

# Layout Managers

# LayoutManager Interface

- ❖ The `LayoutManager` interface defines the object responsible for the components layout.
- ❖ Each container is registered with a `LayoutManager` object.
- ❖ We can register a new `LayoutManager` object by calling the `Container.setLayout()` method.  
`public void setLayout(LayoutManager manager)`

# LayoutManager Interface

- ❖ Calling the `Container.setLayout()` method and passing 'null' will deactivate the layout manager.

Doing so will allow us to specify the exact dimension and location of each component.

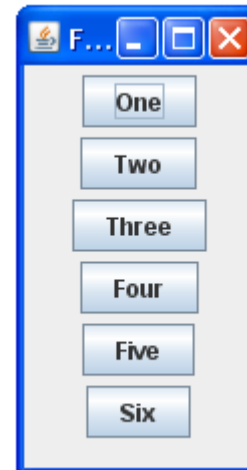
- ❖ The biggest advantage of using layout manager is their elimination of the need to compute each component optimal dimension and location.

Using a layout manager our application will automatically adjust itself to the exact dimension of the screen in accordance with the execution environment.

# FlowLayout Class

- ❖ `FlowLayout` is `JPanel` default `LayoutManager`.
- ❖ Adding components to container that its layout manager is a `FlowLayout` object, the components will be added left to right and top to bottom. The size of each component will be its preferred size.

# FlowLayout Class



# FlowLayout Class

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

public class FlowLayoutDemo
{
    private JFrame frame;
    private JButton bt1, bt2, bt3, bt4, bt5, bt6;
    public FlowLayoutDemo()
    {
        bt1 = new JButton("One");
        bt2 = new JButton("Two");
        bt3 = new JButton("Three");
        bt4 = new JButton("Four");
        bt5 = new JButton("Five");
        bt6 = new JButton("Six");
        frame = new JFrame("FlowLayoutDemo");
        frame.setLayout(new FlowLayout());
    }
}
```

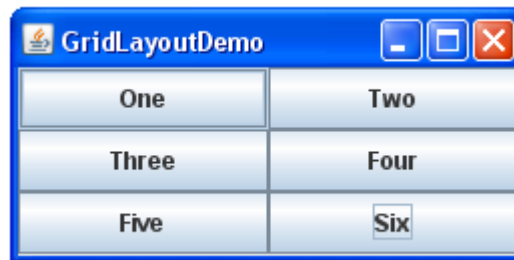
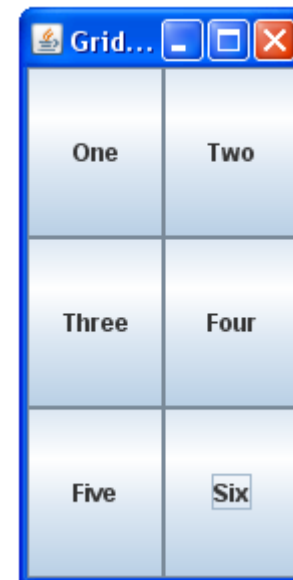
# FlowLayout Class

```
        frame.add(bt1);
        frame.add(bt2);
        frame.add(bt3);
        frame.add(bt4);
        frame.add(bt5);
        frame.add(bt6);
    }
    public void go()
    {
        frame.setSize(600,400);
        frame.setVisible(true);
    }
    public static void main(String args[])
    {
        FlowLayoutDemo demo = new FlowLayoutDemo();
        demo.go();
    }
}
```

# GridLayout Class

- ❖ The `GridLayout` layout manager lays the components organized in rows and in columns.
- ❖ Each cell has the same size. Each components is placed within one cell and spreads to cover its entire area.
- ❖ Components are added left to right top to bottom.
- ❖ When instantiating the `GridLayout` class we specify the number of rows and the number of columns we want to set in our container.

# GridLayout Class



# GridLayout Class

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

public class GridLayoutDemo
{
    private JFrame frame;
    private JButton bt1, bt2, bt3, bt4, bt5, bt6;
    public GridLayoutDemo()
    {
        bt1 = new JButton("One");
        bt2 = new JButton("Two");
        bt3 = new JButton("Three");
        bt4 = new JButton("Four");
        bt5 = new JButton("Five");
        bt6 = new JButton("Six");
        frame = new JFrame("GridLayoutDemo");
    }
}
```

