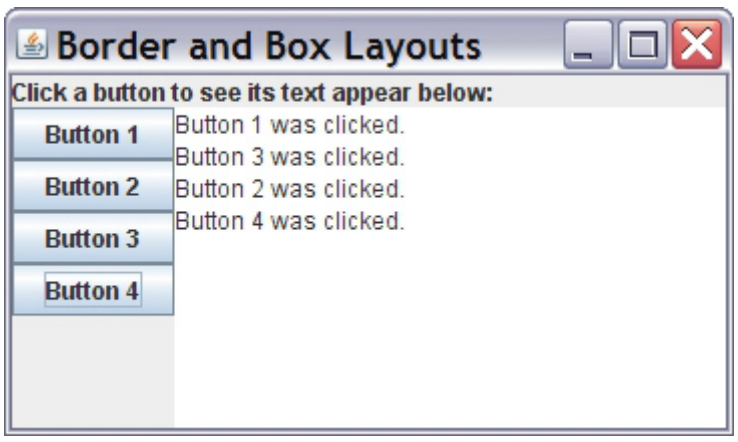
Chapter 3: Panels

Panels

Now that you've seen how to use the four most common layout managers in Java, let's see how we can do even more with them by using panels.

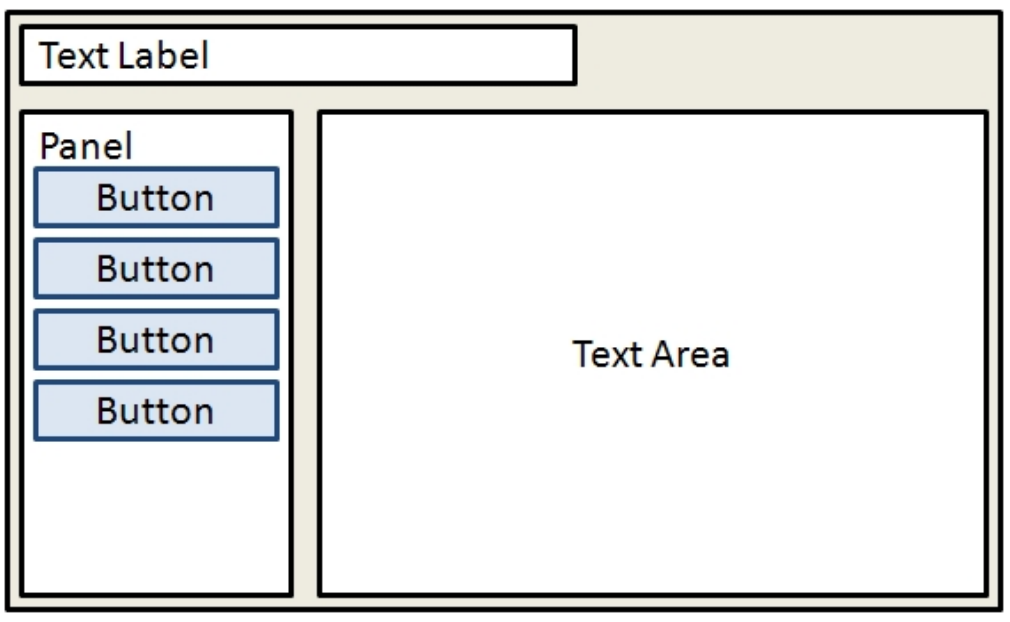
I said earlier that I'd explain how to use a *panel* to add multiple components to a layout region that only accepts one. A panel is a component that holds other components. Not only do panels hold multiple other components, but they can each have their own layout managers that can be different from the frame's manager. We build panels from Java's JPanel class. I'll show you how in just a minute. First, let's decide what we want to do in our window.

How about if we create a window that looks like this?



Combined layouts

This window has a text label at the top with instructions, four buttons on the left, and a big text area on the right where text will appear when we click the buttons. To make this happen, we'll need to use two different layouts and a panel. Here's how we'll organize it:



Layout design

For the panel, we'll use a BoxLayout with the components stacked vertically. The panel will be just wide enough for the buttons, and the width of the panel will determine the width of the West region of the outer layout. The North region will take on the height of the text label, and the Center region with its text area will fill the rest of the window.

