SCHULICH School of Engineering

ENSF 593/594

13 – Java Graphical User Interface (Java GUI) – Part I

GUI Fundamentals

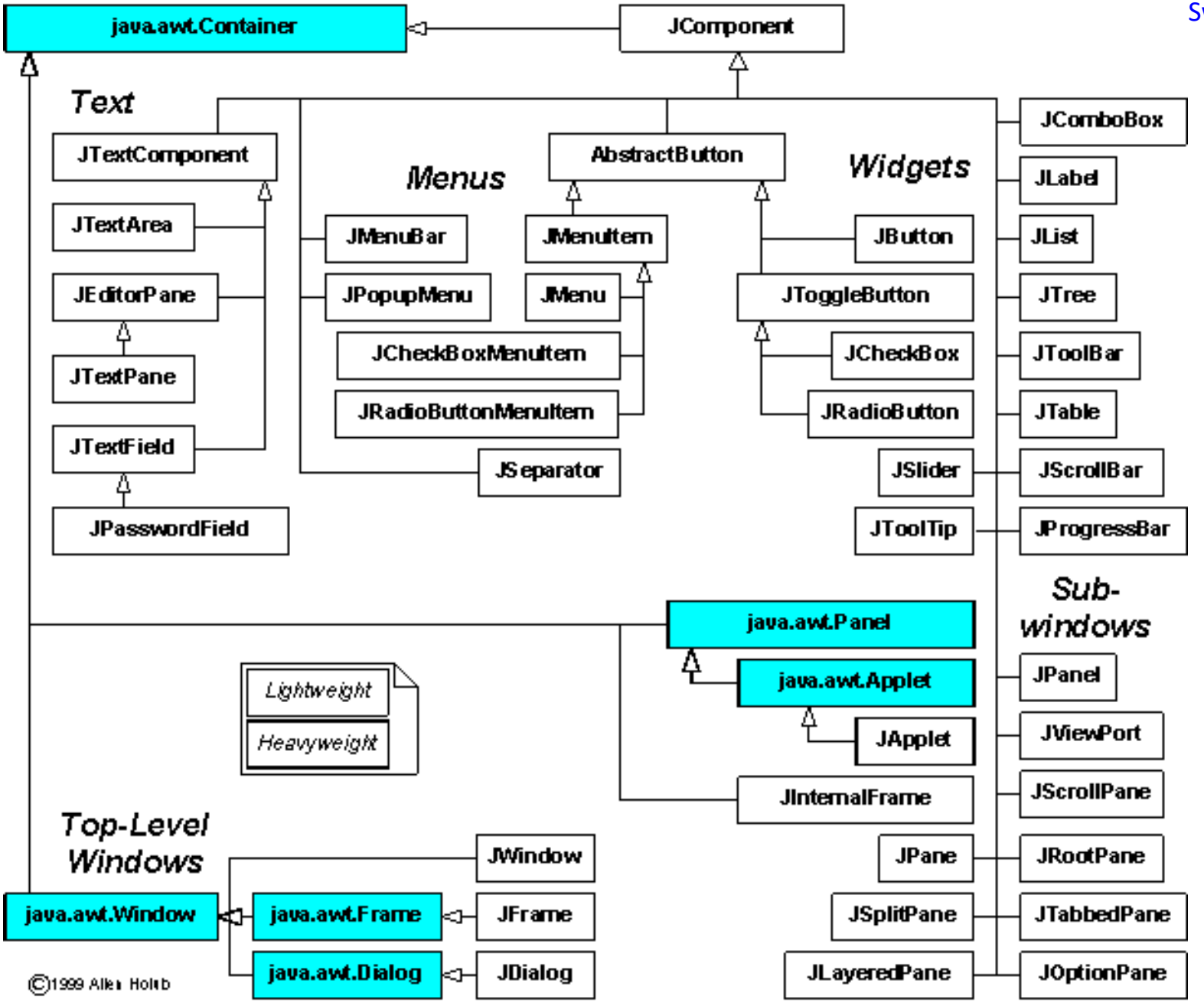
Java Foundation Classes (JFC)

