Richard G. Baldwin
baldwin.richard@iname.com

Java Programming, Lecture Notes # 1005

July 31, 2000

Preface

This series of lessons entitled "Swing from A to Z" discusses the capabilities and features of Swing in quite a lot of detail.

For persons who need details

This series is intended for those persons who need to understand Swing at a detailed level. It is recommended that in addition to studying this set of lessons, you also study my earlier lessons having the following titles:

- The AWT and Swing, A Preview
- Swing and the Delegation Event Model
- Swing, New Event Types in Swing
- Swing, Understanding getContentPane() and other JFrame Layers
- The Swing Package, A Preview of Pluggable Look and Feel
- Swing, Hidden Buttons with Icons, Icon Images, Borders, Tool Tips, Nested Buttons, and Other Fun Stuff
- Swing, Creating and Using Trees
- Swing, Custom Rendering of Tree Nodes
- Swing, Simplified Lists in Swing
- Swing, Understanding Component MVC Models
- Swing, Custom Rendering of JList Cells
- Swing, Custom List Selection Model for JList Objects

Those lessons will introduce you to the use of Swing while avoiding much of the detail included in this series.

Introduction

Swing components are beans

Swing components are JavaBean components. When we discuss JavaBean components, we are usually interested in the interface to those components as determined by the *methods*, *events*, and *properties* of the bean.

Swing components extend JComponent

Most swing components extend **JComponent** either directly or indirectly.

JComponent extends **Container**, which in turn extends **Component**.

Therefore, most Swing components inherit methods, events, and properties (ME&P) from these three classes.

Of course, many Swing components supplement these inherited ME&P in the definition of other classes that extend **JComponent**.

Understanding Swing components as a group

If you understand the ME&P that Swing components inherit from these three classes, you already know a lot about a Swing component even before you begin examining it individually. In other words, we can learn a lot about Swing components by examining the three classes listed above.

Will examine these classes

The next several lessons will discuss ME&P that apply to a large number of Swing components, because they are inherited from the classes **JComponent**, **Container**, and **Component**.

Need some components to work with

In order to discuss and illustrate these features of all Swing components, I need to write and explain simple programs making use of some of the components that inherit those ME&P. In other words, I need some components to use for illustration.

Purpose of the lesson

The purpose of this lesson is to introduce four fairly simple components that I will use for illustration purposes in subsequent lessons. Those components are:

- JFrame
- JButton
- JLabel
- JTextField

I will provide a simple application that makes use of these components as "drop in" replacements for the AWT components having similar names:

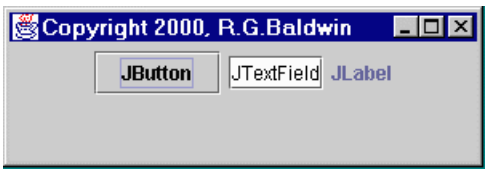
- Frame
- Button
- Label
- TextField

Sample Program

The sample program is named Swing11.java. A complete listing of the program is provided in Figure 8.

A screen shot

A screen shot of the output from the program is shown below.



This output is rendered in the Sun *Metal* look and feel, which is a special look and feel developed by Sun specifically for Java.

Although Swing provides a lot of capability for "dressing up" the GUI (such as the addition of fancy borders on the components), this is a plain vanilla default rendering of the Metal look and feel.

Four swing components were used

This program places a **JButton**, a **JTextField**, and a **JLabel** in the client area of a **JFrame**, 300 pixels wide and 100 pixels tall.

The components are placed in the frame using a **FlowLayout** manager.

Event handling behavior

A single action listener object is registered on both the button and the text field.

If you click on the button, or press the space bar while the button has the focus, the text on the face of the button is displayed in the label.

If you press the enter key while the text field has the focus, the contents of the text field are displayed in the label.

If you click the close button in the upper right corner of the **JFrame**, the program will terminate.

Very similar to an AWT program

As you will see when I discuss the code in the following section, the code to accomplish this is essentially the same as would be used with a **Frame**, a **Button**, a **TextField**, and a **Label** with the AWT.

One major difference

However, there is one difference required by Swing, which I will highlight in my discussion.

