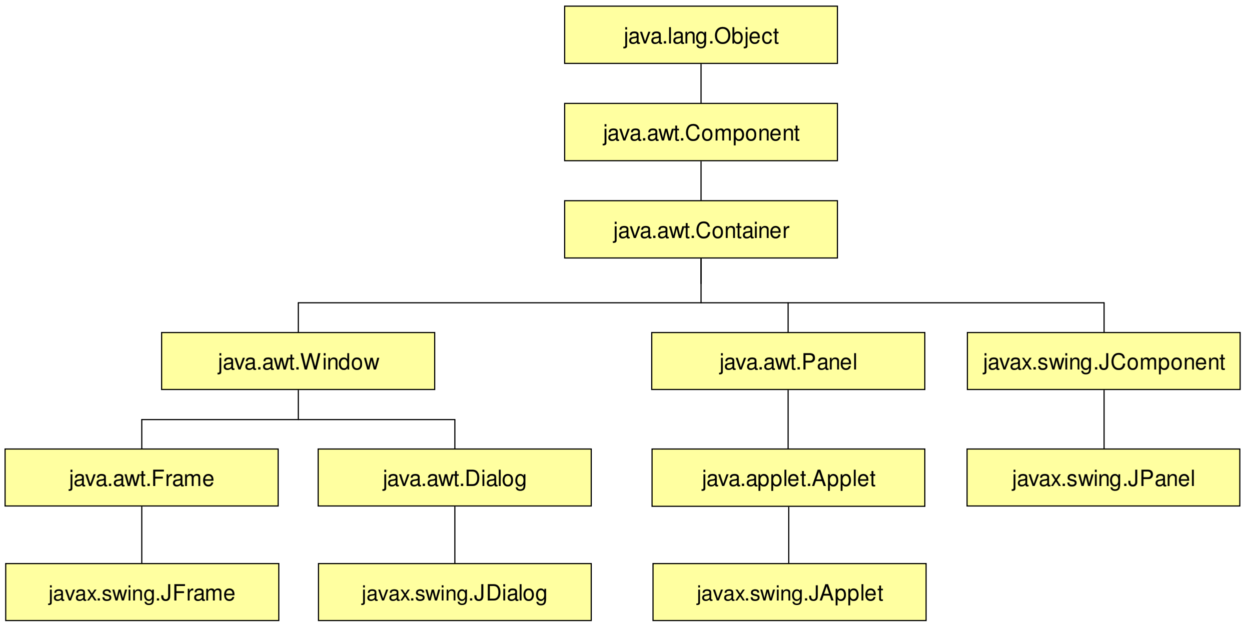
LAYOUT MANAGERS

A ***layout manager*** controls how GUI components are organized within a GUI container. Each Swing container (e.g. **`JFrame`**, **`JDialog`**, **`JApplet`** and **`JPanel`**) is a subclass of **`java.awt.Container`** and so has a layout manager that controls it.



Flow Layout

The simplest layout manager is `java.awt.FlowLayout`, which adds components to the container from left-to-right, top-to-bottom. It is the default layout for GUI container objects of classes `Applet` or `JPanel`.

Example

Assume that an application has built a window with the following code:

```
JFrame win = new JFrame( "Layout Demo" );
win.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

The code below builds a red label. By default, labels are transparent, so we must make it opaque for the color to show. The displayed label is shown at right.

```
JLabel east = new JLabel( "EAST" );
east.setOpaque( true );
east.setBackground( Color.RED );
```

Similarly, the code below makes four additional labels of varying colors:

```
JLabel west = new JLabel( "WEST" );
west.setOpaque( true );
west.setBackground( Color.BLUE );
JLabel north = new JLabel( "NORTH" );
north.setOpaque( true );
north.setBackground( Color.GREEN );
JLabel south = new JLabel( " SOUTH" );
south.setOpaque( true );
south.setBackground( Color.YELLOW );
JLabel center = new JLabel( " CENTER" );
center.setOpaque( true );
center.setBackground( Color.ORANGE );
```

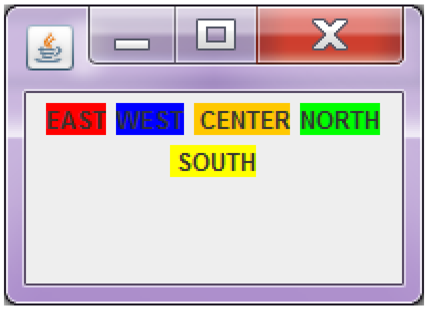
Using a flow layout manager for the window, the following code adds the five labels to the window.

