UNIT 2 & UNIT 3:

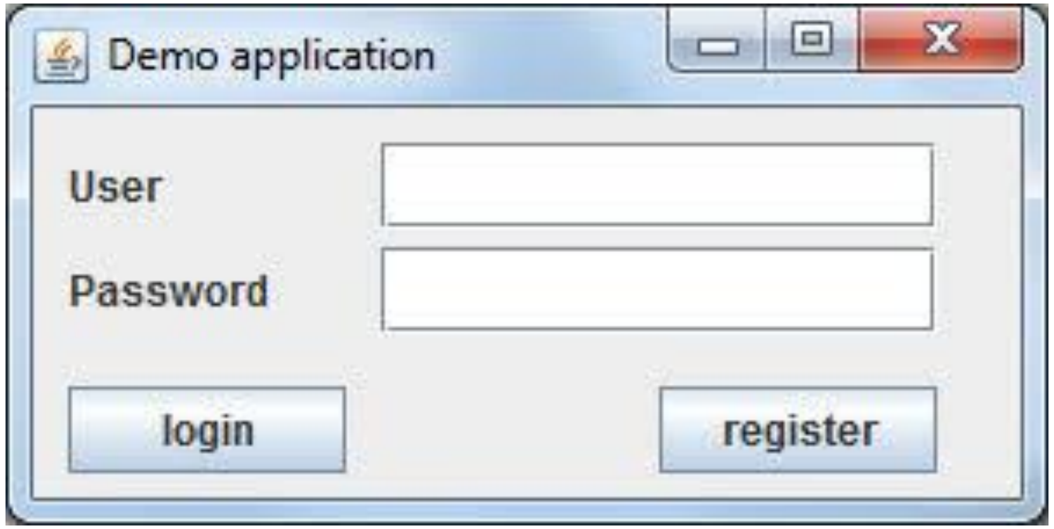
User interface component with Swing
Event Handling

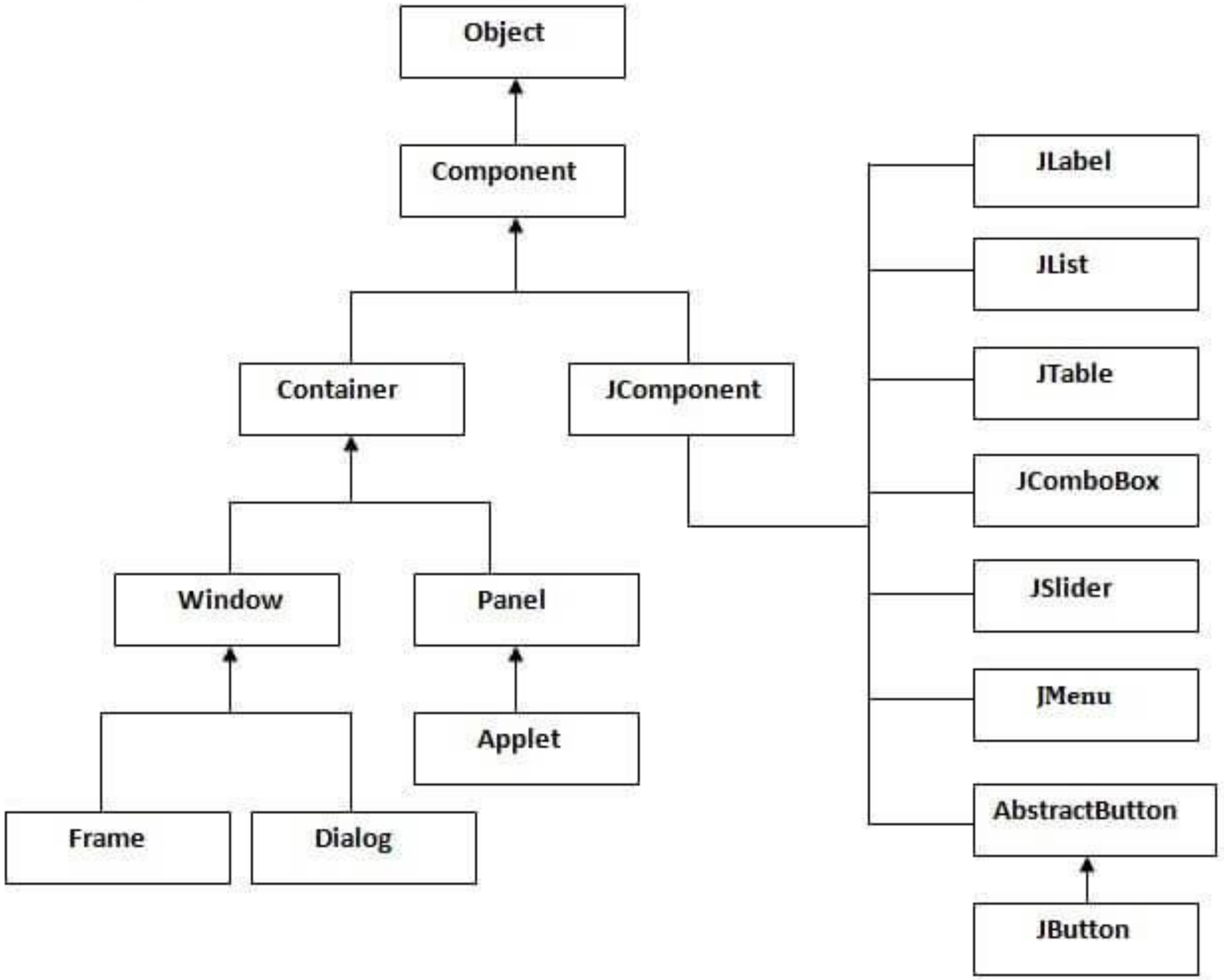
Presented To: Ascol 7th Sem

Presented By: Yuba Raj Devkota

2.1 Java Swing

- **Java Swing** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.





Hierarchy of Java Swing classes

Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

Java AWT	Java Swing
• AWT components are **platform-dependent**. • AWT components are **heavyweight**. • AWT **doesn't support pluggable look and feel**. • AWT provides **less components** than Swing. • AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	• Java swing components are **platform-independent**. • Swing components are **lightweight**. • Swing **supports pluggable look and feel**. • Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. • Swing **follows MVC**.

2.3 Text Input

Text Fields, Password Fields, Text Areas, Scroll Pane, Label and Labeling Components

- A text field is a basic text control that enables the user to type a small amount of text.
- When the user indicates that text entry is complete (usually by pressing Enter), the text field fires an action event.
- If you need to obtain more than one line of input from the user, use a text area.

JTextField : basic text fields.
JPasswordField: A JTextField subclass that does not show the characters that the user types.

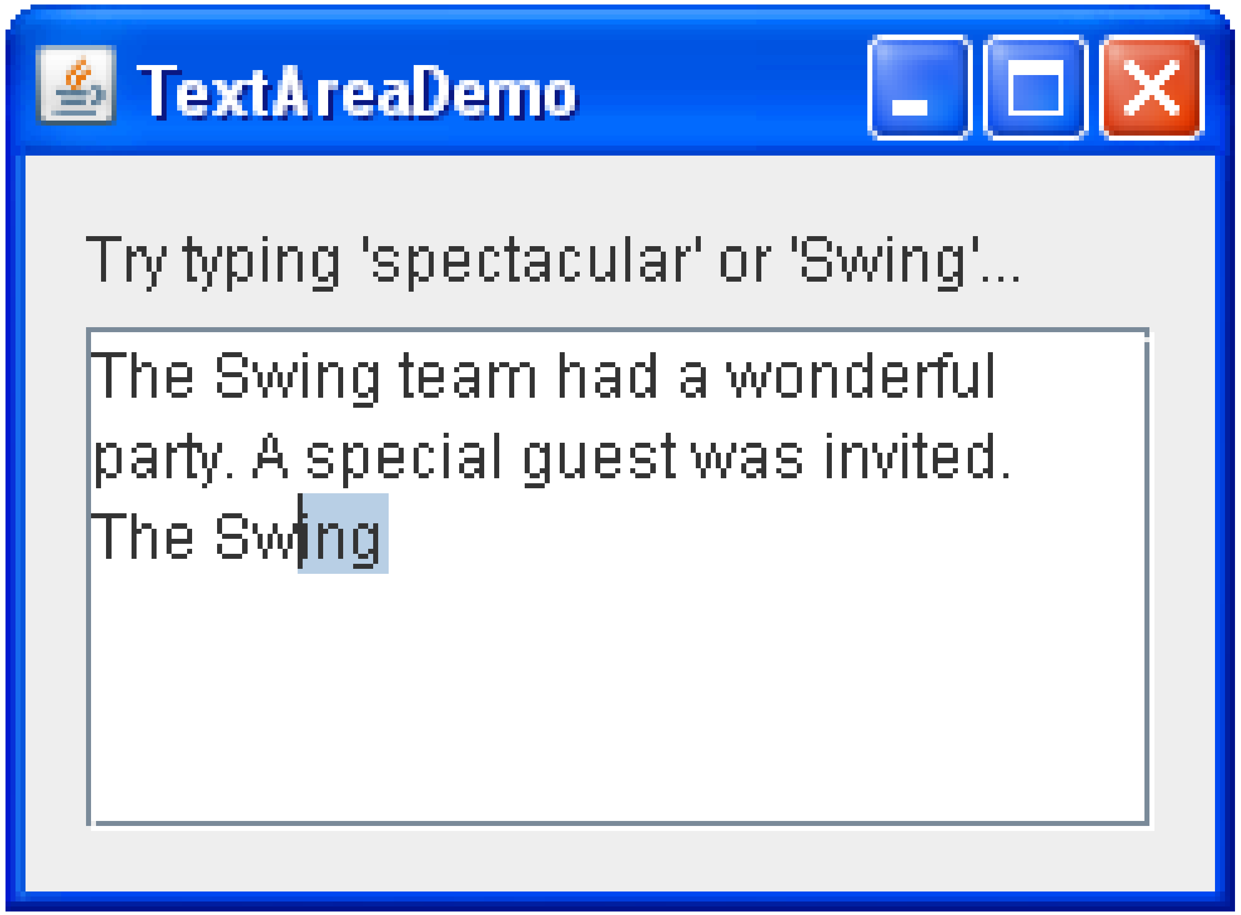
```
textField = new JTextField(20);
```

The actionPerformed method handles action events
from the text field:

```
private final static String newline = "\n";
...
public void actionPerformed(ActionEvent evt) {
    String text = textField.getText();
    textArea.append(text + newline);
    textField.selectAll();
}
```

- The JTextArea class provides a component that displays multiple lines of text and optionally allows the user to edit the text.
- If you need to obtain only one line of input from the user, you should use a text field.
- If you want the text area to display its text using multiple fonts or other styles, you should use an editor pane or text pane.
- If the displayed text has a limited length and is never edited by the user, use a label.

```
textArea = new JTextArea(5, 20);
JScrollPane scrollPane = new JScrollPane(textArea);
textArea.setEditable(false);
```