# GridLayout Class

```
        frame.setLayout(new GridLayout(3,2));
        frame.add(bt1);
        frame.add(bt2);
        frame.add(bt3);
        frame.add(bt4);
        frame.add(bt5);
        frame.add(bt6);
    }
    public void go()
    {
        frame.setSize(600,400);
        frame.setVisible(true);
    }
    public static void main(String args[])
    {
        GridLayoutDemo demo = new GridLayoutDemo();
        demo.go();
    }
}
```

# BorderLayout Class

- ❖ The `BorderLayout` layout manager lays the components organized in the following regions:

`BorderLayout.NORTH`

`BorderLayout.SOUTH`

`BorderLayout.WEST`

`BorderLayout.EAST`

`BorderLayout.CENTER`

- ❖ Each region can include up to one component only.
- ❖ Regions without any component are conquered by the `BorderLayout.CENTER` region.

# BorderLayout Class

- ❖ When adding a component to container that its layout manager is a `BorderLayout` object we should specify the region to which the component should be added.

...

```
JFrame frame = new JFrame();
```

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

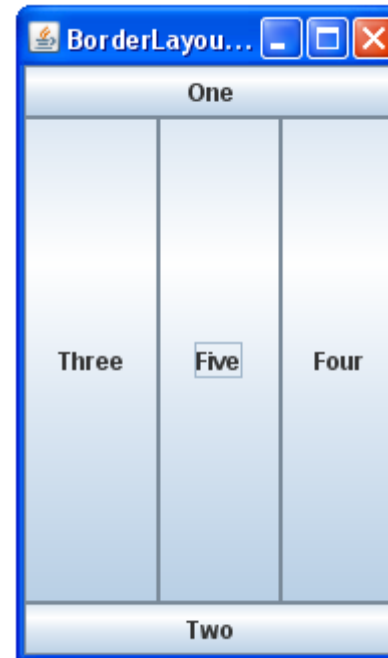
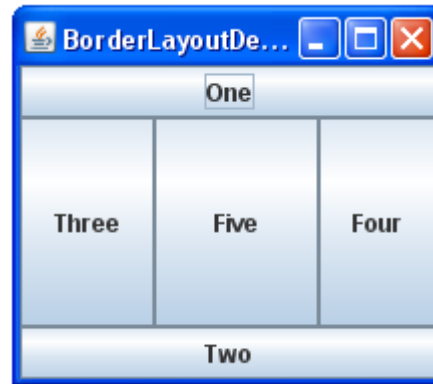
```
frame.add(new JButton("One"), BorderLayout.North);
```

...

# BorderLayout Class

- ❖ Components added to `BorderLayout.NORTH` and to `BorderLayout.SOUTH` regions will span to the entire width. Their height will be their preferred one.
- ❖ Components added to `BorderLayout.WEST` and to `BorderLayout.EAST` regions will span to the entire available height. Their width will be their preferred one.
- ❖ `BorderLayout` is the default layout manager of the `JFrame` content pane.

