Programming in Java

Dr. Nyein Aye Maung Maung
Dr. Eng (Ritsumeikan University, Japan)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Course Schedule

Week	Topics
Week 1	Overview of JAVA, Data Types, Variables and Arrays
Week 2	Operators and Control Statements
Week 3	Classes and A closer look at Methods and Classes
Week 4	Inheritance, Polymorphism, Abstraction and Encapsulation
Week 5	Packages and Interfaces
Week 6	Exception Handling and Multi-threaded Programming
Week 7	String Handling
Week 8	Exploring java.lang and More utilities classes
Week 9	Java Collections Framework
Week 10	Java I/O
Week 11	Basic Graphical User Interface
Week 12	Event Handling
Week 13	Database Programming
Week 14	Applet and Networking

Lecture 11

Java Graphical User Interface

Outline of Class

- Swing and AWT
- GUI API Hierarchy
- GUI Components

Java Graphical User Interface

- There are two sets of Java APIs for graphics programming:
  - AWT (Abstract Windowing Toolkit) and Swing.
- AWT API was introduced in JDK 1.0. AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects. Most of the AWT components have become obsolete and should be replaced by newer Swing components.
- Swing API introduces a much more comprehensive set of graphics libraries that enhances the AWT

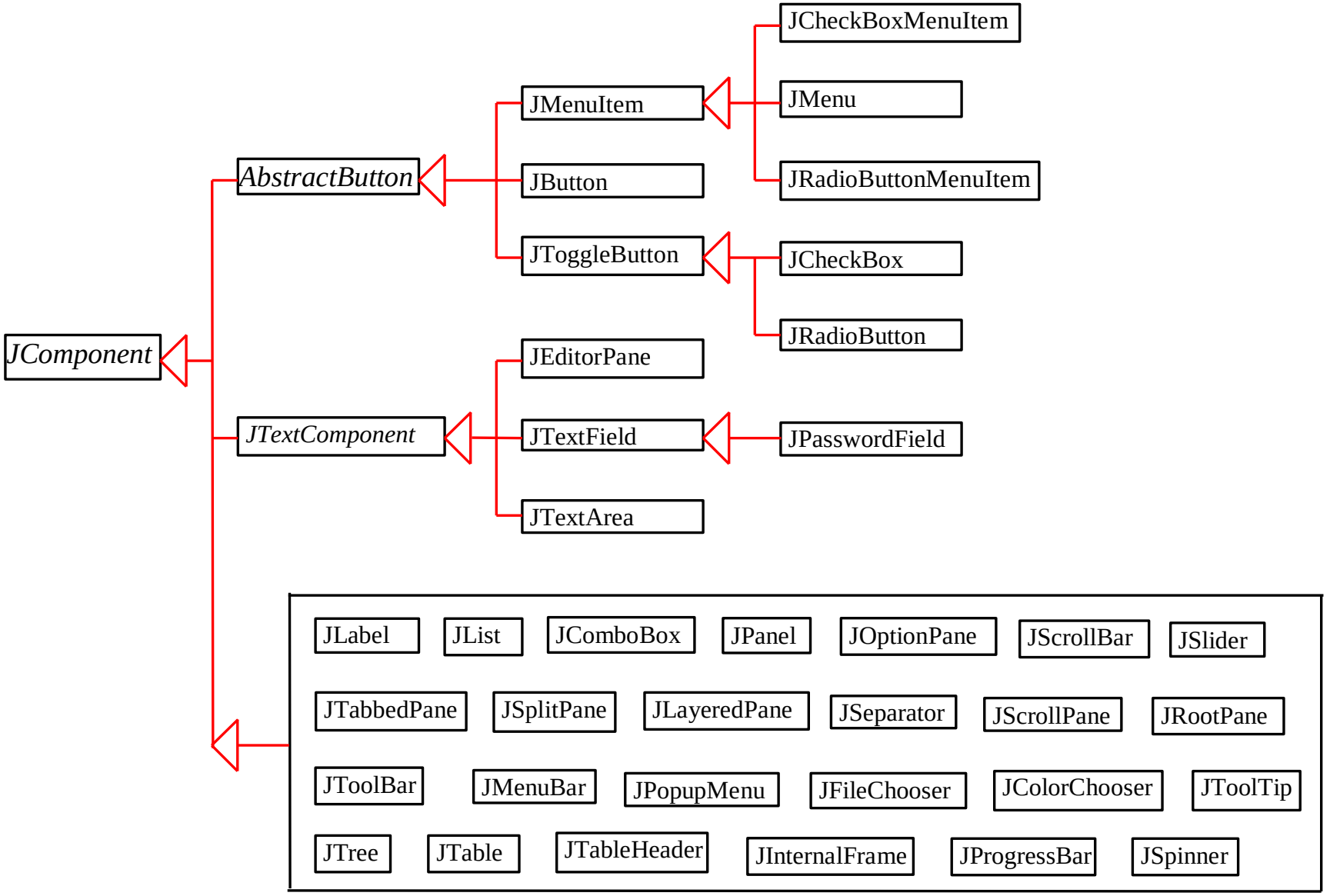
Java GUI API

- The GUI API contains classes that can be classified into three groups:
  - **Component classes**
    - Components are elementary GUI entities (such as Button, Label, and TextField.)
  - **Container classes**
    - Containers (such as Frame and Panel) are used to hold components in a specific layout
  - **Helper classes**
    - classes, such as Graphics , Color , Font , FontMetrics , and Dimension , are called helper classes used to support GUI components

(1) Containers

- An instance of Container can hold instances of Component
  - **javax.swing.Jframe**
    - a top-level container for holding other Swing user-interface components in Java GUI applications.
  - **javax.swing.Jpanel**
    - an invisible container for grouping user-interface components. Panels can be nested. You can place panels inside another panel. JPanel is also often used as a canvas to draw graphics.
  - **javax.swing.Japplet**
    - a base class for creating a Java applet using Swing components.
  - **javax.swing.Jdialog**
    - a popup window generally used as a temporary window to receive additional information from the user or to provide notification to the user.

(2) Components



(3) GUI Helper Classes

- The helper classes are not subclasses of Component. They are used to describe the properties of GUI components such as graphics context, colors, fonts, and dimension.
- **java.awt.Graphics**
  - an abstract class that provides the methods for drawing strings, lines, and simple shapes.
- **java.awt.Color**
  - deals with the colors of GUI components. For example, you can specify background or foreground colors in components like JFrame and JPanel, or you can specify colors of lines, shapes, and strings in drawings.
- **java.awt.Font**
  - specifies fonts for the text and drawings on GUI components. For example, you can specify the font type (e.g., SansSerif), style (e.g., bold), and size (e.g., 24 points) for the text on a button.