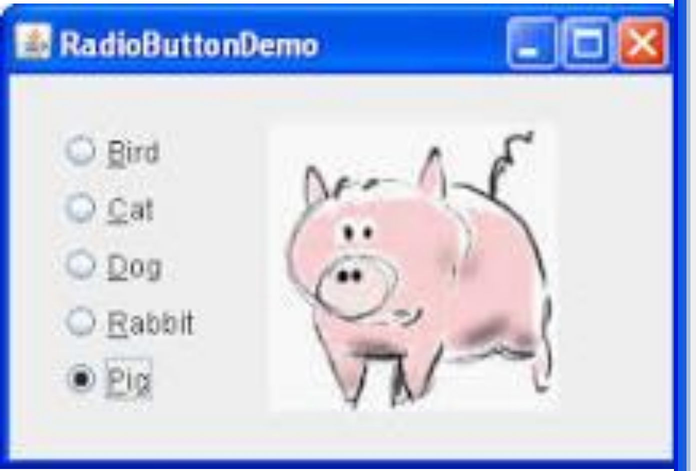


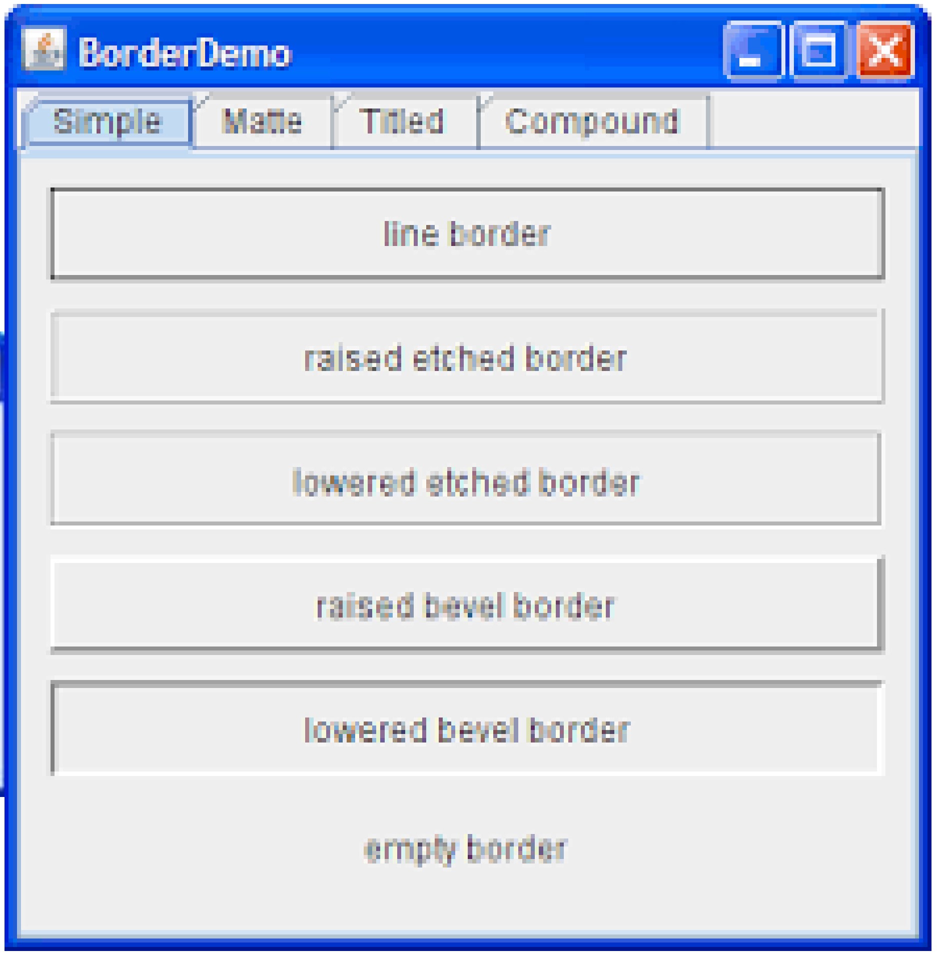
Try typing 'spectacular' or 'Swing'...

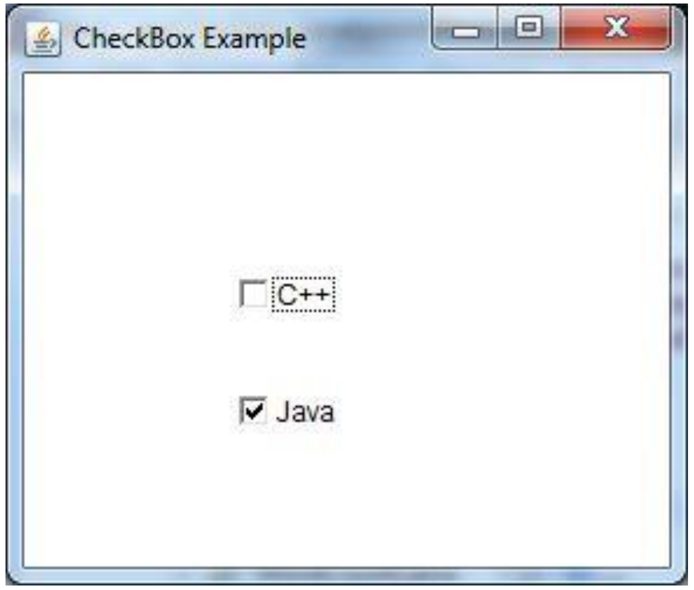
The Swing team had a wonderful
party. A special guest was invited.
The Swing

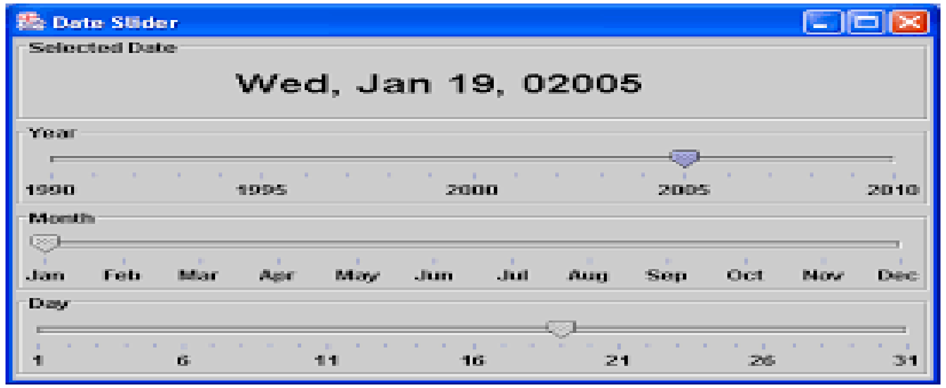
2.4 Choice Components

Check Boxes, Radio Buttons, Borders, Sliders







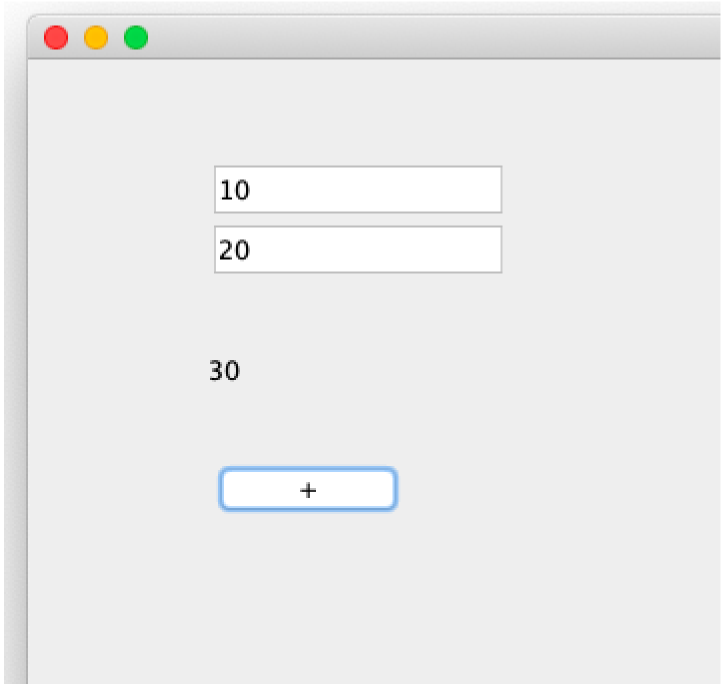


Java swing program to add two numbers

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

class Test extends JFrame implements ActionListener {

    JButton jb1;
    JTextField jt1, jt2;
    JLabel lbl;

    Test() {

        jt1 = new JTextField();
        jt1.setBounds(90, 50, 150, 30);
        add(jt1);

        jt2 = new JTextField();
        jt2.setBounds(90, 80, 150, 30);
        add(jt2);
```

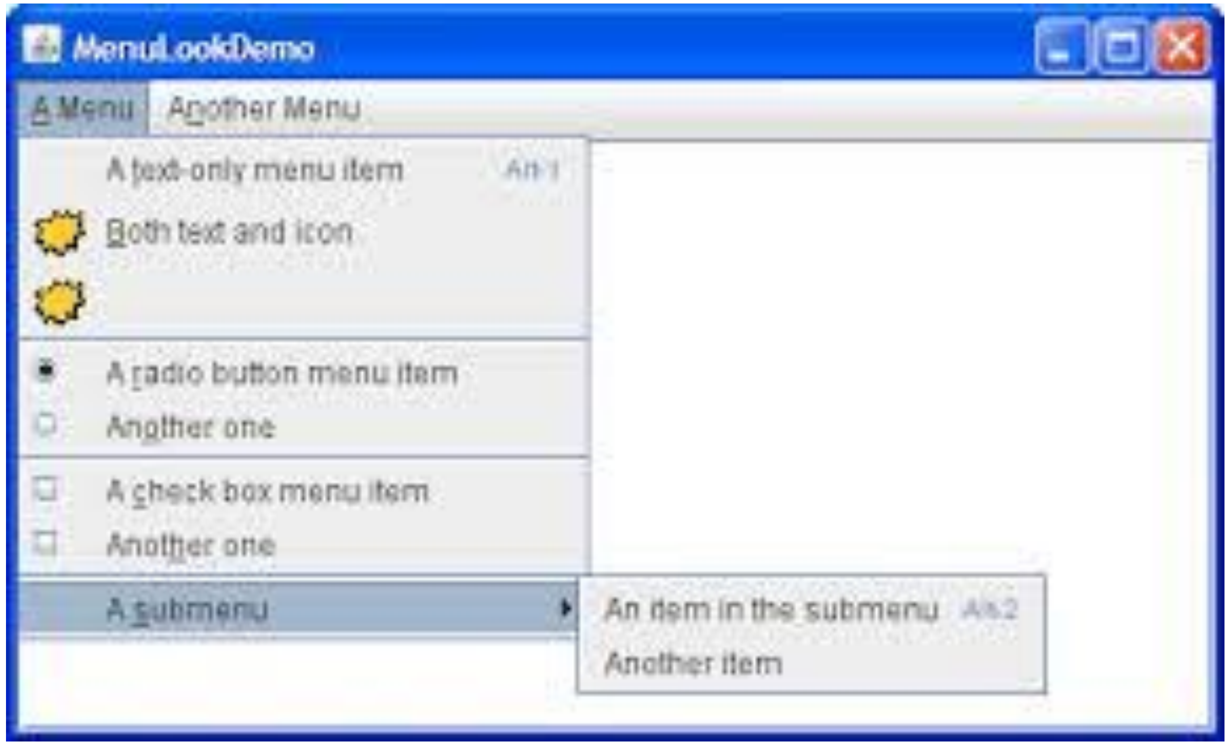
```
        lbl = new JLabel("Result :");
        lbl.setBounds(90, 140, 150, 30);
        add(lbl);

        jb1 = new JButton("+");
        jb1.setBounds(90, 200, 100, 30);
        add(jb1);

        jb1.addActionListener(this);

        setLayout(null);
        setSize(600, 400);
        setVisible(true);

    }
```

```
    public void actionPerformed(ActionEvent e) {

        int a = Integer.parseInt(jt1.getText());
        int b = Integer.parseInt(jt2.getText());
        int c = 0;

        if (e.getSource().equals(jb1)) {
            c = a + b;
            lbl.setText(String.valueOf(c));
        }
    }

    public static void main(String args[]) {
        Test t = new Test();
    }
}
```



2.5 Menus

Menu Building, Icons in Menu Items, Check box and Radio Buttons in Menu Items, op-up Menus, Keyboard Mnemonics and Accelerators, Enabling and Disabling menu Items, Toolbars, Tooltips

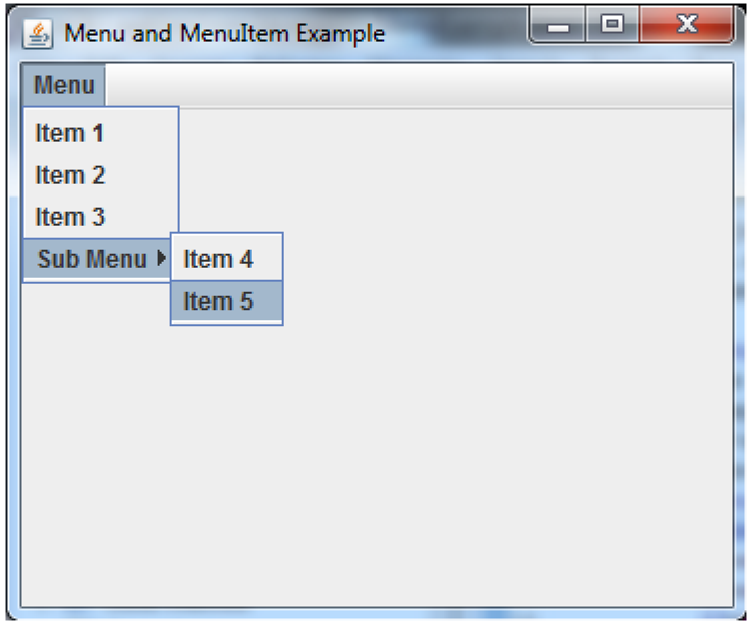


