Swing from A to Z

Alignment Properties and BoxLayout, Part 2

By Richard G. Baldwin

Java Programming, Lecture Notes # 1031

October 9, 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 the previous lesson entitled Swing from A to Z, 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

Preview

In this lesson, I will show you how to use a **Box** container with its default **BoxLayout** manager.

I will show you how to place components on the horizontal axis, and how to establish their vertical positions relative to one another by setting the **alignmentY** property value.

The Box container and BoxLayout manager

Also in the previous lesson, I introduced you to the **Box** container and the **BoxLayout** manager. I showed you the screen shot in Figure 1, which shows five components placed in a **JFrame** object using the **BoxLayout** manager.

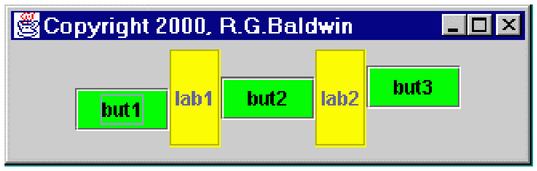


Figure 1 Screen Shot with BoxLayout Manager

I explained that the **BoxLayout.X_AXIS** constant was used when constructing the **BoxLayout** to cause the components to be placed along the horizontal axis. *(It is also possible to place components along a vertical axis in BoxLayout.)*

The alignmentX and alignmentY properties

The primary purpose of this lesson is to investigate the use of the **alignmentX** and **alignmentY** properties that many components inherit from the **JComponent** class.

In the program used to produce the above screen shot, the **alignmentY** property was used to control the vertical positions of the five components relative to one another.

I will show you the code that was used to produce this screen shot later in this lesson.

Components stay in a line

Also in the previous lesson, I explained that if the user manually resizes a container that contains components laid out according to **BoxLayout**, the components stay in their line formation, even if this means that some of the components are no longer visible.

Figure 2 is a screen shot of the same GUI after having manually resized it to make it narrower.

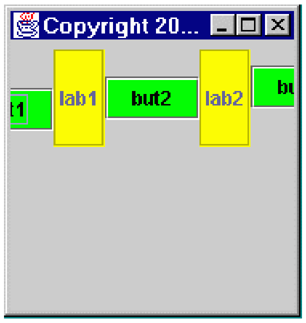


Figure 2 Manually Resized Version of the GUI from Figure 1

As you can see, making the container narrower caused the components on the ends to become partially obscured due to their determination to *"stay in the line, no matter what."*

Stay on the rails, no matter what

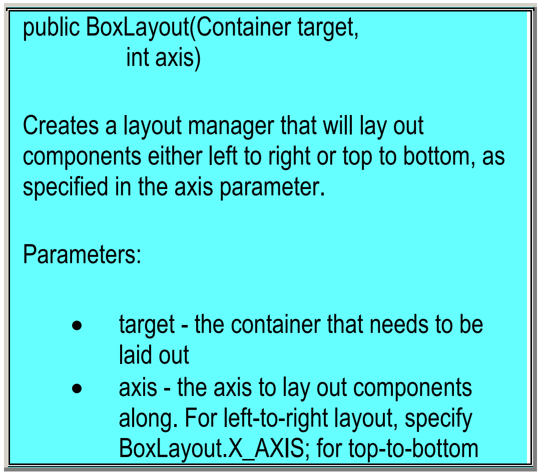
In case you didn't recognize it, the above line is a takeoff on a book that I used to read to my children about a little railroad engine that had trouble remembering that he had to "stay on the rails, no matter what." I'll bet that book was read to many of you by your parents as well.

The BoxLayout manager

BoxLayout is the default layout manager for a **Box** container. It can also be applied to other containers as well, such as **JPanel**.

BoxLayout constructor

Information from Sun regarding the constructor for **BoxLayout** is shown below. We will need this later when viewing code that uses **BoxLayout**.



public BoxLayout(Container target,
                 int axis)

Creates a layout manager that will lay out components either left to right or top to bottom, as specified in the axis parameter.

Parameters:

- target - the container that needs to be laid out
- axis - the axis to lay out components along. For left-to-right layout, specify BoxLayout.X_AXIS; for top-to-bottom

layout, specify BoxLayout.Y_AXIS

Setting the alignment

Here is what Sun has to say about alignment in a container being laid out using a **BoxLayout** manager.

BoxLayout attempts to arrange components at their preferred widths (for left to right layout) or heights (for top to bottom layout).

For a left to right layout, if not all the components are the same height, BoxLayout attempts to make all the components as high as the highest component. If that's not possible for a particular component, then BoxLayout aligns that component vertically, according to the component's Y alignment.

By default, a component has an Y alignment of 0.5, which means that the vertical center of the component should have the same Y coordinate as the vertical centers of other components with 0.5 Y alignment.

Similarly, for a vertical layout, BoxLayout attempts to make all components in the column as wide as the widest component; if that fails, it aligns them horizontally according to their X alignments.

Set and get alignment property

The **JComponent** class provides *setter* methods for controlling the values of the **alignmentX** and **alignmentY** properties of objects that extend **JComponent**. These methods expect to receive a **float** parameter ranging from 0.0 to 1.0.

What do the alignment values mean?

A Y-value of 0.0 represents alignment at the bottom, while a Y-value of 1.0 represents alignment at the top.

An X-value of 0.0 represents alignment at the left, while an X-value of 1.0 represents alignment on the right.

Values in between 0.0 and 1.0 represent proportional movement from bottom to top, or from left to right.

A value of 0.5 represents center alignment in both cases. As mentioned above, the default value for Y alignment is 0.5. This will be illustrated in the sample program that follows.

Sample Program

This program illustrates the use of a **Box** container.

The program also illustrates the use of the **setAlignmentY()** method inherited from the **JComponent** class to adjust the vertical positions of some components in a **Box** container.