(3) GUI Helper Classes (cont')

- **java.awt.FontMetrics**
  - an abstract class used to get the properties of the fonts.
- **java.awt.Dimension**
  - encapsulates the width and height of a component (in integer precision) in a single object.
- **java.awt.LayoutManager**
  - specifies how components are arranged in a container.

Basic GUI Programming Steps in Java

1. Declare a container and components
2. Add components to one or more containers using a layout manager
3. Register event listener(s) with the components
4. Create event listener method(s)

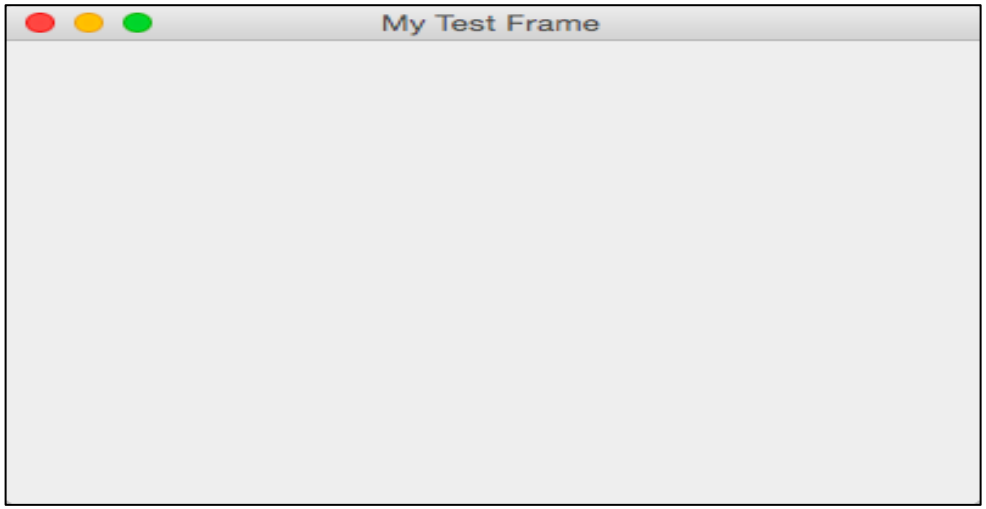
Frames

- A frame is a window for holding other GUI components

javax.swing.JFrame	
+JFrame()	Creates a default frame with no title.
+JFrame(title: String)	Creates a frame with the specified title.
setSize (width: int, height: int): void	Specifies the size of the frame.
+setLocation(x: int, y: int): void	Specifies the upper-left corner location of the frame.
+setVisible(visible: boolean): void	Sets true to display the frame.
+setDefaultCloseOperation(mode: int): void	Specifies the operation when the frame is closed.
+setLocationRelativeTo (c: Component): void	Sets the location of the frame relative to the specified component. If the component is null, the frame is centered on the screen.

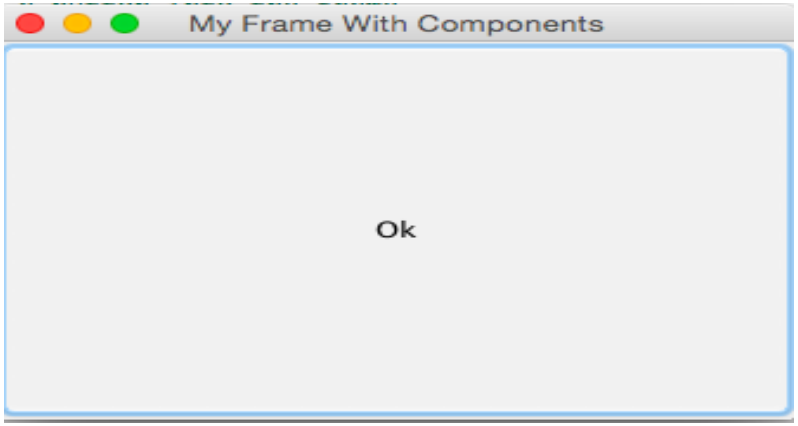
Example: JFrame

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

  public static void main(String[] args) {

    JFrame frame = new JFrame("MyFrame"); // Create a frame
    frame.setSize(400, 300); // Set the frame size
    frame.setLocationRelativeTo(null); // New since JDK 1.4
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true); // Display the frame

  }
}
```



Adding components to a frame

```
import javax.swing.*;
public class MyFrameWithComponents {
  public static void main(String[] args) {
    JFrame frame = new JFrame("My Frame With Components");
    // Add a button into the frame
    JButton jbtOK = new JButton("OK");
    frame.add(jbtOK);
    frame.setSize(400, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setVisible(true);
  }

}
```



Layout Managers

- Provided for arranging GUI components
- Provide basic layout capabilities
- Processes layout details
- Interface **`LayoutManager`**

Layout manager	Description
FlowLayout	Default for javax.swing.JPanel. Places components sequentially (left to right) in the order they were added. It is also possible to specify the order of the components by using the Container method add, which takes a Component and an integer index position as arguments.
BorderLayout	Default for JFrames (and other windows). Arranges the components into five areas: NORTH, SOUTH, EAST, WEST and CENTER.
GridLayout	Arranges the components into rows and columns.

(i) FlowLayout

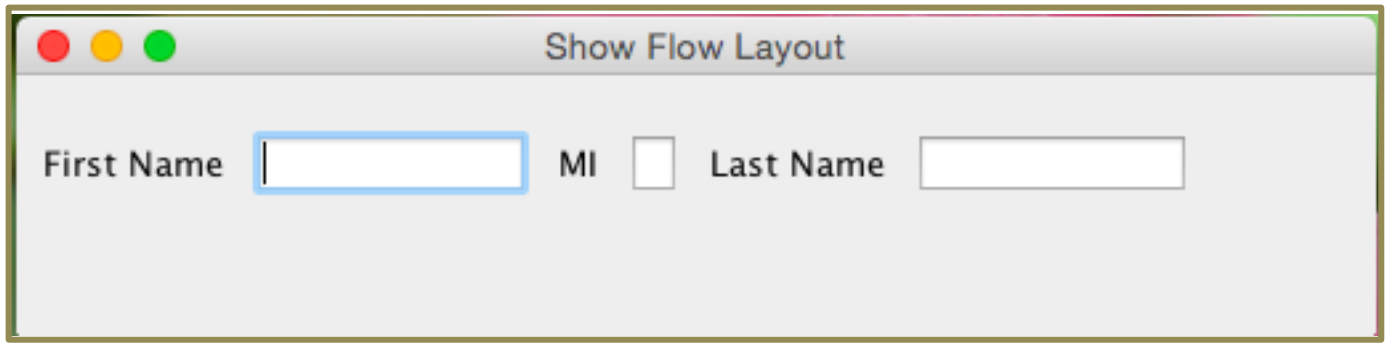
- FlowLayout has alignment, hgap, and vgap properties.
- Use setAlignment, setHgap, and setVgap methods to specify the alignment and the horizontal and vertical gaps.

java.awt.FlowLayout	
-alignment: int	The alignment of this layout manager (default: CENTER).
-hgap: int	The horizontal gap of this layout manager (default: 5 pixels).
-vgap: int	The vertical gap of this layout manager (default: 5 pixels).
+FlowLayout()	Creates a default FlowLayout manager.
+FlowLayout(alignment: int)	Creates a FlowLayout manager with a specified alignment.
+FlowLayout(alignment: int, hgap: int, vgap: int)	Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap.

Example 1. Write a program that adds three labels and text fields into the frame with a FlowLayout to accept user's First Name, Middle Name and Last name.

