# Swing from A to Z

# The border Property

# Part 5, Nested Compound Borders

***By Richard G. Baldwin***

Java Programming, Lecture Notes # 1024

September 18, 2000

---

## Preface

This lesson is Part 5 in a miniseries of several parts designed to illustrate the *border* property and the use of that property to construct fancy borders on Swing components.

I recommend that you study previous lessons on the border property beginning with The border Property, Part 1 before embarking on this lesson.

I also recommended that in addition to studying this set of lessons, you also study my earlier lessons on Swing, which are available at Gamelan.  A consolidated index to those earlier lessons is available at my personal web site.

### 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.

## Introduction

This lesson illustrates the use of nested compound borders.

## Sample Program

### A screen shot

The name of the sample program that I will discuss to illustrate nested compound borders is **Swing14**.

A screen shot of the GUI that is produced by that program is shown in Figure 1.

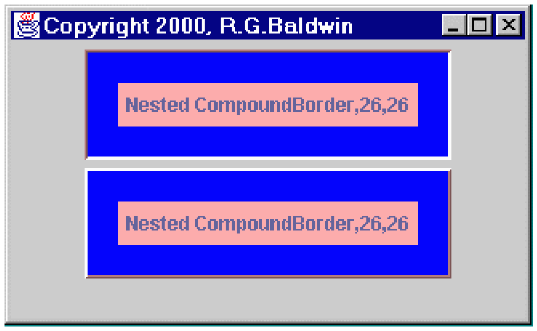
Copyright 2000, R.G.Baldwin
Nested CompoundBorder,26,26
Nested CompoundBorder,26,26

**Figure 1 Two JLabel objects**

The program creates and displays two different **JLabel** objects, applying a different border style to each of them.

## Interesting Code Fragments

I will discuss the program in fragments. A complete listing of the program is provided in Listing 4 near the end of this listing

This program is very similar to the program named **Swing13** that I discussed in detail in the first four parts of this miniseries. Therefore, I will skip those parts of the program that were discussed in the previous lessons

### The top Swing component

Listing 1 shows the code fragment that prepares the border for the top Swing component in the screen shot of Figure 1.

```
    CompoundBorder theBorder = 
      new CompoundBorder(
    new BevelBorder(BevelBorder.LOWERED),
     new CompoundBorder(
         new MatteBorder(19,19,19,19,
             Color.blue),
                 new EmptyBorder(5,5,5,5)));


Listing 1
```

## A CompoundBorder object

This statement instantiates an object of the **CompoundBorder** class, which will be used as the border for a **JLabel** object.

## The constructor

The constructor for **CompoundBorder** requires two parameters, each of which must be references to **Border** objects (**Border** is an interface).

The code fragment in Listing 1 shows the first parameter in red, and shows the second parameter in green. I did this to make them easier to separate visually.

## The *outside* border

The first parameter, or outside border, is a *LOWERED* **BeveledBorder** object. This causes the blue area in the screen shot of Figure 1 to appear to be depressed into the surface of the **JFrame**.

## The *inside* border

The second parameter, or inside border, is itself a reference to a **CompoundBorder** object. Thus, a **CompoundBorder** is nested inside of another **CompoundBorder**. The constructor for this object also requires two parameters.

## A MatteBorder object

The first parameter to the constructor for the nested **CompoundBorder** object is a reference to a solid blue **MatteBorder** object. This is what produces the blue area in the screen shot.

## An EmptyBorder object

The second parameter to the nested **CompoundBorder** object is a reference to an **EmptyBorder** object. This produces a blank margin five pixels in width between the original **JLabel** object and the solid blue matte border.

## Adding the component to the *contentPane*

Listing 2 shows the invocation of the **makeLabel()** method that applies the **Border** object constructed above to a **JLabel** object and adds it to the *contentPane* for later rendering. (See Part 1 of this miniseries on borders for an explanation of this method.)

```
    getContentPane().add(makeLabel(
         "Nested CompoundBorder",theBorder));
```

**Listing 2**

### The bottom component

Listing 3 shows the code fragment that prepares the border for the bottom component in the screen shot.

```
     theBorder = new CompoundBorder(
       new BevelBorder(BevelBorder.RAISED),
         new CompoundBorder(new MatteBorder(
           19,19,19,19,Color.blue),
                    new EmptyBorder(5,5,5,5)));


Listing 3
```

This fragment is the same as the one for the top component, except that this fragment uses a *RAISED* **BevelBorder** for the outside border instead of a *LOWERED* **BevelBorder**.

## Summary

In this miniseries on borders, I have introduced you to each of the standard **Border** classes, and have illustrated one or more variations on each of them.

I have also pointed out that compound borders can be nested to produce very complex borders, and have illustrated two different nested compound borders.

I have mentioned that if the standard borders won't fulfill your needs, you can define your own class that implements the **Border** interface and use an object of that class for your custom border.

## Where To From Here?

I have one more topic to cover before I leave this miniseries on borders: the **BorderFactory** class.  I will cover that topic in the next lesson.

## Complete Program Listing

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

```
/*File Swing14
Rev 3/30/00
Copyright 2000, R.G.Baldwin


Illustrates nesting of CompoundBorder
objects.  This program creates and
displays two different border styles
constructed by nesting CompoundBorder
objects.



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 Swing14 extends JFrame{


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


  public static void main(String args[]) {
      new Swing14();
  }//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()
  //---------------------------------------//


  Swing14(){//constructor


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


    CompoundBorder theBorder =
      new CompoundBorder(
        new BevelBorder(BevelBorder.LOWERED),
          new CompoundBorder(
            new MatteBorder(19,19,19,19,
              Color.blue),new EmptyBorder(
                                  5,5,5,5)));


    getContentPane().add(makeLabel(
         "Nested CompoundBorder",theBorder));


    theBorder = new CompoundBorder(
      new BevelBorder(BevelBorder.RAISED),
        new CompoundBorder(new MatteBorder(
          19,19,19,19,Color.blue),
                  new EmptyBorder(5,5,5,5)));


    getContentPane().add(makeLabel(
         "Nested CompoundBorder",theBorder));

    setTitle("Copyright 2000, R.G.Baldwin");
    setSize(329,200);
    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 Swing14
```

**Listing 4**

## 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-