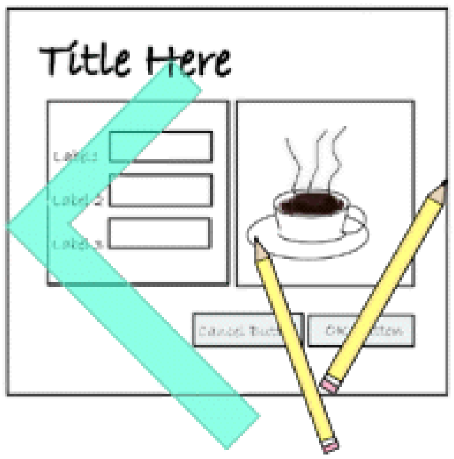
Objective Viewpoint

Java AWT Layout Management 101

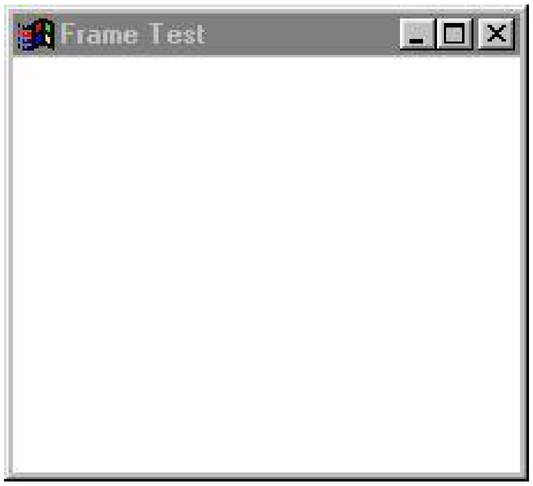
by George Crawford III



This article provides a brief summary of basic layout management in the Java Abstract Window Toolkit (AWT) and is intended to serve as a foundation for more sophisticated AWT programming.

Introduction to Frames

Frames are AWT containers that are used to display a collection of graphical AWT components. Figure 1 shows an example of a Java frame on Windows 95.



[Figure 1]

The code needed to display this frame is shown in Figure 2. The majority of GUI-based Java applications will duplicate the approach illustrated here.

```
import java.awt.*;

public class FrameExample extends Frame {
    public FrameExample(String title) {
        super(title);
    }
    public static void main(String[] args) {
          FrameExample frameExample = new FrameExample("Example Frame");
          // sets the width and height, respectively:
          frameExample.setSize(200, 300);
          frameExample.show();
    }
}
```

Figure 2: FrameExample code

As with most GUI-based Java applications, the `FrameExample` program extends the `java.awt.Frame` class. The `Frame` class has a constructor method that sets the `Frame` instance title. Likewise, the `FrameExample` class accepts the title for the frame as a parameter. The title is set by calling the constructor of the superclass using the `super` keyword.

Creating a framed application is easy. The typical procedure for frame development is as follows:

1. Create an instance of the main `Frame`:
   `FrameExample frameExample = new FrameExample("Example Frame");`
2. Set the `Frame` width and height:
   `frameExample.setSize(200, 300);`
3. Display the `Frame`:
   `frameExample.show();`

Layouts

To add components to a Frame instance, you need to specify the type of component layout scheme to be used. A `LayoutManager` is used to specify to a container how components within that container should be arranged. The next few sections explain and demonstrate the various layout managers included in the AWT.

FlowLayout

The `FlowLayout` manager is the simplest of the layout managers. In this scheme, components are arranged in the order they are placed within the container. If a row becomes completely filled, the remaining components are grouped together on the next row. Figure 3 shows an example of a `FlowLayout`, using button components.

