Swing from A to Z

Glue, Struts, and BoxLayout

By Richard G. Baldwin

Java Programming, Lecture Notes # 1032

November 6, 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 different figures and listings while you are reading about them.

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*.

The Table of Contents on my site provides links to each of the lessons at Gamelan.com.

The lessons identified on that list will introduce you to the use of Swing while avoiding much of the detail included in this series.

Introduction

The BoxLayout manager

Also, in the earlier lesson entitled Alignment Properties and BoxLayout, Part 1, I introduced you to the **Box** container and the **BoxLayout** manager.

In the previous lesson entitled Alignment Properties and BoxLayout, Part 2, I promised you that this lesson would deal with *glue* and *struts*.

Preview

In this lesson, I will refresh your memory on how to use a **Box** container with its default **BoxLayout** manager.

I will also show you how to place components on the horizontal axis, and how to insert glue and struts between the components so as to produce the behavior for which the glue and strut components are intended.

What are glue and struts?

The first thing that I want to do is to show you an example of glue and struts in action. Let's begin with the screen shot shown in Figure 1.

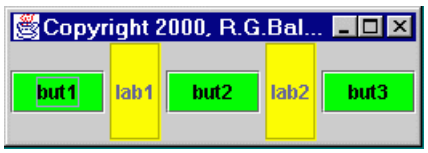

Figure 1 An Example of Glue and Struts

The screen shot in Figure 1 shows three buttons and two labels placed in a **Box** container using **BoxLayout** as the layout manager.

Invisible components

Although it isn't obvious in Figure 1, the container also contains four invisible strut components and two invisible glue components. These invisible components are used to separate the buttons and the labels that you can see. Each of the invisible strut components has a width of three pixels.

Where are the invisible components hiding?

The type and position of the components from left to right is:

button, glue, strut, label, strut, button, strut, label, strut, glue, button

As you can see, the glue and strut components, when present, separate the button and label components.

Is the order important?

The locations of the glue and strut components relative to the buttons and labels is very important.

However, the order in which the glue and strut components occur relative to each other is not important. In fact, they *appear* to occupy the same physical space regardless of the order in which you add them to the layout. *(This appearance is an illusion. They don't really occupy the same physical space. I will explain this in more detail in a subsequent lesson.)*

What is a strut component?

You can think of a strut component as an invisible, *non-compressible* component of a given size that you can insert between two other components. It acts as an invisible spacer that prevents the components from being pushed together.

You can't see the wind either

Just like the wind, you can't see a strut, but you can see the result of having a strut in the screen shot shown in Figure 2.

Try to push buttons and labels closer together

This screen shot shows the result of manually reducing the size of the **JFrame** in an attempt to push the button and label components closer together.

As you can see, the components refuse to be pushed closer together. This is because they are separated from each other by an invisible spacer component called a strut.

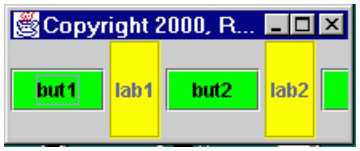

Figure 2 Illustration that Struts Keep Components Separated

What is glue?

Actually, glue is not a very good name for this component. The name *spring* may have been more appropriate, because the component tends to act like a spring.

Glue can expand

A glue component is a component that can expand when needed to fill the space between two other components.

Where is the glue?

As I mentioned earlier, two glue components were placed in the **Box** along with the other components. The two glue components were placed between the label components and the buttons on each end.

How does glue behave?

The screen shot in Figure 3 shows the effect of the glue components when the user manually resizes the **JFrame** to make it wider than its original size.

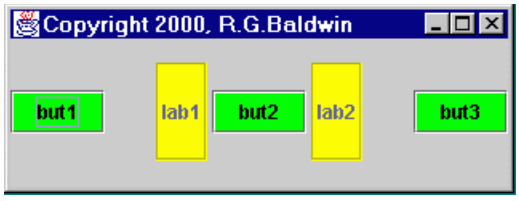

Figure 3 Illustration of the Effect of Glue Components

Glue components got wider

As you can see, the glue components separating the buttons and the labels increased in width to fill the newly available horizontal space.

This allowed the button and label components to retain their original size.

It also allowed the center button (which was separated from the two labels on either side by a strut but no glue) to maintain its same position relative to those two labels.

Sample Program

The program that produced the above screen shots is named **Swing16**. This program illustrates the use of glue and struts.

Interesting Code Fragments

I will break this program down and discuss it in fragments. A listing of the entire program is provided in Listing 6.

The controlling class

Listing 1 shows the beginning of the controlling class. Because the controlling class extends **JFrame**, an object of the controlling class is a top-level GUI.

```
class Swing16 extends JFrame{

  public static void main(String
args[]) {
      new Swing16();
  }//end main()

Listing 1
```

Listing 1 also shows the main method, which instantiates an object of the controlling class.

Constructor uses a factory method

The constructor for the controlling class begins in Listing 2.

```
  Swing16(){//constructor

    //Instantiate a new horizontal Box
    // object.
    Box aBox =
Box.createHorizontalBox();

    //Add the Box to the contentPane
    getContentPane().add(aBox);

Listing 2
```

The constructor invokes a factory method of the **Box** class to cause a new **Box** object to be instantiated.

An alternative approach

As an alternative, I could have used the new operator, as in the following expression, to instantiate the object.

```
new Box(BoxLayout.X_AXIS)
```

BoxLayout with horizontal placement

Recall that the layout manager for a **Box** object is **BoxLayout**, and it can't be changed.

In this case, the layout manager was initialized to cause the components to be arranged in a horizontal line.

First component on the left...

The first component added to the **Box** appears on the left end of the line, and the last component added to the **Box** appears on the right end of the line.