```
JMenuBar menuBar;
JMenu menu, submenu;
JMenuItem menuItem;
JRadioButtonMenuItem rbMenuItem;
JCheckBoxMenuItem cbMenuItem;
.........
//Create the menu bar.
menuBar = new JMenuBar();
...........
//Build the first menu.
menu = new JMenu("A Menu");
menuBar.add(menu);
..............
//a group of JMenuItems
menuItem = new JMenuItem("A text-only menu
item",KeyEvent.VK_T);
...............
menuItem = new JMenuItem("Both text and icon",
        new ImageIcon("images/middle.gif"));
...................
menu.addSeparator();
```

Java JMenuItem and JMenu Example

```
import javax.swing.*;
class MenuExample
{
        JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;
        MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        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");
```

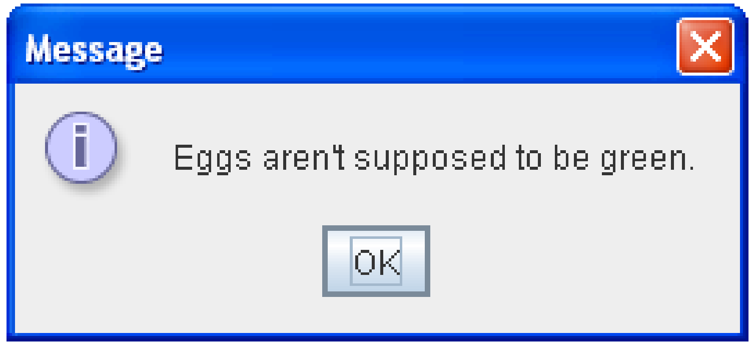
```
        submenu.add(i4); submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}}
```



2.6 Dialog Boxes

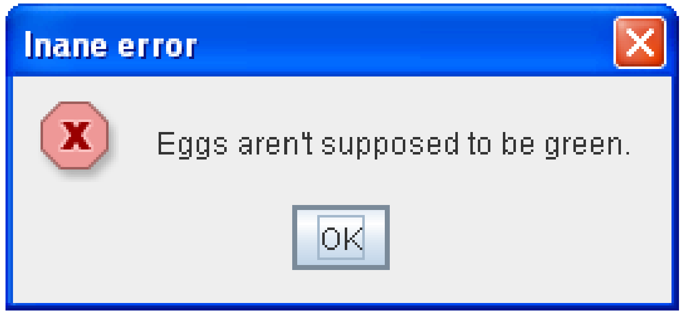
Option Dialogs, Creating Dialogs, Data Exchange, File Choosers, Color Choosers

```
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.");
```



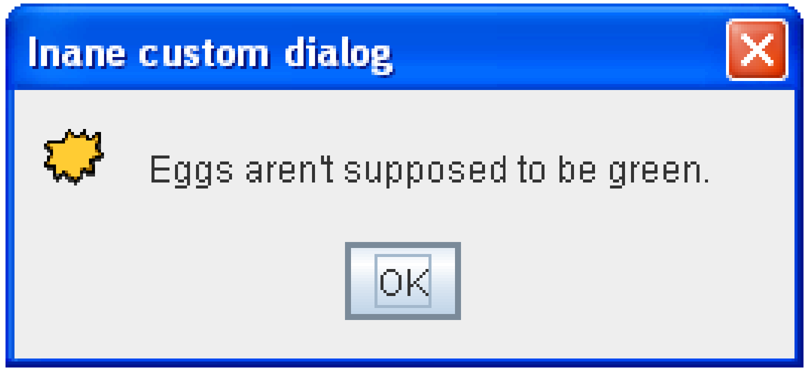
Eggs aren't supposed to be green.

```
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane error",
    JOptionPane.ERROR_MESSAGE);
```



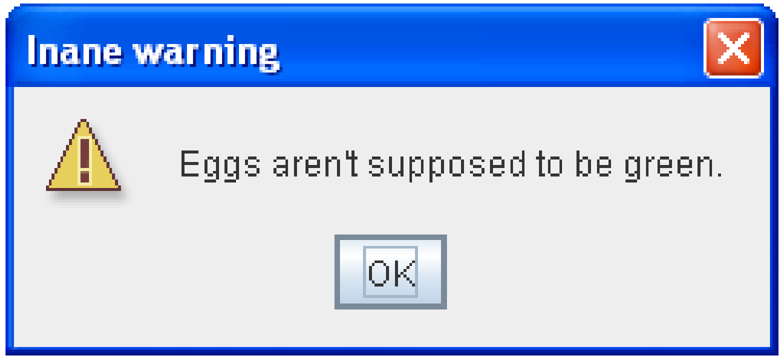
Eggs aren't supposed to be green.

```
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane custom dialog",
    JOptionPane.INFORMATION_MESSAGE,
    icon);
```



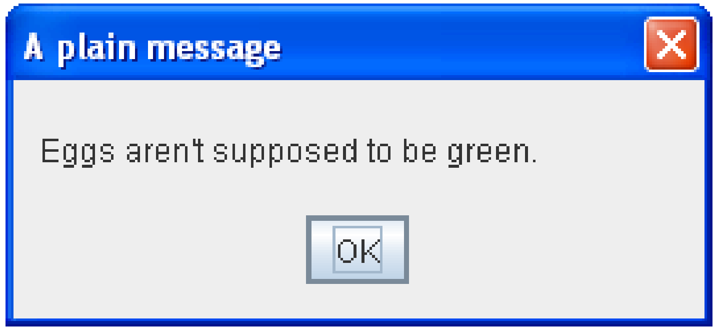
Eggs aren't supposed to be green.

```
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane warning",
    JOptionPane.WARNING_MESSAGE);
```



Eggs aren't supposed to be green.

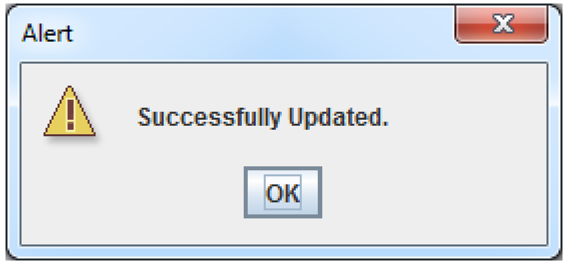
```
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "A plain message",
    JOptionPane.PLAIN_MESSAGE);
```



Eggs aren't supposed to be green.

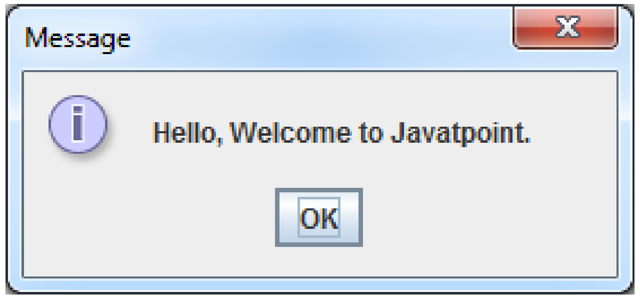
Java JOptionPane Example: showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```


Alert

Successfully Updated.

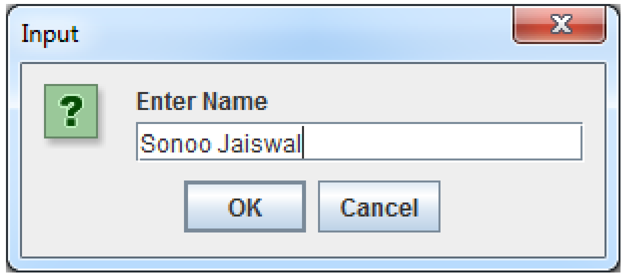
```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Successfully Updated.","Alert",JOptionPane.WARNING_MESSAG
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```


Message

Hello, Welcome to Javatpoint.

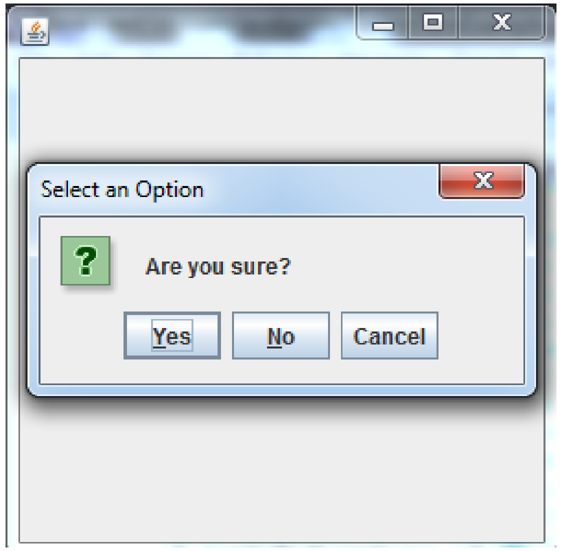
Java JOptionPane Example: showInputDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    String name=JOptionPane.showInputDialog(f,"Enter Name");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```



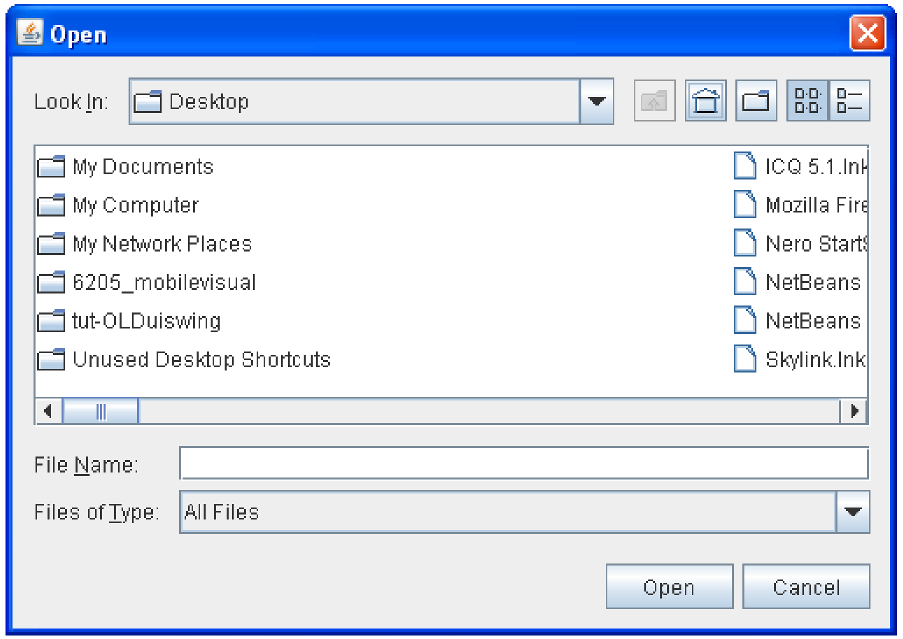
Java JOptionPane Example: showConfirmDialo

```
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter{
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    f.addWindowListener(this);
    f.setSize(300, 300);
    f.setLayout(null);
    f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    f.setVisible(true);
}
public void windowClosing(WindowEvent e) {
    int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
if(a==JOptionPane.YES_OPTION){
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
public static void main(String[] args) {
    new  OptionPaneExample();
}
}
```

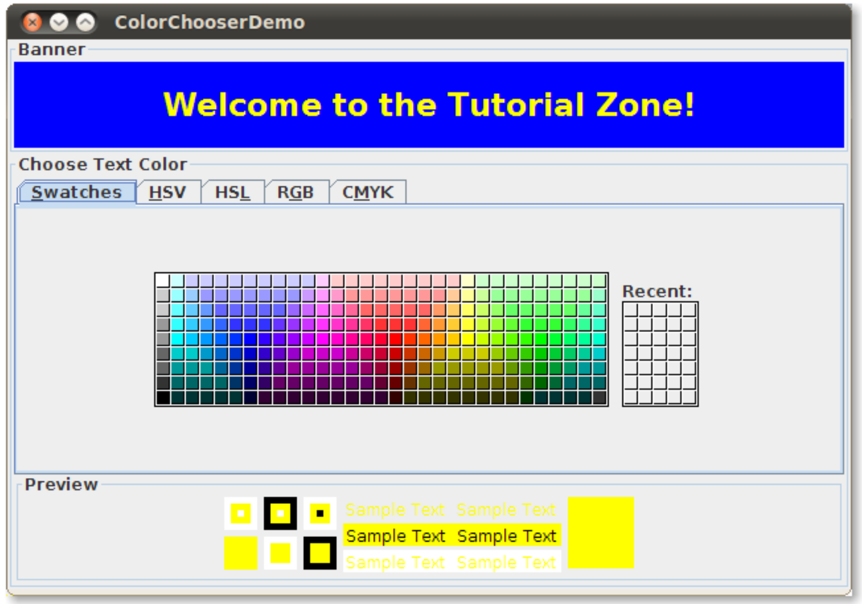


File Choosers/ Color Choosers

```
//Create a file chooser
final JFileChooser fc = new JFileChooser();
...
//In response to a button click:
int returnVal = fc.showOpenDialog(aComponent);
```



```
banner = new JLabel("Welcome to the Tutorial Zone!",
                    JLabel.CENTER);
    banner.setForeground(Color.yellow);
    . . .
    tcc = new JColorChooser(banner.getForeground());
    . . .
    add(tcc, BorderLayout.PAGE_END);
```

