Arranging Components : Layout Manager

- **Layout** means the arrangement of components within the container.

- **Layout manager** automatically positions all the components within the container. (***i.e. Default Layout Manager***)

- **LayoutManager:**
  - A layout manager is an instance of any class that implements the **LayoutManager** interface.

Arranging Components : Layout Manager

- Every ***Container*** has a **layout manager.**
- The ***default layout*** for a Panel *and* Applet is **FlowLayout**
- The ***default layout*** for a **Window *and* Frame** is a **BorderLayout**
- We could set it explicitly with: **setLayout**()
  - **void setLayout (LayoutManager layoutObj)**

  -Here, *layoutObj* is a reference to the desired layout manager
- If you do this, you will need to determine the shape and position of each component manually, using the **setBounds( )** method defined by **Component**.

Syntax: **void setBounds**(x-coordinates,y-coordinates, width, height)

Different Layout Manager

- **FlowLayout**
  - The **FlowLayout** is the ***default layout***.
  - It layouts the components in a directional/horizontally flow.
- **BorderLayout**
  - The **BorderLayout** arranges the components to fit in the ***five*** regions: ***east, west, north, south and center.***
- **GridLayout**
  - The **GridLayout** manages the components in form of a **rectangular grid.**

Different Layout Manager

- **CardLayout**
  - The **CardLayout** object treats each component in the container as a card.
  - Only one card is visible at a time.

- **GridBagLayout**
  - This is the most flexible layout manager class.
  - The object of **GridBagLayout** aligns the component vertically, horizontally or along their baseline without requiring the components of same size.

FlowLayout

- ***FlowLayout*** is the **default** layout manager.
- ***FlowLayout*** implements a simple layout style, which is similar to how words flow in a text editor.
- Components are added ***left-to-right***, ***top to bottom***.
- If no room, a new row is started
- Exact layout depends on size of Applet
- Components are made as small as possible
- ***FlowLayout*** is convenient but often not good.

FlowLayout

- **Constructors for FlowLayout:**
  - **FlowLayout**()
  - **FlowLayout**(int how)
  - **FlowLayout**(int how, int horz, int vert)

- The **first form** creates the default layout, which centers components and leaves ***five pixels*** of space between each component.

- The **second form** lets you specify how each line is aligned. Valid values for how are as follows:
  - **FlowLayout.LEFT**
  - **FlowLayout.RIGHT**
  - **FlowLayout.CENTER**
  - **FlowLayout.LEADING**
  - **FlowLayout.TRAILING**

FlowLayout (Example)

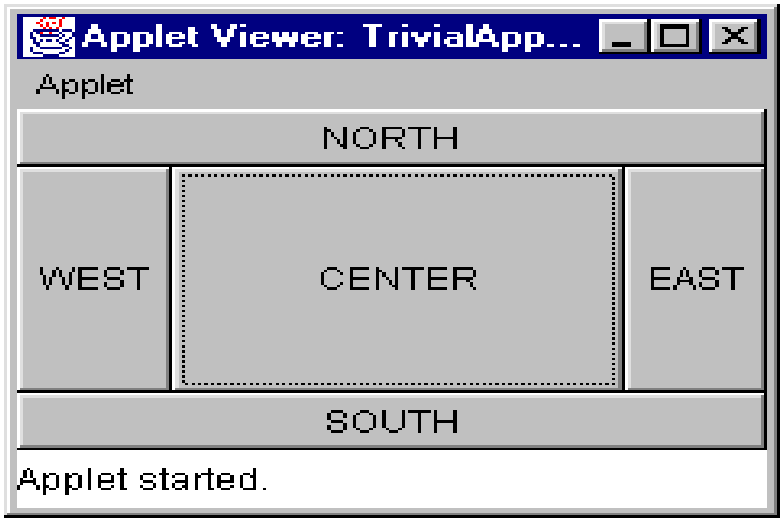
```java
//Demonstrate use of FlowLayout
import java.awt.*;
import java.applet.*;
public class FlowLayoutExample extends Applet
{
    public void init ()
    {
        setLayout(new FlowLayout());
        add(new Button("One"));
        add(new Button("Two"));
        add(new Button("Three"));
        add(new Button("Four"));
        add(new Button("Five"));
        add(new Button("Six"));
    }
}

/*
<applet code="FlowLayoutExample" width=100 height=100>
</applet>
*/
```