- Abstract Window Toolkit (AWT)
  - original user interface toolkit
- Swing
  - package javax.swing.*, introduced in Java 1.2

What is Swing

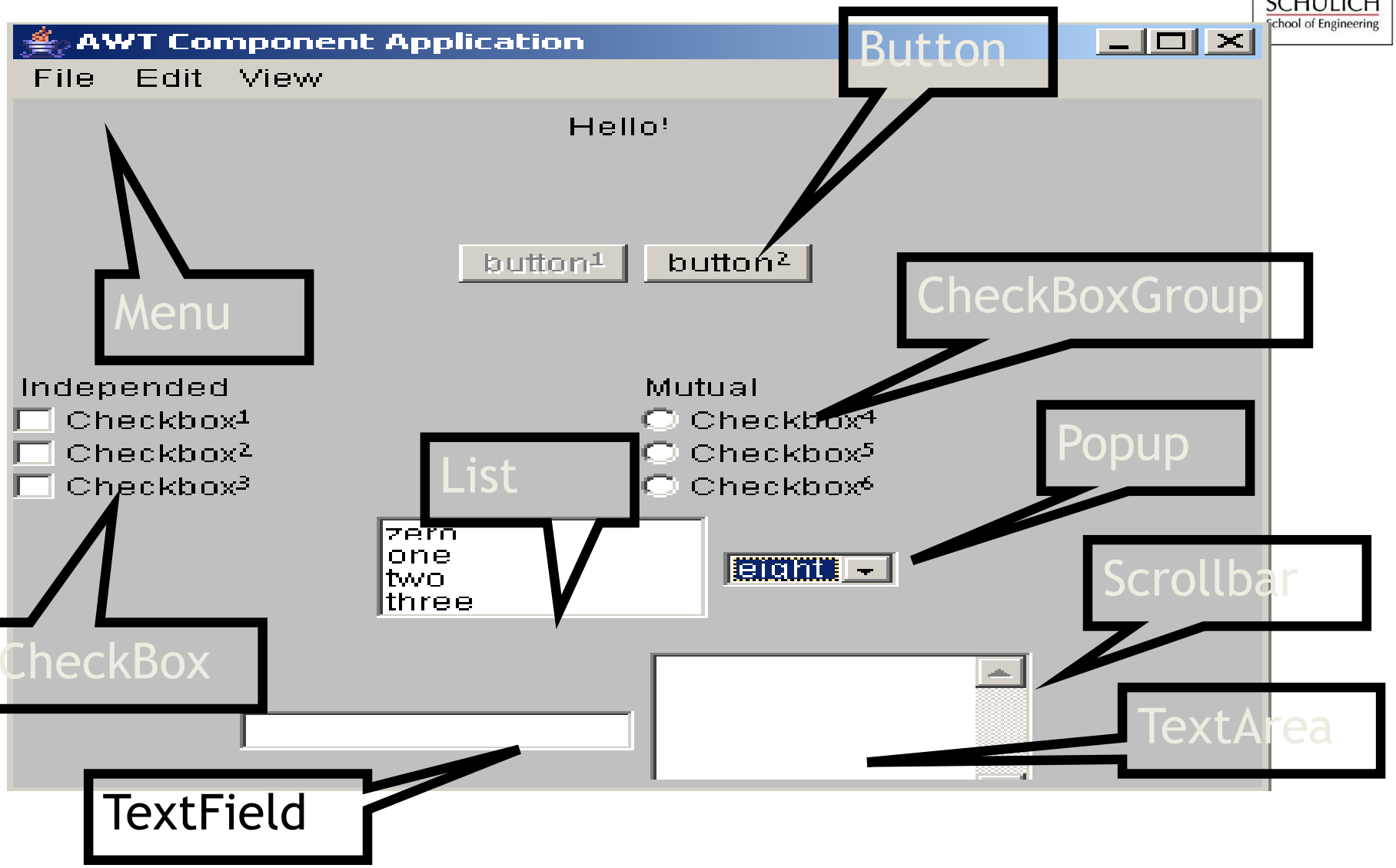
- Provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (awt).
- Emulates a native look and feel of several platforms.
- Supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.