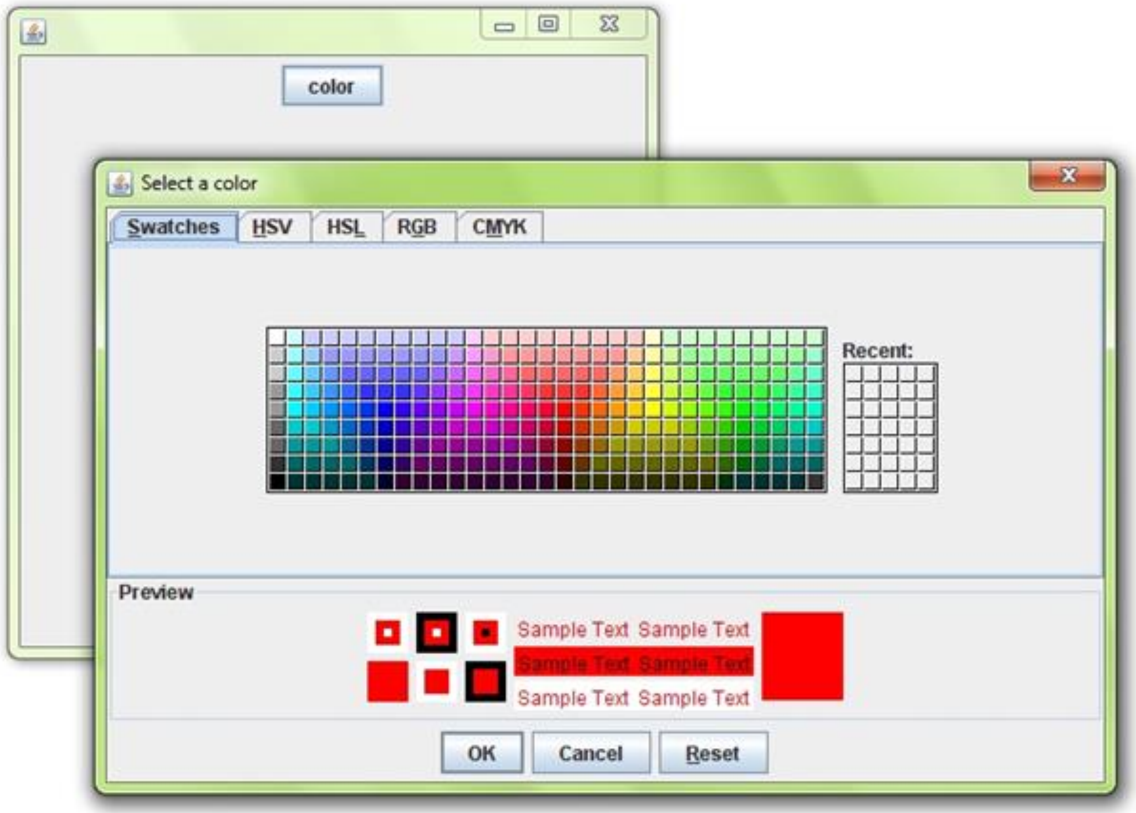


Java JColorChooser Example

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class ColorChooserExample extends JFrame implements ActionListener {
JButton b;
Container c;
ColorChooserExample(){
    c=getContentPane();
    c.setLayout(new FlowLayout());
    b=new JButton("color");
    b.addActionListener(this);
    c.add(b);
}
```

```
public void actionPerformed(ActionEvent e) {
Color initialcolor=Color.RED;
Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
c.setBackground(color);
}
```

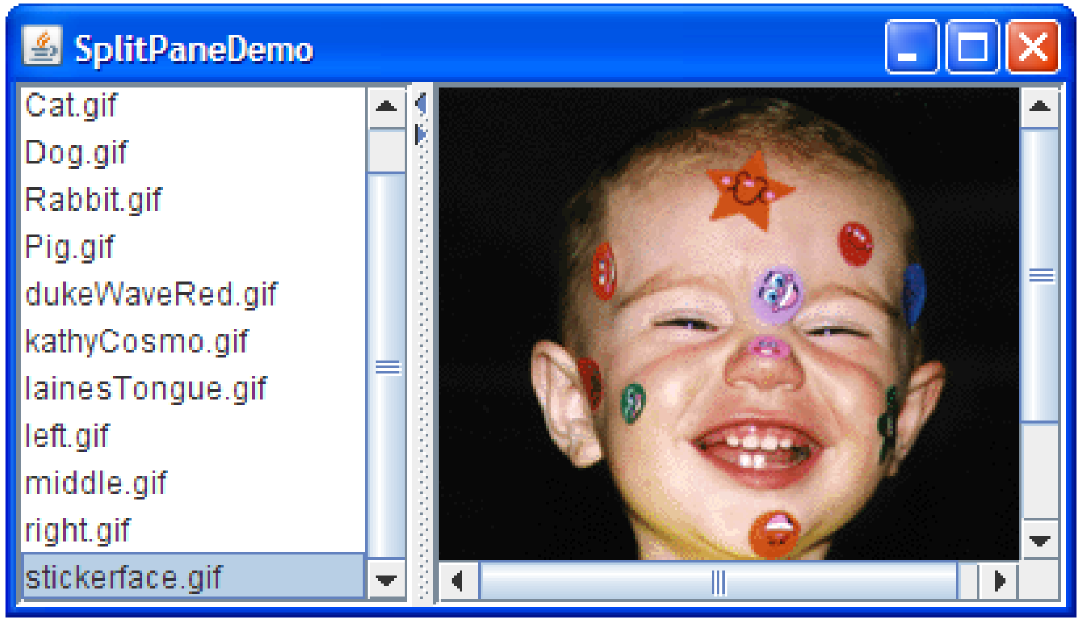
```
public static void main(String[] args) {
    ColorChooserExample ch=new ColorChooserExample();
    ch.setSize(400,400);
    ch.setVisible(true);
    ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```



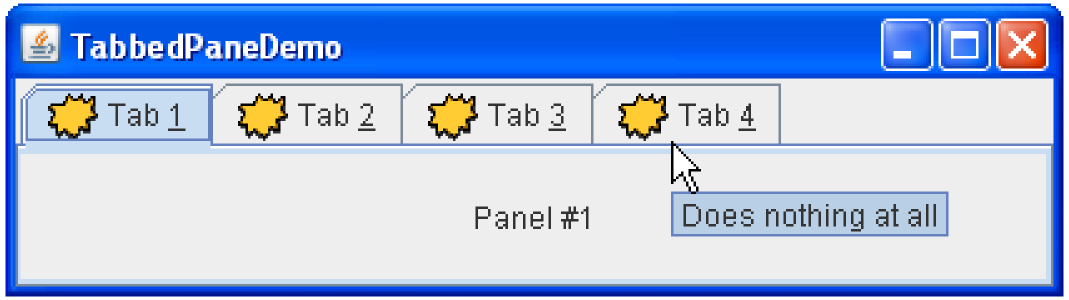
2.7 Components Organizers

split Panes, Tabbed Panes, Desktop Panes and Internal Frames, Cascading and Tiling

```
splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                    listScrollPane, pictureScrollPane);
splitPane.setOneTouchExpandable(true);
splitPane.setDividerLocation(150);

//Provide minimum sizes for the two components in the split
pane
Dimension minimumSize = new Dimension(100, 50);
listScrollPane.setMinimumSize(minimumSize);
pictureScrollPane.setMinimumSize(minimumSize);
```



```
    JTabbedPane tabbedPane = new JTabbedPane();
    ImageIcon icon = createImageIcon("images/middle.gif");

    JComponent panel1 = makeTextPanel("Panel #1");
    tabbedPane.addTab("Tab 1", icon, panel1,
              "Does nothing");
    tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);
```

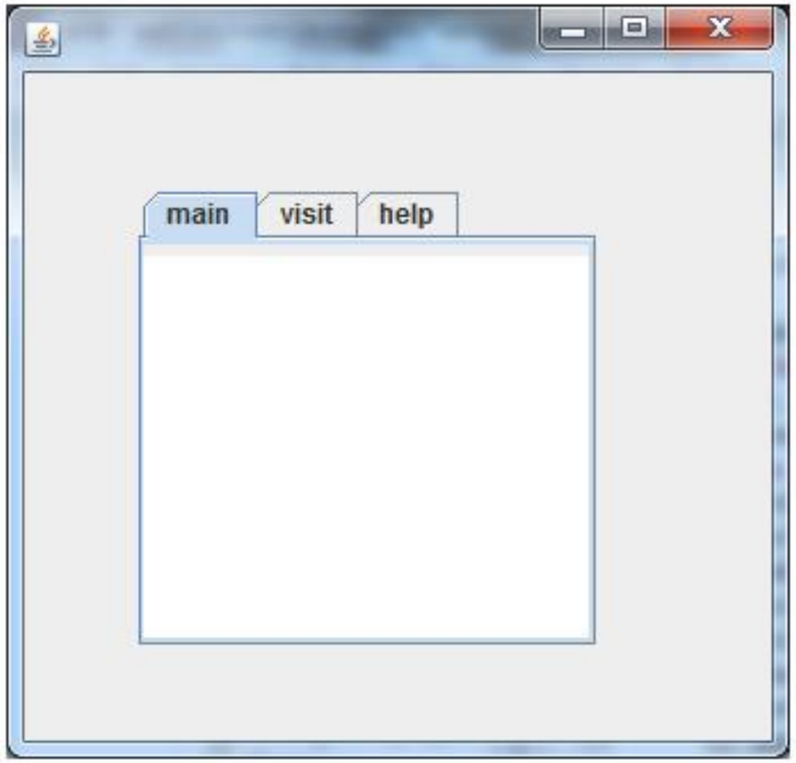


Java JTabbedPane Example

```
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
    f=new JFrame();
    JTextArea ta=new JTextArea(200,200);
    JPanel p1=new JPanel();
    p1.add(ta);
    JPanel p2=new JPanel();
    JPanel p3=new JPanel();
    JTabbedPane tp=new JTabbedPane();
    tp.setBounds(50,50,200,200);
    tp.add("main",p1);
    tp.add("visit",p2);
    tp.add("help",p3);
    f.add(tp);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
```

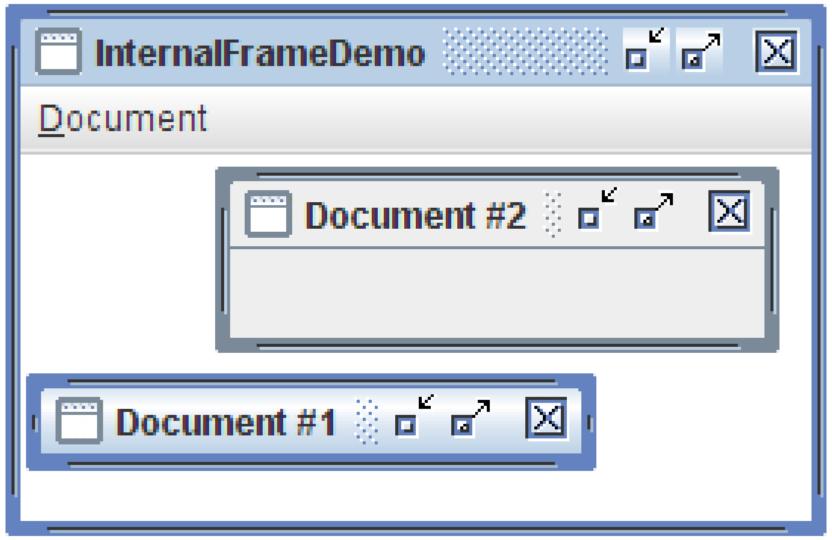
```
public static void main(String[] args) {
    new TabbedPaneExample();
}}
```