```
import javax.swing.*;

import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame {

  public ShowFlowLayout() {

    // Set FlowLayout, aligned left with horizontal gap 10 and vertical gap 20 between comp
onents

    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));

    // Add labels and text fields to the frame

    add(new JLabel("First Name"));

    add(new JTextField(8));

    add(new JLabel("MI"));

    add(new JTextField(1));

    add(new JLabel("Last Name"));

    add(new JTextField(8));

  }
```

```
  /** Main method */
  public static void main(String[] args) {
    ShowFlowLayout frame = new ShowFlowLayout();
    frame.setTitle("ShowFlowLayout");
    frame.setSize(200, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```



(ii) GridLayout

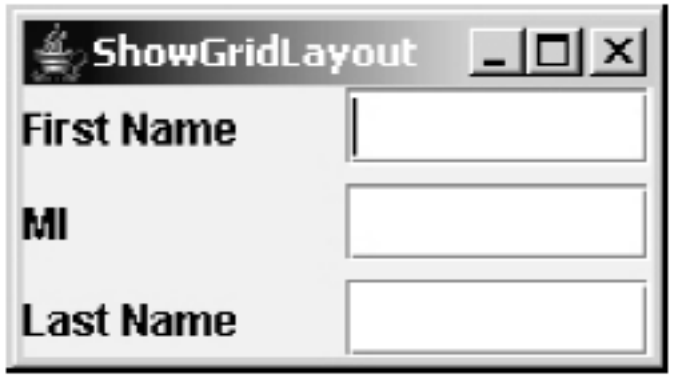
- The GridLayout manager arranges components in a grid (matrix) formation.
- The components are placed in the grid from left to right, starting with the first row, then the second, and so on, in the order in whi ch they are added.

java.awt.GridLayout	
-rows: int	The number of rows in this layout manager (default: 1).
-columns: int	The number of columns in this layout manager (default: 1).
-hgap: int	The horizontal gap of this layout manager (default: 0).
-vgap: int	The vertical gap of this layout manager (default: 0).
+GridLayout()	Creates a default GridLayout manager.
+GridLayout(rows: int, columns: int)	Creates a GridLayout with a specified number of rows and columns.
+GridLayout(rows: int, columns: int, hgap: int, vgap: int)	Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap.