# BorderLayout Class



# BorderLayout Class

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

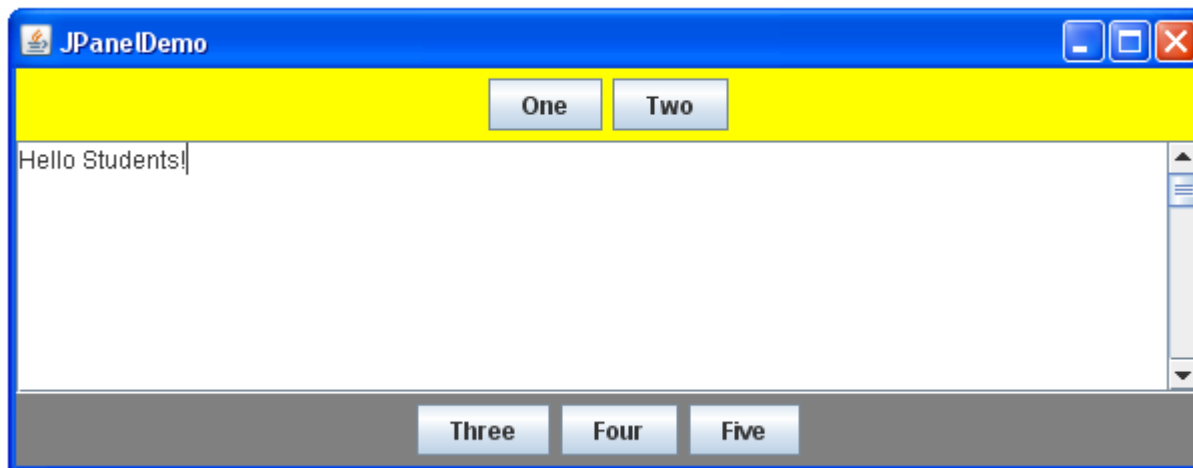
public class BorderLayoutDemo
{
    private JFrame frame;
    private JButton bt1, bt2, bt3, bt4, bt5;
    public BorderLayoutDemo()
    {
        bt1 = new JButton("One");
        bt2 = new JButton("Two");
        bt3 = new JButton("Three");
        bt4 = new JButton("Four");
        bt5 = new JButton("Five");
        frame = new JFrame("BorderLayoutDemo");
    }
}
```

# BorderLayout Class

```
        frame.setLayout(new BorderLayout());
        frame.add(bt1, BorderLayout.NORTH);
        frame.add(bt2, BorderLayout.SOUTH);
        frame.add(bt3, BorderLayout.WEST);
        frame.add(bt4, BorderLayout.EAST);
        frame.add(bt5, BorderLayout.CENTER);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void go()
    {
        frame.setSize(600, 400);
        frame.setVisible(true);
    }
    public static void main(String args[])
    {
        BorderLayoutDemo demo = new BorderLayoutDemo();
        demo.go();
    }
}
```

# JPanel Class

- ❖ JPanel is both a component and a container. As a component we can place it on other containers. As a container we can place other components on it.



# JPanel Class

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

public class JPanelDemo
{
    private JFrame frame;
    private JTextArea textArea;
    private JScrollPane scrollPane;
    private JPanel panelTop, panelBottom;
    private JButton bt1, bt2, bt3, bt4, bt5;
    public JPanelDemo()
    {
        panelTop = new JPanel();
        panelBottom = new JPanel();
        panelTop.setBackground(Color.YELLOW);
        panelBottom.setBackground(Color.GRAY);
        textArea = new JTextArea(40,24);
        scrollPane = new JScrollPane(textArea);
        bt1 = new JButton("One");
        bt2 = new JButton("Two");
        bt3 = new JButton("Three");
        bt4 = new JButton("Four");
        bt5 = new JButton("Five");
    }
}
```

# JPanel Class

```
    frame = new JFrame("JPanelDemo");
    frame.setLayout(new BorderLayout());
    panelTop.add(bt1, BorderLayout.NORTH);
    panelTop.add(bt2, BorderLayout.SOUTH);
    panelBottom.add(bt3, BorderLayout.WEST);
    panelBottom.add(bt4, BorderLayout.EAST);
    panelBottom.add(bt5, BorderLayout.CENTER);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.add(panelTop, BorderLayout.NORTH);
    frame.add(panelBottom, BorderLayout.SOUTH);
    frame.add(scrollPane, BorderLayout.CENTER);
}
public void go()
{
    frame.setSize(600, 400);
    frame.setVisible(true);
}
public static void main(String args[])
{
    JPanelDemo demo = new JPanelDemo();
    demo.go();
}
}
```

# GridBagLayout Class

- ❖ The `GridBagLayout` manager arranges the components in a rectangular grid.