Swing Classes

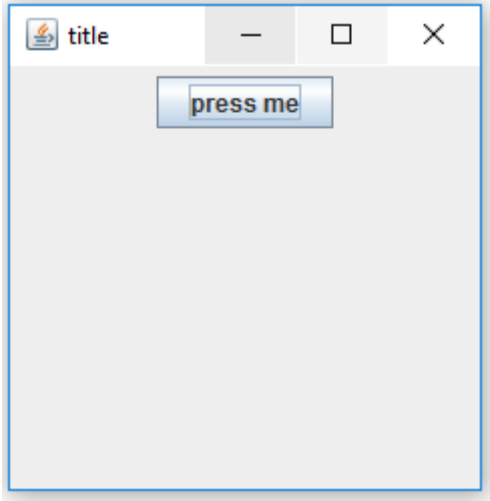


http://www.holub.com/goodies/java.swing.html

What are Java Components



Simple GUI Code




```
import javax.swing.*;
public class SimpleGUI {

        public static void main(String[] args){
                JFrame f = new JFrame("title");
                JPanel p = new JPanel();
                JButton b = new JButton("press me");

                p.add(b);// add button to panel
                f.setContentPane(p);   // add panel to frame

                f.setSize(200, 200);
                f.setVisible(true);
        }
}
```