```
win.setLayout( new FlowLayout( ) );
win.add( east );
```

```
     win.add( west );
     win.add( center );
     win.add( north );
     win.add( south );
```

Flow layout adds the labels in the order specified by the code, left to right, top to bottom. The completed window is shown to the right.

The flow layout manager makes each component no larger than it needs to be to display its contents.



Border Layout

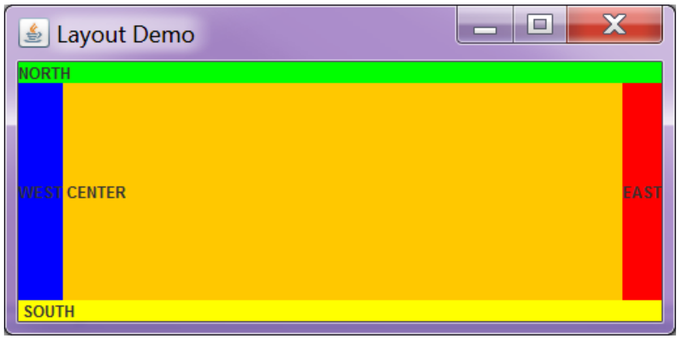
Another simple layout manager is **`java.awt.BorderLayout`**, which is the default layout for GUI container objects of classes **`JFrame`** or **`JDialog`**.

Border layout splits the GUI container into five areas – east, west, north, south and center – oriented so that north is the top. In the call to the **`add`** method, you must specify to which area of the container the component is to be added, otherwise it is added to the center area by default.

Example

Border layout adds the labels to the container areas specified by the call to meth **`add`**. The window build by the code below is shown to the right.

The border layout manager expands the size of the component to fit its area.



```
win.setLayout( new BorderLayout( ) );
win.add( east, BorderLayout.EAST );
win.add( west, BorderLayout.WEST );
win.add( center, BorderLayout.CENTER );
win.add( north, BorderLayout.NORTH );
win.add( south, BorderLayout.SOUTH );
```

Grid Layout

The third simple layout manager is `java.awt.GridLayout`, which adds components to a grid in ***row-major order*** (i.e. left to right and top to bottom). The class has a choice of three constructors:

Constructors for java.awt.GridLayout

`GridLayout( int r, int c )`
`// Build a grid with r rows and c columns.`

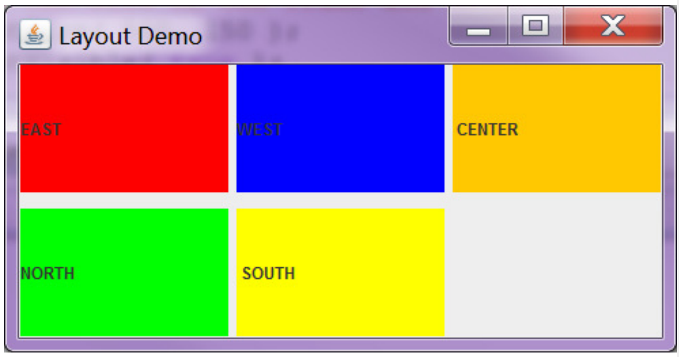
`GridLayout( )`
`// Build a grid with 1 row and a column for each added`
`// component.`

`GridLayout( int r, int c, int xgap, int ygap )`
`// Build a grid with r rows and c columns. 'xgap' is the spacing`
`// between columns and 'ygap' is the spacing between rows.`

Example

Grid layout adds the labels to the container using the grid specified by the call to its constructor. The window build by the code below is shown to the right.

Grid layout makes each component in the grid equal sized.