Each component can occupy different multiple number of rows and columns.

Unlike the `GridLayout`, using the `GridBagLayout` each component can have a different size.

- ❖ The position and the behavior of each component is specified by a `GridBagConstraints` object.

Each time we add a component we specify the `GridBagConstraints` object that should be used.

# GridBagLayout Class

- ❖ Each time we add a component we can configure the `GridBagConstraints` object in a different way.  
This way, each component can be positioned in a different position and occupy a different number of rows and columns.
- ❖ The number of columns and rows of our container is based on the number of components we added and the `GridBagConstraints` objects we used when adding them.  
The `GridBagLayout.MAXGRIDSIZE` defines the maximum capacity of a screen as 512 rows by 512 columns.

# GridBagLayout Class



# GridBagLayout Class

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

public class GridBagLayoutDemo
{
    private JFrame frame;
    private JButton btOne, btTwo, btThree, btFour, btFive;
    public GridBagLayoutDemo()
    {
        btOne = new JButton("One");
        btTwo = new JButton("Two");
        btThree = new JButton("Three");
        btFour = new JButton("Four");
        btFive = new JButton("Five");
        frame = new JFrame("GridBagLayout Demo");
        frame.setLayout(new GridBagLayout());
    }
}
```

# GridBagLayout Class

```
GridBagConstraints gbc = new GridBagConstraints(  
    0, //component will be placed in col num 0  
    0, //component will be placed in row num 0  
    1, //component will occupy one cell wide  
    1, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0,  
    0  
);  
frame.add(btOne,gbc);
```

# GridBagLayout Class

```
gbc = new GridBagConstraints(  
    1, //component will be placed in col num 1  
    0, //component will be placed in row num 0  
    1, //component will occupy one cell wide  
    1, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0, //no internal horizontal padding space  
    0 //no internal vertical padding space  
);  
frame.add(btTwo,gbc);
```

# GridBagLayout Class

```
gbc = new GridBagConstraints(  
    2, //component will be placed in col num 2  
    0, //component will be placed in row num 0  
    1, //component will occupy one cell wide  
    1, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0, //no internal horizontal padding space  
    0 //no internal vertical padding space  
);  
frame.add(btThree,gbc);
```

# GridBagLayout Class

```
gbc = new GridBagConstraints(  
    0, //component will be placed in col num 0  
    1, //component will be placed in row num 1  
    2, //component will occupy two cell wide  
    2, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0, //no internal horizontal padding space  
    0 //no internal vertical padding space  
);  
frame.add(btFour,gbc);
```

# GridBagLayout Class

```
gbc = new GridBagConstraints(  
    2, //component will be placed in col num 2  
    1, //component will be placed in row num 1  
    1, //component will occupy one cell wide  
    1, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0, //no internal horizontal padding space  
    0 //no internal vertical padding space  
);  
frame.add(btFive,gbc);
```

# GridBagLayout Class

```
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void go()
    {
        frame.setVisible(true);
    }