Example 2. Write a program that adds three labels and text fields into the frame with a GridLayout to accept user's First Name, Middle Name and Last name.

```
import javax.swing.*;

import java.awt.GridLayout;

public class ShowFlowLayout extends JFrame {

  public ShowFlowLayout() {

    // Set FlowLayout, aligned left with horizontal gap 10 and vertical gap 20 between comp
onents

    setLayout(new GridLayout(3,2,5,5));

    // Add labels and text fields to the frame

    add(new JLabel("First Name"));

    add(new JTextField(8));

    add(new JLabel("MI"));

    add(new JTextField(1));

    add(new JLabel("Last Name"));

    add(new JTextField(8));

  }
```

```
  /** Main method */
  public static void main(String[] args) {
    ShowFlowLayout frame = new ShowFlowLayout();
    frame.setTitle("ShowGridLayout");
    frame.setSize(200, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

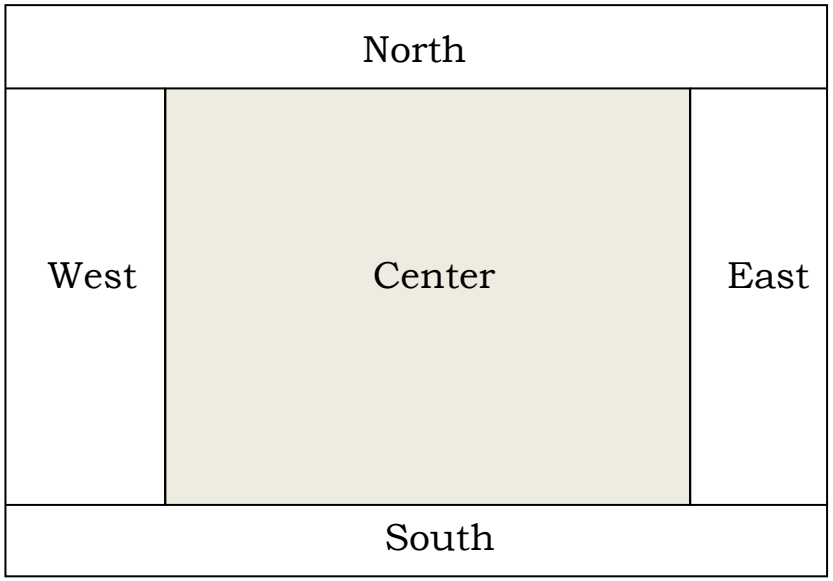


(iii) BorderLayout

Divide the container into five areas: East, South, West, North, and Center. Components are added to a BorderLayout by using the add method.

add(Component, constraint), where constraint is

BorderLayout.EAST,
BorderLayout.SOUTH,
BorderLayout.WEST,
BorderLayout.NORTH, or
BorderLayout.CENTER.

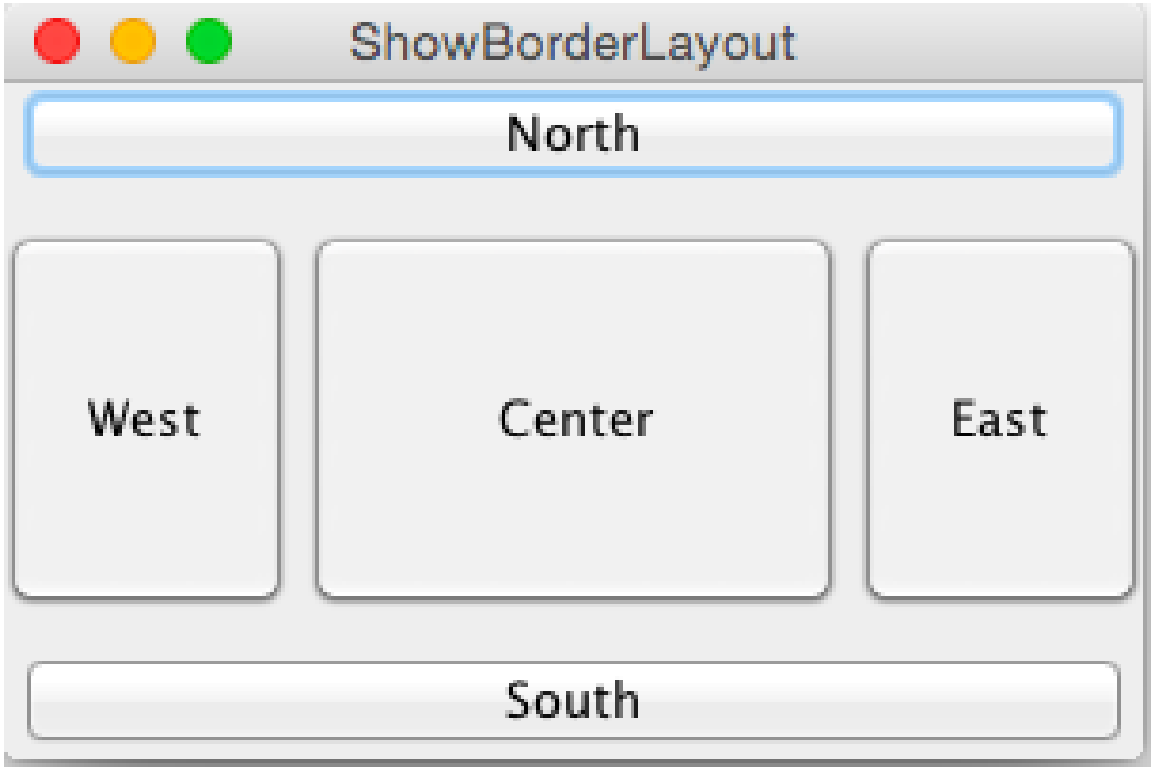


java.awt.BorderLayout	
-hgap: int	The horizontal gap of this layout manager (default: 0).
-vgap: int	The vertical gap of this layout manager (default: 0).
+BorderLayout()	Creates a default BorderLayout manager.
+BorderLayout(hgap: int, vgap: int)	Creates a BorderLayout manager with a specified number of horizontal gap, and vertical gap.

- **Example 3:** Write a program that adds five buttons labeled East , South , West , North , and Center to the frame with a BorderLayout manager.

```
import javax.swing.*;
import java.awt.BorderLayout;
public class ShowBorderLayout extends JFrame {
  public ShowBorderLayout() {
    // Set BorderLayout with horizontal gap 5 and vertical gap 10
    setLayout(new BorderLayout(5, 10));
    // Add buttons to the frame
    add(new JButton("East"), BorderLayout.EAST);
    add(new JButton("South"), BorderLayout.SOUTH);
    add(new JButton("West"), BorderLayout.WEST);
    add(new JButton("North"), BorderLayout.NORTH);
    add(new JButton("Center"), BorderLayout.CENTER);
  }
  /** Main method */
  public static void main(String[] args) {
    ShowBorderLayout frame = new ShowBorderLayout();
    frame.setTitle("ShowBorderLayout");
    frame.setSize(300, 200);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

- Output



Using Panels as Sub-Containers

- Panels act as sub-containers for grouping user interface components.
- It is recommended that you place the user interface components in panels and place the panels in a frame. You can also place panels in a panel.

- Example: Panel Demo

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

public class TestPanels extends JFrame {
  public TestPanels() {
    // Create panel p1 for the buttons and set GridLayout
    JPanel p1 = new JPanel();
    p1.setLayout(new GridLayout(4, 3));

    // Add buttons to the panel
    for (int i = 1; i <= 9; i++) {
      p1.add(new JButton("" + i));
    }

    p1.add(new JButton("" + 0));
    p1.add(new JButton("Start"));
    p1.add(new JButton("Stop"));

    // Create panel p2 to hold a text field and p1
    JPanel p2 = new JPanel(new BorderLayout());
    p2.add(new JTextField("Time to be displayed here"),
      BorderLayout.NORTH);
    p2.add(p1, BorderLayout.CENTER);

    // add contents into the frame
    add(p2, BorderLayout.EAST);
    add(new JButton("Food to be placed here"),
      BorderLayout.CENTER);
  }
```

```
  /** Main method */
  public static void main(String[] args) {
    TestPanels frame = new TestPanels();
    frame.setTitle("The Front View of a Microwave Oven");
    frame.setSize(400, 250);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

Components

i. Button
ii. Check Box
iii. Radio Button
iv. Labels
v. Text Fields
vi. Text Areas
vii. Combo Boxes
viii. Lists
ix. Scroll Bars
x. Sliders
xi. Tables

i. Button

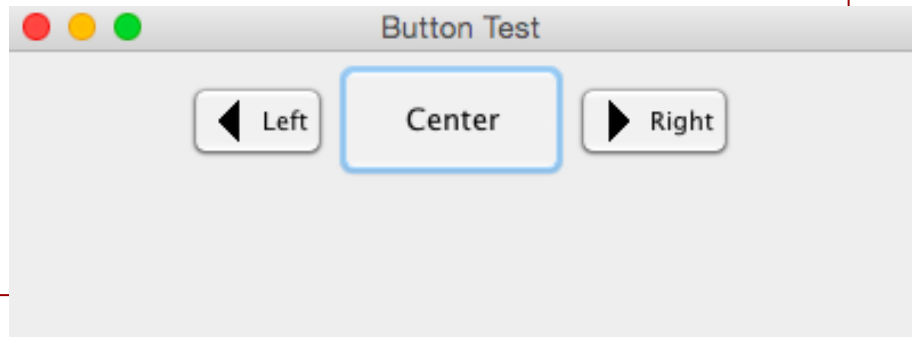
- To create a push button, use the **JButton** class
- Constructors

+JButton()	Creates a default button without a ny text or icons.
+JButton(icon: javax.swing.Icon)	Creates a button with an icon.
+JButton(text: String)	Creates a button with text.
+JButton(text: String, icon: Icon)	Creates a button with text and an i con.