Output:



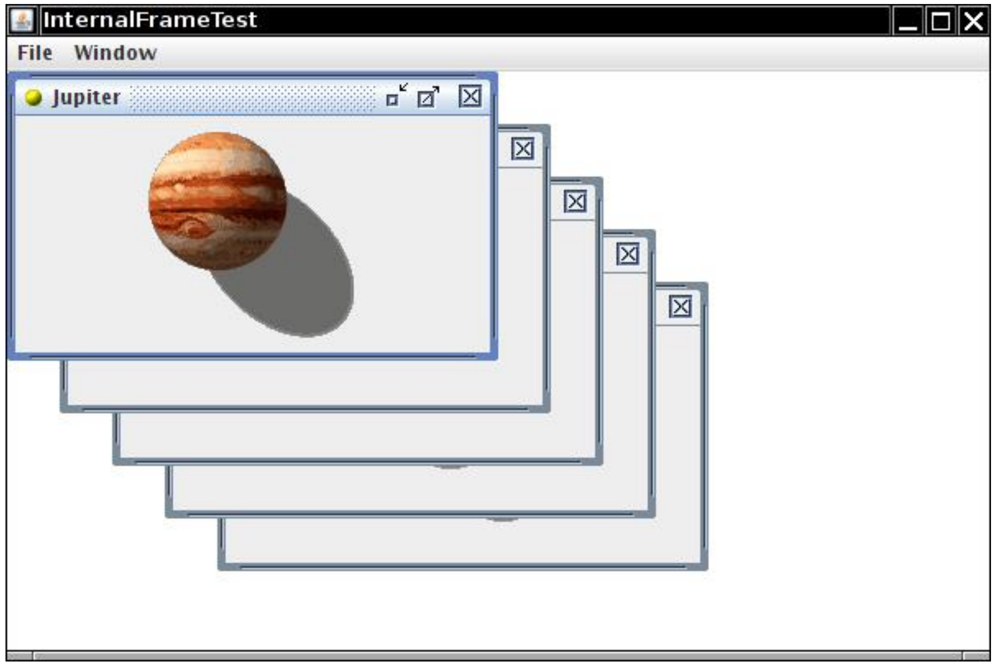
Desktop Panes and Internal Frames

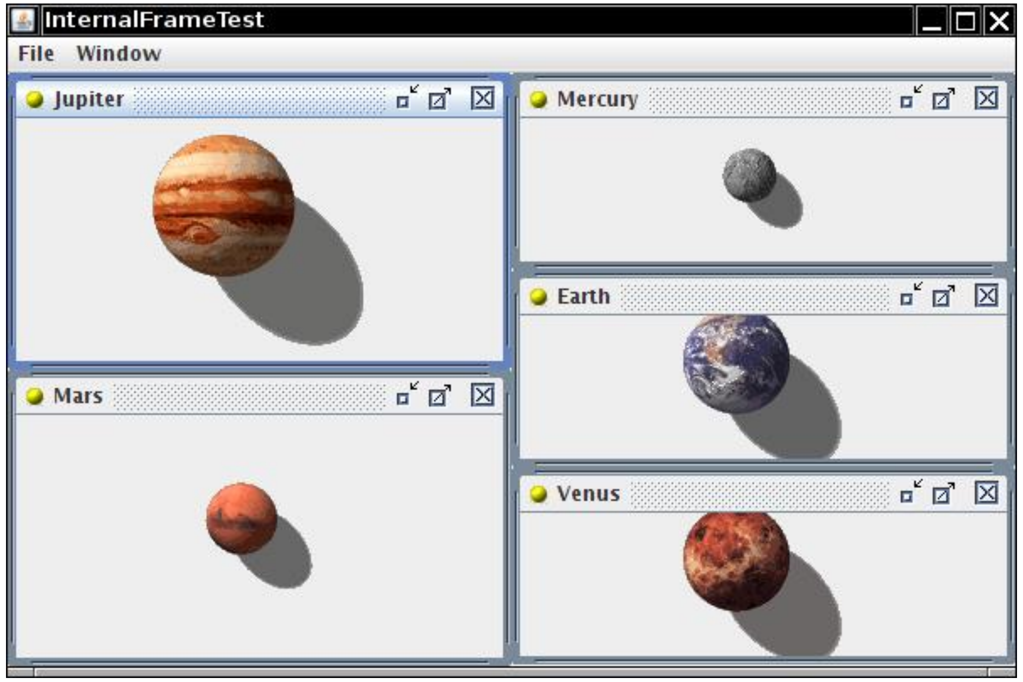
```
.//In the constructor of InternalFrameDemo, a JFrame subclass:
    desktop = new JDesktopPane();
    createFrame(); //Create first window
    setContentPane(desktop);
    ...
    //Make dragging a little faster but perhaps uglier.
    desktop.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);
```



With the JInternalFrame class you can display a JFrame-like window within another window. Usually, you add internal frames to a desktop pane. The desktop pane, in turn, might be used as the content pane of a JFrame. The desktop pane is an instance of JDesktopPane, which is a subclass of JLayeredPane that has added API for managing multiple overlapping internal frames.

Cascading and Tiling





- One of the things that a desktop manage will typically do (although it may not) is resuffle windows and icons around when the pane has been resized.

- A desktop manager will possibly do things like re-arranging icons so that they stay at the bottom of the pane. If the manage supports "tiled“ internal frames, then it might re arrange the positions of internal frames. One thing that even the default desktop manager javax.swing.DefaultDesktopManger should to is resize maximizedinternbal frames so that they stay maximized.

```
package examQuestions;

import javax.swing.*;

public class InternalFrameDemo extends JFrame {
    InternalFrameDemo(){
        for (int i=0; i<5; i++){
            JInternalFrame jInternalFrame = new JInternalFrame("Internal Frame "+i);
            jInternalFrame.setLocation(i*50+10, i*50+10);
            jInternalFrame.setSize(300, 300);
            this.add(jInternalFrame);
            jInternalFrame.setVisible(true);
        }
        setVisible(true);
        setSize(1000,1000);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        new InternalFrameDemo();
    }
}
```

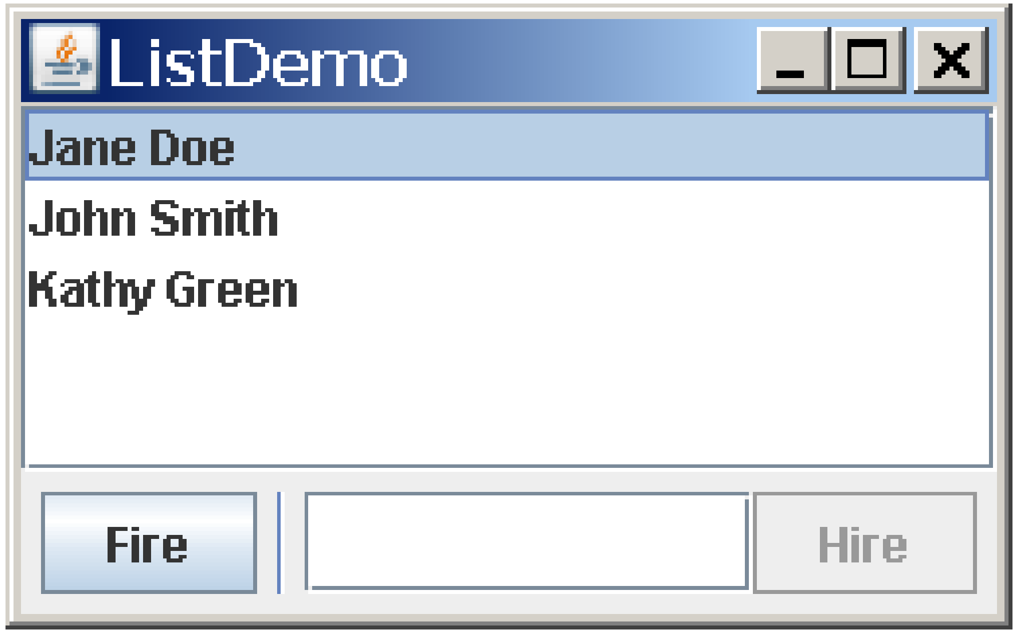
2.8 Advance Swing Components

List, Trees, Tables, Progress Bars

List

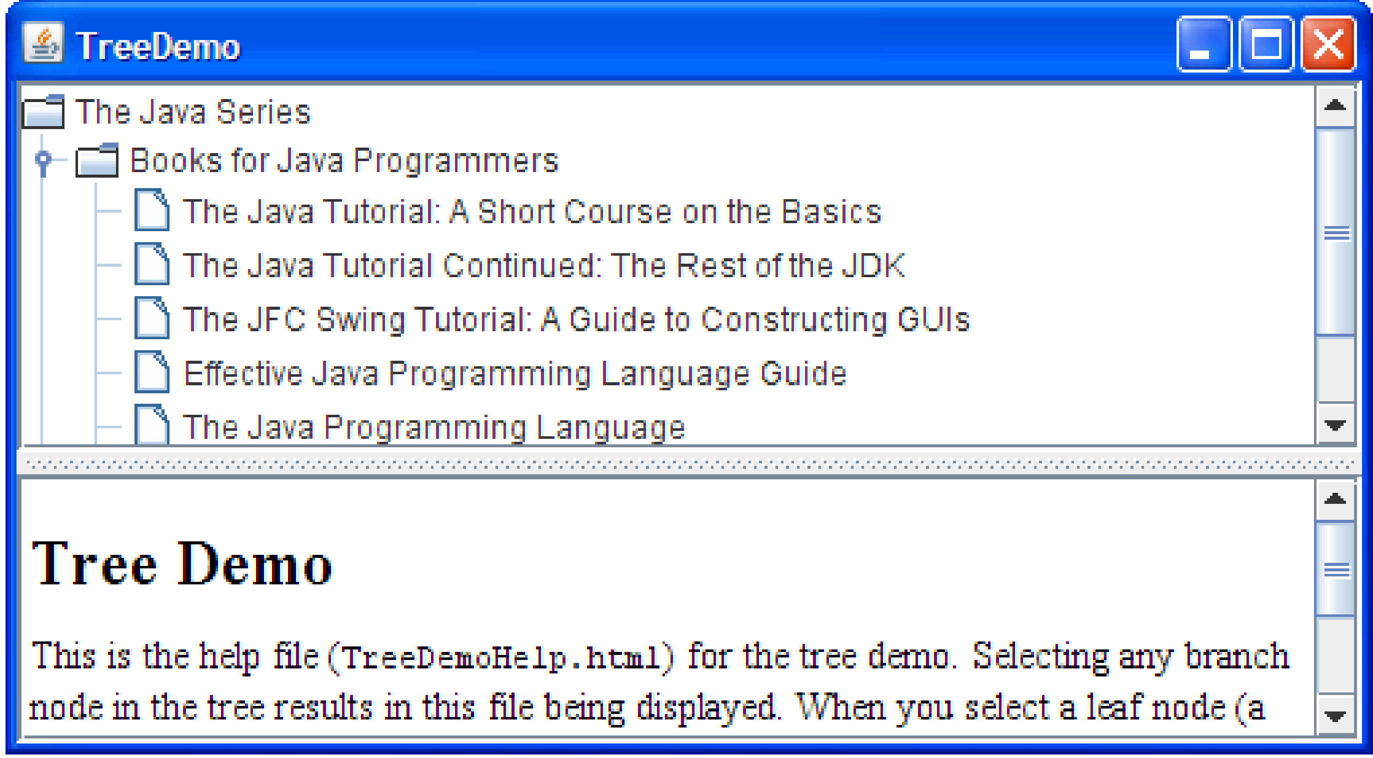
A JList presents the user with a group of items, displayed in one or more columns, to choose from. Lists can have many items, so they are often put in scroll panes.

```
list = new JList(data); //data has type Object[]
list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
list.setLayoutOrientation(JList.HORIZONTAL_WRAP);
list.setVisibleRowCount(-1);
...
JScrollPane listScroller = new JScrollPane(list);
listScroller.setPreferredSize(new Dimension(250, 80));
```



Trees

With the JTree class, you can display hierarchical data. A JTree object does not actually contain your data; it simply provides a view of the data. Like any non-trivial Swing component, the tree gets data by querying its data model.