FlowLayout (Example)

```java
//Use left-aligned flow layout
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="FlowLayoutDemo" width=240 height=200></applet>*/
public class FlowLayoutDemo extends Applet implements ItemListener
{
    String msg=" ";  Checkbox windows, android, mac, solaris;
    public void init()
    {
        //set left-aligned flow layout
        setLayout(new FlowLayout(FlowLayout.LEFT));

        windows=new Checkbox("Windows",null, true);
        android=new Checkbox("Android");
        mac=new Checkbox("Mac OS");
        solaris=new Checkbox("Solaris");

        add(windows);
        add(android);
        add(mac);
        add(solaris);

        //register to receive item events
        windows.addItemListener(this);
        android.addItemListener(this);
        mac.addItemListener(this);
        solaris.addItemListener(this);
    }
    //Repaint when status of a check box changes
    public void itemStateChanged(ItemEvent ie)
    {  repaint(); }
    //Display current state of the check boxes.
    public void paint(Graphics g)
    {
        msg="Current State: ";
        g.drawString(msg,6,80);
        msg=" Windows: "+windows.getState();
        g.drawString(msg,6,100);
        msg=" Android: "+android.getState();
        g.drawString(msg,6,120);
        msg=" Mac OS: "+mac.getState();
        g.drawString(msg,6,140);
        msg="Solaris: "+solaris.getState();
        g.drawString(msg,6,160);
    }
}
```

BorderLayout

- At most five components can be added.
- If you want more components, add a Panel, then add components to it.
- **setLayout** (**new BorderLayout());**



add (new **Button**("NORTH"),***BorderLayout.NORTH***);

BorderLayout

- Constructors for ***BorderLayout*** are:
  - **BorderLayout()**
  - **BorderLayout(int horz, int vert)**

- ***BorderLayout*** defines the following commonly used constants that specify the regions:
  - **BorderLayout.CENTER**
  - **BorderLayout.EAST**
  - **BorderLayout.WEST**
  - **BorderLayout.NORTH**
  - **BorderLayout.SOUTH**

BorderLayout

- When adding components, you will use these constants with the following form of **add(),** which is defined by container.

void add(Component compRef,Object region)

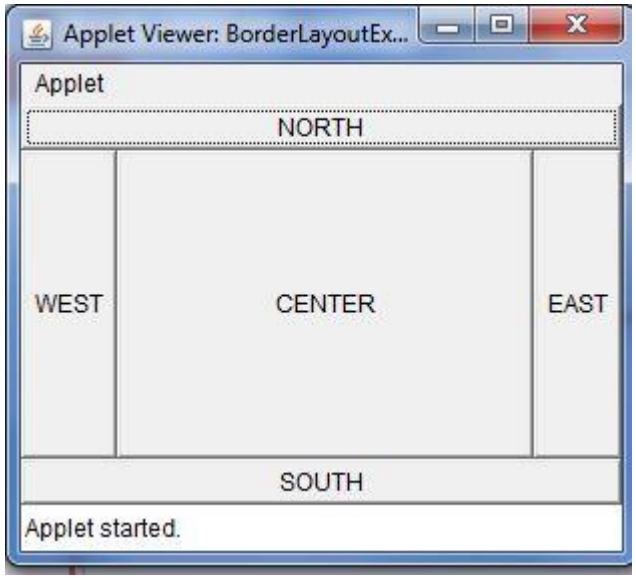
-here, **compRef** is a reference to the component to be added, and the **region** specifies where the component will be added.

BorderLayout With Five Buttons

```
public void init()
{
    setLayout(new BorderLayout());
    add(new Button("NORTH"), BorderLayout.NORTH);
    add(new Button("SOUTH"), BorderLayout.SOUTH);
    add(new Button("EAST"), BorderLayout.EAST);
    add(new Button("WEST"), BorderLayout.WEST);
    add(new Button("CENTER"),BorderLayout.CENTER);
}
```