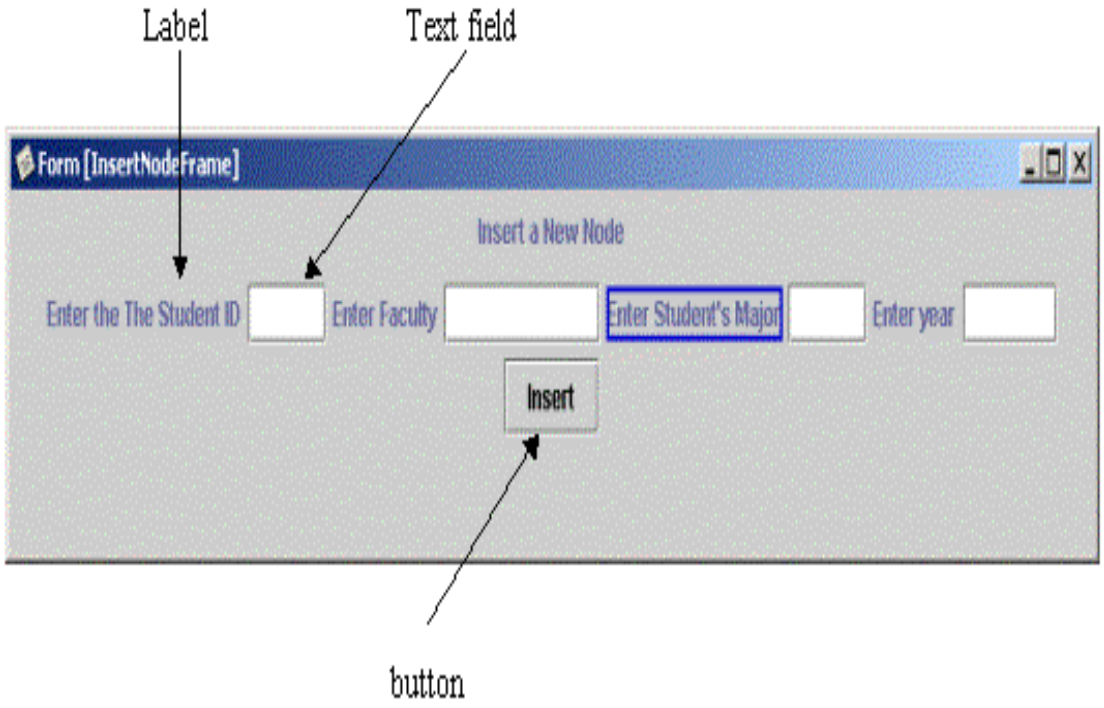
Different Levels of Containment

- **Top-level container:**
  - place for other Swing components to paint themselves
  - e.g., JFrame, JDialog, Japplet
- **Intermediate container:**
  - simplify positioning of atomic components
  - e.g., JPanel, JSplitPane, JTabbedPane.
- **Atomic components:**
  - self-sufficient components that present information to and get input from the user
  - e.g., JButton, JLabel, JComboBox, JTextField, JTable

What is Java Frame

Java Frames

A frame, is top level window that has a border, a title, and may contain buttons, text fields, or other GUI components. GUI applications usually use at least one frame.



Java Frame – Simple Code

```
import javax.swing.JButton;

public class MyFrame extends javax.swing.JFrame {
        private Integer num;
        private JButton y;

        public MyFrame(Integer x) {
                setSize(900,600);
                y = new JButton ("OK");
                num = x;
                ….
        }
}
```

Java Dialog Boxes

- Another important GUI in java is a dialog box. A dialog box can be created, by using a `java.Dialog` or `javax.swing.JDialog` class.
- Normally dialog boxes are used to **show a message**, like an error message to a user, or receiving a user's input.
- Two commonly used dialog boxes that can be created easily by calling methods:
  - `showMessageDialog`
  - `showInputDialog`

Methods that Create Dialog Boxes

- To be able to create standard input, output, of message boxes you need to import another class for swing package called ***JOptionPane***.
  - `import javax.swing.JOptionPane`
- Methods such as **`showMessageDialog`** or **`showInputDialog`** are **static methods** to be used

Different Types of Dialog Boxes:

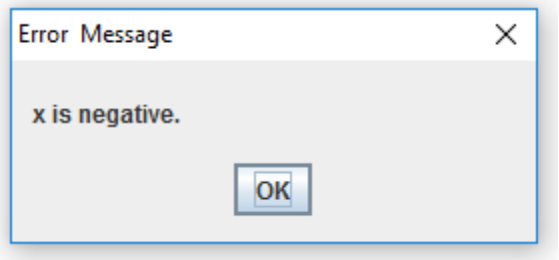
- ERROR_MESSAGE
- INFORMATION_MESSAGE
- WARNING_MESSAGE
- QUESTION_MESSAGE
- PLAIN_MESSAGE

Dialog Box Example 1

The following code:

```
if ( x < 0) {
        JOptionPane.showMessageDialog(null,   "x is negative.",
                "Error Message",  JOptionPane.PLAIN_MESSAGE);
}
```

Will produce the flowing dialog box:

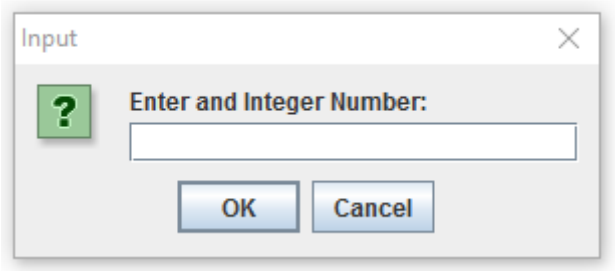


Dialog Box Example 2

The following code:

```
num = Integer.parseInt(JOptionPane.showInputDialog("Enter and
                                                    Integer Number: "));
```

Will produce the flowing dialog box:



What is Panel

javjava.awt.Container
javax.swing.JComponent

- Panel is the simplest container class.
- provides space in which an application can attach any other component, including other panels.

Layout Managers

- The components are laid out on a container using a layout manager.
- The size and position of components are controlled by a layout manager.
- It is also possible to define new layout managers.
- Every container is assigned a default layout manager when it is first created.
- For `Window`, the default layout manager is a `BorderLayout`.