    public static void main(String args[])
    {
        Runnable runner = new Runnable()
        {
            public void run()
            {
                GridBagLayoutDemo demo = new GridBagLayoutDemo();
                demo.go();
            }
        };
        EventQueue.invokeLater(runner);
    }
}
```

# Layout Managers

09/22/08

© 2008 Haim Michael

1

## LayoutManager Interface

- ❖ The `LayoutManager` interface defines the object responsible for the components layout.
- ❖ Each container is registered with a `LayoutManager` object.
- ❖ We can register a new `LayoutManager` object by calling the `Container.setLayout()` method.  
`public void setLayout(LayoutManager manager)`

## LayoutManager Interface

- ❖ Calling the `Container.setLayout()` method and passing 'null' will deactivate the layout manager.

Doing so will allow us to specify the exact dimension and location of each component.

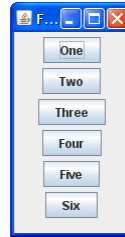
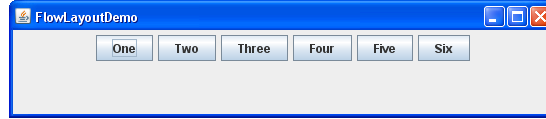
- ❖ The biggest advantage of using layout manager is their elimination of the need to compute each component optimal dimension and location.

Using a layout manager our application will automatically adjust itself to the exact dimension of the screen in accordance with the execution environment.

## FlowLayout Class

- ❖ `FlowLayout` is `JPanel` default `LayoutManager`.
- ❖ Adding components to container that its layout manager is a `FlowLayout` object, the components will be added left to right and top to bottom. The size of each component will be its preferred size.

# FlowLayout Class



## FlowLayout Class

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

public class FlowLayoutDemo
{
    private JFrame frame;
    private JButton bt1, bt2, bt3, bt4, bt5, bt6;
    public FlowLayoutDemo()
    {
        bt1 = new JButton("One");
        bt2 = new JButton("Two");
        bt3 = new JButton("Three");
        bt4 = new JButton("Four");
        bt5 = new JButton("Five");
        bt6 = new JButton("Six");
        frame = new JFrame("FlowLayoutDemo");
        frame.setLayout(new FlowLayout());
    }
}
```

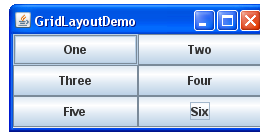
## FlowLayout Class

```
        frame.add(bt1);
        frame.add(bt2);
        frame.add(bt3);
        frame.add(bt4);
        frame.add(bt5);
        frame.add(bt6);
    }
    public void go()
    {
        frame.setSize(600,400);
        frame.setVisible(true);
    }
    public static void main(String args[])
    {
        FlowLayoutDemo demo = new FlowLayoutDemo();
        demo.go();
    }
}
```

## GridLayout Class

- ❖ The `GridLayout` layout manager lays the components organized in rows and in columns.
- ❖ Each cell has the same size. Each components is placed within one cell and spreads to cover its entire area.
- ❖ Components are added left to right top to bottom.
- ❖ When instantiating the `GridLayout` class we specify the number of rows and the number of columns we want to set in our container.

# GridLayout Class



## GridLayout Class

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

public class GridLayoutDemo
{
    private JFrame frame;
    private JButton bt1, bt2, bt3, bt4, bt5, bt6;
    public GridLayoutDemo()
    {
        bt1 = new JButton("One");
        bt2 = new JButton("Two");
        bt3 = new JButton("Three");
        bt4 = new JButton("Four");
        bt5 = new JButton("Five");
        bt6 = new JButton("Six");
        frame = new JFrame("GridLayoutDemo");
    }
}
```

## GridLayout Class

```
        frame.setLayout(new GridLayout(3,2));
        frame.add(bt1);
        frame.add(bt2);
        frame.add(bt3);
        frame.add(bt4);
        frame.add(bt5);
        frame.add(bt6);
    }
    public void go()
    {
        frame.setSize(600,400);
        frame.setVisible(true);
    }
    public static void main(String args[])
    {
        GridLayoutDemo demo = new GridLayoutDemo();
        demo.go();
    }
}
```

## BorderLayout Class

- ❖ The `BorderLayout` layout manager lays the components organized in the following regions:

`BorderLayout.NORTH`

`BorderLayout.SOUTH`

`BorderLayout.WEST`

`BorderLayout.EAST`

`BorderLayout.CENTER`

- ❖ Each region can include up to one component only.
- ❖ Regions without any component are conquered by the `BorderLayout.CENTER` region.

## BorderLayout Class

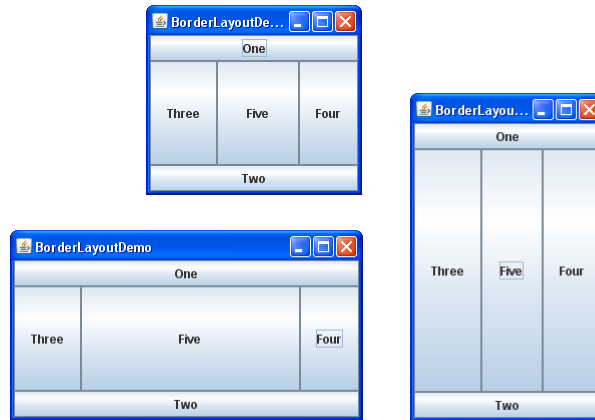
- ❖ When adding a component to container that its layout manager is a `BorderLayout` object we should specify the region to which the component should be added.