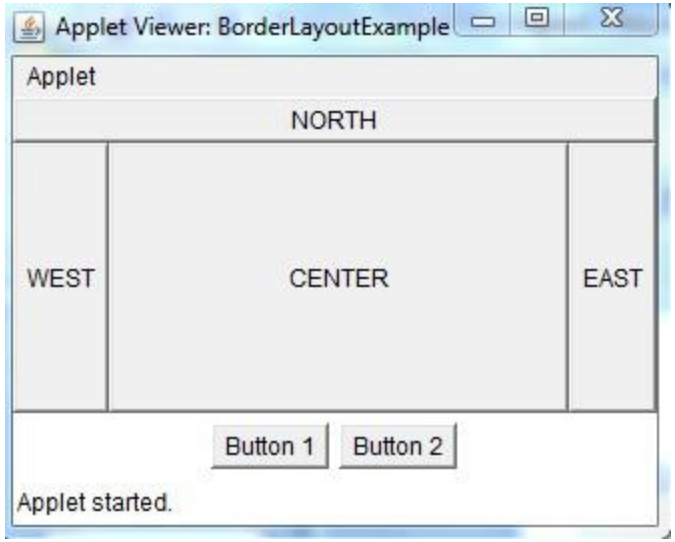
BorderLayout

```java
//Demonstrate BorderLayout
import java.awt.*;
import java.applet.*;
public class BorderLayoutExample extends Applet
{
    public void init()
    {
        setLayout(new BorderLayout());
        add(new Button("NORTH"), BorderLayout.NORTH);
        add(new Button("SOUTH"), BorderLayout.SOUTH);
        add(new Button("EAST"), BorderLayout.EAST);
        add(new Button("WEST"), BorderLayout.WEST);
        add(new Button("CENTER"), BorderLayout.CENTER);
    }
}
/*
<applet code="BorderLayoutExample" width=300 height=200>
</applet>
*/
```



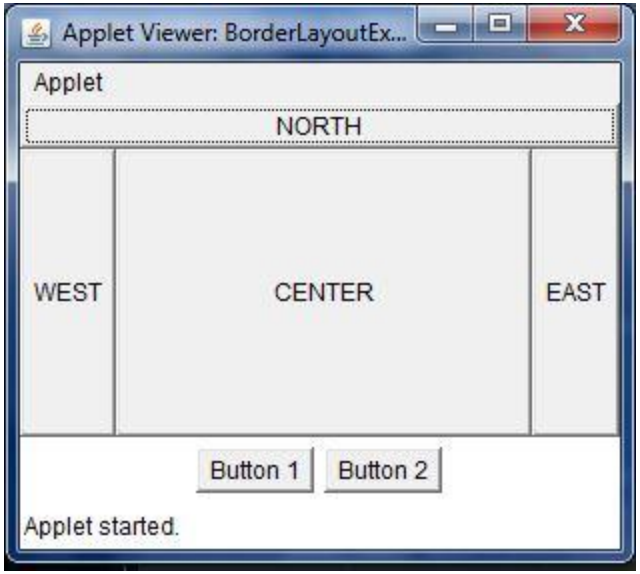
USING A PANEL

```
Panel p = new Panel();
add (p, BorderLayout.SOUTH);
p.add (new Button("Button 1"));
p.add (new Button("Button 2"));
```



USING A PANEL (EXAMPLE)

```
//Demonstrate BorderLayout
import java.awt.*;
import java.applet.*;
public class BorderLayoutExample1 extends Applet
{
    public void init()
    {
        setLayout(new BorderLayout());
        add(new Button("NORTH"), BorderLayout.NORTH);
        add(new Button("EAST"), BorderLayout.EAST);
        add(new Button("WEST"), BorderLayout.WEST);
        add(new Button("CENTER"), BorderLayout.CENTER);

        Panel p=new Panel();
        add(p,BorderLayout.SOUTH);
        p.add(new Button("Button 1"));
        p.add(new Button("Button 2"));
    }
}
/*
<applet code="BorderLayoutExample1" width=300 height=200>
</applet>
*/
```



Using Insets

- Sometimes you will want to leave a *small amount of space* between the container that holds your components and the window that contains it.

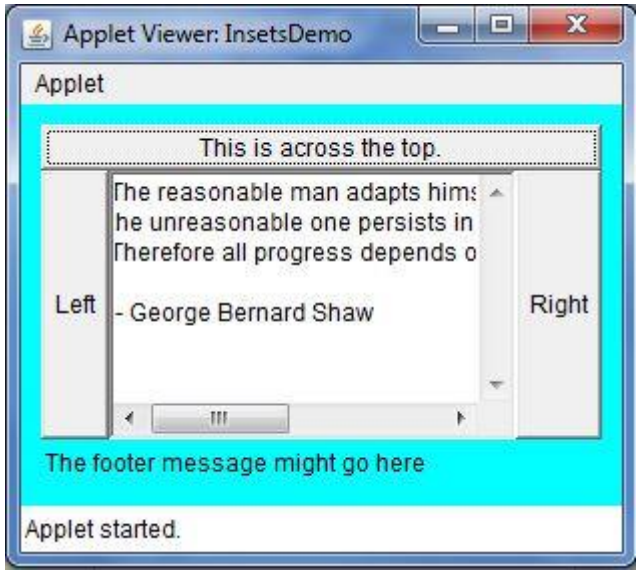
- To do this, override the **getInsets()** method defined by the container.

