Swing from A to Z

Using Focus in Swing, Part 2

By Richard G. Baldwin

Java Programming, Lecture Notes # 1042

November 27, 2000

- Preface
- Introduction
- Sample Program
- Interesting Code Fragments
- Summary
- What's Next
- Complete Program Listing

Preface

This series of lessons entitled *Swing from A to Z*, discusses the capabilities and features of Swing in quite a lot of detail. This series is intended for those persons who need to understand Swing at a detailed level.

Viewing tip

You may find it useful to open another copy of this lesson in a separate browser window. That will make it easier for you to scroll back and forth among the figures and listings while you are reading about them, without losing your place.

Recommended supplementary reading

In an earlier lesson entitled Alignment Properties and BoxLayout, Part 1, I recommended a list of my earlier Swing tutorials for you to study prior to embarking on a study of this set of lessons.

Where are they located?

You will find those lessons published at Gamelan.com. I also maintain a consolidated Table of Contents at *Baldwin's Java Programming Tutorials* that may make it easier for you to locate specific lessons. The Table of Contents on my site provides links back to each of the lessons at Gamelan.com.

Introduction

The previous lesson

In a previous lesson entitled Using Focus in Swing, Part 1, I introduced you to:

- The five focus properties of the **JComponent** class that are inherited by most Swing components.
- The focus traversal cycle.
- The use of the **requestFocus()** method.

The focus properties

The five focus properties of the **JComponent** class that are inherited by most Swing components are repeated below for convenient viewing.

- focusCycleRoot
- focusTraversable
- managingFocus
- requestFocusEnabled
- nextFocusableComponent

Promises, promises...

In the previous lesson, I also provided the AWT baseline for focus traversal. I promised that this lesson would extend the concept to include focus traversal in Swing, and would contrast the two different versions of the focus traversal cycle.

I also promised that subsequent lessons will discuss the use of all five of the focus properties of Swing, and will show you how to create and use your own focus manager as a replacement for the default focus manager.

Swing components are beans

Swing components are JavaBean components, and most extend **JComponent** either directly or indirectly. We are usually interested in the *methods*, *events*, and *properties* of the bean. Most Swing components inherit the methods, events, and properties from **JComponent** and its superclasses.

Understanding Swing components as a group

If you understand the characteristics that Swing components inherit from **JComponent**, 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 **JComponent** class.

Sample Program

This sample program, named **Swing24**, is the second in a series of programs designed to illustrate the use of focus in Swing.

The screen shot in Figure 1 shows what the screen looks like when this sample program starts running.

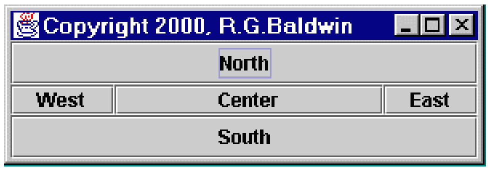

Figure 1 A Screen Shot of the Running Program

A side trip

Strictly as an aside, Figure 2 shows a screen shot from the previous lesson.

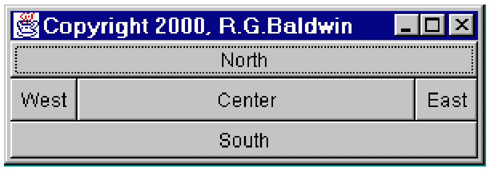

Figure 2 A Screen Shot from the Previous Lesson

What's the difference?

Do you notice any difference between the two screen shots? The main difference is that Figure 1 shows the Swing GUI rendered in the Sun *Metal* look and feel, while Figure 2 shows the AWT GUI rendered under the Windows operating system, giving it a Windows look and feel.

Swing provides *pluggable look and feel*

With the AWT, the rendering of components is determined by the operating system. However, with Swing, you have the ability to change the look and feel at runtime largely independent of the operating system. *(See my earlier lessons to learn more about pluggable look and feel.)*

Now back to the main thread

Now lets return from our side trip and get back to the business at hand. The previous lesson illustrated two important aspects of AWT and Swing focus handling:

Order of focus traversal

First, the order of focus traversal for the AWT is the order in which the components are placed in the container. This lesson will explain the order of focus traversal for Swing, which is different from the AWT.

Requesting the focus

Second, it is possible, with either the AWT or Swing, to request that the focus be shifted to a specific component by invoking the **requestFocus()** method on that component.

