Swing from A to Z

The border Property

Part 2, BevelBorder and EmptyBorder

By Richard G. Baldwin

Java Programming, Lecture Notes # 1021

August 28, 2000

- Preface
- Introduction
- Sample Program
- Interesting Code Fragments
- Summary
- Where To From Here?
- Complete Program Listing

Preface

This lesson is Part 2 of a miniseries designed to illustrate the *border* property and the use of that property to construct fancy borders on Swing components.

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, without losing your place.

Recommended supplementary reading

It is strongly recommended that you study Part 1 entitled Swing from A to Z, The border Property, Part1, EtchedBorder before embarking on this part.

It is also recommended that you read the following lesson, which you will find at *Baldwin's Java Programming Tutorials*.

> **Swing, Hidden Buttons with Icons, Icon Images, Borders, Tool Tips, Nested Buttons, and Other Fun Stuff**

That lesson illustrates some very interesting uses of borders with buttons to cause buttons to rise up from the surface when you point at them with the mouse.

Introduction

What's in this lesson?

This is the second of several sequential lessons that emphasize an understanding of the *border* property along with the use of that property to construct components having different border styles.

Several parts are needed

Because of the large amount of material involved, I have decided to break this discussion into several parts.

What was in Part 1?

Part 1 set the background for future discussions, and also dealt specifically with the use of the **EtchedBorder** class to create borders of a style known as etched borders.

What about Part 2?

This lesson, Part 2, deals specifically with the use of the **BevelBorder** class to create two different styles of beveled, three-dimensional borders.

This lesson also illustrates the use of the **EmptyBorder** class to provide blank space around the component to which the border is being applied.

Sample Program

A screen shot

The name of the sample program that I will discuss to illustrate borders is **Swing13**.

A screen shot of the GUI that is produced when the program is started is shown below.

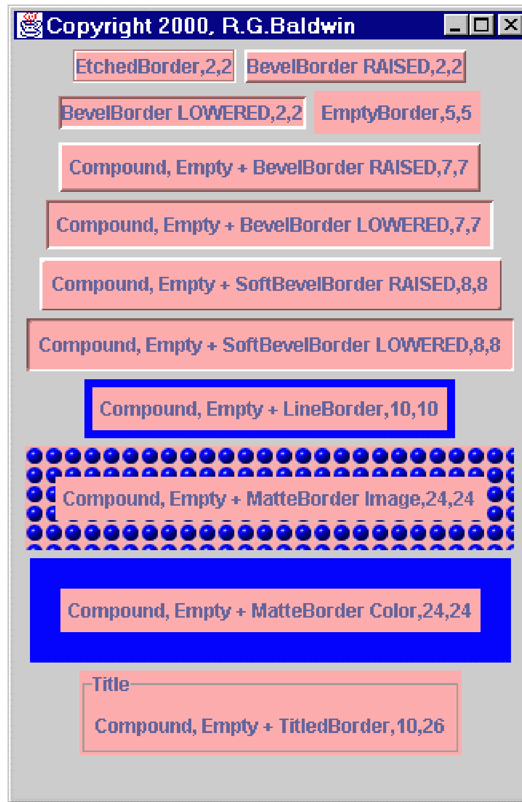

Figure 1 A screen shot of the program running.

Twelve JLabel objects

The program creates and displays twelve different **JLabel** objects, applying a different border style to each of them. As you can see, with Swing, there are many different ways to display a label simply by taking control of the **border** property.

The first object in the upper left-hand corner was discussed in Part 1.

This lesson will discuss the right-most object in the first row, and both objects in the second row.

Interesting Code Fragments

I will continue discussing the program in fragments. A complete listing of the program is provided in Listing 4. I will skip those parts of the program that were discussed in Part 1.

The constructor

Following some preliminary material, the constructor, which you can view in its entirety in Listing 4, contains twelve fairly complex statements. Each of the twelve statements causes a **JLabel** object to be displayed with a specified border style.

The text displayed in each **JLabel** object label consists of the concatenation of a specified text value, and the left and top insets for the specified border style.

Open in another browser window

At this point, it may be helpful for you to open a copy of this lesson in another browser window so you can see the screen shot of the GUI while I discuss the different border styles.