- The constructor for **Insets** is shown here:
  - **Insets(int top, int left, int bottom, int right)**

- The **getInsets()** method has this general form:
    - **Insets getInsets()**

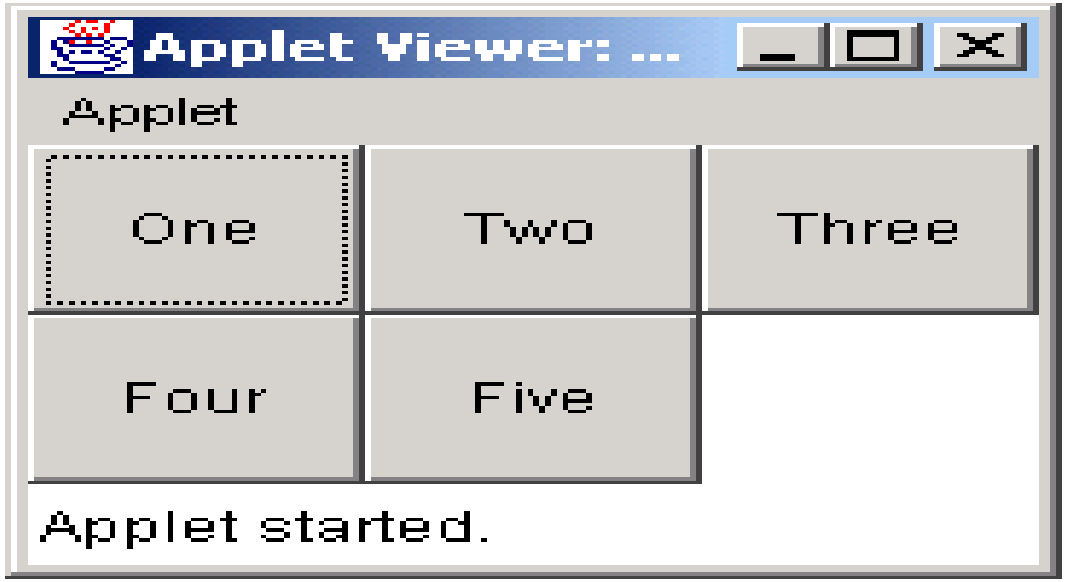
Using Insets (Example)

```
//Demonstrate BorderLayout with Insets
import java.awt.*;
import java.applet.*;
public class InsetsDemo extends Applet
{
    public void init()
    {
        //set background color so insets can be easily seen
        setBackground(Color.cyan);

        setLayout(new BorderLayout());

        add(new Button("This is across the top."), BorderLayout.NORTH);
        add(new Label("The footer message might go here"), BorderLayout.SOUTH);
        add(new Button("Right"), BorderLayout.EAST);
        add(new Button("Left"), BorderLayout.WEST);

        String msg= "The reasonable man adapts " +
            "himself to the world;\n" +
            "the unreasonable one persists in " +
            "trying to adapt the world to himself.\n" +
            "Therefore all progress depends " +
            "on the unreasonable man.\n\n" +
            " - George Bernard Shaw\n\n";

        add(new TextArea(msg), BorderLayout.CENTER);
    }
    //add insets
    public Insets getInsets()
    {
        return new Insets(10,10,10,10);
    }
}
/*
<applet code="InsetsDemo" width=300 height=200>
</applet>
*/
```



GridLayout

- **GridLayout** lays out components in a two-dimensional grid.
- When you instantiate a **GridLayout**, you define the number of rows and columns.
- All sections of the grid are equally sized and as large as possible