[Figure 3]

The code to produce this result is shown in Figure 4. The program is invoked as follows:

```
java FlowLayoutExample [count]
```

where `[count]` is an optional parameter that specifies the number of buttons to be added to the frame (the default value is ten).

```
import java.awt.*;
```

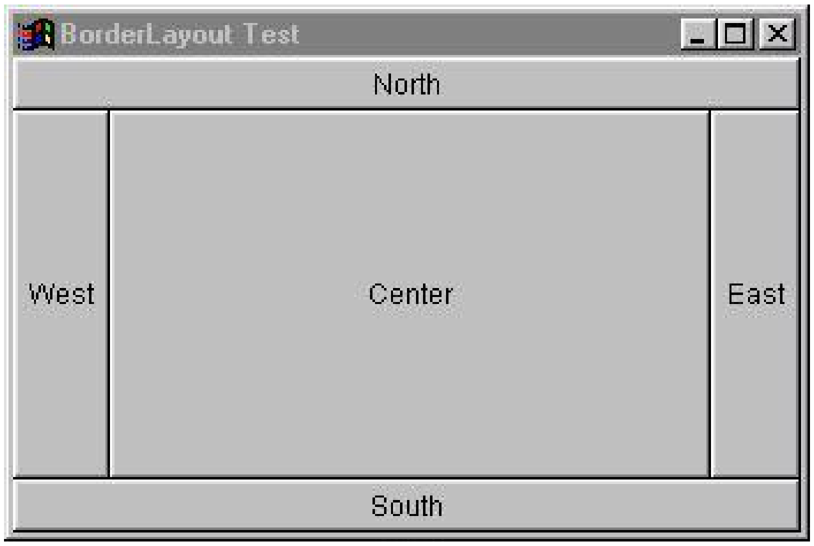
```
public class FlowLayoutExample extends Frame {
    public FlowLayoutExample(String title, int count) {
        super(title);
        setLayout(new FlowLayout());
        for(int i=1; i <= count; i++)
            add(new Button(Integer.toString(i)));
    }
    public static void main(String[] args) {
        int count = (args.length == 0) ? Integer.parseInt(args[0]) : 10;
        FlowLayoutExample fle = new FlowLayoutExample("FlowLayoutExample", count);
        fle.setSize(300, 200);
        fle.show();
    }
}
```

Figure 4: FlowLayoutExample code

The constructor accepts two parameters: the frame title and the number of buttons to be added to the frame. Like `FrameExample`, the first line calls the constructor of the superclass to set the frame title. The second line assigns the frame's layout manager. The default layout manager for a frame is the `BorderLayout` manager, discussed in the next section. The next line adds the specified number of buttons to the frame, using the `add(Component)` method in class `Container` (from which `Frame` is derived).

BorderLayout

The `BorderLayout` manager arranges components within specified regions of a container. Valid regions are "North", "South", "East", "West", and "Center". Figure 5 shows an example of the `BorderLayout` layout manager.



[Figure 5]

Notice that components placed within a region of the frame are extended to fit that region. Only one component should be positioned in a particular region at a time. If more than one component is added to a region, the most recent component will obscure the previously added ones.

The code for `BorderLayoutExample` is shown in Figure 6. Since the default layout manager for frames is `BorderLayout`, we do not explicitly set the layout manager.

```
public class BorderLayoutExample extends Frame {
    public BorderLayoutExample(String title) {
        super(title);
          add("North", new Button("North"));
          add("South", new Button("South"));
          add("East", new Button("East"));
          add("West", new Button("West"));
          add("Center", new Button("Center"));
    }
    public static void main(String[] args) {
        BorderLayoutExample ble = new BorderLayoutExample("BorderLayoutExample");
          ble.setSize(300, 200);
          ble.show();
    }
}
```

Figure 6: BorderLayoutExample code

In addition to a component, the `add()` method used for `BorderLayout` requires a `String` parameter that indicates the region where the component should be placed.

GridLayout

A container using the `GridLayout` scheme arranges components in rows and columns in row-major order. Each component is sized to fit its respective grid cell. An example of a `GridLayout` is shown in Figure 7.