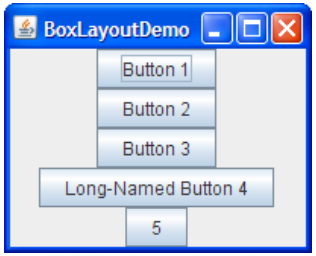
Different Types of Java Layout

- Several AWT and Swing classes provide layout managers for general use:
  - BorderLayout
  - BoxLayout
  - CardLayout
  - FlowLayout
  - GridBagLayout
  - GridLayout
  - GroupLayout
  - SpringLayout

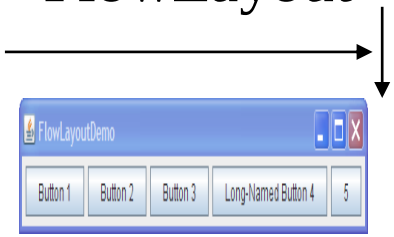
- For details see:
  http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html#grid

Examples of Layouts

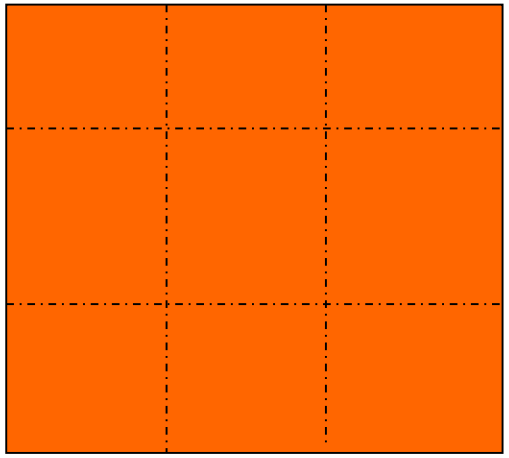