Interesting Code Fragments

I will break this program down and discuss it in fragments. A listing of the entire program is provided in Listing 7 near the end of the lesson.

This program is named **Swing15**.

The controlling class

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

```
class Swing15 extends JFrame{

  public static void main(String
args[]) {
      new Swing15();
  }//end main()

Listing 1
```

You have seen code like this many times in the past. The only significant thing about this code is, because the controlling class extends **JFrame**, an object of the controlling class is also a top-level GUI that can be placed on the computer's desktop.

The constructor

Listing 2 shows the beginning of the constructor. If you have studied the supplementary Swing lessons recommended earlier, you will recognize that there is also nothing particularly new about this code. The main reason that I highlighted it is not to explain what it is, but rather to explain what it is not.

```
  Swing15(){//constructor

    //Set for center alignment in the
    // contentPane
    getContentPane().setLayout(
           new
FlowLayout(FlowLayout.CENTER));

Listing 2
```

Not the same alignment

Although the word *alignment* is used in the comments describing this code, this is not the same as the **alignmentX** and **alignmentY** properties inherited from the **JComponent** class. Rather, this deals with a property named **alignment**, which is a property of objects of the **FlowLayout** class.

FlowLayout constructor sets alignment property

In this case, the alignment is being set through a parameter to the **FlowLayout** constructor. However, the **FlowLayout** class also provides a setter method named **setAlignment(int align)** that can be used to set the **alignment** property after the object has been instantiated. *(Note that this setter method requires an int as a parameter instead of a float.)*

Instantiate a Box object

The code in Listing 3 instantiates a new **Box** container object and adds it to the content pane.

```
    //Instantiate a Box container
    Box aBox = new
Box(BoxLayout.X_AXIS);

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

Listing 3
```

The parameter to the constructor for the **Box** specifies that the components be arranged in a horizontal line within the container. *(Recall that the default layout manager for a Box is BoxLayout, and this cannot be changed.)*

Instantiate the JButton objects

The code fragment in Listing 4

- Instantiates three **JButton** objects.
- Sets their vertical alignment using the **setAlignmentY()** method inherited from **JComponent**. *(Note that this method requires a float parameter instead of an int.)*
- Sets their **background** property to the color green.

```
    JButton but1 = new JButton("but1");
    but1.setAlignmentY(0.25f);
    but1.setBackground(Color.green);

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

    JButton but3 = new JButton("but3");
    but3.setAlignmentY(0.75f);
    but3.setBackground(Color.green);

Listing 4
```

Where does the background property come from?

The **background** property is also inherited into **JButton**. In this case, the property is inherited from the class named **Component**, which is a superclass of **JComponent**.

Default value of alignmentY

Note that the default value for the **alignmentY** property is 0.5f. Since this is the desired value for the second button, it isn't necessary to set the property for that button. Thus, the new property values are set only for the first and third buttons.

Now let's see some labels

The code fragment in Listing 5

- Instantiates two **JLabel** objects.
- Sets their **border** property to a **CompoundBorder** to make them taller than the buttons.
- Sets their **background** property to 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);

Listing 5
```

I discussed the use of the **border** property in several previous lessons.

Does not set alignmentY property

Note that this code does not set the **alignmentY** property for the **JLabel** objects. Rather, this property is allowed to retain its default value of 0.5f.

By viewing the screen shot presented in Figure 1, you will see that the two **JLabel** objects and the single **JButton** object with the default value for **alignmentY**, are aligned at their vertical centers. This agrees with the information from Sun that I presented earlier.

One button is lower; another is higher

By viewing the screen shot in Figure 1, you will also notice that the center of the button with the **alignmentY** value of 0.25f is aligned about half way between the bottom and the center of the **JLabel**.

The button with the **alignmentY** value of 0.75f is aligned about half way between the center and the top of the **JLabel.**

Finish constructing the GUI

Finally, the code fragment in Listing 6 adds the buttons and the labels to the **Box**.

```
    aBox.add(but1);
    aBox.add(lab1);
    aBox.add(but2);
    aBox.add(lab2);
    aBox.add(but3);

Listing 6
```

The remaining code in the program simply takes care of some utility matters that aren't worth discussing here. You can view that code in Listing 7 near the end of the lesson.

Summary

In this lesson, I showed you 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 establish their vertical positions relative to one another by setting the **alignmentY** property value.

What's Next?

In the next lesson, I will show you how to use glue and struts to control the separation between components.

Complete Program Listing

A complete listing of the program is provided in Listing 7.

```
/*File Swing15
Rev 3/30/00
Copyright 2000, R.G.Baldwin

Illustrates use of setAlignmentY to adjust
the vertical position of some buttons in a
Box container.  Also illustrates the use
of a Box container.

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 Swing15 extends JFrame{

  //---------------------------------------//

  public static void main(String args[]) {
      new Swing15();
  }//end main()
  //---------------------------------------//

  Swing15(){//constructor

    //Set for center alignment in the
    // contentPane
    getContentPane().setLayout(
          new FlowLayout(FlowLayout.CENTER));

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

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

    //Instantiate three JButton objects,
    // make them green, and set their
    // vertical alignment.  Note that the
    // default vertical alignment is 0.5f.
    JButton but1 = new JButton("but1");
    but1.setAlignmentY(0.25f);
    but1.setBackground(Color.green);

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

    JButton but3 = new JButton("but3");
    but3.setAlignmentY(0.75f);
    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.
    aBox.add(but1);
    aBox.add(lab1);
    aBox.add(but2);
    aBox.add(lab2);
    aBox.add(but3);


    setTitle("Copyright 2000, R.G.Baldwin");
    setSize(329,100);
    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
```

```
}//end class Swing15
```

Listing 7

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-