In typical Swing fashion, the constructor also adds the **Box** object to the content pane.

Need some green buttons

The code in Listing 3 simply instantiates three green **JButton** objects. This is not particularly interesting. It is shown here simply for the sake of continuity.

```
    JButton but1 = new JButton("but1");
    but1.setBackground(Color.green);

    JButton but2 = new JButton("but2");
    but2.setBackground(Color.green);

    JButton but3 = new JButton("but3");
    but3.setBackground(Color.green);
```

Listing 3

And some yellow labels...

The code in Listing 4 instantiates two yellow **JLabel** objects. A **CompoundBorder** is used to make the labels taller than the buttons.

```
    JLabel lab1 = new JLabel("lab1");
    lab1.setBorder(
        new CompoundBorder(
            new EtchedBorder(),
                new
EmptyBorder(20,2,20,2)));
    lab1.setBackground(Color.yellow);
    lab1.setOpaque(true);

    JLabel lab2 = new JLabel("lab2");
    lab2.setBorder(
        new CompoundBorder(
            new EtchedBorder(),
```

```
                new
EmptyBorder(20,2,20,2)));
    lab2.setBackground(Color.yellow);
    lab2.setOpaque(true);

Listing 4
```

Again, there is nothing particularly interesting here. This code fragment was included in the discussion for the sake of continuity.

The final construction

We have arrived at the code that illustrates the main purpose of this lesson.

The code in Listing 5 performs the final construction of the GUI.

```
    aBox.add(but1);

    aBox.add(Box.createGlue());
    aBox.add(Box.createHorizontalStrut(3));

    aBox.add(lab1);
    aBox.add(Box.createHorizontalStrut(3));
    aBox.add(but2);
    aBox.add(Box.createHorizontalStrut(3));
    aBox.add(lab2);
    aBox.add(Box.createHorizontalStrut(3));
    aBox.add(Box.createGlue());
    aBox.add(but3);

Listing 5
```

This is where the buttons and labels get added to the **Box**.

Now for the glue and struts

More importantly, this is where the glue and the struts get inserted between the buttons and the labels.

Strut components, each three pixels wide, are inserted between each of the buttons and labels to keep them separated by three pixels.

Glue components are inserted between the buttons on each end and their neighboring labels.

Where is the Glue or Strut class or interface?

If you go to the documentation, you probably won't find classes or interfaces named Glue or Strut. What you will find are the factory methods, highlighted with boldface in Listing 5, to create and return references to these invisible components as type **Component**.

Add the glue and the strut components

These invisible components are created and added to the **Box** in the appropriate order, by the code in Listing 5, to cause them to be inserted between the buttons and the labels as indicated earlier in this lesson.

Remaining Code

The remaining code in the program is uninteresting, and therefore won't be discussed further here. You can view all of the code in the program in Listing 6.

Summary

In this lesson, I have illustrated once again how to use a **Box** container with its default **BoxLayout** manager.

I showed you how to place components on the horizontal axis, and how to insert glue and struts between the components so as to produce the behavior shown in the earlier screen shot.

What's Next?

In the next lesson, I will discuss the use of *minimum*, *maximum*, and *preferred* sizes with struts in **BoxLayout**.

Complete Program Listing

A complete listing of the program is provided in Listing 6.

```
/*File Swing16
Rev 3/30/00
Copyright 2000, R.G.Baldwin

Illustrates use of glue and struts to control
the separation between components.  In order
to see the effect of using glue, you must
manually resize the JFrame object to make
it larger and smaller.

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

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

```
import javax.swing.*;
import javax.swing.border.*;

class Swing16 extends JFrame{

  public static void main(String args[]) {
      new Swing16();
  }//end main()
  //---------------------------------------//

  Swing16(){//constructor

    //Instantiate a new horizontal Box
    // object.  Could also use the
    // constructor
    // new Box(BoxLayout.X_AXIS);
    Box aBox = Box.createHorizontalBox();

    //Add the Box to the contentPane
    getContentPane().add(aBox);

    //Instantiate three JButton objects,
    // make them green.
    JButton but1 = new JButton("but1");
    but1.setBackground(Color.green);

    JButton but2 = new JButton("but2");
    but2.setBackground(Color.green);

    JButton but3 = new JButton("but3");
    but3.setBackground(Color.green);

    //Instantiate two JLabel objects  Use a
    // compound border to make them taller
    // than the buttons.  Color them yellow.
    JLabel lab1 = new JLabel("lab1");
    lab1.setBorder(
        new CompoundBorder(
            new EtchedBorder(),
                new EmptyBorder(20,2,20,2)));
    lab1.setBackground(Color.yellow);
    lab1.setOpaque(true);

    JLabel lab2 = new JLabel("lab2");
    lab2.setBorder(
        new CompoundBorder(
            new EtchedBorder(),
                new EmptyBorder(20,2,20,2)));
    lab2.setBackground(Color.yellow);
    lab2.setOpaque(true);

    //Add the buttons and the labels to the
    // Box.  Insert glue between the labels
    // and the buttons on each end.  Insert
    // horizontal struts between each of the
    // components to control the minimum
```

```
    // spacing between them.
    aBox.add(but1);
    aBox.add(Box.createGlue());
    aBox.add(Box.createHorizontalStrut(3));
    aBox.add(lab1);
    aBox.add(Box.createHorizontalStrut(3));
    aBox.add(but2);
    aBox.add(Box.createHorizontalStrut(3));
    aBox.add(lab2);
    aBox.add(Box.createHorizontalStrut(3));
    aBox.add(Box.createGlue());
    aBox.add(but3);

    setTitle("Copyright 2000, R.G.Baldwin");
    //Pack the JFrame down around the
    // components
    pack();
    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 Swing16
```

Listing 6

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-