BoxLayout



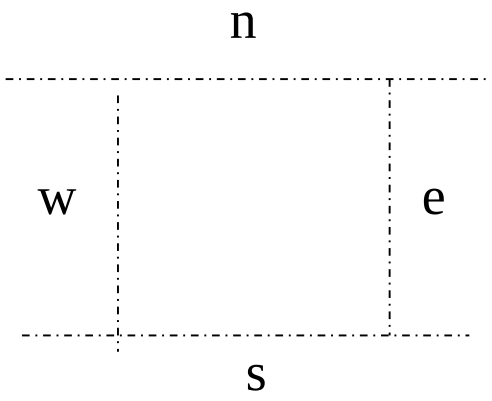
FlowLayout

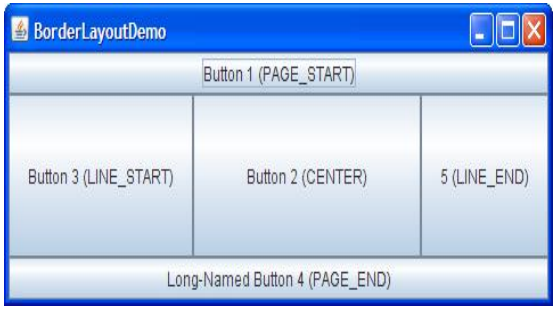


GridLayout

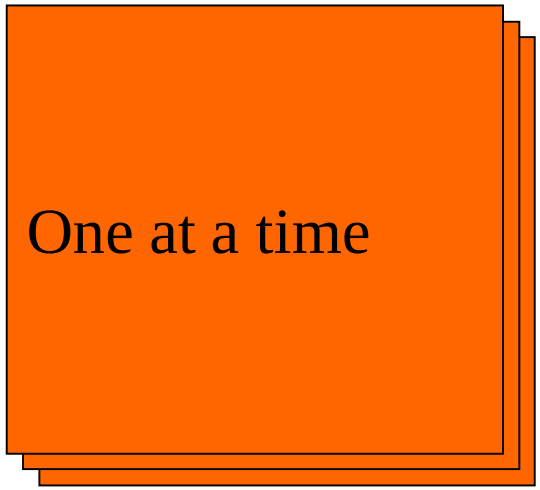


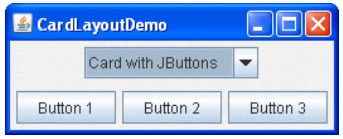
BorderLayout

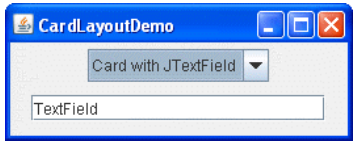




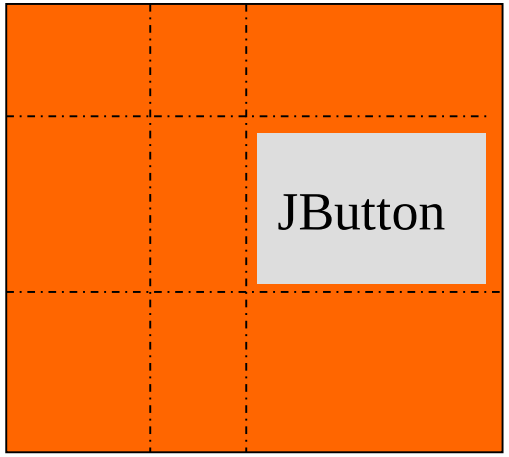
CardLayout







GridBagLayout



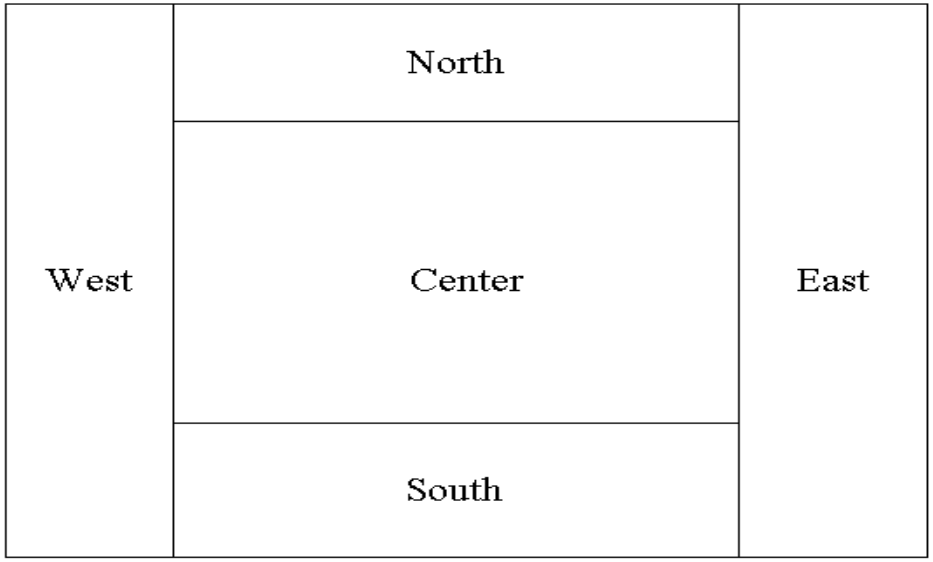
http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