The default starting point

With both Swing and the AWT, the default is for the first component placed in the container to have the focus when the program starts running. In this case, that would be the East Button *(later you will see that the East button is the first component placed in the container).*

Overriding the default

However, a **ComponentEvent** handler in this program requests that the focus be shifted to the North Button immediately after the GUI is made visible. As shown in Figure 1, this causes the focus to reside on the North button by the time the GUI is available to the user.

Focus traversal cycle

Focus traversal can be observed by repeatedly pressing the Tab key. Pressing Shift-Tab will reverse the order of focus traversal.

What is the order of focus traversal?

If, like the AWT, the order of traversal were based on the order that the components are placed in the container, then the order of focus traversal for the GUI in Figure 1 would be North, Center, East, West, South, and back to North.

Not like the AWT

However, with the default Swing focus manager, the order of traversal is not based on the order that the components are placed in the container.

Rather, the order of traversal proceeds from the upper left to the lower right regardless of the order that the components are placed in the container.

Therefore, for the Swing GUI shown in Figure 1, the order of traversal is North, West, Center, East, South, and back to North.

Modifying the order of focus traversal

There are at least two different ways to modify the order of focus traversal in Swing.

One way is to manipulate the **nextFocusableComponent** property. The second way is to replace the default focus manager with a new focus manager of your own design. Both approaches will be illustrated in subsequent lessons.

Event handlers

There are no event handlers registered on the five buttons placed in the **JFrame** in Figure 1.

However, as mentioned earlier, a component listener is registered on the **JFrame** to invoke **requestFocus()** on the North button when the **JFrame** becomes visible. *(You can only request focus on a component after it becomes visible. If you request focus before the component becomes visible, the request has no effect.)*

Interesting Code Fragments

I will break this program down and discuss it in fragments. A listing of the entire program is provided in Listing 5.

The controlling class

Listing 1 shows the beginning of the controlling class named **Swing24** along with the **main()** method.

The fragment also shows the instantiation of a new **JButton** object with a label of "North". A subsequent code fragment will invoke the **requestFocus()** method to purposely shift the focus to this **JButton** component.

```
/*File Swing24.java Rev 8/17/00
**********************************/

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

class Swing24 extends JFrame{
  JButton northBtn = new
JButton("North");

  public static void main(String
args[]) {
    new Swing24();
  }//end main()

Listing 1
```

The controlling class extends JFrame

Note that the controlling class extends the **JFrame** class. **JFrame** is one of the top-level classes in Swing. Therefore the instance of the controlling class created in the **main()** method appears on the screen as a GUI.

The constructor

Listing 2 shows the beginning of the constructor.

```
  Swing24(){//Constructor
    getContentPane().
             add(new
JButton("East"),"East");
    getContentPane().
             add(new
JButton("West"),"West");
    getContentPane().
           add(new
JButton("South"),"South");
    getContentPane().add(northBtn,"North");
    getContentPane().
         add(new
JButton("Center"),"Center");

    setTitle("Copyright 2000,
R.G.Baldwin");
    setSize(300,100);


Listing 2
```

The layout manager

Hopefully by now, you are familiar with the use of **getContentPane()** in Swing. According to Sun, *"The default contentPane() will have a BorderLayout manager set on it."*

Place components in the JFrame

The first five statements in the constructor (highlighted in boldface) instantiate four anonymous **JButton** objects and place them, along with the **JButton** object instantiated earlier, in the East, West, South, North, and Center locations of the **JFrame** component.

First component is placed in the East

Note in particular that the first component is placed in the east position. By default, this is the button that would have the focus when the GUI first appears on the screen. However, the **requestFocus()** method will be used to prevent this button from having the focus when the GUI becomes available to the user.

Traversal doesn't match order of placement

As explained earlier, unlike the AWT, the focus traversal order of components in a Swing container does not match the order in which the components are placed in the container. If it did, this would cause the focus traversal order among the buttons to be: East, West, South, North, Center, and back to East.

The actual focus traversal cycle

However, the focus traversal order provided by the default Swing focus manager is from left to right, top to bottom. Therefore, the actual focus traversal order of this GUI is North, West, Center, East, South, and back to North.

Set title and size properties

The remaining code in Listing 2 simply sets the title property and the size property of the **JFrame**. There's nothing exciting or unusual about that.