```
...  
JFrame frame = new JFrame();  
frame.setLayout(new BorderLayout());  
frame.add(new JButton("One"), BorderLayout.North);  
...
```

## BorderLayout Class

- ❖ Components added to `BorderLayout.NORTH` and to `BorderLayout.SOUTH` regions will span to the entire width. Their height will be their preferred one.
- ❖ Components added to `BorderLayout.WEST` and to `BorderLayout.EAST` regions will span to the entire available height. Their width will be their preferred one.
- ❖ `BorderLayout` is the default layout manager of the `JFrame` content pane.

# BorderLayout Class



## BorderLayout Class

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

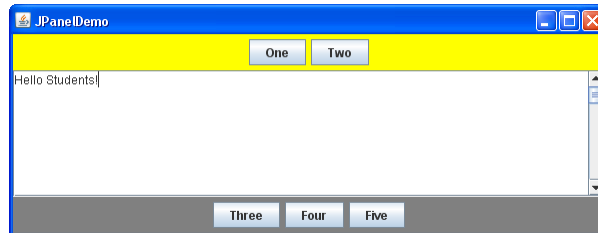
public class BorderLayoutDemo
{
    private JFrame frame;
    private JButton bt1, bt2, bt3, bt4, bt5;
    public BorderLayoutDemo()
    {
        bt1 = new JButton("One");
        bt2 = new JButton("Two");
        bt3 = new JButton("Three");
        bt4 = new JButton("Four");
        bt5 = new JButton("Five");
        frame = new JFrame("BorderLayoutDemo");
    }
}
```

## BorderLayout Class

```
frame.setLayout(new BorderLayout());
frame.add(bt1, BorderLayout.NORTH);
frame.add(bt2, BorderLayout.SOUTH);
frame.add(bt3, BorderLayout.WEST);
frame.add(bt4, BorderLayout.EAST);
frame.add(bt5, BorderLayout.CENTER);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public void go()
{
    frame.setSize(600, 400);
    frame.setVisible(true);
}
public static void main(String args[])
{
    BorderLayoutDemo demo = new BorderLayoutDemo();
    demo.go();
}
}
```

## JPanel Class

- ❖ `JPanel` is both a component and a container. As a component we can place it on other containers. As a container we can place other components on it.



## JPanel Class

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

public class JPanelDemo
{
    private JFrame frame;
    private JTextArea textArea;
    private JScrollPane scrollPane;
    private JPanel panelTop, panelBottom;
    private JButton bt1, bt2, bt3, bt4, bt5;
    public JPanelDemo()
    {
        panelTop = new JPanel();
        panelBottom = new JPanel();
        panelTop.setBackground(Color.YELLOW);
        panelBottom.setBackground(Color.GRAY);
        textArea = new JTextArea(40,24);
        scrollPane = new JScrollPane(textArea);
        bt1 = new JButton("One");
        bt2 = new JButton("Two");
        bt3 = new JButton("Three");
        bt4 = new JButton("Four");
        bt5 = new JButton("Five");
    }
}
```

## JPanel Class

