Chapter 2: Java's Layouts

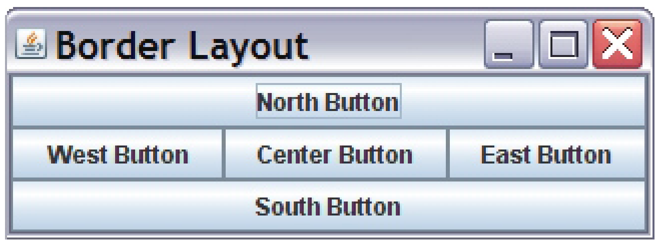
Java's Layouts

When it comes to managing components in a window, Java uses *layout managers*. Java has eight of them, and third-party authors have created others. Each layout manager produces a specific window layout that we can fine-tune to make our windows look the way we want. For this lesson, we'll stick to Java's layout managers. If you want to look at some third-party managers and compare them, they are easy to find by doing an online search for "java layout manager."

We're only going to look at the four most-used Java layout managers in this lesson: *BorderLayout*, *BoxLayout*, *FlowLayout*, and *GridLayout*. Here's a brief description of each one, along with code that shows how to use them.

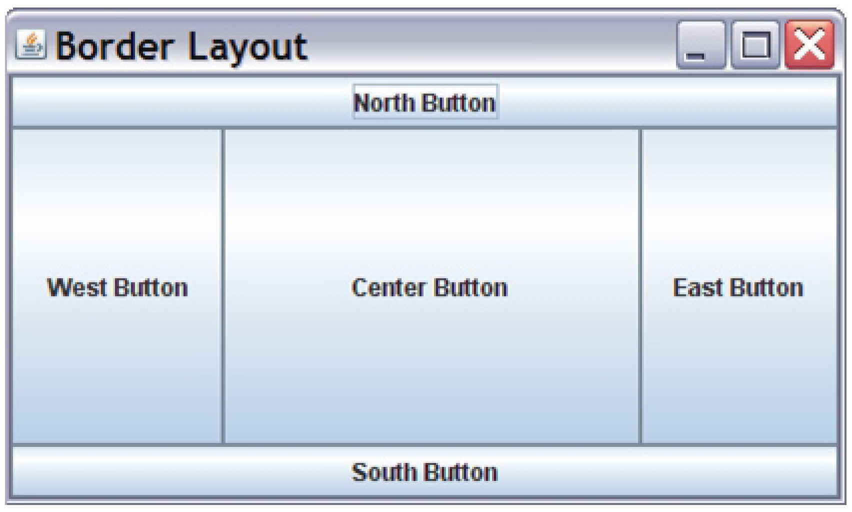
BorderLayout

BorderLayout divides its window into five parts that have fixed positions relative to each other. Each part can hold one component, and the size of the part controls the component's size. This figure shows the five parts, each with a button added to it:



Java's BorderLayout

We can change the size of each part, but not its position. The north and south areas always keep the same height, but we can change their width. The east and west areas keep the same width, but we can change their height. The center area is the only one whose height and width can both change. For example, if I stretch the above window, here is what it looks like:



Stretched BorderLayout

This may seem like a rigid layout, but it is one of the most commonly used. It's actually more flexible than it might seem, since Java hides any empty outer areas (North, South, East, or West). If there's nothing in them, they do not appear when Java displays the window.

Let's look at the code it takes to generate a window using the BorderLayout manager. Specifically, we'll look at the code to generate the first window we saw above. Right now, we'll focus on the display, and then we'll come back to the listeners and the actions that the buttons perform later.

We can start with the basic shell that we used in the last lesson, with a main() method that creates an object and a start() method that builds the window. Our beginning empty class looks like this:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LayoutBorder
{
  private JFrame frame;

  public static void main (String[] args)
  {
    LayoutBorder guiLayout = new LayoutBorder();
    guiLayout.start();
  }

  public void start()
  {
    frame = new JFrame("Border Layout");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container contentPane = frame.getContentPane();

    frame.pack();
    frame.setVisible(true);
  }
}
```

Other than the window title and some class and variable names, we saw all of this code in the last lesson. As it is, this code does not add any components to the window—it only produces an empty window. So let's add some code that will use the BorderLayout manager to put buttons into the five areas of the window.

The first thing we need to do is tell Java that we want to use a layout manager for our window and which manager we want to use. Every GUI Container object contains a *setLayout()* method to assign a layout manager to the container. Its single argument is the layout manager to assign. We will use contentPane's setLayout() method with a BorderLayout object like this:

```
contentPane.setLayout(new BorderLayout());
```

This line has to follow the declaration of contentPane, so we'll add it as the fourth line of our start method.

Once we have the layout manager, we can add buttons to the pane like we did in the previous lesson. When we're dealing with the BorderLayout manager, though, we need to add an additional parameter that tells the layout manager which area to put the button in. To get the window we saw above, we need to add one button to each region, with the region's name as part of the button name, like this:

