Swing from A to Z

Using Focus in Swing, Part 1

By Richard G. Baldwin

Java Programming, Lecture Notes # 1040

November 20, 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. 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, 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, I told you that one of the major weaknesses of the AWT, as compared to other GUI programming environments, is the handling of focus.

Swing is better

Swing rectifies that weakness by providing a significantly improved capability for the programmer to control the focus. This lesson is the first lesson in a new miniseries on the use of Swing and focus.

Swing components are beans

In another earlier lesson entitled Swing from A to Z, Some Simple Components, I told you that Swing components are JavaBean components. *(You will find quite a lot of information about JavaBean components in my earlier lessons.)*

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 the methods, events, and properties from these three classes. *(Many Swing components also supplement these inherited characteristics in the definition of other classes that extend JComponent.)*

Understanding Swing components as a group

If you understand the characteristics 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.

JComponent and focus

If we examine the API documentation from Sun, we find the following methods in the **JComponent** class.

- **isFocusCycleRoot()** - Override this method and return true if your component is the root

of a component tree with its own focus cycle.
- **isFocusTraversable()** - Identifies whether or not this component can receive the focus.
- **isManagingFocus()** - Override this method and return true if your JComponent manages focus.
- **isRequestFocusEnabled()** - Return whether the receiving component can obtain the focus by calling requestFocus
- **getNextFocusableComponent()** - Return the next focusable component or null if the focus manager should choose the next focusable component automatically

Components inherit these methods

Every component class that extends **JComponent** inherits these methods, and is free to use them as is, or to override them to provide behavior that is more appropriate for objects instantiated from that class.

Methods define properties

Hopefully, you will recognize that these methods define a set of properties having the following names *(if not, you may want to review some of my earlier lessons on the properties of JavaBean components):*

- focusCycleRoot
- focusTraversable
- managingFocus
- requestFocusEnabled
- nextFocusableComponent

Using the focus properties

The lessons in this miniseries will teach you how to use these properties in your Swing programs. The miniseries will also contain lessons on how to design and use your own **FocusManager** class as an alternative to the default Swing focus manager.

Sample Program

This sample program, named **Swing22**, is the first 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.

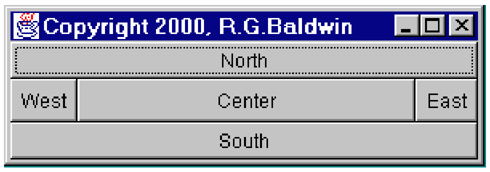

Figure 1 A Screen Shot of the Running Program

Doesn't deal directly with Swing yet...

However, this program doesn't deal directly with Swing just yet. Rather, it uses the AWT as a baseline to illustrate the focus traversal cycle.

The program illustrates how the focus traverses its cycle among AWT components. This will be contrasted *(in a subsequent lesson)* with the way in which the focus traverses its cycle in Swing.

Two important points

This program illustrates 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. *(We will see later that Swing treats the order of focus traversal differently from the AWT.)*

Requesting the focus

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

What is the default starting point?

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.

Order of focus traversal

Thus, based on the order that the components were placed in the container, the order of focus traversal is North, Center, East, West, South, and back to North.

How do I traverse the focus?

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.

Event handlers

There are no event handlers registered on the five buttons placed in the **Frame**. However, as mentioned above, a component listener is registered on the **Frame** to invoke **requestFocus()** on the North button when the **Frame** becomes visible. *(You can only request focus on a component after it becomes visible.)*

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 **Swing22** along with the **main()** method. The fragment also shows the instantiation of a new **Button** object with a label of "North". A subsequent code fragment will invoke the **requestFocus()** method to purposely shift the focus to this **Button** component.

```
/*File Swing22.java Rev 8/17/00
Copyright 2000, R.G.Baldwin
**********************************/

import java.awt.*;
import java.awt.event.*;

class Swing22 extends Frame{
  Button northBtn = new
Button("North");

  public static void main(String
args[]) {
    new Swing22();
  }//end main()
```

Listing 1

The controlling class extends Frame

Note that the controlling class extends the **Frame** class. **Frame** is a top-level class of the AWT. Therefore the instance of the controlling class created in the **main()** method appears on the screen as a graphical user interface (GUI) once it becomes visible.