Interesting Code Fragments

The first fragment, shown in Figure 1, shows the import directives required for this program. I used boldface to highlight the directive required for Swing.

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

Figure 1

Not the same as the older version

If you compare this with a Swing program written to use the version of Swing that existed prior to the release of JDK 1.2, you will notice that the import directive is different. Sun changed the location of the Swing class libraries with the release of JDK 1.2.

The swing components

The fragment in Figure 2 shows the definition of the controlling class. Note that this class extends **JFrame**.

```
class Swing11 extends JFrame
                implements ActionListener{
  JButton jButton = new JButton("JButton");
  JTextField jTextField = new JTextField(
                                  "JTextField");
  JLabel jLabel = new JLabel("JLabel");
```

Figure 2

The fragment also declares three reference variables for the types **JButton**, **JTextField**, and **JLabel**.

New objects of these three classes are instantiated, and references to the objects are assigned to the reference variables.

Names are similar to AWT components

The names of the Swing classes are the same as the names of corresponding AWT classes (**Frame**, **Button**, **TextField**, and **Label**) except that the names of the Swing classes are prepended with an upper-case J.

Swing provides "drop in" replacements for AWT components

Swing provides a replacement component for all, or at least most, of the components in the AWT component library. (In a few cases, such as radio buttons, the Swing approach is different from the AWT approach.) The naming convention involving the prepending of an upper-case "J" to the AWT class name is used in most cases.

Implements ActionListener

The controlling class in this simple program implements the **ActionListener** interface.

That means that an object of the controlling class is an action listener that can be registered on any component that has the ability to multicast action events.

Must define actionPerformed() method

This also means that the controlling class must provide a concrete definition for the method named **actionPerformed()**, which is declared in the **ActionListener** interface.

The constructor

The code fragment in Figure 3 shows the beginning of the constructor for the controlling class. This code is straightforward.

```
  Swing11(){//constructor
    jButton.addActionListener(this);
    jTextField.addActionListener(this);
```

Figure 3

Registering action listeners on sources

In this code, the object of the controlling class (**this**) is registered as an action listener on the button and on the text field.

When do action events occur?

The button multicasts an action event when it is clicked with the mouse, or when the space bar is pressed while the button has the focus.

The text field multicasts an action event when the *Enter* key is pressed while the text field has the focus.

The one difference relative to AWT

The fragment in Figure 4 shows the only thing that is different between this Swing program and a similar program that could be written using the corresponding components from the AWT.

```
getContentPane().setLayout(
                    new FlowLayout());
getContentPane().add(jButton);
getContentPane().add(jTextField);
getContentPane().add(jLabel);
setTitle("Copyright 2000, R.G.Baldwin");
setSize(300,100);
setVisible(true);
```

Figure 4

The AWT version

When programming with the AWT, to add a component to a **Frame**, or to set the layout manager on a **Frame**, you simply invoke either the **add()** method or the **setLayout()** method on a reference to the **Frame** object.

The Swing version

However, to add a component to a **JFrame**, or to set the layout manager on the **JFrame** object, you must first invoke **getContentPane()** on a reference to the **JFrame** object.

Why getContentPane()

This has to do with the fact that with a **Frame** in AWT, every child of the **Frame** is located on the same layer.

However, with Swing and the **JFrame**, you can place components on any one of about 65,000 different layers (**getContentPane()** represents only one of those layers).

Move components between layers at runtime

In addition, you can move components from one layer to another at runtime. If the components overlap, those components on the layers with higher numbers will cover the components on layers with lower numbers.

See an earlier tutorial for details

I discuss this in detail in the tutorial lesson entitled *"Swing, Understanding getContentPane() and other JFrame Layers"*, so I won't discuss it further here. Just remember that you need to do it.