For our program, let's start with the shell of the program we used for the BorderLayout example, before we added any buttons. It looks like this:

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;



public class LayoutCombo

{

    private JFrame frame;



    public static void main (String[] args)

    {

      LayoutCombo guiLayout = new LayoutCombo();

      guiLayout.start();

    }



    public void start()

    {

      frame = new JFrame("Border and Box Layouts");

      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      Container contentPane = frame.getContentPane();

      contentPane.setLayout(new BorderLayout());



      frame.pack();

      frame.setVisible(true);

    }

}
```

Now, what data objects will we need for our window? We know we'll need a text label and four buttons. We already know how to create them. But how do we create a panel and a large text area? We'll use the JPanel class to create a panel, and (appropriately enough) the JTextArea class for the text area. Let's go ahead and declare the variables we'll use at the top of our class:

```
private JLabel label;

private JButton button1, button2, button3, button4;

private JPanel panel;

private JTextArea textArea;
```

These declarations go with the frame declaration that is already in the program.

Now that we have the variables we need, let's go down to the start() method and put all these details together. First, right after the line that sets the layout for the window pane, let's add this to create our label and place it at the top of the screen:

```
label = new JLabel("Click a button to see its text appear below:");

contentPane.add(label, BorderLayout.NORTH);
```

Next, let's create our panel, set its layout, and add our buttons to it. The JPanel constructor does not need any parameters, and we use the same add() method that contentPane uses to add components to a panel. So setting up the panel looks like this:

```
panel = new JPanel();

panel.setLayout(new BoxLayout(panel,BoxLayout.Y_AXIS));



button1 = new JButton("Button 1");

panel.add(button1);

button2 = new JButton("Button 2");

panel.add(button2);

button3 = new JButton("Button 3");

panel.add(button3);

button4 = new JButton("Button 4");

panel.add(button4);

contentPane.add(panel, BorderLayout.WEST);
```

That code should go right after our label creation lines.

We'll add code for the text area next. The default JTextArea constructor creates an area that will take up whatever space is available in the window. I want to create an area a little bit bigger than that so we have some extra room. Here's the constructor call I'm going to use:

```
textArea = new JTextArea(10,25);
```

This constructor call will create a text area that is 10 rows tall and 25 columns wide. When setting this up, Java calculates a row height based on the height of the font used in the window. The column width gets set for the width of the letter *m*. Ten rows and 25 columns give me a text area about the size we want. Once we've created the text area, we can add it to the window:

```
contentPane.add(textArea, BorderLayout.CENTER);
```

If we run the program now, we'll see the same window we saw in the image at the beginning of the section, minus the text in the text area. We've created all our components, and they look right. The only problem is that they don't do anything. (If you have problems running it at this point and want to compare it to my code, click here.)

Solution: Try not to peek!

If we have all the components, what's left to do to make it work? Remember the listeners we attached to buttons in the last lesson? We need to do the same for each of these four buttons. Before we do, though, we need to take care of a couple of related items.

Adding action listeners means we need two other items as well. First, we need to add the code `implements ActionListener` to the first line of our class declaration, just like we did in the last lesson.

Second, we need to add a method named actionPerformed() to our class because that's what the ActionListener interface requires. That method will contain the action we want to perform when a user clicks a button. In this case, we'll need the method to add text to the text area when each button gets clicked. And that means we need to know which button was clicked.

How do we do that? This is where the ActionEvent that we receive in the actionPerformed() method comes into play. An ActionEvent object has a method named getSource() that will tell us the object that initiated the action. Then we can compare it to our buttons to see which one was clicked.

Here's one way to write that method:

```
public void actionPerformed(ActionEvent e)

{

    if (e.getSource() == button1)

    textArea.append("Button 1 was clicked.\n");

    else if (e.getSource() == button2)

    textArea.append("Button 2 was clicked.\n");

    else if (e.getSource() == button3)

    textArea.append("Button 3 was clicked.\n");

    else if (e.getSource() == button4)

    textArea.append("Button 4 was clicked.\n");

    else

    textArea.append("Should not get here!\n");}
```

Each *if* calls the ActionEvent's getSource() method, then compares it to one of our buttons. If they match, we add the appropriate text to our text area using its append() method.

Now we're finally ready to add our listeners. We need to add them to the buttons, just like we did in the last lesson:

```
button1.addActionListener(this);

button2.addActionListener(this);

button3.addActionListener(this);

button4.addActionListener(this);
```

You can add these lines almost anywhere before the window is displayed. I added the line for each button between the line that created the button and the one that added it to the panel.

The window should now work as advertised. Each click on one of the buttons should add a line of text to the text area naming the button that was just clicked. If your program does not work that way, you can compare it to my latest version here or ask me about it in the Discussion Area.

Solution: Try not to peek!