- Icons, Pressed Icons, and Rollover Icons
  - Default icon
  - Pressed icon: display when a button is pressed
  - Rollover icon: displayed when the mouse is over the button but not pressed

- Example: JButton

```
import javax.swing.*;
import java.awt.*;
public class TestSwing extends JFrame
{
 TestSwing()
 {
  ImageIcon left=new ImageIcon("images/left.gif");
  ImageIcon right=new ImageIcon("images/right.gif");
  JButton bt1 = new JButton("Left",left);     //Button with Icon
  JButton bt2 = new JButton("Right",right); //Button with Icon
  JButton bt3=new JButton("Center"); //Default Button
  bt3.setPreferredSize(new Dimension(100,50)); //Define prefer size
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);     //setting close operation.
  setLayout(new FlowLayout());         //setting layout using FlowLayout object
  setTitle("Button Test");
  setSize(400, 400);              //setting size of Jframe
  add(bt1);
  add(bt3); //adding Yes button to frame.
  add(bt2);        //adding No button to frame.
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new TestSwing();
 }
}
```



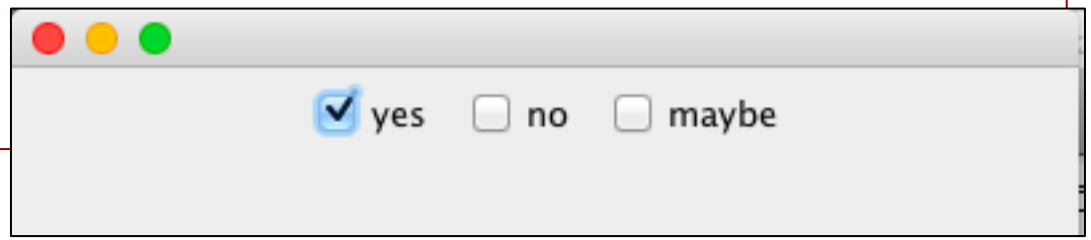
ii. Check Box

- To create a check box button, use the **JCheckBox** class
- Constructors

+JCheckBox()	**Creates a default check box without any text or icon.**
+JCheckBox(text: String)	**Creates a check box with text.**
+JCheckBox(text: String, selected: boolean)	**Creates a check box with text and specifies whether the check box is initially selected.**
+JCheckBox(icon: Icon)	**Creates a check box with an icon.**
+JCheckBox(text: String, icon: Icon)	**Creates a check box with text and an icon.**
+JCheckBox(text: String, icon: Icon, selected: boolean)	**Creates a check box with text and an icon, and specifies whether the check box is initially selected.**

- Example: JCheckBox

```
import javax.swing.*;
import java.awt.*;
public class CheckBox extends JFrame
{
 public CheckBox()
 {
  JCheckBox jcb = new JCheckBox("yes",true);    //creating
                                                JCheckBox with default selected
  add(jcb);                                              //adding JCheckBox to frame.
  jcb = new JCheckBox("no");
  add(jcb);
  jcb = new JCheckBox("maybe");
  add(jcb);
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new CheckBox();
 }
}
```



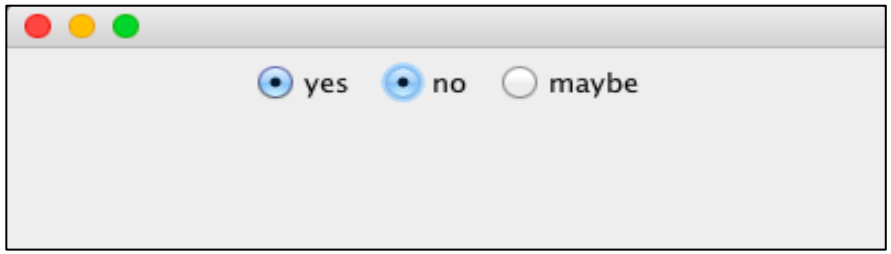
iii. Radio Button

- To create a radio button, use the JRadioButton class
- Constructors

+JRadioButton()	Creates a default radio button without any text or icon.
+JRadioButton(text: String)	Creates a radio button with text.
+JRadioButton(text: String, selected:boolean)	Creates a radio button with text and specifies whether the radio button is initially selected.
+JRadioButton(icon: Icon)	Creates a radio button with an icon.
+JRadioButton(text: String, icon: Icon)	Creates a radio button with text and an icon.
+JRadioButton(text: String, icon: Icon, selected: boolean)	Creates a radio button with text and an icon, and specifies whether the radio button is initially selected.

- To group radio buttons, you need to create an instance of java.swing.ButtonGroup

- Example: JRadioButton




```
import javax.swing.*;
import java.awt.*;
public class RadioButton extends JFrame
{
 public RadioButton()
 {
  JRadioButton jcb = new JRadioButton("yes",true);
  JRadioButton jcb2 = new JRadioButton("no");
  JRadioButton jcb3 = new JRadioButton("maybe");

  add(jcb);
  add(jcb2);
  add(jcb3);
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new RadioButton();
 }
}
```

Add following codes here to avoid allowing multiple selection
\\add all radio button in ButtonGroup.

```
ButtonGroup g=new ButtonGroup();
g.add(jcb);
 g.add(jcb2);
g.add(jcb3);
```