An anonymous inner class

The fragment in Figure 5 is an anonymous inner class that causes the program to terminate when the user presses the close button on the **JFrame**. There is nothing about this that is peculiar to Swing, so I won't discuss it further here. *(Note that JDK 1.3 provides another approach to terminating the program when the user presses the close button on the **JFrame**, but the code in Figure 5 works with JDK 1.3 as well.)*

```
  this.addWindowListener(
    new WindowAdapter(){
      public void windowClosing(
                        WindowEvent e){
        System.exit(0);
      }//end windowClosing()
    }//end WindowAdapter
  );//end addWindowListener
}//end constructor
```

Figure 5

The main() method

The code in Figure 6 is a plain vanilla **main()** method that instantiates an object of the controlling class, causing the GUI to appear on the screen. Again, there is nothing special about this, so I won't discuss it further.

```
public static void main(String args[]) {
    new Swing11();
}//end main()
```

Figure 6

The action event handler

Finally, the code in Figure 7 is the **actionPerformed()** method that is invoked whenever an action event occurs on either the button or on the text field. This is plain vanilla Delegation Event Model material, so I won't discuss it further.

```
  public void actionPerformed(ActionEvent e){
    jLabel.setText(e.getActionCommand());
  }//end actionPerformed

}//end class Swing11
```

Figure 7

Summary

If you are familiar with event driven programming using the Delegation Event Model with the AWT, but you are not familiar with Swing, you should have seen only one thing that might be new to you.

getContentPane() is peculiar to Swing

The only thing about this program that should have been new to you is the requirement to use **getContentPane()** whenever you need to add a component to the **JFrame** object, or you need to set a layout manager on the **JFrame** object.

Otherwise, this program looks just like an AWT program with new names for the classes from which the button, text field, and label are instantiated.

The purpose of the lesson

The purpose of this lesson was not to teach you anything significantly new. Rather, it was simply to introduce you to the use of Swing **JFrame**, **JButton**, **JTextField**, and **JLabel** in place of AWT **Frame**, **Button**, **TextField**, and **Label**.

Will use these components for illustration of Swing features

As mentioned earlier, I will use these components in the next few lessons as I explain some of the methods, events, and properties that Swing components inherit from the **JComponent**, **Container**, and **Component** classes.

Complete Program Listing

A complete listing of the program is provided in Figure 8.

```
/*File Swing11
Rev 3/26/00
Copyright 2000, R.G.Baldwin


Illustrates very simple use of JButton,
JTextField, and JLabel.


Tested using JDK 1.2.2 under WinNT 4.0 WkStn
```

```
**********************************/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;//jdk 1.2 version

class Swing11 extends JFrame
                implements ActionListener{
  JButton jButton = new JButton("JButton");
  JTextField jTextField = new JTextField(
                          "JTextField");
  JLabel jLabel = new JLabel("JLabel");

  //---------------------------------------//
  Swing11(){//constructor
    jButton.addActionListener(this);
    jTextField.addActionListener(this);

    getContentPane().setLayout(
                        new FlowLayout());
    getContentPane().add(jButton);
    getContentPane().add(jTextField);
    getContentPane().add(jLabel);
    setTitle("Copyright 2000, R.G.Baldwin");
    setSize(300,100);
    setVisible(true);

    //.....................................//
    //Anonymous inner terminator class
    this.addWindowListener(
      new WindowAdapter(){
        public void windowClosing(
                          WindowEvent e){
          System.exit(0);
        }//end windowClosing()
      }//end WindowAdapter
    );//end addWindowListener
  }//end constructor

  //---------------------------------------//
  public static void main(String args[]) {
      new Swing11();
  }//end main()

  //---------------------------------------//
  public void actionPerformed(ActionEvent e){
    jLabel.setText(e.getActionCommand());
  }//end actionPerformed
  //---------------------------------------//
}//end class Swing11
```

Figure 8

About the author

Richard Baldwin *is a college professor and private consultant whose primary focus is a combination of Java and XML. In addition to the many platform-independent benefits of Java*

applications, he believes that a combination of Java and XML will become the primary driving force in the delivery of structured information on the Web.

Richard has participated in numerous consulting projects involving Java, XML, or a combination of the two. He frequently provides onsite Java and/or XML training at the high-tech companies located in and around Austin, Texas. He is the author of Baldwin's Java Programming Tutorials, which has gained a worldwide following among experienced and aspiring Java programmers. He has also published articles on Java Programming in Java Pro magazine.

Richard holds an MSEE degree from Southern Methodist University and has many years of experience in the application of computer technology to real-world problems.

baldwin.richard@iname.com

-end-