```
frame = new JFrame("JPanelDemo");
frame.setLayout(new BorderLayout());
panelTop.add(bt1, BorderLayout.NORTH);
panelTop.add(bt2, BorderLayout.SOUTH);
panelBottom.add(bt3, BorderLayout.WEST);
panelBottom.add(bt4, BorderLayout.EAST);
panelBottom.add(bt5, BorderLayout.CENTER);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.add(panelTop, BorderLayout.NORTH);
frame.add(panelBottom, BorderLayout.SOUTH);
frame.add(scrollPane, BorderLayout.CENTER);
}
public void go()
{
    frame.setSize(600,400);
    frame.setVisible(true);
}
public static void main(String args[])
{
    JPanelDemo demo = new JPanelDemo();
    demo.go();
}
}
```

## GridBagLayout Class

- ❖ The `GridBagLayout` manager arranges the components in a rectangular grid.

Each component can occupy different multiple number of rows and columns.

Unlike the `GridLayout`, using the `GridBagLayout` each component can have a different size.

- ❖ The position and the behavior of each component is specified by a `GridBagConstraints` object.

Each time we add a component we specify the `GridBagConstraints` object that should be used.

## GridBagLayout Class

- ❖ Each time we add a component we can configure the `GridBagConstraints` object in a different way. This way, each component can be positioned in a different position and occupy a different number of rows and columns.
- ❖ The number of columns and rows of our container is based on the number of components we added and the `GridBagConstraints` objects we used when adding them. The `GridBagLayout.MAXGRIDSIZE` defines the maximum capacity of a screen as 512 rows by 512 columns.

# GridBagLayout **Class**



09/22/08

© 2008 Haim Michael

23

## GridBagLayout Class

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

public class GridBagLayoutDemo
{
    private JFrame frame;
    private JButton btOne, btTwo, btThree, btFour, btFive;
    public GridBagLayoutDemo()
    {
        btOne = new JButton("One");
        btTwo = new JButton("Two");
        btThree = new JButton("Three");
        btFour = new JButton("Four");
        btFive = new JButton("Five");
        frame = new JFrame("GridBagLayout Demo");
        frame.setLayout(new GridBagLayout());
    }
}
```

## GridBagLayout Class

```
GridBagConstraints gbc = new GridBagConstraints(  
    0, //component will be placed in col num 0  
    0, //component will be placed in row num 0  
    1, //component will occupy one cell wide  
    1, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0,  
    0  
);  
frame.add(btOne,gbc);
```

## GridBagLayout Class

```
gbc = new GridBagConstraints(  
    1, //component will be placed in col num 1  
    0, //component will be placed in row num 0  
    1, //component will occupy one cell wide  
    1, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0, //no internal horizontal padding space  
    0 //no internal vertical padding space  
);  
frame.add(btTwo,gbc);
```

## GridBagLayout Class

```
gbc = new GridBagConstraints(  
    2, //component will be placed in col num 2  
    0, //component will be placed in row num 0  
    1, //component will occupy one cell wide  
    1, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0, //no internal horizontal padding space  
    0 //no internal vertical padding space  
);  
frame.add(btThree, gbc);
```

## GridBagLayout Class

```
gbc = new GridBagConstraints(  
    0, //component will be placed in col num 0  
    1, //component will be placed in row num 1  
    2, //component will occupy two cell wide  
    2, //component will occupy one cell height  
    0,  
    0,  
    GridBagConstraints.CENTER,  
    GridBagConstraints.BOTH,  
    new Insets(0,0,0,0),  
    0, //no internal horizontal padding space  
    0 //no internal vertical padding space  
);  
frame.add(btFour,gbc);
```

## GridBagLayout Class

```
gbc = new GridBagConstraints(
    2, //component will be placed in col num 2
    1, //component will be placed in row num 1
    1, //component will occupy one cell wide
    1, //component will occupy one cell height
    0,
    0,
    GridBagConstraints.CENTER,
    GridBagConstraints.BOTH,
    new Insets(0,0,0,0),
    0, //no internal horizontal padding space
    0 //no internal vertical padding space
);
frame.add(btFive,gbc);
```

## GridBagLayout Class

```
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void go()
    {
        frame.setVisible(true);
    }

    public static void main(String args[])
    {
        Runnable runner = new Runnable()
        {
            public void run()
            {
                GridBagLayoutDemo demo = new GridBagLayoutDemo();
                demo.go();
            }
        };
        EventQueue.invokeLater(runner);
    }
}
```

09/22/08

© 2008 Haim Michael

30