Swing from A to Z

Minimum, Maximum, and Preferred Sizes

By Richard G. Baldwin

Java Programming, Lecture Notes # 1033

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 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

Also, in the same earlier lesson, I introduced you to the **Box** container and the **BoxLayout** manager. I will use both of them in the sample program in this lesson.

In the previous lesson entitled Glue, Struts, and BoxLayout, I promised you that this lesson would deal with *minimum*, *maximum*, and *preferred* sizes.

Preview

Every component that extends **JComponent** has the following three properties:

- preferredSize
- minimumSize
- maximumSize

Different layout managers behave differently with respect to these three properties.

In this lesson, I will show you how to control the values of these three properties. I will also show you how a **BoxLayout** behaves with respect to these properties when the physical size of the container is changed.

What are minimum, maximum, and preferred sizes?

All components that extend **JComponent** inherit the three properties listed above.

The **JComponent** class provides setter and getter methods for all three of these properties. *(This is a major improvement over the original AWT, which treated them as read-only properties.)*

Why do we need them?

These three properties are needed to support the use of layout managers.

Whenever a user (or the program code) resizes a container that contains one or more components, the layout manager may need to change the size of some of the components.

These three properties are intended to be used by layout managers to make changes in component sizes somewhat in accordance with the wishes of the programmer.

How are the size properties maintained?

The three size properties are all maintained as objects of the class **Dimension**.

What is a Dimension object?

An object of the **Dimension** class is a container for two public instance variables named **width** and **height**. These variables are both defined as type **int**.

According to Sun:

> The Dimension class encapsulates the width and height of a component (in integer precision) in a single object. The class is associated with certain properties of components. Several methods defined by the Component class and the LayoutManager interface return a Dimension object.

How are they accessed?

Because **width** and **height** are public, they can be accessed directly without the requirement to use an accessor method, such as **getWidth()**.

In addition, setter and getter methods are available for both **width** and **height**. Interestingly, the setter and getter methods take or return **double** values, and apparently provide automatic conversion between **double** and **int** on the fly.

Are the size properties always honored?

Some layout managers, such as **GridLayout**, completely ignore the size properties.

FlowLayout, attempts to honor both dimensions of **preferredSize**, and possibly has no need to honor either **minimumSize** or **maximumSize**.

Other layout managers fall somewhere in between.

What about BorderLayout?

For example, **BorderLayout**

- Ignores the **width** dimension of **preferredSize** for North and South components.
- Ignores the **height** dimension of **preferredSize** for East and West components.
- Ignores both dimensions of **preferredSize** for Center components.

Apparently **BorderLayout** also ignores both dimensions of **minimumSize** and **maximumSize** in all cases.

BoxLayout

As we will see in this lesson, **BoxLayout** honors the width dimension of all three properties for components placed on a horizontal line.

Must the size properties always be set?

Every component has default values for all three of the size properties. It is not necessary for you to set them if you are satisfied with the default values. The manner in which the default values are determined varies from one component to another.

An example of a default value

For example, the default value for **preferredSize** for a given component, is usually a size that provides a visually pleasing component.

For those components that have a text caption, such as **JButton** and **JLabel**, the default preferred size is based on the size required to properly display the text caption in the current font.

Sample Program

This sample program, named **Swing17** illustrates the use and manipulation of the width dimension of the **preferredSize**, **minimumSize**, and **maximumSize** properties.

Figure 1 shows three **JLabel** components placed on a horizontal line in a **BoxLayout** in a **JFrame**.

Figure 1 A Screen Shot Showing Three JLabel Components in a BoxLayout in a JFrame

(As mentioned in the earlier viewing tip, you may find it useful to open another copy of this lesson in a separate browser window so that you can easily view the screen shots while reading about them)

Separated by ten-pixel struts

The three components are separated from each other and from the ends of the frame by struts. Each, strut is ten pixels wide.

Figure 1 shows what this GUI looks like when it first appears on the screen, before it is manually resized.

preferredSize for component1

The left-most component (**component1**) has the width dimension of its **preferredSize** property set to 1.1 times the default preferred width.

As you can see in Figure 1, this causes a small amount of blank green space to appear to the right of the caption on the component. *(Note that the preferredSize property for the other two components was not modified. Hence, they don't exhibit any blank green space.)*

What about minimumSize and maximumSize?

The width dimensions of the **minimumSize** and **maximumSize** properties for **component1** were set equal to the modified preferred size. Therefore, the size of **component1** doesn't change when the user resizes the **JFrame**.

minimumSize for other two components

The two right-most components (**component2** and **component3**) have the **width** dimension of their **minimumSize** property set to 0.6 times the preferred width of **component1**.

maximumSize for other two components

The **width** dimension of the **maximumSize** property for the other two components was set to 1.2 times the preferred width of **component1**.

Expand the JFrame

The screen shot entitled Figure 2 shows what happens if you manually expand the size of the **JFrame**.

Figure 2 The result of manually expanding the JFrame from Figure 1.

The width of **component1** (the **JLabel** on the left in Figure 2) doesn't change when the **JFrame** is manually expanded. This is because its maximum width was set equal to its preferred width.