```
contentPane.add(new JButton("North Button"),

                BorderLayout.NORTH);

contentPane.add(new JButton("South Button"),

                BorderLayout.SOUTH);

contentPane.add(new JButton("East Button"),

                BorderLayout.EAST);

contentPane.add(new JButton("West Button"),

                BorderLayout.WEST);

contentPane.add(new JButton("Center Button"),

                BorderLayout.CENTER);
```

These lines follow the setLayout() call and precede the call to the pack() method. That's all there is to it! Our finished start() method looks like this:

```
public void start()

{

    frame = new JFrame("Border Layout");

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    Container contentPane = frame.getContentPane();

    contentPane.setLayout(new BorderLayout());


    contentPane.add(new JButton("North Button"),

                    BorderLayout.NORTH);

    contentPane.add(new JButton("South Button"),

                    BorderLayout.SOUTH);

    contentPane.add(new JButton("East Button"),

                    BorderLayout.EAST);

    contentPane.add(new JButton("West Button"),

                    BorderLayout.WEST);

    contentPane.add(new JButton("Center Button"),

                    BorderLayout.CENTER);


  frame.pack();

  frame.setVisible(true);

}
```

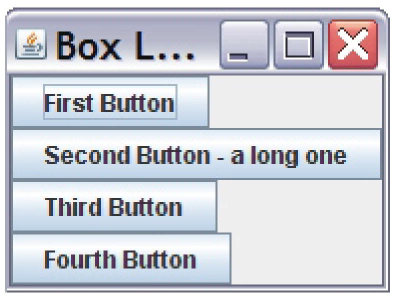
Go ahead and run this program now. You should see the BorderLayout window I showed you above. Please let me know in the Discussion Area if you see something else.

Before we move on to the next layout manager, there are just a few notes on BorderLayout that I'd like to review:

1. Each BorderLayout region can only contain one component.
2. If you leave one of the outer areas empty, it won't show up in the window. So you can have a window with North, West, and Center areas but no East or South areas, or any other combination that makes sense in your program.
3. Even though a region can only hold one component, don't think of this as a limitation. Later in this lesson, we'll see how to make a component (a *panel*) that can hold other components and even use a different layout manager. Then we can add that external component (with its inner components) to a BorderLayout region, effectively overcoming the one-component limit.

BoxLayout

The next layout manager we'll look at is BoxLayout. BoxLayout allows each component to keep its own size. It stacks components in the order we add them, and we can choose to stack them either vertically or horizontally. We'll use BoxLayout's vertical organization more often than its horizontal layout. Here's an example of how it looks:



Java's BoxLayout

The code for the above BoxLayout window is very similar to the code we used for the BorderLayout window. The only differences are in the lines we introduced in the last section that controlled the BorderLayout. Most of these differences affect the parameters. Let me show you what I mean.

The first line we need to look at is the one that connects the layout manager to the container it will manage (contentPane, in this case). For a BoxLayout manager, that line looks like this:

```
contentPane.setLayout(new BoxLayout(contentPane,BoxLayout.Y_AXIS));
```

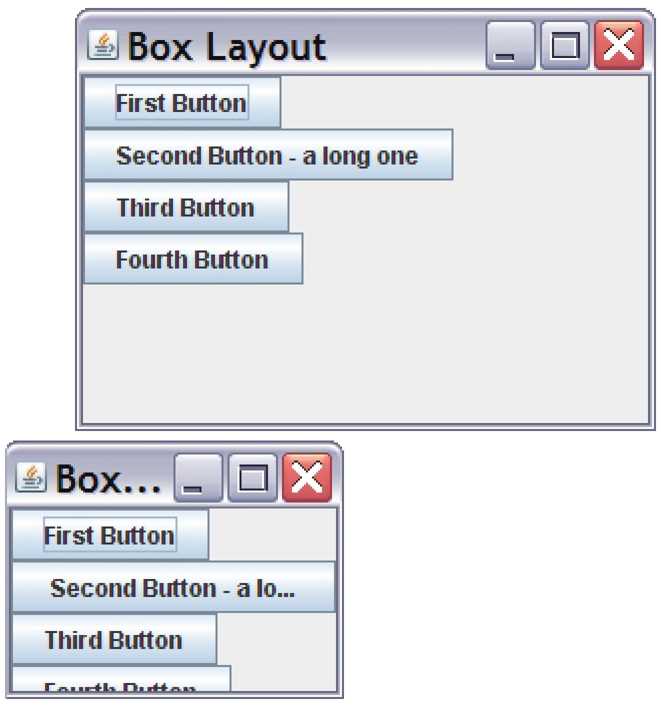
The call to setLayout still needs a LayoutManager for its parameter. In this example, we created a BoxLayout object for that parameter. The BoxLayout constructor, however, needs some parameters that the BorderLayout constructor did not. The first of those (contentPane in parentheses in the line above) is the container whose layout we want to manage. The second parameter (BoxLayout.Y_AXIS) tells the layout manager whether to stack components vertically or horizontally. In this example, we chose to stack them vertically. If we wanted to lay them out horizontally, we would use a parameter of BoxLayout.X_AXIS instead.

The way we add components to the window in BoxLayout is also very similar to how we did it in BorderLayout. The only difference is that we don't have to tell the layout manager which region to add a component to, since there is only one big region. So the lines to add buttons look like this:

```
contentPane.add(new JButton("First Button"));

contentPane.add(new JButton("Second Button - a long one"));

contentPane.add(new JButton("Third Button"));

contentPane.add(new JButton("Fourth Button"));
```

If we replace the lines we added to place the buttons in the previous example with the five lines you see here, and change the title of the window, we should get the window I showed you above. Try running this program now. The pane shows the buttons in the order we added them, keeping their original sizes, as you can see from the image above.

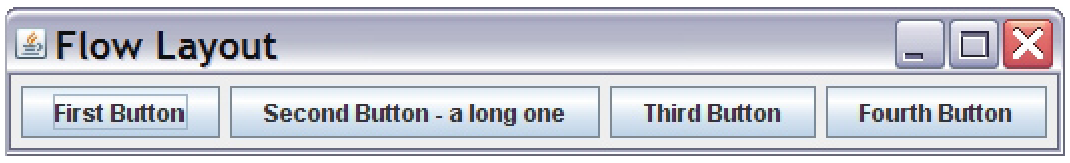
You can also see from the two images below that the components in a BoxLayout are not resized or rearranged if the window size changes.



Resized BoxLayout windows

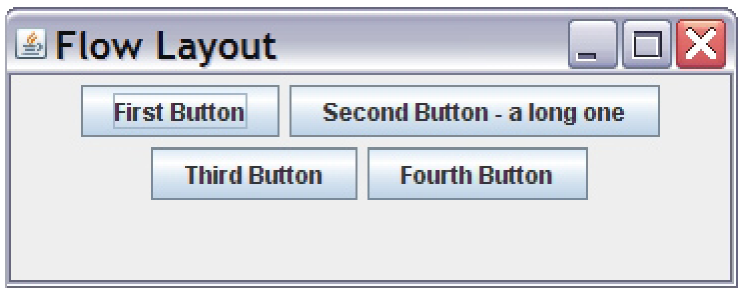
FlowLayout

FlowLayout, like BoxLayout, places all components into the window in the order we add them. It also allows them to keep their original sizes. It is different from BoxLayout in a couple of ways, too. First, it simply *flows* all components from left to right. Here's a snapshot of a FlowLayout window with the same four buttons we used in the BoxLayout:



Java's FlowLayout

The second difference is that if the window size changes so that it cannot hold all the components from left to right, it wraps them and centers them as you can see here:



Resized FlowLayout

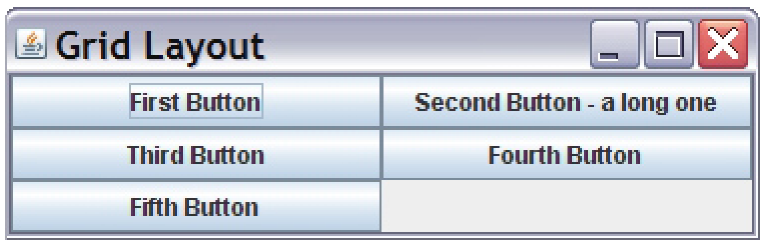
We only need to change one line of code (other than the window title, which I am not counting as a significant change) to change a BoxLayout to a FlowLayout. As you might have guessed, it's the line that calls the pane's setLayout() method to connect the layout manager to the container. Here is the new version:

```
contentPane.setLayout(new FlowLayout());
```

Like BorderLayout, FlowLayout's constructor does not need any arguments. Try using FlowLayout in your code to see how it works and how it differs from BoxLayout and BorderLayout.

GridLayout

The last of the layouts we'll talk about, GridLayout, puts components into a grid. We can specify the number of rows and columns we want the grid to contain, and all components will have equal size based on the size of the largest component. Here's an example:



Java's GridLayout

We only need to make one significant change to use GridLayout instead of FlowLayout or BoxLayout. It's in the same line of code we changed in the last example—the line that calls the pane's setLayout() method. Here's the call for GridLayout:

```
contentPane.setLayout(new GridLayout(3,2));
```

GridLayout's constructor needs two numeric arguments. The first number tells it the number of rows to put in the grid. The second gives it the number of columns. As we add components to the pane, GridLayout places them from left to right and top to bottom, as the image above shows.

Go ahead and set up a GridLayout in your program to check it out. In fact, if you want to have a little more fun, set up a three-by-three grid of buttons with no text, and make the buttons change their text to *X* or *O* when you click them. See if you can figure out how to make the first button you click show an *X*, the second one show an *O*, and so on. Then you can play tic-tac-toe on it! If you have any problem creating the above window, be sure to let me know, or check out my solution here.

Solution: Try not to peek!

Got all that? In the next chapter, we'll tackle panels and see how we can use them to put multiple components anywhere we want to!