[Figure 7]

The code for the `GridLayout` example appears in Figure 8. The program is invoked as follows:

```
java GridLayoutExample [rows] [columns]
```

where `[rows]` and `[columns]` are optional parameters that indicate the number of rows and columns to appear in the grid.

```
import java.awt.*;
public class GridLayoutExample extends Frame {
    public GridLayoutExample(String title, int rows, int columns) {
        super(title);
          setLayout(new GridLayout(rows, columns));
          for(int i = 0; i < rows; i++)
            for(int j = 0; j < columns; j++)
              add(new Button(Integer.toString(i)+"," + Integer.toString(j)));
    }
    public static void main(String[] args) {
        int rows = (args.length == 0) ? Integer.parseInt(args[0]) : 10;
        int columns = (args.length == 2) ? Integer.parseInt(args[1]) : 10;
        GridLayoutExample gle = new GridLayoutExample("GridLayoutExample", rows,
columns);
          gle.setSize(300, 300);
          gle.show();
    }
}
```

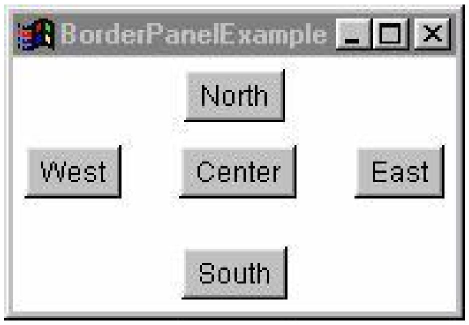
Figure 8: GridLayoutExample Code

Panels

Panels are invisible containers that are used to further refine the arrangement of components within another container. Since a panel is itself a container, you can set a panel's layout manager just as you would with other graphical containers. In addition, containers can have other containers (including other panels). This recursive structure gives the programmer increased flexibility in arranging the GUI.

How to Use Panels

As a simple example of `Panel` usage, consider the `BorderLayoutExample` application. When run, the buttons within the frame are extended to fit the regions. To remedy this, you can place each button inside a panel. Figure 9 shows the result of using panels in this way.



[Figure 9]

The program is shown in Figure 10. To avoid redundancy in the code, a simple method called `addComponent()` that adds a single component to a panel inside the specified region is used (in real-world applications, you will likely want support for multiple components). The conventions used for panels are actually no different from the protocols used for components, except that panels must be added to the enclosing component; otherwise, the components inside the panel will not be visible.

```
public class BorderPanelExample extends Frame {
    public BorderPanelExample(String title) {
        super(title);
          addComponent("North", new Button("North"));
          addComponent("South", new Button("South"));
          addComponent("East", new Button("East"));
```

```
            addComponent("West", new Button("West"));
            addComponent("Center", new Button("Center"));
    }
    public void addComponent(String region, Component component) {
            Panel panel = new Panel();
            panel.add(component);
            add(region, panel); // make sure you add the panel!
    }
    public static void main(String[] args) {
          BorderPanelExample bpe =
              new BorderPanelExample("BorderPanelExample");
          bpe.setSize(200, 150);
          bpe.show();
    }
}
```

Figure 10: BorderPanelExample code

Conclusion

This column is intended to give you a quick start in GUI application development in Java. The basic layout mechanisms in the AWT were covered to help assist in understanding the Java layout management scheme. In our next article, we will examine two more complex layout managers: the `GridBagLayout` and the `CardLayout`. We will also cover the JDK 1.1 event model.

References

1.
Cornell, Gary and Horstmann, C. Core Java: Volume 1 - Fundamentals, Sun Microsystems Press, 1997.

2.
Hall, Marty. Core Web Programming, Prentice Hall, Inc., Upper Saddle River, NJ, 1998.

3.
Morrison, Michael et al. Java 1.1 Unleashed, Sams.net, Indianapolis, In., 1997.

George Crawford III is a software engineer at MPI Software Technology, Inc. and is completing his M.S. degree in computer science at Mississippi State University. His technical areas of interest include software design, software process management, distributed object technology, and games programming. He has been using Java since its alpha release in 1995.