What about the other two components?

However, the width of each of the two right-most components increases up to its maximum size. This is evidenced by the fact that these two components now show more blank green space than the blank green space shown by **component1**. *(Recall that originally component2 and component3 didn't show any blank green space.)*

What happens if you continue expanding?

Figure 3 shows that continuing to increase the width of the **JFrame** doesn't cause the widths of any of the three components to increase after they reach their maximum width. This is evidenced by the increase in the blank gray space on the right end of Figure 3.

Figure 3 The Result of Expanding the JFrame Even Further

What if you decrease the width?

Figure 4 shows what happens if you manually decrease the width of the **JFrame**.

Figure 4 The Result of Manually Decreasing the Width of the JFrame

As you can see, this causes the two right-most components to decrease in width. In fact, the decrease makes it impossible for these two components to display their captions, so Swing automatically changes the displayed caption to make it end with **...**

This decrease in width was allowed because of the value previously set for the **minimumSize** properties for these two components (0.6 times the preferred width of component1).

However, the left-most component (**component1**) doesn't decrease in width, because its minimum width was set to be the same as its preferred width.

Is minimum width *really* minimum width?

Figure 5 shows that if you continue to decrease the width of the **JFrame**, none of the components decrease in width after they hit the width specified by the **width** dimension of their **minimumSize** property.

Figure 5 The Result of Manually Decreasing the Width of the JFrame Even Further

As shown in Figure 5, once all three components hit their minimum widths, the right-most component gets clipped by the edge of the **JFrame**. This occurs when the **JFrame** is no longer wide enough to accommodate all three components at their minimum widths.

Finally, Figure 6 shows that as you continue to decrease the width of the **JFrame**, after the right-most component disappears, the middle component gets clipped.

Figure 6 The Result of Manually Decreasing the Width of the JFrame Even Further

The width of the struts does not decrease

Note that the space between the components defined by the struts does not decrease.

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.

The controlling class

Listing 1 shows the beginning of the controlling class and the **main()** method. You have seen code like this in several previous lessons. Therefore, I won't discuss it further here.

```
class Swing17 extends JFrame{

  public static void main(String
args[]) {
      new Swing17();
  }//end main()

Listing 1
```

The constructor

The beginning of the constructor is shown in Listing 2. By now, this is also plain vanilla code. I won't discuss it further. I will simply let the comments speak for themselves.

```
  Swing17(){//constructor

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

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

    //Instantiate three JLabel objects,
    // make them green.
    JLabel component1 =
                    new
JLabel("component1");
    component1.setOpaque(true);
    component1.setBackground(Color.green);

    JLabel component2 =
                    new
```

```
JLabel("component2");
    component2.setBackground(Color.green);
    component2.setOpaque(true);

    JLabel component3 =
                        new
JLabel("component3");
    component3.setBackground(Color.green);
    component3.setOpaque(true);

Listing 2
```

Modify the preferredSize property

The code to modify the value of the **preferredSize** property is shown in Listing 3. This is the beginning of the code that is new to this lesson.

```
    //Get and modify the preferred size of
    // component1
    Dimension preferredSize =
                component1.getPreferredSize();
    preferredSize.width =
              (int)(preferredSize.width*1.1);
    component1.setPreferredSize(
                               preferredSize);

Listing 3
```

Invoke getPreferredSize()

The code begins by invoking the **getPreferredSize()** method on the reference to the **JLabel** object. The reference to the **Dimension** object that is returned is stored in the variable named **preferredSize**. *(I did it this way because I wanted to take advantage of the default preferred size in setting a new value for the preferred size. In other words, I didn't want to have to start from ground zero in coming up with a new preferred size.)*

Change the width dimension

The next step is to increase the value of the instance variable named **width** in the **Dimension** object by a factor of 1.1.

Invoke setPreferredSize()

The third step in Listing 3 is to invoke **setPreferredSize()** on the **JLabel** component passing the modified **Dimension** object as a parameter.

A permanent change

This causes a permanent change in the value of the **preferredSize** property for that component.

This change is reflected in Figure 1 where the width of the left-most component is about ten-percent greater than the width required to display its caption.

Calculate new values for other components

The code in Listing 4 calculates values for the minimum and maximum widths that will be used later for setting the **minimumSize** and **maximumSize** properties for the other two components. *(Again, I did it this way because I wanted to cause these property values to be based on the preferred size of component1.)*

```
    Dimension minSize = new Dimension();
    minSize.width =
              (int)(preferredSize.width*0.6);
    minSize.height = preferredSize.height;

    Dimension maxSize = new Dimension();
    maxSize.width =

              (int)(preferredSize.width*1.2);
    maxSize.height = preferredSize.height;

Listing 4
```

No surprises here

Since you already know that **width** and **height** are public instance variables of objects of the class **Dimension**, you should find the code in Listing 4 to be completely straightforward.