TreeDemo
The Java Series
Books for Java Programmers
The Java Tutorial: A Short Course on the Basics
The Java Tutorial Continued: The Rest of the JDK
The JFC Swing Tutorial: A Guide to Constructing GUIs
Effective Java Programming Language Guide
The Java Programming Language

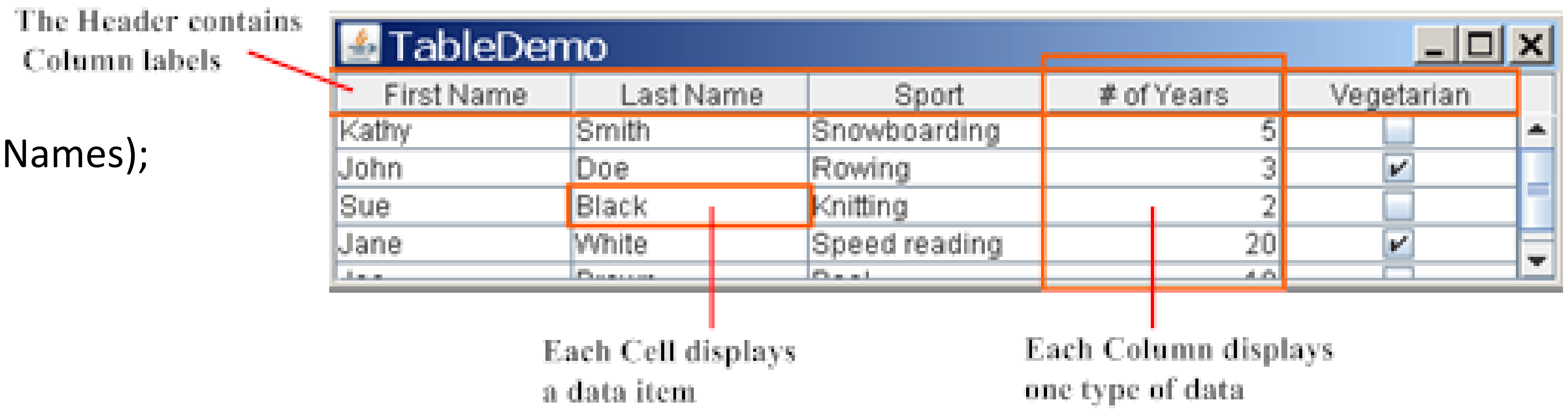
Tree Demo

This is the help file (TreeDemoHelp.html) for the tree demo. Selecting any branch node in the tree results in this file being displayed. When you select a leaf node (a

```
private JTree tree;
...
public TreeDemo() {
    ...
    DefaultMutableTreeNode top =
        new DefaultMutableTreeNode("The Java Series");
    createNodes(top);
    tree = new JTree(top);
    ...
    JScrollPane treeView = new JScrollPane(tree);
```

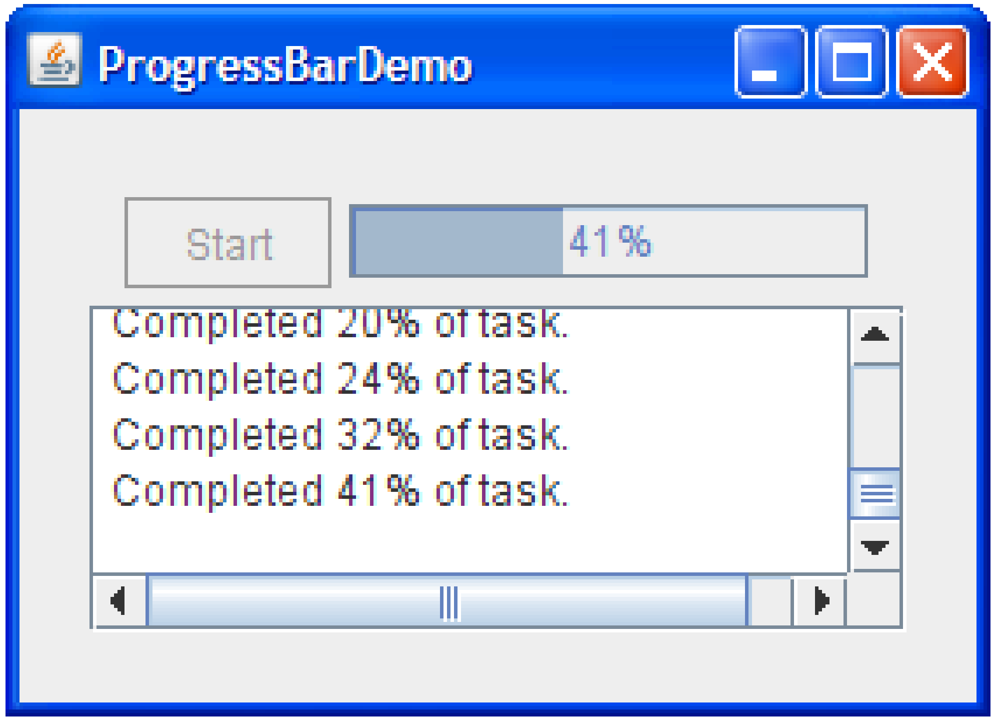
Tables, Progress Bars

```
JTable table = new JTable(data, columnNames);
```


The Header contains Column labels
TableDemo
Each Cell displays a data item
Each Column displays one type of data

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	☐
John	Doe	Rowing	3	✓
Sue	Black	Knitting	2	☐
Jane	White	Speed reading	20	✓

```
//Where member variables are declared:
JProgressBar progressBar;
...
//Where the GUI is constructed:
progressBar = new JProgressBar(0, task.getLengthOfTask());
progressBar.setValue(0);
progressBar.setStringPainted(true);
```



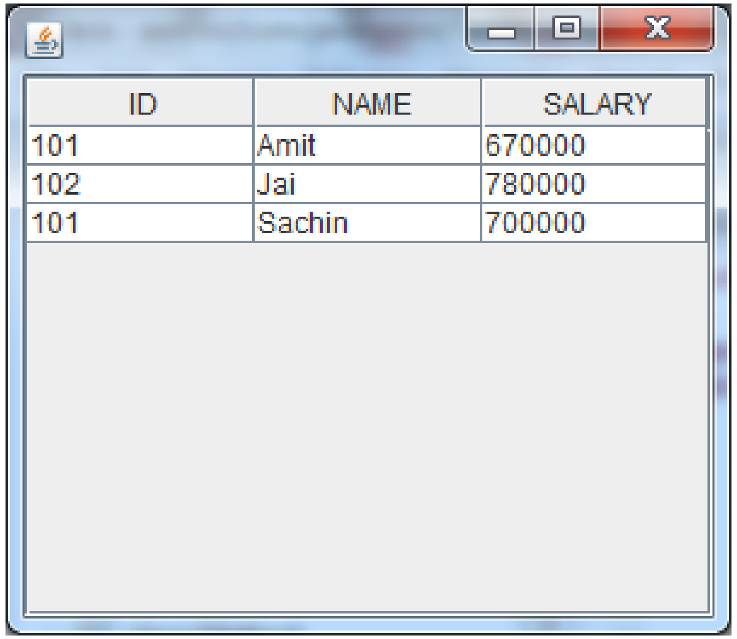
```
Completed 20% of task.
Completed 24% of task.
Completed 32% of task.
Completed 41% of task.
```

Java JTable Example

```
import javax.swing.*;
public class TableExample {
    JFrame f;
    TableExample(){
    f=new JFrame();
    String data[][]={ {"101","Amit","670000"},
                          {"102","Jai","780000"},
                          {"101","Sachin","700000"}};
    String column[]={"ID","NAME","SALARY"};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
    JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
}
```

```
public static void main(String[] args) {
    new TableExample();
}
}
```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

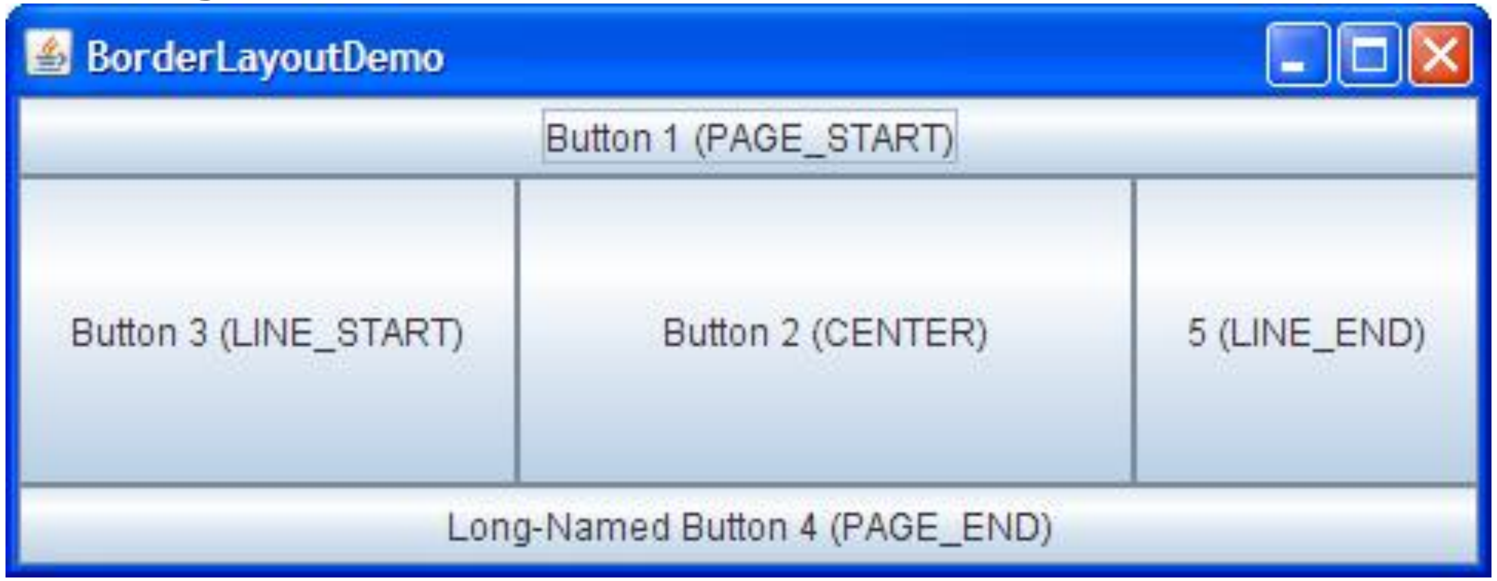
2.2 Layout Management

- The LayoutManagers are used to arrange components in a particular manner.
- LayoutManager is an interface that is implemented by all the classes of layout managers.
- There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.GridLayout
3. java.awt.GridBagLayout
4. javax.swing.GroupLayout
5. Using No Layout Managers
6. Custom Layout Managers

BorderLayout

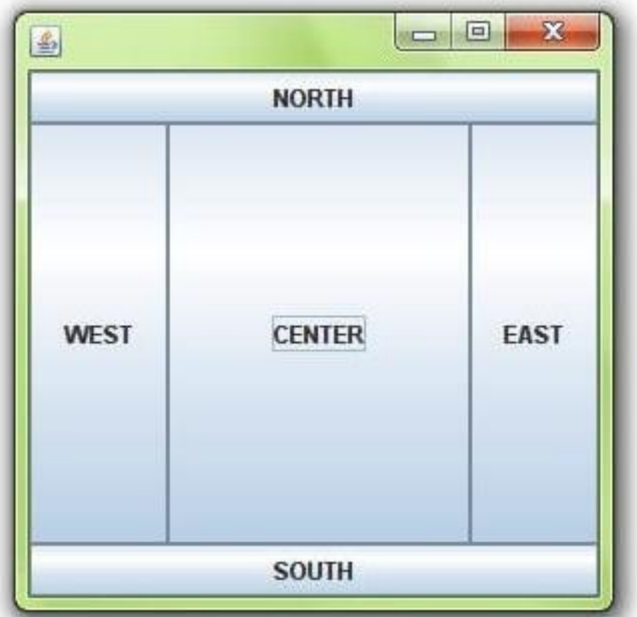
- A BorderLayout places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area.
- Tool bars that are created using JToolBar must be created within a BorderLayout container, if you want to be able to drag and drop the bars away from their starting positions.