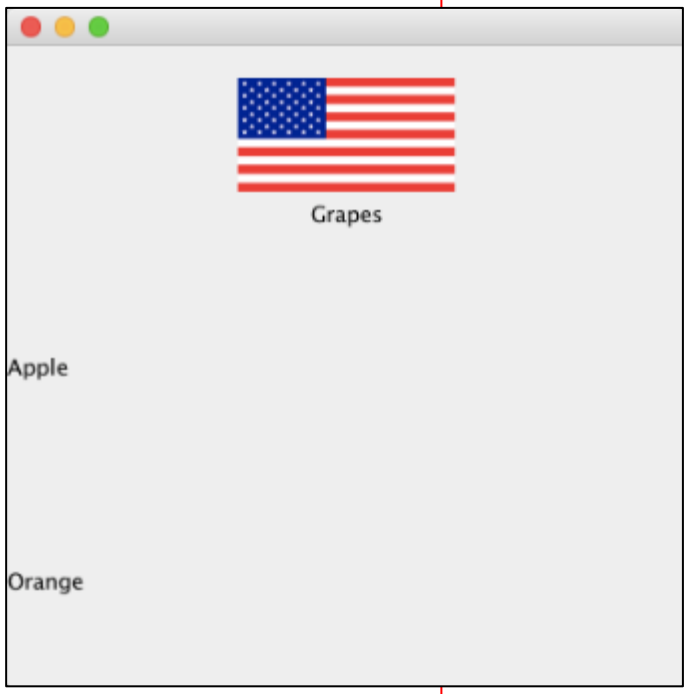
iv. Label

- To create a label, use the JLabel class.
- Constructors

+JLabel()	Creates a default label without any text or icons.
+JLabel(icon: javax.swing.Icon)	Creates a label with an icon.
+JLabel(icon: Icon, hAlignment: int)	Creates a label with an icon and the specified horizontal alignment.
+JLabel(text: String)	Creates a label with text.
+JLabel(text: String, icon: Icon, hAlignment: int)	Creates a label with text, an icon, and the specified horizontal alignment.
+JLabel(text: String, hAlignment: int)	Creates a label with text and the specified horizontal alignment.

- Example: JLabel

```
import javax.swing.*;
import java.awt.*;
public class Label extends JFrame
{
 public Label()
 {
  ImageIcon icon = new ImageIcon("images/usIcon.gif");
  JLabel jlbl = new JLabel("Grapes",icon,JLabel.CENTER);//Label with icon
    //Set label's text alignment and gap between text and icon
  jlbl.setHorizontalTextPosition(JLabel.CENTER);
  jlbl.setVerticalTextPosition(JLabel.BOTTOM);
  jlbl.setIconTextGap(5);
  JLabel jlbl2=new JLabel("Apple",JLabel.LEFT);
  JLabel jlbl3=new JLabel("Orange",JLabel.LEFT);
  setLayout(new GridLayout(3,1));
  add(jlbl);
  add(jlbl2);
  add(jlbl3);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new Label();
 }
}
```



v. Text Field

- To create a text field, use the JTextField class.
- Constructors

+JTextField()	Creates a default empty text field with number of columns set to 0.
+JTextField(column: int)	Creates an empty text field with a specified number of columns.
+JTextField(text: String)	Creates a text field initialized with the specified text.
+JTextField(text: String, columns: int)	Creates a text field initialized with the specified text and columns.

- Password Field
  - Use for password inputs
  - JPasswordField extends JTextField

```
JPasswordField value = new JPasswordField();
```

- Example: Text Field

```
import javax.swing.*;
import java.awt.*;
public class TxtTield extends JFrame
{
 public TxtTield()
 {
  setLayout(new FlowLayout());
  JTextField jtf = new JTextField(20);     //creating JTextField.
  JTextField jtf1 = new JTextField("Input Here");
  jtf1.setPreferredSize(new Dimension(200,50));
  add(jtf);
  add(jtf1);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new TxtTield();
 }
}
```

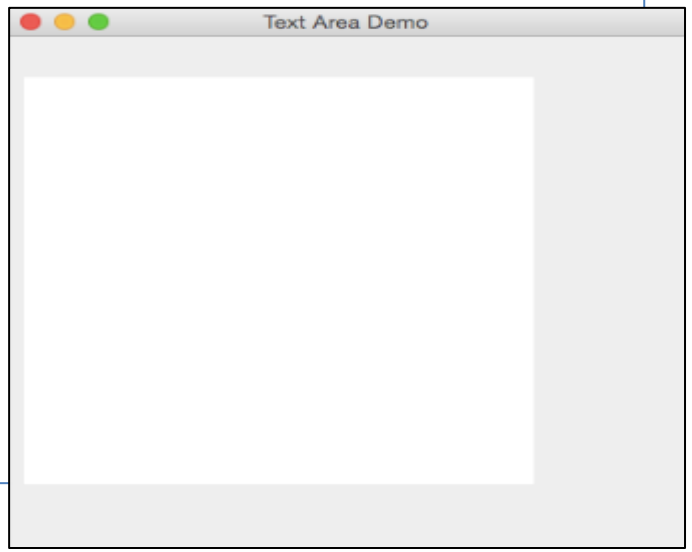
vi. Text Area

- A **JTextArea** enables the user to enter multiple lines of text.
- Constructors

+JTextArea()	Creates a default empty text area.
+JTextArea(rows: int, columns: int)	Creates an empty text area with the specified number of rows and columns.
+JTextArea(text: String)	Creates a new text area with the specified text
+JTextArea(text: String, rows: int, columns: int)	Creates a new text area with the specified text and number of rows and columns.
+getLineCount(): int	Returns the actual number of lines contained in the text area.

- Example: Text Area

```
import javax.swing.*;
public class TextArea {
    JTextArea area;
    JFrame f;
    TextArea(){
                        f=new JFrame();
                        area=new JTextArea(300,300);
                        area.setBounds(10,30,300,300);
                        area.setLineWrap(true);
                        area.setWrapStyleWord(true);
                        f.add(area);
                        f.setSize(400,400);
                        f.setLayout(null);
                        f.setVisible(true);
            }
    public static void main(String[] args) {
        new TextArea();
    }
}
```



vii. Combo Box

- A combo box , also known as a choice list or drop-down list , contains a list of items from which the user can choose

+JComboBox()	Creates a default empty combo box.
+JComboBox(items: Object[])	Creates a combo box that contains the elements in the specified array.
+addItem(item: Object): void	Adds an item to the combo box.
+getItemAt(index: int): Object	Returns the item at the specified index.
+getItemCount(): int	Returns the number of items in the combo box.
+getSelectedIndex(): int	Returns the index of the selected item.
+setSelectedIndex(index: int): void	Sets the selected index in the combo box.
+getSelectedItem(): Object	Returns the selected item.
+setSelectedItem(item: Object): void	Sets the selected item in the combo box.
+removeItem(anObject: Object): void	Removes an item from the item list.
+removeItemAt(anIndex: int): void	Removes the item at the specified index in the combo box.
+removeAllItems(): void	Removes all the items in the combo box.