Invoke setMinimumSize() and setMaximumSize()

The code in Listing 5 invokes the **setMinimumSize()** and **setMaximumSize()** methods on each of the three objects to set the **minimumSize** and **maximumSize** properties of the components to the predetermined values.

```
    //Set minimum and maximum sizes for all
    // three components.  Cause all three to
    // be the same for component1
    component1.setMinimumSize(preferredSize);
    component2.setMinimumSize(minSize);
    component3.setMinimumSize(minSize);

    component1.setMaximumSize(preferredSize);
    component2.setMaximumSize(maxSize);
    component3.setMaximumSize(maxSize);
```

```
Listing 5
```

These properties are set to match the preferred size for the left-most component (**component1**). They are set to the values calculated above for the other two components.

Place components in the Box container

Listing 6 shows the code that adds the three components to the display with a strut separating the three components from each other and from the two ends of the box. You have seen this code in previous lessons, so I won't discuss it further.

```
    //Add the components to the Box. Insert
    // horizontal struts between the
    // components to control the minimum
    // spacing between them.
    aBox.add(Box.createHorizontalStrut(10));
    aBox.add(component1);
    aBox.add(Box.createHorizontalStrut(10));
    aBox.add(component2);
    aBox.add(Box.createHorizontalStrut(10));
    aBox.add(component3);
    aBox.add(Box.createHorizontalStrut(10));


Listing 6
```

The remaining code is completely uninteresting. You can view that code in Listing 7.

Summary

Every component that extends **JComponent** has the following three properties:

- preferredSize
- minimumSize
- maximumSize

In this lesson, I have shown you how to control the values of these three properties. I have also shown you how a **BoxLayout** behaves with respect to these properties when the physical size of the container is changed. I also pointed out that different layout managers behave differently with respect to these three properties.

What's Next?

In the next lesson, I will show you how to create your own invisible spacers. I will use them in a sample program that takes some of the mystery out of glue and struts.

The sample program will also demonstrate that **BoxLayout** is not confined to being used with a **Box** container. In fact, just for fun, I will use a **JButton** as a container with a **BoxLayout** manager.

Complete Program Listing

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

```
/*File Swing17.java
Copyright 2000, R.G.Baldwin
Rev 8/11/00

Illustrates the use of minimum, preferred,
and maximum sizes.

Three JLabel components are placed in a
horizontal line in a Box layout in a JFrame.
The components are separated from each other
and from the end of the frame by struts.
Each, strut is ten pixels wide.

The left-most component has its preferred
width set to 1.1 times the default preferred
width. The minimum and maximum sizes for this
component are fixed at the preferred size.
Therefore, its size doesn't change when the
user resizes the JFrame.

The two right-most components have their
minimum widths set to 0.6 times the preferred
width of the left-most component.  Their
maximum widths are set to 1.2 times the
preferred width of the left-most component.

Manually expand the size of the JFrame to see
that the widths of the two right-most
components increase to their maximum sizes
and then don't increase any further after
that.

Manually decrease the size of the JFrame to
see that the two right-most components
decrease in width until they reach their
minimum size and then don't decrease in size
any further.  Beyond this point, the right-
most component gets clipped by the edge of
the JFrame when the JFrame is no longer large
enough to accommodate all three components at
their minimum sizes.

When the right-most component disappears, the
middle component gets clipped. The space
between the components defined by the struts
does not decrease.
```

```
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 Swing17 extends JFrame{

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

  public static void main(String args[]) {
      new Swing17();
  }//end main()
  //---------------------------------------//

  Swing17(){//constructor

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

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

    //Instantiate three JLabel objects,
    // make them green.
    JLabel component1 =
                    new JLabel("component1");
    component1.setOpaque(true);
    component1.setBackground(Color.green);

    JLabel component2 =
                    new JLabel("component2");
    component2.setBackground(Color.green);
    component2.setOpaque(true);

    JLabel component3 =
                    new JLabel("component3");
    component3.setBackground(Color.green);
    component3.setOpaque(true);

    //Get and modify the preferred size of
    // component1
    Dimension preferredSize =
               component1.getPreferredSize();
    preferredSize.width =
              (int)(preferredSize.width*1.1);
    component1.setPreferredSize(
                              preferredSize);

    //Calculate minimum and maximum sizes
    // based on preferred size of component1.
```

```
    Dimension minSize = new Dimension();
    minSize.width =
              (int)(preferredSize.width*0.6);
    minSize.height = preferredSize.height;

    Dimension maxSize = new Dimension();
    maxSize.width =
              (int)(preferredSize.width*1.2);
    maxSize.height = preferredSize.height;

    //Set minimum and maximum sizes for all
    // three components.  Cause all three to
    // be the same for component1
    component1.setMinimumSize(preferredSize);
    component2.setMinimumSize(minSize);
    component3.setMinimumSize(minSize);

    component1.setMaximumSize(preferredSize);
    component2.setMaximumSize(maxSize);
    component3.setMaximumSize(maxSize);

    //Add the components to the Box. Insert
    // horizontal struts between the
    // components to control the minimum
    // spacing between them.
    aBox.add(Box.createHorizontalStrut(10));
    aBox.add(component1);
    aBox.add(Box.createHorizontalStrut(10));
    aBox.add(component2);
    aBox.add(Box.createHorizontalStrut(10));
    aBox.add(component3);
    aBox.add(Box.createHorizontalStrut(10));

    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 Swing17
```

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-