Swing from A to Z: Demystifying Glue and Struts, Part 1

Published December 4, 2000
By Richard G. Baldwin

Java Programming, Lecture Notes # 1034

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 your place.

Recommended supplementary reading

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

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

Where are they located?

You will find those lessons published at Gamelan.com. However, as of the date of this writing, Gamelan doesn't maintain a consolidated index of my Java tutorial lessons, and sometimes my lessons are difficult to locate there. You will find a consolidated index at *Baldwin's Java Programming Tutorials*.

The index on my site provides links to the lessons at Gamelan.com.

Preview

In this lesson, I will show you some screen shots from a program that is designed to take the mystery out of glue and struts. This program doesn't use glue and struts as produced by the factory methods that were illustrated in the lesson entitled *Glue, Struts, and BoxLayout*. Instead, this program makes use of invisible spacer components of my own design that behave similarly to glue and struts but provide more capability.

Introduction

In the lesson entitled *Alignment Properties and BoxLayout, Part 1*, I introduced you to the **BoxLayout** manager. I will use **BoxLayout** in the sample program discussed in this lesson.

The mystery of glue and struts

In another lesson entitled *Glue, Struts, and BoxLayout*, I introduced you to the use of glue and struts. I don't know about you, but when I first encountered the use of glue and struts, they seemed pretty mysterious to me. However, we didn't have the tools to take the mystery out of glue and struts until we studied the size properties in the lesson entitled *Minimum, Maximum, and Preferred Sizes*.

My promise to you

In the lesson entitled *Minimum, Maximum, and Preferred Sizes*, I promised you that this lesson would take the mystery out of glue and struts. I also promised that we would have some fun in the process. Hopefully I can keep that promise.

What are minimum, maximum, and preferred sizes?

All components that extend **JComponent** inherit the following three properties that support the use of layout managers:

- preferredSize
- minimumSize
- maximumSize

Why are they needed?

Whenever the size of a container changes, the layout manager *may* use these properties to change component sizes in accordance with the wishes of the programmer.

Different layout managers do different things

Different layout managers use these properties in different ways. The **BoxLayout** manager honors the width dimension of all three properties for components placed on a horizontal line. *(It probably also honors the height dimension for components placed on a vertical line, but we haven't demonstrated that.)*

Sample Program

This sample program, named **Swing20** is designed to take the mystery out of glue and struts as used with **BoxLayout**. I will show you some screen shots produced by this program in this lesson, and will get into the details of the code in the next lesson.

Must resize to see the effect

This program illustrates the use of invisible fixed-width and elastic spacers to control the separation between components. In order to see the effect of the spacers, you must manually resize the **JFrame** object to make it larger and smaller.

BoxLayout can be used with many containers

This program also demonstrates that **BoxLayout** can be used with containers other than **Box**, *(even containers that might not ordinarily be used for this purpose)*.

Let's make things interesting

Just to make things interesting, this program places a **JButton** on the content pane in a **JFrame** and uses that **JButton** as a container for several other components. *(You should recall that a JButton, like most components in Swing, is a container.)*

Why use a JButton as a container?

I'm not suggesting that a **JButton** has any particular usefulness when used as a container in this fashion. I simply elected to use it to demonstrate that **BoxLayout** can be used with a variety of different container types.

A JButton with a BoxLayout manager

The layout manager for the **JButton** is set to **BoxLayout.**

BoxLayout allows you to specify whether the components will be arranged on a horizontal line, or on a vertical line. In this case, a horizontal line placement was specified.

What does it look like?

Three green **JButton** objects and two yellow **JLabel** objects were placed in the large **JButton** as shown in the screen shot identified as Figure 1.

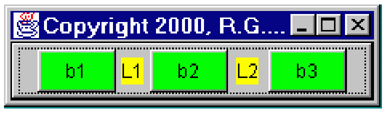

Figure 1. A screen shot at startup

Also contains invisible spacer components

In addition, several invisible spacers were placed between the buttons and the labels as shown in Figure 1.

Custom-designed spacers were used

However, in this case, the invisible spacers were not *glue* and *struts* as described in the lesson entitled *Glue, Struts, and BoxLayout*. Rather, the invisible spacers were custom designed using two different approaches.

Two of these new spacer types have more capability than the standard glue and struts discussed in the lesson entitled *Glue, Struts, and BoxLayout*.

Is this a better alternative?

I'm not necessarily presenting this as a better alternative to glue and struts. Rather, I am presenting it as a way for you to better understand the inner-workings of glue and struts.

Three different types of spacers

When we get to the program code in the next lesson, you will see that three different types of spacers were created and used:

- Fixed Spacer
- Elastic Spacer
- Combination Spacer

How do they behave?

The fixed spacers behave in a manner very similar to struts when used with **BoxLayout**.

The elastic spacers behave like glue, *except that you can specify the maximum amount of "stretch"*. When the elastic spacers hit that limit, they won't stretch any more.

The combination spacers combine the behavior of fixed spacers and elastic spacers in a single spacer object.