Constructor or Method	Purpose
[BorderLayout(int *horizontalGap*, int *verticalGap*)]()	Defines a border layout with specified gaps between components.
[setHgap(int)]()	Sets the horizontal gap between components.
[setVgap(int)]()	Sets the vertical gap between components.

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

public class Border {
JFrame f;
Border(){
   f=new JFrame();

   JButton b1=new JButton("NORTH");;
   JButton b2=new JButton("SOUTH");;
   JButton b3=new JButton("EAST");;
   JButton b4=new JButton("WEST");;
   JButton b5=new JButton("CENTER");;

   f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST);
   f.add(b4,BorderLayout.WEST);
   f.add(b5,BorderLayout.CENTER);
```

```java
   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args) {
   new Border();
}
}
```

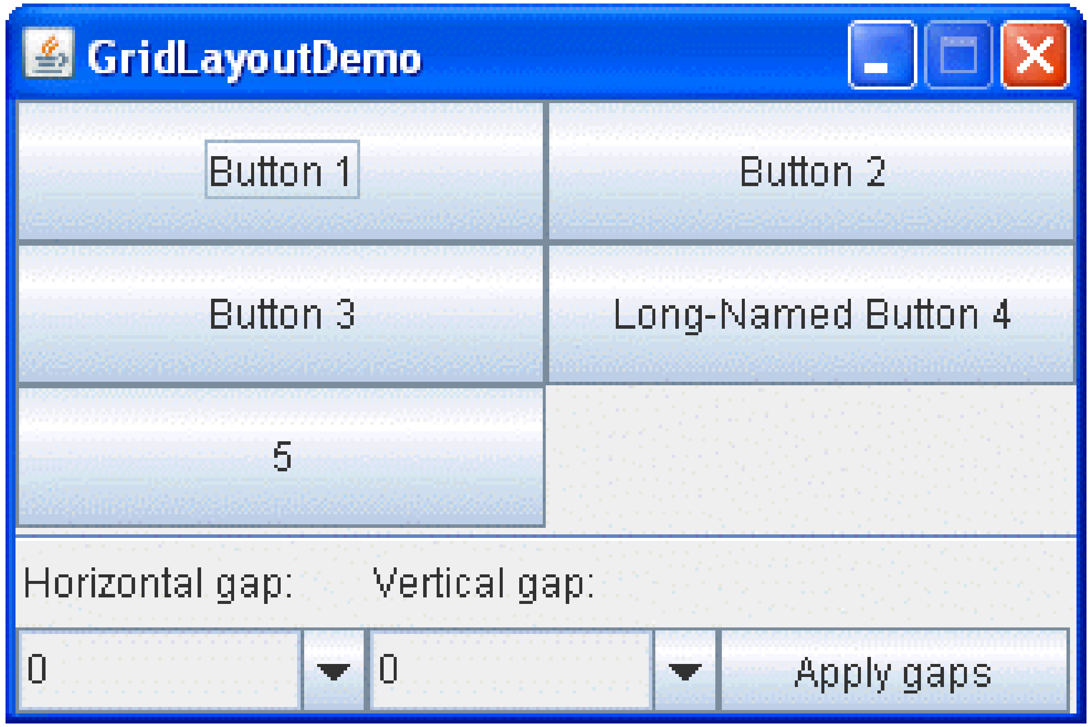


GridLayout

GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.

The Constructors used are:

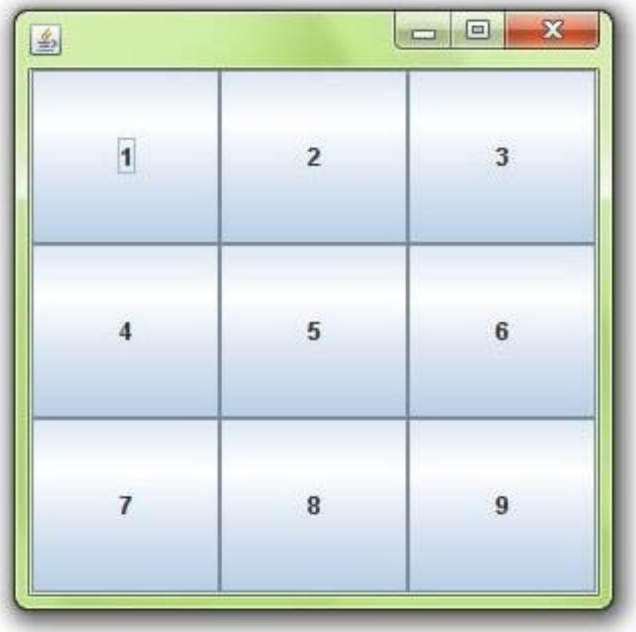
- GridLayout(int rows, int cols)
- GridLayout(int rows, int cols, int hgap, int vgap) .



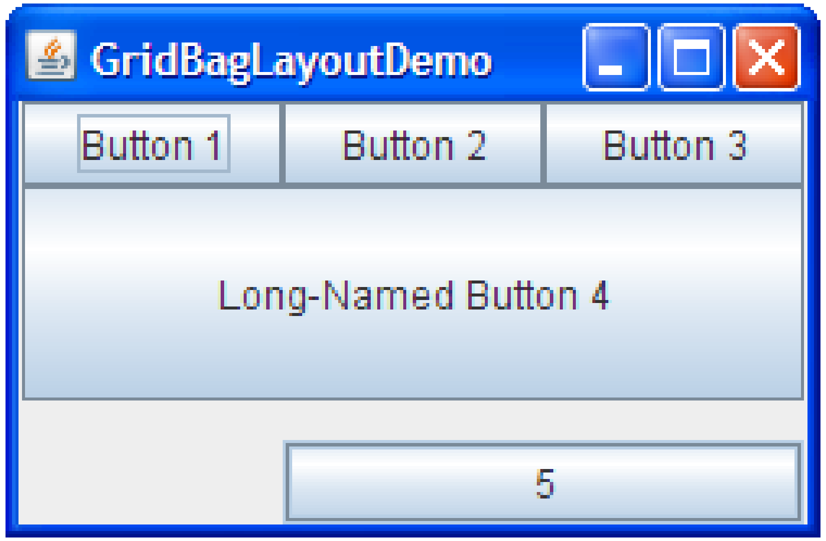
```
GridLayout experimentLayout = new GridLayout(0,2);

    compsToExperiment.setLayout(experimentLayout);
    compsToExperiment.add(new JButton("Button 1"));
    compsToExperiment.add(new JButton("Button 2"));
    compsToExperiment.add(new JButton("Button 3"));
    compsToExperiment.add(new JButton("Long-Named Button 4"));
    compsToExperiment.add(new JButton("5"));
```

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

public class MyGridLayout{
JFrame f;
MyGridLayout(){
   f=new JFrame();

   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");
      JButton b6=new JButton("6");
      JButton b7=new JButton("7");
   JButton b8=new JButton("8");
      JButton b9=new JButton("9");
```

```java
    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
    f.add(b6);f.add(b7);f.add(b8);f.add(b9);

    f.setLayout(new GridLayout(3,3));
    //setting grid layout of 3 rows and 3 columns

    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new MyGridLayout();
}
}
```



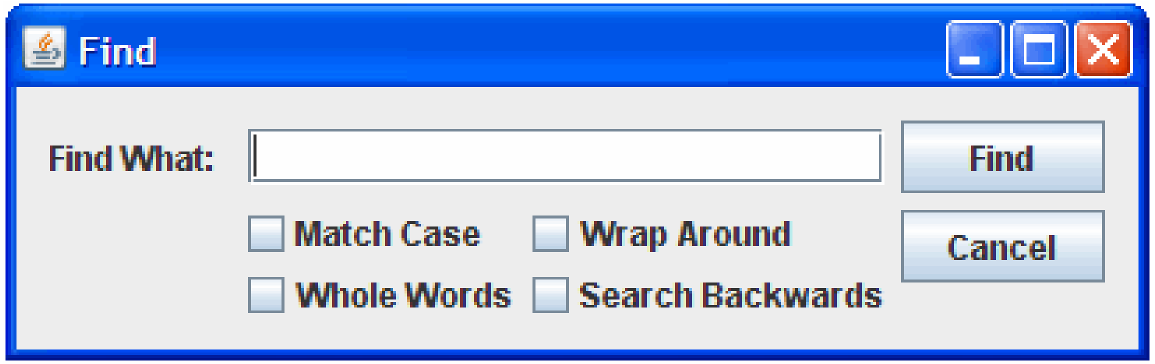
GridBagLayout



- GridBagLayout is a sophisticated, flexible layout manager.
- It aligns components by placing them within a grid of cells, allowing components to span more than one cell.
- The rows in the grid can have different heights, and grid columns can have different widths.

```
JPanel pane = new JPanel(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();

//For each component to be added to this container:
//...Create the component...
//...Set instance variables in the GridBagConstraints instance...
pane.add(theComponent, c);
```

GroupLayout



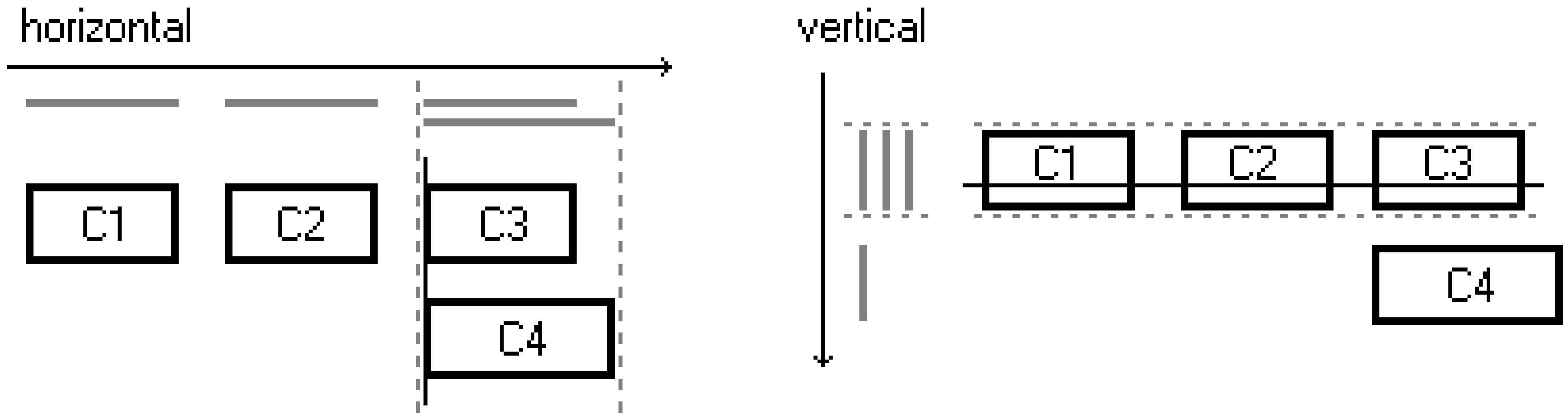
- GroupLayout is a layout manager that was developed for use by GUI builder tools, but it can also be used manually.
- GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently.
- Consequently, however, each component needs to be defined twice in the layout. The Find window shown above is an example of a GroupLayout.
- You do not need to worry about the *vertical* dimension when defining the *horizontal* layout, and vice versa, as the layout along each axis is totally independent of the layout along the other

```
GroupLayout layout = new GroupLayout(panel);
panel.setLayout(layout);
```

We specify automatic gap insertion:

```
layout.setAutoCreateGaps(true);
layout.setAutoCreateContainerGaps(true);
```

- GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently.
- GroupLayout uses two types of arrangements -- sequential and parallel, combined with hierarchical composition.