```
win.setLayout( new GridLayout( 2, 3, 6, 12 ) );
win.add( east );
win.add( west );
win.add( center );
win.add( north );
win.add( south );
```

Box Layout

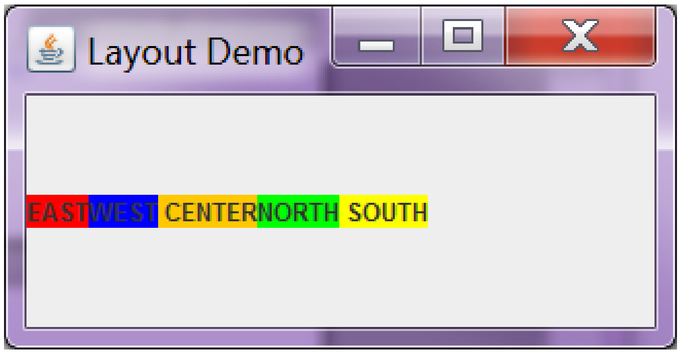
`javax.swing.BoxLayout` adds components to the container either vertically or horizontally. The components do not wrap as with flow layout. For example, a horizontal arrangement stays horizontal even when the container is resized. Here is the class constructor:

```
BoxLayout( Container container, int orientation )
```

For the first argument, pass the content pane of the container that is to use the box layout. The second argument is an integer code indicating vertical or horizontal orientation. The **`BoxLayout`** class provides ease-of-use constants for this argument:

Two of the Orientation Constants for BoxLayout	
`X_AXIS`	Lay components out horizontally from left to right.
`Y_AXIS`	Lay components out vertically from top to bottom.

Example
The window build by the code below is shown to the right.