Fixed width spacers having different widths

Now, please refer back to Figure 1. This is the image that first appears on the screen when you start the program *(before any manual resizing takes place)*.

Space increases from left to right

Note that each pair of components is separated by progressively more space moving from left to right across the image. The components were separated by fixed-width spacers having the following widths: 3, 4, 5, and 6 pixels.

What if you make it narrower?

Now refer to Figure 2. This screen shot shows the behavior of the **BoxLayout** manager with this set of components when the width of the container is manually decreased.

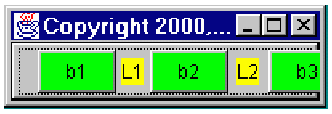

Figure 2. A screen shot after making the Frame narrower.

Spaces don't decrease

As you would expect from your studies on glue and struts, since the invisible spacers between the button and label components have a fixed width, the space between them doesn't decrease.

Right-most component gets clipped

Rather, the visible components remain the same distance apart, and the right-most component gets clipped when the container is no longer wide enough to accommodate the total width of the buttons, the labels, and the fixed-width spacers.

An interesting effect

There is one interesting aspect of Figure 2 that you may have noticed. Even though the large **JButton** object is the actual container, it appears that the right-most component is not clipped at the "edge" of the large **JButton**. Rather, it is clipped at the edge of the client area of the **JFrame**.

An artifact of the 3D rendering

This is an artifact of the three-dimensional rendering of the large **JButton**. The outer edge of the **JButton** is at the inner edge of the client area of the **JFrame** *(inside its borders)*. All of the decorations that cause the **JButton** to exhibit the optical illusion of being three-dimensional are inside its outer edges.

Clipped at the actual outer edge

The right-most component is clipped at the actual outer edge of the **JButton**, but this is beyond the "apparent" edge of the three-dimensional representation of the **JButton**.

Can demonstrate using Metal look and feel

You can demonstrate this by modifying the program (discussed in a subsequent lesson) to cause it to use a Metal look and feel instead of a Windows look and feel. A **JButton** in the Metal look and feel has very little three-dimensional decoration, and the clipping location appears closer to where you would expect to see it.

What if you increase the width?

The screen shot in Figure 3 shows what happens when you increase the width of the **JFrame**, and hence increase the width of the large **JButton** serving as a container for the other components.

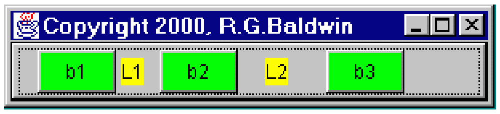

Figure 3. A screen shot after making the Frame wider.

Elastic spacers were used

Elastic spacers were inserted on both sides of the button labeled **b2**, and on the left side of the button labeled **b3**.

Can only stretch so far

However, unlike glue, these spacers have an upper limit on their elasticity.

Going from left to right, the upper limit is: 6, 12, and 24 pixels respectively.

The maximum stretch limits

In other words, the first elastic spacer on the left is allowed to stretch to a maximum width of six pixels. Each of the other elastic spacers is allowed to stretch twice as far as the elastic spacer to its left.

As you can see in Figure 3, the space between each pair of components is approximately (but not exactly) twice the width of the space to its left. *(I will have more to say about this in the next lesson.)*

What about the big space on the right?

I did not insert a spacer on the right-hand side of **b3**. However, the space on the right-hand side of **b3** is the largest of all. Furthermore, it continues to grow as the width of the **JFrame** continues to increase. This indicates that the elastic spacers have all reached their limit. The remaining required space is being provided on the right end of the line by the **BoxLayout** manager.

A single combination spacer

When we get to a discussion of the program code, you will see that a single spacer of the Combination Spacer type was inserted to the left of the button labeled **b3**. This single spacer component had a fixed minimum width of six pixels and a maximum width of 24 pixels.

A single fixed spacer

A single fixed-width spacer component (without an Elastic Spacer) was inserted to the right of the button labeled **b1**. This component had a fixed width of three pixels. Hence, the space between **b1** and **L1** did not increase when the width of the **JFrame** was manually increased.

Two pairs of spacers

A pair of spacers was inserted on each side of the button labeled **b2**. Each pair consisted of one spacer of the Fixed Spacer type and one spacer of the Elastic Spacer type.

The widths of the Fixed Spacers were 4 and 5 pixels respectively.

The maximum widths of the Elastic Spacers were 6 and 12 pixels respectively.

Elastic spacers have zero minimum width

In all cases, the minimum width of the elastic spacers was zero pixels.

Summary

In this lesson, I have shown you some screen shots from a program that is designed to take the mystery out of *glue* and *struts*. This program doesn't use glue and struts as produced by the factory methods that were illustrated in the lesson entitled *Glue, Struts, and BoxLayout*.

Instead, this program makes use of invisible spacer components of my own design that behave similarly to glue and struts but provide more capability. I am hopeful that an examination of the code for these spacers will serve to make the behavior of glue and struts more understandable.

What's Next?

In the next lesson, I will introduce you to the actual code that was used to produce the screen shots that were shown in this lesson. In that lesson, you will learn how to design your own invisible spacer components using two different approaches.

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-