BevelBorder RAISED,2,2

Listing 1 shows the code fragment that caused the label in the upper-right corner of the screen shot to be added to the *contentPane* of the **JFrame** container.

```
  getContentPane().add(makeLabel(
  "BevelBorder RAISED",new BevelBorder(
                   BevelBorder.RAISED)));

Listing 1
```

The specified text content

The first parameter to the **makeLabel()** method is highlighted in Italics in the code fragment of Listing 1. This parameter eventually becomes part of the text content of the label.

This string is concatenated with the inset values for the specified border style. The concatenated string is set into the *text* property for the label. This, in turn, causes the combination of the specified text string and the inset values to be displayed as the text value for the label.

Thus, the text displayed in the upper right-hand label is

BevelBorder RAISED,2,2

This indicates that the top and left insets for the **BevelBorder** style are each two pixels.

The specified border style

The second parameter to the **makeLabel()** method, highlighted in boldface in Listing 1, is a reference to a new object of the class **BevelBorder**.

Here is what Sun has to say about the **BevelBorder** class.

A class which implements a simple 2 line bevel

border.

As is sometimes the case with the Sun documentation, this isn't very illuminating.

Constructors

There are three constructors for the **BevelBorder** class. I used the simplest of the three in Listing 1.

Raised or lowered?

All three constructors allow you to provide an integer constant that specifies whether the component should appear to protrude out of the screen (RAISED) or be depressed into the screen (LOWERED).

For this component, I specified that the label should be RAISED. As you can see from the screen shot, this label appears to protrude out of the screen.

Color, highlight, and shadows

The other two constructors for the **BevelBorder** class allow you to customize the color, highlight, and shadow information, presumably to produce a three-dimensional border that is more to your liking.

BevelBorder LOWERED,2,2

Listing 2 shows the code that caused a "LOWERED" version of the label with the **BevelBorder** to be added to the *contentPane* of the **JFrame** object.

```
  getContentPane().add(makeLabel(
  "BevelBorder LOWERED",new BevelBorder(
                   BevelBorder.LOWERED)));

Listing 2
```

This code fragment is exactly like the previous code fragment except for the value of the parameter passed to the constructor for the **BevelBorder** class (LOWERED instead of RAISED).

See it in the screen shot

This label appears in the screen shot on the left side of the second row.

The size of the label

As was the case in the previous lesson, the default size of the label with the beveled border is barely large enough to accommodate its text, particularly on the ends.

The inset values

The inset of the left border and the inset of the top border each have a value of two pixels for the beveled border.

Although I didn't display the inset for the right border and the bottom border, they are probably the same.

EmptyBorder,5,5

A solution, or at least the beginning of a solution, to the size problem is shown by the code in Listing 3.

```
getContentPane().add(makeLabel(
  "EmptyBorder",new EmptyBorder(
                    5,5,5,5)));


Listing 3
```

The code in Listing 3 caused the label shown in the right-hand side of the second row of the screen shot to be added to the *contentPane* container. This label has an **EmptyBorder**.

What does Sun have to say?

Here is what Sun has to say about this border.

> A class which provides an empty, transparent border which takes up space but does no drawing.

Regardless on Sun's brevity on the matter, this is an important border style.

The constructor for EmptyBorder

The arguments to the constructor allow you to specify a band of blank space on each of the four sides of the component. You can specify different values for each side if you want to.

When used alone, this border provides an improvement in the appearance of a label with an opaque background where the text is normally crammed up against the ends.

Very useful with CompoundBorder

As we will see in the next lesson, this border style can be used with a **CompoundBorder** object to produce components whose appearance is much improved over the first three labels in the screen shot.

Two overloaded constructors

Two overloaded constructors are available for the **EmptyBorder** class.

One constructor accepts a single parameter of the **Insets** class. The other constructor accepts four **int** parameters.

Whichever constructor you use, you specify the amount of blank space in pixels on each of the four sides of the component. In Listing 3, I specified a blank space of five pixels on each of the four sides of the **JLabel** object.

Summary

This lesson has continued the discussion of the *border* property, and has shown you how to use that property to cause fancy borders to be rendered on Swing components.