BorderLayout Manager

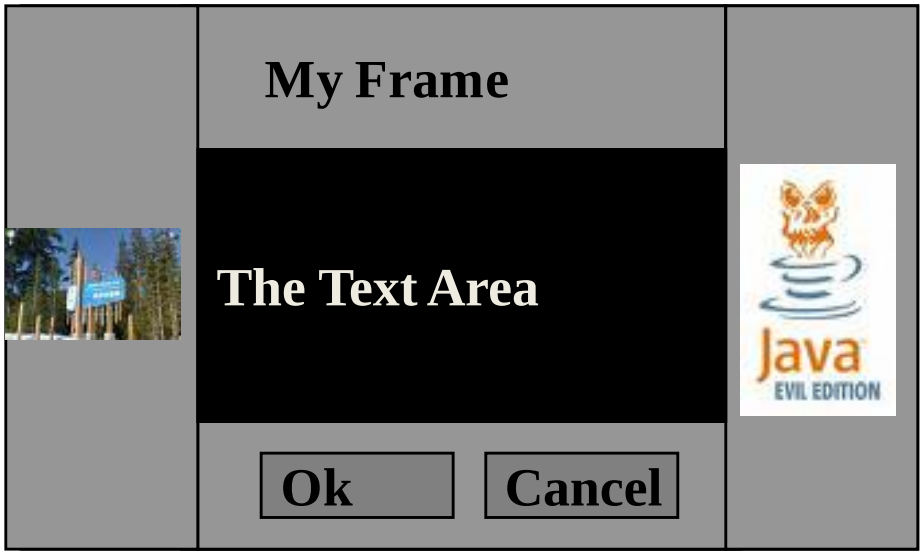
- A BorderLayout places one component in the center of a container. The central component is surrounded by up to four other components that border it to the "North", "South", "East", and "West",
- Each of the four bordering components is optional.
- The layout manager first allocates space to the bordering components. Any space that is left over goes to the center component.
- If a container uses a BorderLayout, then components should be added to the container using a version of the add() method that has two parameters.

- Look at the API for the add method and constructor.

Default Layout: Border Layout



Schematic View of a Frame Using Boarder Layout



BorderLayout Example

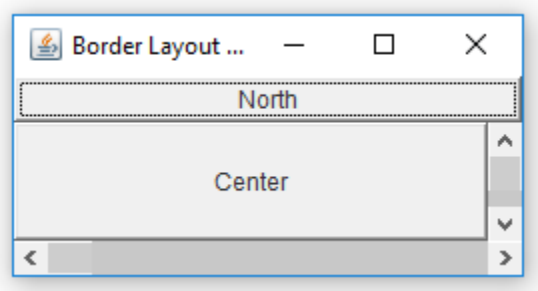
```
import java.awt.*;
import javax.swing.*;

class  BorderLayoutExample extends JFrame
{
   public BorderLayoutExample (int  widthInPixels,  int heightInPixels )
  {
      setTitle ( "Border Layout Example");
      setSize(widthInPixels, heightInPixels);

      setLayout (new BorderLayout ( ) );

      add( "North", new Button( "North" ) );
      add( "Center", new Button( "Center" ) );
      add( "East", new Scrollbar( Scrollbar.VERTICAL ) );
      add( "South", new Scrollbar( Scrollbar.HORIZONTAL ) );
      setVisible(true);
   }

   public static void main (String [] str){
   BorderLayoutExample e = new BorderLayoutExample(350, 300);
   }
}
```



FlowLayout Manager

- The components are arranged in the container from left to right in the order in which they were added. When one row becomes filled, a new row is started.

- `FlowLayout` is the default Layout Manager for Frames and Panels.

- Look at the API for `FlowLayout` constructor and the add method

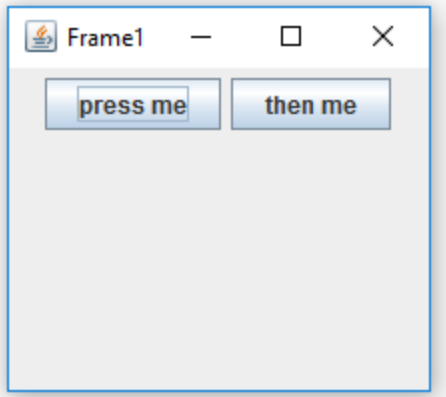
Code: FlowLayout

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

public class MyClass extends JFrame {

  public static void main(String[] str) {
    JFrame f = new JFrame("Frame1");
    JPanel p = new JPanel( );
    FlowLayout L = new FlowLayout( );
    JButton b1 = new JButton("press me");
    JButton b2 = new JButton("then me");