The constructor

Listing 2 shows the beginning of the constructor.

```
  Swing22(){//Constructor
    add(new Button("East"),"East");
    add(new Button("West"),"West");
    add(new Button("South"),"South");
    add(northBtn,"North");
    add(new Button("Center"),"Center");

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

Listing 2
```

The layout manager

The default layout manager for a **Frame** object is **BorderLayout**. *(At the risk of boring you, let me mention one more time that you can learn more about BorderLayout and other layout managers in my earlier lessons.)*

Place components in the Frame

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

Focus traversal order matches order of placement

As explained earlier, the focus traversal order of components in an AWT container matches the order in which the components are placed in the container. This causes the focus traversal order among the buttons to be: East, West, South, North, Center, and back to East.

Set title and size properties

The remaining code in Listing 2 simply sets the title property and the size property of the **Frame**.

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. *(By now you should know where you can learn more about anonymous inner classes if this syntax is not familiar to you.)*

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

    //Make the Frame visible
    setVisible(true);

Listing 3
```

What does this code do?

This code invokes the **requestFocus()** method on the reference to the North button to cause the focus to shift to that button. However, as mentioned earlier, this can only be done (successfully) after the component becomes visible. *(If you request focus on a component that is not visible, the focus is not shifted to the component.)*

A componentShown() event

Whenever a component becomes visible, that component multicasts a **ComponentEvent** to all registered listeners, and invokes the **componentShown()** method on those listeners. That event is used in this program to determine when the **Frame** actually becomes visible.

Shift the focus

When the **Frame** becomes visible, the **requestFocus()** method is invoked on the reference to the North button to cause the focus to shift to that button.

Make the Frame visible

The remaining code in Listing 3 sets the visible property to true, causing the **Frame** to become visible on the screen. This is what actually triggers the **componentShown()** event discussed above.

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 **Frame**.

```
    //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 Swing22
```

Listing 4

Another approach

JDK 1.3 provides an alternative way to accomplish the same result, but so far, I am still using the old way for the benefit of those persons who have not yet downloaded and installed JDK 1.3. *(In some future lesson, I will show you the alternative way to terminate an application using JDK 1.3.)*

Summary

In this lesson, I have 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.

What's Next?

In this lesson, I provided the AWT baseline for focus traversal. In the next lesson, I will extend the concept to include focus traversal in Swing, and will contrast the two different versions of the focus traversal cycle.

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.

Complete Program Listing

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

```
/*File Swing22.java Rev 8/17/00
```

```
Copyright 2000, R.G.Baldwin

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

However, this particular program is not about
Swing.  Rather, it illustrates how the focus
traverses its cycle among AWT components.
This will be contrasted with the manner in
which the focus traverses its cycle in
Swing.

Illustrates two important aspects of AWT
focus handling:

First, the order of focus traversal is 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 Swing, 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.
A ComponentEvent handler requests focus on
the North Button immediately after the GUI
is made visible, which causes the focus to
reside on the North Button by the time the
GUI is available to the user.

Thus, the order of focus traversal is
North, Center, East, West, 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 buttons placed in the Frame.
However, a component listener is registered
on the Frame to invoke requestFocus() on the
North button when the Frame 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.*;

class Swing22 extends Frame{
  Button northBtn = new Button("North");

  public static void main(String args[]) {
    new Swing22();
  }//end main()
  //---------------------------------------//

  Swing22(){//Constructor
    add(new Button("East"),"East");
    add(new Button("West"),"West");
    add(new Button("South"),"South");
    add(northBtn,"North");
    add(new Button("Center"),"Center");

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

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

    //Make the Frame 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 Swing22
```

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-