```
Container box = win.getContentPane( );
box.setLayout( new BoxLayout( box, BoxLayout.X_AXIS ) );
win.add( east );
win.add( west );
win.add( center );
win.add( north );
win.add( south );
```

Java provides the class **`javax.swing.Box`**, which allows you to build a container whose layout manager is box layout. Most GUI programmers use this instead of **`javax.swing.BoxLayout`** because it provides handy methods for configuring the box. For more information, see the topic *javax.swing.Box*.

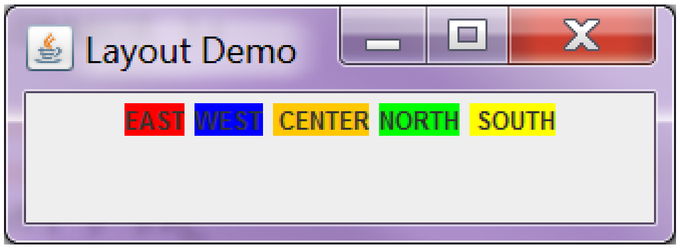
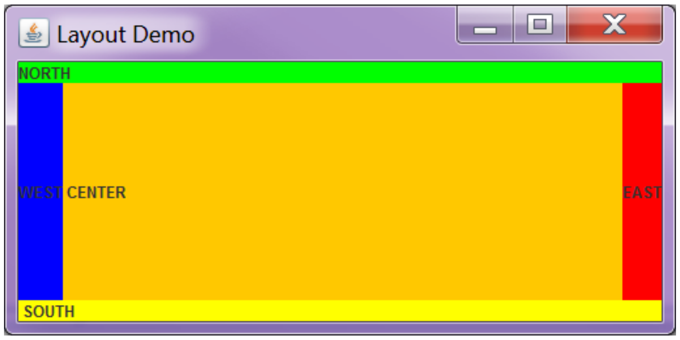
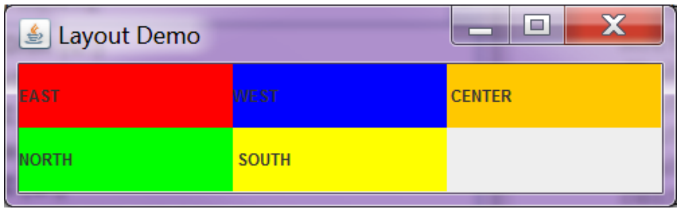
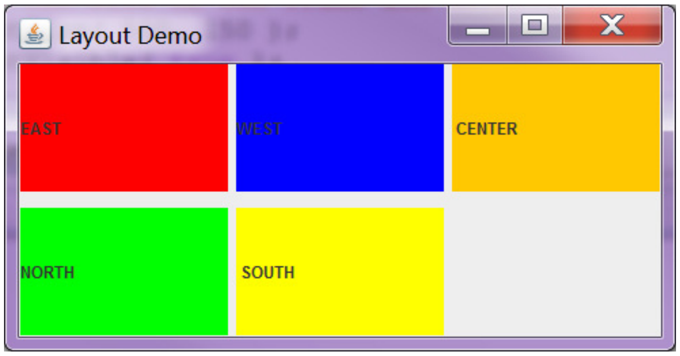
Grid Bag Layout

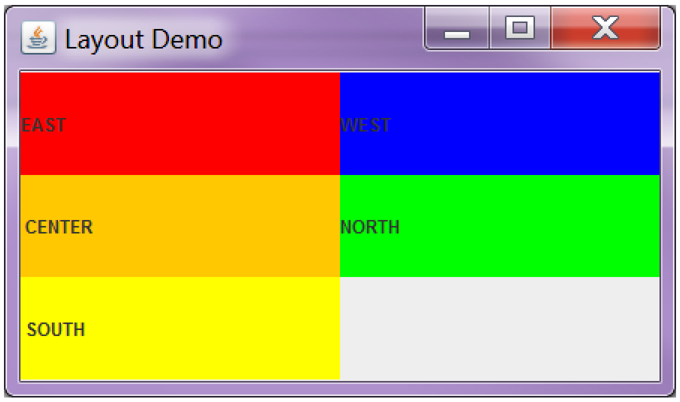
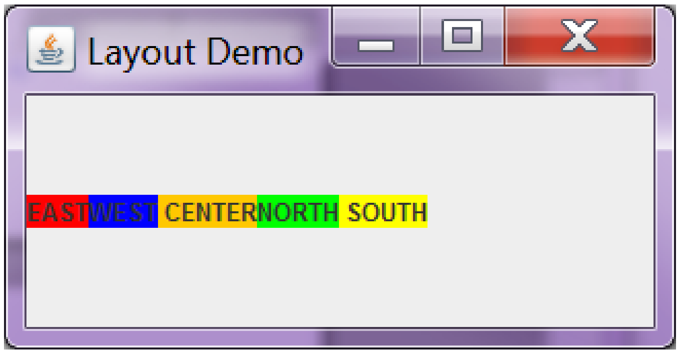
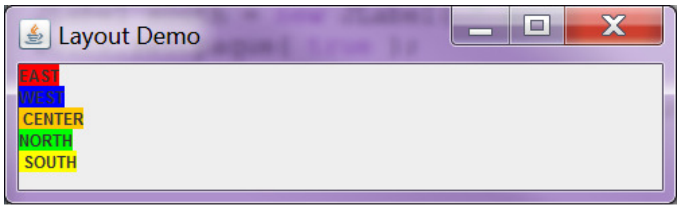
`java.awt.GridBagLayout` allows very sophisticated placement of components within a container. It's essentially a grid, but the grid rows and columns may have varying heights and widths and components may span more than one grid cell. Consequently, it's a rather challenging layout manager to learn and use. If you wish to tackle it, see *How to Use GridBagLayout* in *The Java™ Tutorials* (google *java tutorial gridbaglayout*).

Other Layout Managers

`javax.swing.GroupLayout` and `javax.swing.SpringLayout` were created to be used by GUI builder tools such as that found in NetBeans, although you can use them manually. See *How to Use GroupLayout* and *How to Use SpringLayout* in *The Java™ Tutorials*.

Exercises

1.	Using the code from the first example, write a complete Java program (application or applet) that uses a flow layout manager to build and display the GUI shown to the right.	
2.	Modify your solution to exercise #1 so that it uses a border layout manager to build and display the GUI shown to the right.	
3.	In your solution to exercise #2, remove the second parameter from the first two calls to the frame's **`add`** method. Recompile and rerun the application. Observe the displayed GUI. Explain its appearance.	
4.	Modify your solution to exercise #1 so that it uses the two-parameter **`GridLayout`** constructor to build a grid layout manager that displays the GUI shown to the right.	
5.	In your solution to exercise #4, replace the two-parameter **`GridLayout`** constructor by the four-parameter constructor. The program should display the GUI shown to the right. The grid has vertical padding of 10 and horizontal padding of 5.	
6.	In your solution to exercise #4, replace the two-parameter **`GridLayout`** constructor by the constructor with no parameters. Recompile and rerun the program. Observe the displayed GUI. Explain its appearance.	

7.	Modify your solution to exercise #1 so that it uses the grid layout manager to build and display the GUI shown to the right.	
8.	Modify your your solution to exercise #1 so that it uses the box layout manager to build a horizontally oriented GUI as shown to the right.	
9.	Modify your your solution to exercise #1 so that it uses the box layout manager to build a vertically oriented GUI as shown to the right.	 Layout Demo EAST WEST CENTER NORTH SOUTH