So far, four different standard borders have been shown:

- EtchedBorder
- BevelBorder RAISED
- BevelBorder LOWERED
- EmptyBorder

When the first three of these were applied to a **JLabel** object, the results weren't very attractive because the ends of the text were too close to the edge of the label.

The third border can be used to provide empty space on each of the four sides of the component, which can be used to solve the problem described above.

Where To From Here?

The real solution to the problem described above involves the combined use of the **EmptyBorder** and the **CompoundBorder**, which will be the subject of the next lesson.

Complete Program Listing

A complete listing of the program is shown in Listing 4.

```
/*File Swing13
Rev 3/28/00
Copyright 2000, R.G.Baldwin


Illustrates the border property.  This
program creates and displays several
different border types surrounding a
JLabel object.

```

```
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 Swing13 extends JFrame{

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

  public static void main(String args[]) {
      new Swing13();
  }//end main()
  //-----------------------------------------//

  //The purpose of this method is to create
  // and return an opaque pink JLabel with
  // a border.  The text content of the
  // lable is provided as the first
  // parameter.  The border type is provided
  // as the second parameter.  When the
  // label is displayed, the left and top
  // insets are displayed following the
  // text content of the label.
  JLabel makeLabel(
           String content,Border borderType){

    JLabel label = new JLabel();
    label.setBorder(borderType);
    label.setOpaque(true);
    label.setBackground(Color.pink);

    label.setText(content + ","
    +label.getInsets().left + ","
    +label.getInsets().top);

    return label;

  }//end makeLabel()
  //-----------------------------------------//

  Swing13(){//constructor

    getContentPane().setLayout(
                       new FlowLayout());

    getContentPane().add(makeLabel(
         "EtchedBorder",new EtchedBorder()));
    getContentPane().add(makeLabel(
        "BevelBorder RAISED",new BevelBorder(
                       BevelBorder.RAISED)));
    getContentPane().add(makeLabel(
       "BevelBorder LOWERED",new BevelBorder(
                      BevelBorder.LOWERED)));
    getContentPane().add(makeLabel(
         "EmptyBorder",new EmptyBorder(
                              5,5,5,5)));
   getContentPane().add(makeLabel(
      "Compound, Empty + BevelBorder RAISED",
      new CompoundBorder(new BevelBorder(
      BevelBorder.RAISED),new EmptyBorder(
                             5,5,5,5))));
    getContentPane().add(makeLabel(
     "Compound, Empty + BevelBorder LOWERED",
     new CompoundBorder(new BevelBorder(
```

```
      BevelBorder.LOWERED),new EmptyBorder(
                              5,5,5,5))));

    getContentPane().add(makeLabel(
      "Compound, Empty + SoftBevelBorder " +
      "RAISED",
      new CompoundBorder(new SoftBevelBorder(
      SoftBevelBorder.RAISED),new EmptyBorder(
                              5,5,5,5))));
    getContentPane().add(makeLabel(
      "Compound, Empty + SoftBevelBorder " +
      "LOWERED",
      new CompoundBorder(new SoftBevelBorder(
      SoftBevelBorder.LOWERED),
                 new EmptyBorder(
                              5,5,5,5))));
    getContentPane().add(makeLabel(
      "Compound, Empty + LineBorder",
      new CompoundBorder(new LineBorder(
      Color.blue,5),new EmptyBorder(
                              5,5,5,5))));
    getContentPane().add(makeLabel(
      "Compound, Empty + MatteBorder Image",
      new CompoundBorder(new MatteBorder(
      19,19,19,19,new ImageIcon(
      "blue-ball.gif")),new EmptyBorder(
                              5,5,5,5))));

    getContentPane().add(makeLabel(
      "Compound, Empty + MatteBorder Color",
      new CompoundBorder(new MatteBorder(
               19,19,19,19,Color.blue),
                 new EmptyBorder(5,5,5,5))));



    getContentPane().add(makeLabel(
      "Compound, Empty + TitledBorder",
      new CompoundBorder(new TitledBorder(
        "Title"),new EmptyBorder(5,5,5,5))));

    setTitle("Copyright 2000, R.G.Baldwin");
    setSize(329,500);
    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 Swing13
```

Listing 4

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-