Swing from A to Z

Alignment Properties and BoxLayout, Part 1

By Richard G. Baldwin

Java Programming, Lecture Notes # 1030

October 9, 2000

- Preface
- Introduction
- A Sample BoxLayout
- Summary
- What's Next?

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 while you are reading about them, without losing the place where you are reading.

Recommended supplementary reading

It is recommended that prior to embarking on a study of this set of lessons, you first study the following lessons on Swing, which you will find at Gamelan.com. I also maintain a consolidated Table of Contents at *Baldwin's Java Programming Tutorials*. The Table of Contents provides links to each of the lessons at Gamelan.com.

- 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

The lessons listed above 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 introduce you to the **Box** container and the **BoxLayout** manager. I will discuss a number of characteristics of each, and will show you some screen shots that illustrate the use of the **BoxLayout** manager.

The Box container

The **Box** class can be used to produce a lightweight container that uses a **BoxLayout** (see below) object as its layout manager.

Cannot modify the layout manager

Unlike other containers, however, you cannot modify the layout manager of a **Box** object. (For example, attempting to cause the layout manager for a **Box** to be **FlowLayout** produces the following runtime error: *java.awt.AWTError: Illegal request.*

Provides important class methods

While the **Box** class can be used as a container, perhaps its most useful characteristic is serving as the home for several class methods that produce invisible components:

- createGlue()
- createHorizontalGlue()
- createHorizontalStrut(int width)
- createRigidArea(Dimension d)
- createVerticalGlue()
- createVerticalStrut(int height)

These invisible components are very useful for controlling the appearance of component layouts in containers that use **BoxLayout**.

I will discuss the use of these invisible components in a subsequent lesson.

Does not extend JComponent

One shortcoming of **Box** as a container is that it does not extend **JComponent**. Rather, it extends **Container**. As a result, many capabilities imparted by the **JComponent** class (such as the creation of borders) do not apply to a **Box** container.

The BoxLayout manager

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

Places components in a line

BoxLayout is a layout manager that makes it possible to position components in either a horizontal line or in a vertical line.

The components do not wrap (as in **FlowLayout**). Therefore, a group of components in a horizontal or vertical line will remain in the line when the container is resized.

Nested BoxLayout objects

You can nest containers having a **BoxLayout** manager to achieve groupings of horizontal and vertical lines of components. I will provide an illustration of nesting **JPanel** objects using **BoxLayout** in a subsequent lesson.

The **BoxLayout** manager places each of its managed components from left to right, or from top to bottom in the order that they are placed in the container.

BoxLayout constructor

The constructor for **BoxLayout** is a little unusual. As shown below, you must pass two parameters to the constructor when you instantiate the object.

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

The first parameter is a reference to the container whose layout will be managed by the layout manager (this is the unusual part).

The second parameter specifies whether the components will be arranged in a horizontal line or a vertical line.

Setting the alignment

Component alignment is very important in **BoxLayout**. Here is what Sun has to say about alignment in a **BoxLayout**.

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* and *getter* 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.

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.

A Sample BoxLayout

Figure 1 shows a screen shot of three buttons and two labels placed in a **JFrame** using **BoxLayout**. The **BoxLayout.X_AXIS** constant was used when constructing the **BoxLayout** to cause the components to be placed along the horizontal axis.

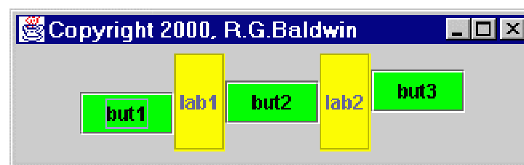

Figure 1 Screen Shot of Horizontal Box Layout

As you can see, the **alignmentY** property was used to control the vertical positions of the five components relative to one another.

Figure 2 shows a screen shot of the same GUI after having manually resized it to make it narrower. If you are familiar with **FlowLayout**, you will know that performing this manual operation on a container controlled by **FlowLayout** would cause the right-most components to move down to produce a second line of components.

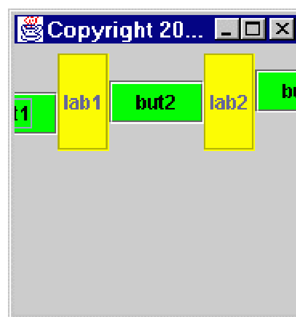

Figure 2 Screen Shot of "Narrowed" Horizontal Box Layout from Figure 1.

However, as mentioned earlier, resizing a **BoxLayout** doesn't have that effect. Rather, the components remain on the specified axis, even if that means that they get clipped at their ends (as is the case here).

Summary

In this lesson, I have introduced you to the **Box** container and the **BoxLayout** manager. I have discussed a number of characteristics of each.

What's Next?

In the next lesson, I will discuss the code that was used to produce the screen shots shown above. Following that, I will discuss a variety of interesting aspects of component alignment as inherited from the **JComponent** class.

Copyright 2000, Richard G. Baldwin. Reproduction in whole or in part in any form or medium without express written permission from Richard Baldwin is prohibited.

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-