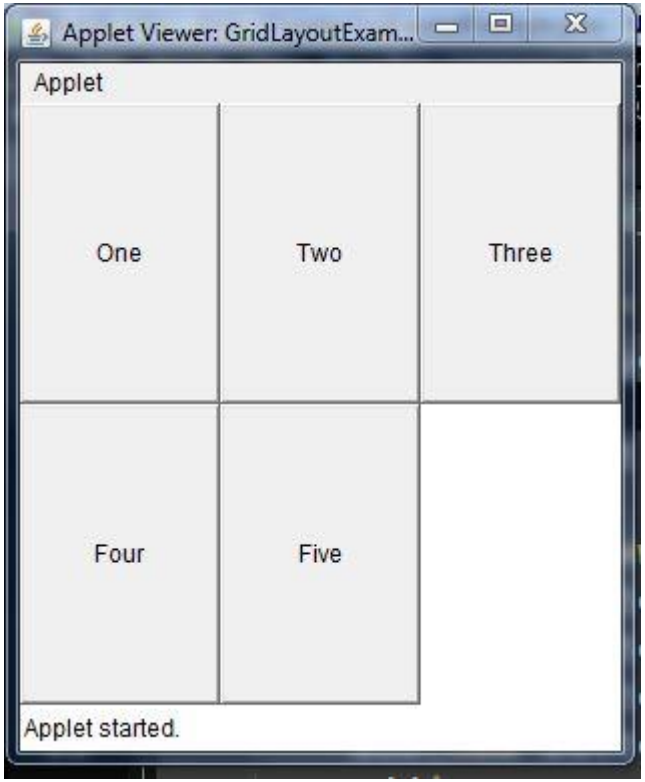


GridLayout

- **The constructors supported by GridLayout are:**
  - GridLayout()
  - GridLayout(int numRows,int numColumns)
  - GridLayout(int numRows,int numColumns,int horz,int vert)

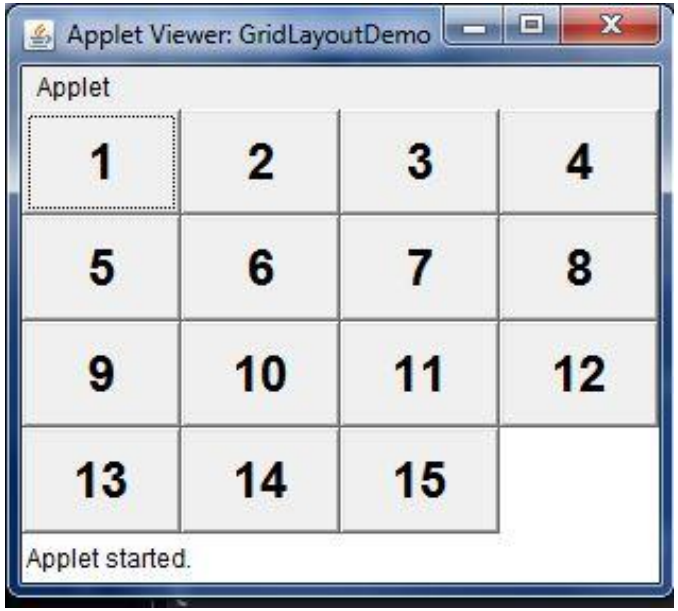
GridLayout (example)

```java
//Demonstrate GridLayout
import java.awt.*;
import java.applet.*;
public class GridLayoutExample extends Applet
{
    public void init ()
    {
        setLayout(new GridLayout(2, 3));
        add(new Button("One"));
        add(new Button("Two"));
        add(new Button("Three"));
        add(new Button("Four"));
        add(new Button("Five"));
    }
}
/*
<applet code="GridLayoutExample" width=300 height=300>
</applet>
*/
```



GridLayout (example)

```
//Demonstrate GridLayout
import java.awt.*;
import java.applet.*;
/*
<applet code="GridLayoutDemo" width=300 height=200>
</applet>
*/
public class GridLayoutDemo extends Applet
{
    static final int n=4;
    public void init()
    {
        setLayout(new GridLayout(n,n));
        setFont(new Font("SansSerif",Font.BOLD,24));

        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                int k=i*n+j;
                if(k>0)
                    add(new Button(""+k));
            }
        }
    }
}
```



CardLayout

- The class **CardLayout** arranges each component in the container as a card.

- *Only one card is visible at a time*, and the container acts as a stack of cards.

- **Constructors:**
  - **CardLayout()**

    -Creates a new card layout with gaps of size zero.
  - **CardLayout(int hgap, int vgap)**

    -Creates a new card layout with the specified horizontal and vertical gaps.

CardLayout

- Cards are typically held in an object of type **Panel**.
- Panel must have **CardLayout** selected as its layout manager.
- **For Add component:**
  - void add(Component *panelObj*, *Object name);*
- **Methods:**
  - void first(**Container *deck*)**
  - void last(**Container *deck*)**
  - void next(**Container *deck*)**
  - void previous(**Container *deck*)**
  - void show(**Container *deck*, *String cardName*)**