   f.setSize(200, 200);
   p.setLayout(L);
   p.add(b1);
   p.add(b2);
   f.setContentPane(p);
   f.setVisible(true);
 }
}
```



Note: Set layout before adding components

GridLayout Manager

- `GridLayout` arranges components into a grid of rows and columns.
- You can specify the number of rows and columns, or the number of rows only and let the layout manager determine the number of columns, or the number of columns only and let the layout manager determine the number of rows.
- The cells in the grid are **equal size**.
- Look at the API for the add method and constructor.

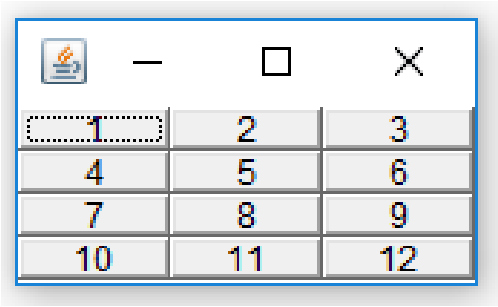
GridLayout Example

```
import java.awt.*;

class  GridLayoutExample  extends Frame
{
    public GridLayoutExample( int  widthInPixels, int heightInPixels )
    {
      setTitle( "Grid Example" );
      setSize ( widthInPixels, heightInPixels);
      int numberOfRows = 4;
      int numberOfColumns = 3;
      setLayout( new GridLayout( numberOfRows,
                                 numberOfColumns ) );

      for (int label = 1; label < 13; label++ )
         add( new Button( String.valueOf( label ) ) );
      setVisible(true);
     }

    public  static  void  main( String  args[] )
   {
      new  GridLayoutExample( 175, 100 );
    }
} // END OF CLAS
```



Learn More

- Visit the following site to learn more about Java Layout Manager:
  - http://download.oracle.com/javase/tutorial/uiswing/layout/

How to Add a New Component

- To be able to use the GUI objects such as a panel, button, textbox, etc, you need to import the following packages:

```
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
…
import javax.swing.JFrame;
```

Declare Components

- The next step is to use the Java class objects to create the component such as buttons, labels textboxes etc:

```
public class MyFrame extends JFrame {
    private JButton openButton;
    private JLabel  titleLabel;
     …  // more reference component, as needed

     … // other method
} // end of class MyFrame
```

Declaring Panels

```
public class MyFrame extends JFrame
{
    ...
    private JLabel titleLabel;
    private JPanel titlePanel;
    private JPanel buttonsPanel;
        ...
        ...
```

Creating Panels

```
    …  // more reference component, as needed
    public MyFrame ()
    {
         titlePanel = new JPanel();
          buttonsPanel = new JPanel();
          … // more object components, as need
    }
    … // other method
}
```

Adding Components to the Panel

- *Next step is to add components to those panel:*

```
public class MyFrame extends JFrame {
    public MyFrame () {
        ...
        titlePanel.add(titleLabel);
        buttonsPanel.add(openButton);
        … // more object components, as need
    }
    … // other method
}
```

Getting Hold of the Container

- Now each panel must be added to the container such as: **JFrame**, **JApplet**, or **JDialog Box**.
  - But first you need to get access to the container object of this container. This can be simply done by calling the method **getContentPane**:

```
Container c;
c = getContentPane();
```

  - A container is a *kind of* Abstract Window Toolkit(AWT) object that can contain other AWT components.
  - Your Window inherits the method **getContentPane** from the parent class (e.g. **Frame**).

How to Add a New Components …

```
public class MyFrame extends JFrame {
        private JButton openButton;
        private JLabel titleLabel;
        Container c;

        public MyFrame() {
            // …
            c = getContentPane();

            c.add(openButton, "north");
            c.add(titleLabel, "south");
        }
    … // other method
}
```

Get the handle to the frame's container

Adding Components to the container

Painting

- Painting occurs when
  - A component is displayed for the first time
  - The component is uncovered or unhidden
  - Something in the display has changed
- Painting starts from the highest component and works down
  - Every container is painted before the components it contains

Next: Event Handling In Java