Example: Combo Box

```
import javax.swing.*;
import java.awt.*;
public class ComboBox extends JFrame
{
 String name[] = {"India","Aus","U.S.A","England","Newzeland"};  //list of items to be
                                                                  appeared in combo box
 public ComboBox()
 {
  JComboBox jc = new JComboBox(name);      //initialzing combo box with list of name
. add(jc);                                  //adding JComboBox to frame.
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new ComboBox();
 }
}
```

viii. List

- A list is a component that basically performs the same function as a combo box, but it enables the user to choose a single value or multiple values

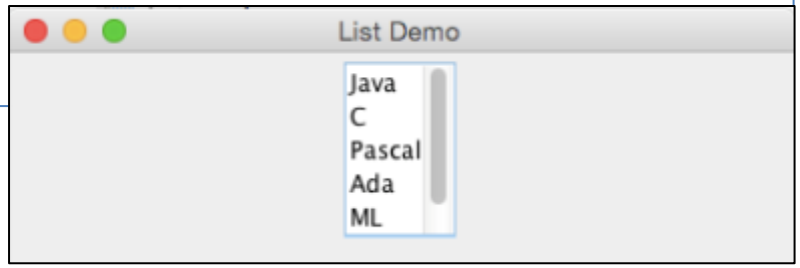
+JList()	Creates a default empty list.
+JList(items: Object[])	Creates a list that contains the elements in the specified array.

- This component contains list of items from which a user can select one or several items at a time.
- JList methods
  - getSelectedIndex() or getSelectedValues(); Returns index or values of selected item
  - setSelectionMode(**SelectionChoices**)

Selection Choices
ListSelectionModel.SINGLE_SELECTION
ListSelectionModel.SINGLE_INTERVAL_SELECTION
ListSelectionModel.MULTIPLE_INTERVAL_SELECTION (default)

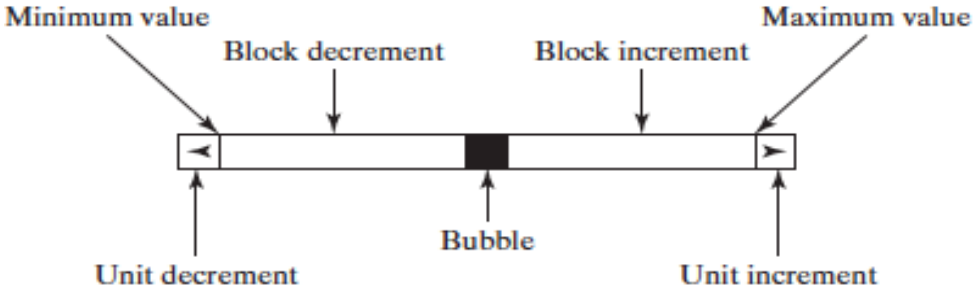
- Example: List

```
import java.awt.FlowLayout;

import javax.swing.*;
public class List{
        public static void main(String[] args)
        {
                JFrame f=new JFrame("List Demo");
                f.setLayout(new FlowLayout());
                String [] list = { "Java", "C", "Pascal", "Ada", "ML", "Prolog" };
                JList lang = new JList(list);
                JScrollPane jsp = new JScrollPane(lang);
                lang.setVisibleRowCount(5);
            lang.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
            f.add(jsp);
            f.setSize(400, 300); // Set the frame size
            f.setLocationRelativeTo(null); // New since JDK 1.4
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setVisible(true); // Display the frame
        }
}
```



IX. Scroll Bar

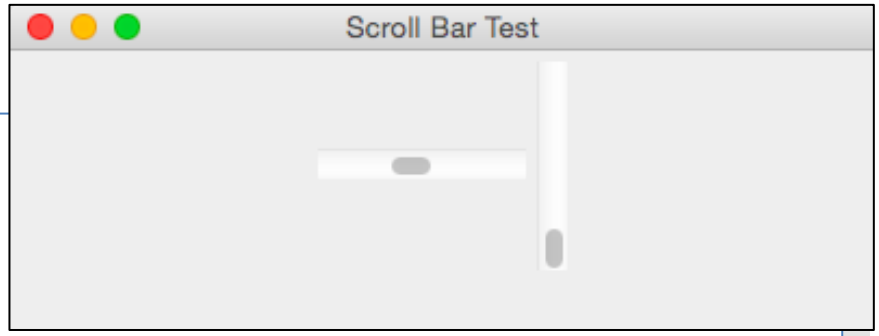
- JScrollBar is a component that enables the user to select from a range of values



A scroll bar represents a range of values graphically.

+JScrollBar()	Creates a default vertical scroll bar.
+JScrollBar(orientation: int)	Creates a scroll bar with the specified orientation.
+JScrollBar(orientation: int, value: int, extent: int, min: int, max: int)	Creates a scroll bar with the specified orientation, value, extent, minimum, and maximum *value* - the starting position of the knob of Scrollbar on the track of a Scrollbar. *extent* - the width of the knob of Scrollbar. *min* - the minimum width of the track on which Scrollbar moves. *max* - the maximum width of the track.

Example: Scroll Bar




```
import java.awt.FlowLayout;
import javax.swing.*;
public class ScrollBar {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                JFrame f=new JFrame("Scroll Bar Test");
                f.setLayout(new FlowLayout());
                JScrollBar scb=new JScrollBar(JScrollBar.HORIZONTAL);
                JScrollBar scb2=new JScrollBar(JScrollBar.VERTICAL);
                scb.setMaximum(200);
                scb.setMinimum(1);
                scb2.setMaximum(100);
                scb2.setMinimum(1);
                f.add(scb);
                f.add(scb2);
                f.setSize(400, 300); // Set the frame size
            f.setLocationRelativeTo(null); // New since JDK 1.4
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setVisible(true); // Display the frame
        }

}
```

X. Table

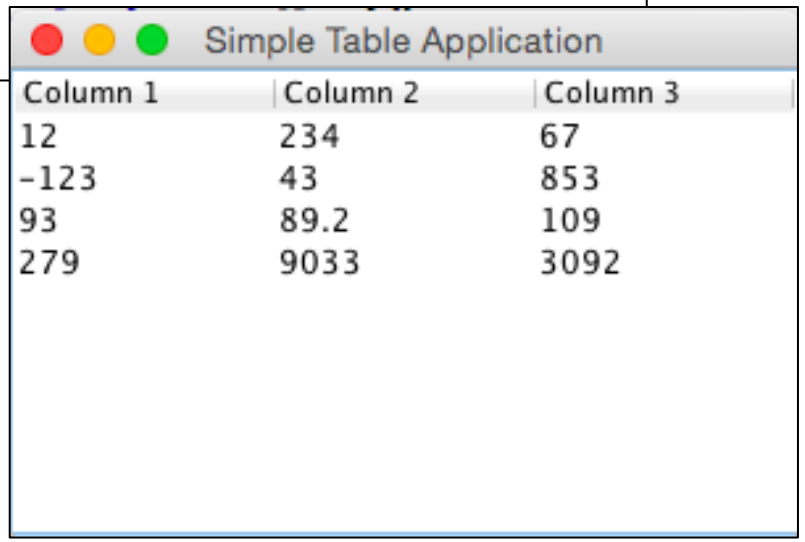
- Jtable class is used to display the data on two dimensional tables of cells.