An anonymous ComponentEvent listener

Listing 3 is an anonymous **ComponentEvent** listener class that is used to cause the focus to reside on the North button when the GUI becomes available to the user.

```
    //Register component listener to request
    // focus on the north JButton after the
    // JFrame becomes visible.
    //....................................//
    this.addComponentListener(
      new ComponentAdapter(){
        public void componentShown(
                                   ComponentEvent e){
          northBtn.requestFocus();
        }//end componentShown
      }//end ComponentAdapter
    );//end addComponentListener
    //....................................//

    //Make the JFrame visible
    setVisible(true);
```

Listing 3

What does this code do?

This code invokes the **requestFocus()** method on the reference to the North button. This causes the focus to shift to that button *(after the GUI becomes visible on the screen).*

This is essentially the same code that was explained in the previous lesson. Therefore, I won't discuss it further here.

Make the JFrame visible

The remaining code in Listing 3 sets the visible property to true, causing the **JFrame** to become visible on the screen. This is what actually triggers the **componentShown()** event that the earlier code in Listing 3 is listening for.

The terminator...

The code in Listing 4 registers an anonymous object that causes the program to terminate when the user clicks the *close* button on the **JFrame**.

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

  }//end constructor
}//end class Swing24
```

Listing 4

Summary

In the previous lesson, I introduced you to:

- The five focus properties of the **JComponent** class that are inherited by most Swing components.
- The focus traversal cycle.
- The use of the **requestFocus()** method.

In that lesson, I also provided the AWT baseline for focus traversal.

In this lesson, I have extended the concept to include focus traversal in Swing, and I have contrasted the two different versions of the focus traversal cycle.

What's Next?

In the next lesson, I will show you how to control the *focus root cycle* in Swing.

What I mean by focus root cycle is, when you create a composite GUI consisting of nested containers, you have the choice of allowing the focus to traverse smoothly into and out of a nested container, or of causing the focus to continually cycle within a nested container once it enters that container.

This will be accomplished using the **isFocusCycleRoot()** method and the **focusCycleRoot** property.

The lessons in this miniseries will discuss the use of all five of the focus properties of Swing, and will also show you how to create and use your own focus manager as a replacement for the default focus manager.

Complete Program Listing

A complete listing of the program is provided in Listing 5.

```
/*File Swing24.java Rev 8/17/00
Copyright 2000, R.G.Baldwin

This is one in a series of programs designed
to illustrate the use of focus in Swing.

This lesson illustrates how the focus
traverses its cycle among Swing components.
This will be contrasted with the manner in
which the focus traverses its cycle in the
AWT.

Illustrates two important aspects of Swing
focus handling:

First, unlike the AWT, the order of focus
traversal in Swing is left to right, top to
bottom, regardless of the order in which the
components are placed in the container.

Second, it is possible to request that focus
be moved to a specific component by invoking
the requestFocus() method on that component.
Like the AWT, the default is for the first
component placed in the container to have
the focus when the program starts running.
In this case, that would be the East JButton.
A ComponentEvent handler requests focus on
the North JButton immediately after the GUI
is made visible which causes the focus to
reside on the North JButton by the time the
GUI is available to the user.

Thus, the order of focus traversal is
```

```
North, West, Center, East, South, and back
to North.  Focus traversal can be observed
by repeatedly pressing the Tab key.
Repeatedly pressing Shift-Tab will reverse
the order of focus traversal among the
components.

There are no event handlers registered on
the five JButtons placed in the JFrame.
However, a component listener is registered
on the JFrame to invoke requestFocus() on the
North JButton when the JFrame becomes
visible.  (You can only request focus on a
component after it becomes visible.)

Tested using JDK 1.2.2 under WinNT 4.0 WkStn
*********************************/

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

class Swing24 extends JFrame{
  JButton northBtn = new JButton("North");

  public static void main(String args[]) {
    new Swing24();
  }//end main()
  //--------------------------------------//

  Swing24(){//Constructor
    getContentPane().
             add(new JButton("East"),"East");
    getContentPane().
             add(new JButton("West"),"West");
    getContentPane().
           add(new JButton("South"),"South");
    getContentPane().add(northBtn,"North");
    getContentPane().
         add(new JButton("Center"),"Center");

    setTitle("Copyright 2000, R.G.Baldwin");
    setSize(300,100);

    //Register component listener to request
    // focus on the north JButton after the
    // JFrame becomes visible.
    //....................................//
    this.addComponentListener(
      new ComponentAdapter(){
        public void componentShown(
                           ComponentEvent e){
          northBtn.requestFocus();
        }//end componentShown
      }//end ComponentAdapter
    );//end addComponentListener
    //.....................................//

    //Make the JFrame visible
    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
}//end class Swing24
```

Listing 5

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-