JTable()	: creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	: creates a table with the specified data.

- Example: Table

```
import java.awt.*;
import javax.swing.*;
class SimpleTableExample extends JFrame
 {
    JPanel      topPanel;
    JTable      table;
    JScrollPane scrollPane;

public SimpleTableExample()
    {
        // Set the frame characteristics
        setTitle( "Simple Table Application" );
        setSize( 300, 200 );
        // Create a panel to hold all other components
        topPanel = new JPanel();
        topPanel.setLayout( new BorderLayout() );
        add( topPanel );
        // Create columns names
        String columnNames[] = { "Column 1", "Column 2", "Column 3" };
        // Create some data
        String dataValues[][] =
        {
            { "12", "234", "67" },
            { "-123", "43", "853" },
            { "93", "89.2", "109" },
            { "279", "9033", "3092" }
        };
```

```
        // Create a new table instance
        table = new JTable( dataValues, columnNames );
        // Add the table to a scrolling pane
        scrollPane = new JScrollPane( table );
        topPanel.add( scrollPane, BorderLayout.CENTER );
    }
//Main entry point for this example
    public static void main( String args[] )
    {
        // Create an instance of the test application
        SimpleTableExample mainFrame     = new SimpleTableExample();
        mainFrame.setVisible( true );
    }
}
```



xi. Menu Bar

- Java provides several classes—JMenuBar, JMenu, JMenuItem, which will implement menus in a frame.
- A menu bar holds menus; the menu bar can only be added to a frame.

Building Menu

1. Create a menu bar as follows:

```
JMenuBar mb=new JMenuBar();
```

2. Creat the menu objects as follows:

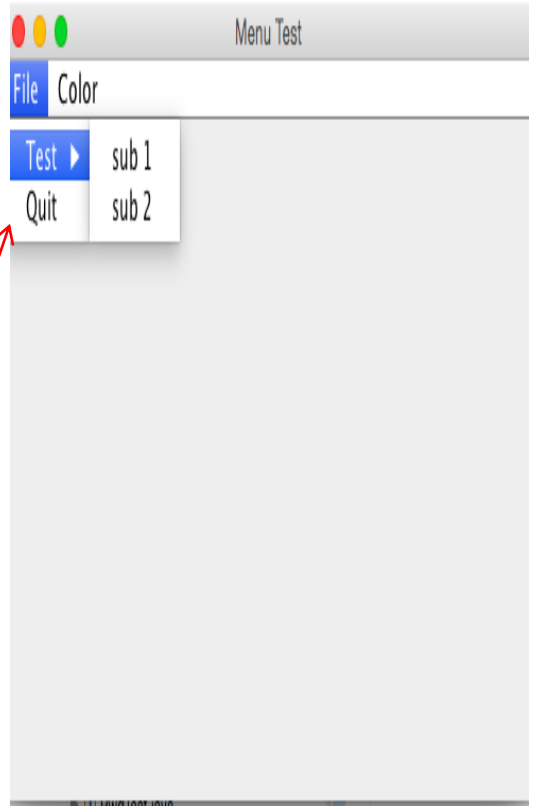
```
JMenu file=new JMenu(“File”);
```

3. Creating the menuItem object as follows:

```
JMenuItem newItem=new JMenuItem(”Quit");
```

4. Creating sub menu items

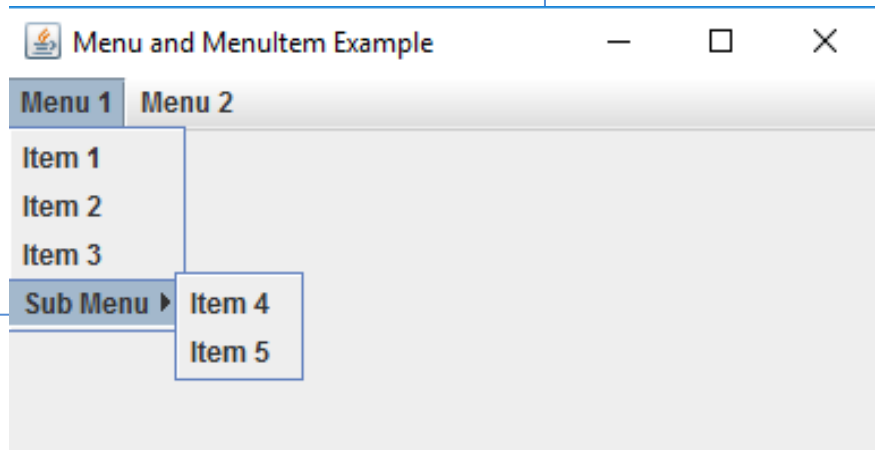
```
JMenu Test=new JMenu("Test");
JMenuItem one=new JMenuItem("sub 1");
JMenuItem two=new JMenuItem("sub 2");
Test.add(one);
Test.add(two);
file.add(Test);
```



Example: Menu Bar

```
import javax.swing.*;
class Menu
{
        JMenu menu1,menu2, submenu;
        JMenuItem i1, i2, i3, i4, i5;
        Menu(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();  //Menubar
        menu1=new JMenu("Menu 1");
        menu2=new JMenu("Menu 2");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
```

```
        menu1.add(i1); //Add Item 1 in Menu 1
        menu1.add(i2); //Add Item 2 in Menu 1
        menu1.add(i3); //Add Item 3 in Menu 1
       //Add Item 4 and 5 in Sub Menu
        submenu.add(i4);
        submenu.add(i5);
          menu1.add(submenu);  //Add sub menu in menu 1
       //Add Menu 1 and Menu 2 to Menu BAr
          mb.add(menu1);
          mb.add(menu2);
           f.setJMenuBar(mb);   //Set Menu bar on Frame
          f.setSize(400,400);
          f.setLayout(null);
          f.setVisible(true);
}
public static void main(String args[])
{
new Menu();
}}
```



Next Week Lecture

- Event-Driven Programming