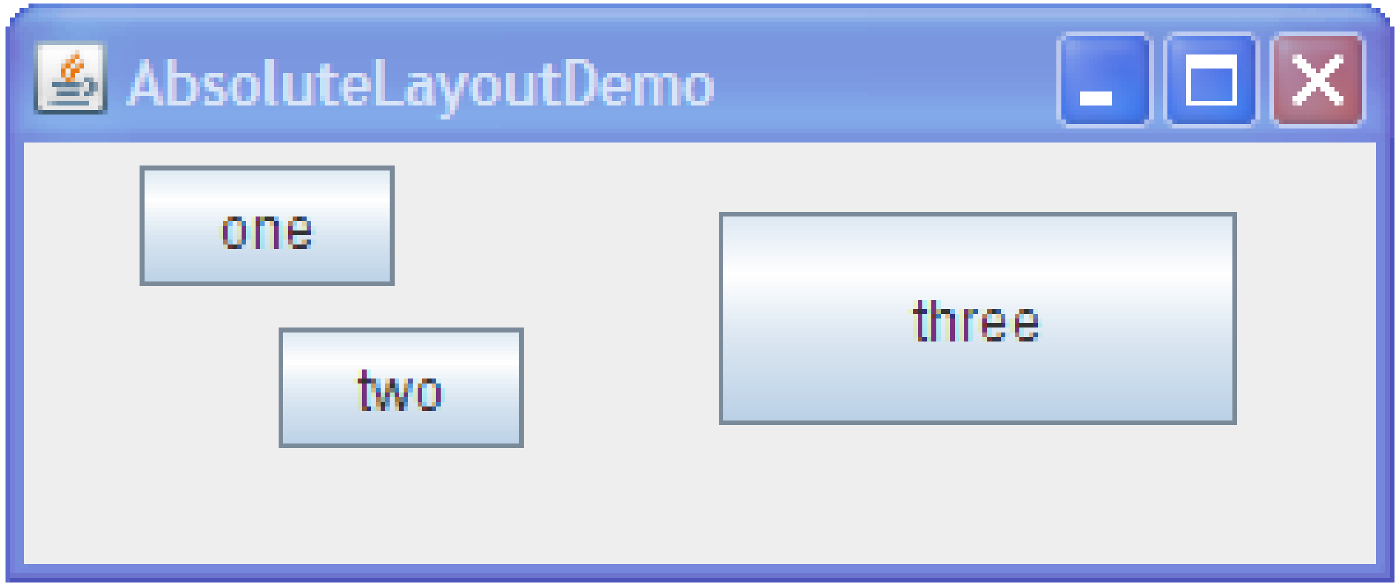
In pseudo code, the layout specification now looks like this:

```
horizontal layout = sequential group { c1, c2, parallel group (LEFT) { c3, c4 } }
vertical layout = sequential group { parallel group (BASELINE) { c1, c2, c3 }, c4 }
```

Doing Without a Layout Manager (Absolute Positioning)

- Although it is possible to do without a layout manager, you should use a layout manager if at all possible. A layout manager makes it easier to adjust to look-and-feel-dependent component appearances, to different font sizes, to a container's changing size, and to different locales. Layout managers also can be reused easily by other containers, as well as other programs.

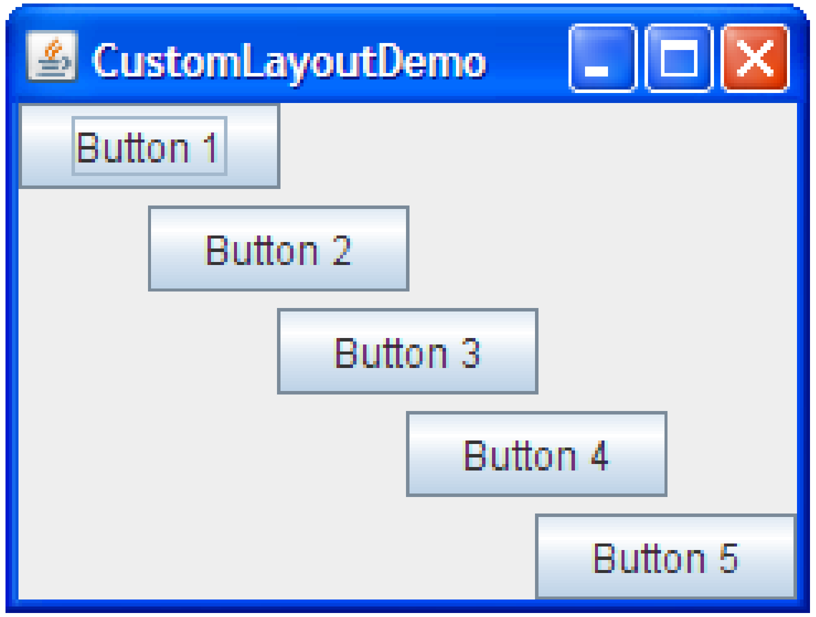
```
pane.setLayout(null);

JButton b1 = new JButton("one");
JButton b2 = new JButton("two");
JButton b3 = new JButton("three");

pane.add(b1);
pane.add(b2);
pane.add(b3);
```



Creating a Custom Layout Manager

- To create a custom layout manager, you must create a class that implements the LayoutManager interface. You can either implement it directly, or implement its subinterface, LayoutManager2.
- Every layout manager must implement at least the following five methods, which are required by the LayoutManager interface:

1. void addLayoutComponent(String, Component)
2. void removeLayoutComponent(Component)
3. Dimension preferredLayoutSize(Container)
4. Dimension minimumLayoutSize(Container)
5. void layoutContainer(Container)

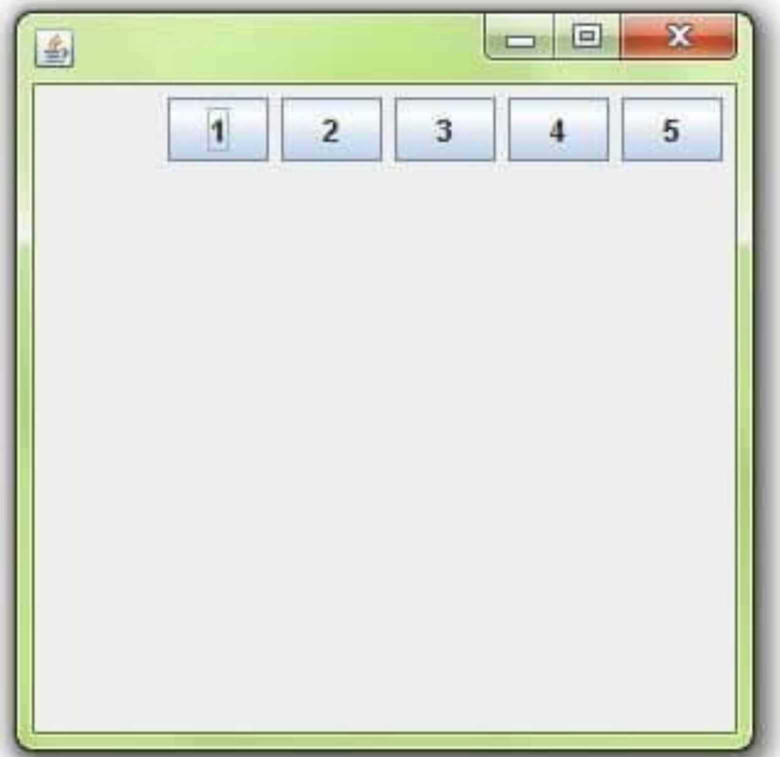


- If you wish to support component constraints, maximum sizes, or alignment, then your layout manager should implement the LayoutManager2 interface. In fact, for these reasons among many others, nearly all modern layout managers will need to implement LayoutManager2. That interface adds five methods to those required by LayoutManager:

1. addLayoutComponent(Component, Object)
2. getLayoutAlignmentX(Container)
3. getLayoutAlignmentY(Container)
4. invalidateLayout(Container)
5. maximumLayoutSize(Container)

FlowLayout

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

public class MyFlowLayout{
JFrame f;
MyFlowLayout(){
   f=new JFrame();

   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");

   f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

   f.setLayout(new FlowLayout(FlowLayout.RIGHT));
   //setting flow layout of right alignment
 f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args) {
   new MyFlowLayout();
}
}
```



Event and Listener (Java Event Handling)

- Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

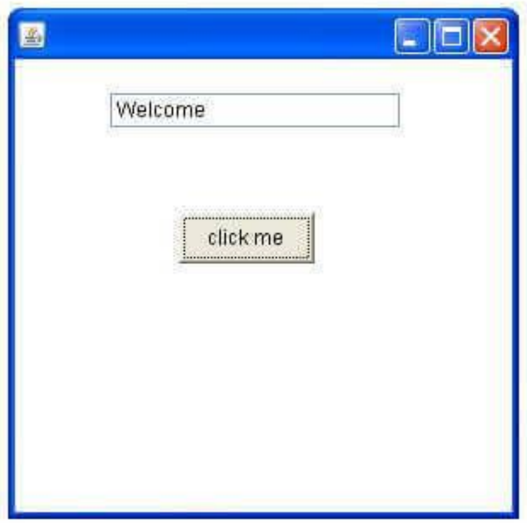
Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Registration Methods

- **Button**
    - public void addActionListener(ActionListener a){}
- **MenuItem**
    - public void addActionListener(ActionListener a){}
- **TextField**
    - public void addActionListener(ActionListener a){}
    - public void addTextListener(TextListener a){}
- **TextArea**
    - public void addTextListener(TextListener a){}
- **Checkbox**
    - public void addItemListener(ItemListener a){}
- **Choice**
    - public void addItemListener(ItemListener a){}
- **List**
    - public void addActionListener(ActionListener a){}
    - public void addItemListener(ItemListener a){}

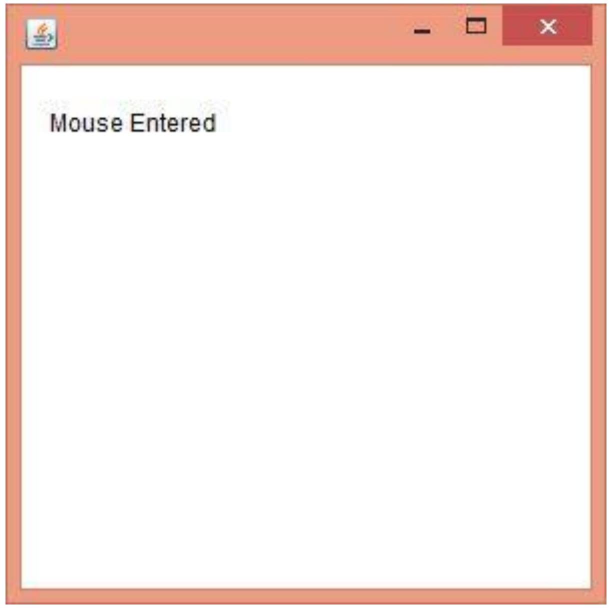
Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```



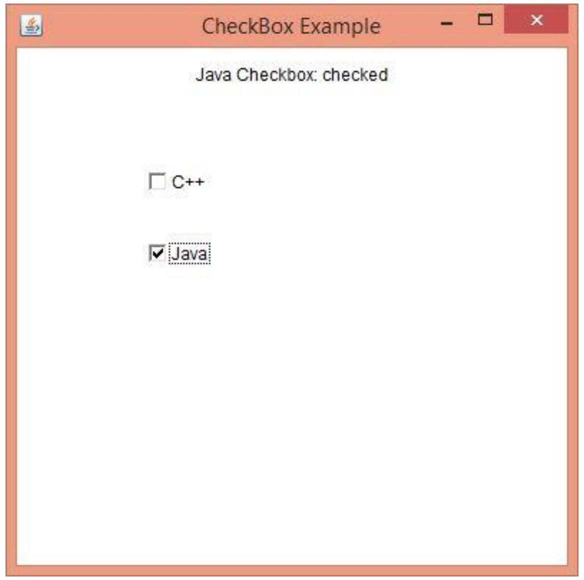
MouseListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements
MouseListener{
    Label l;
    MouseListenerExample(){
        addMouseListener(this);

        l=new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
        l.setText("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e) {
        l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e) {
        l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
        l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
        l.setText("Mouse Released");
    }
public static void main(String[] args) {
    new MouseListenerExample();
}
}
```


Mouse Entered

ItemListener Interface

```
import java.awt.*;
import java.awt.event.*;
public class ItemListenerExample implements ItemListener{

    Checkbox checkBox1,checkBox2;
    Label label;
    ItemListenerExample(){
        Frame f= new Frame("CheckBox Example");
        label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        checkBox1 = new Checkbox("C++");
        checkBox1.setBounds(100,100, 50,50);
        checkBox2 = new Checkbox("Java");
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        checkBox1.addItemListener(this);
        checkBox2.addItemListener(this);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
public void itemStateChanged(ItemEvent e) {
        if(e.getSource()==checkBox1)
            label.setText("C++ Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
        if(e.getSource()==checkBox2)
        label.setText("Java Checkbox: "
        + (e.getStateChange()==1?"checked":"unchecked"));
      }
public static void main(String args[])
{
    new ItemListenerExample();
}
}
```


CheckBox Example

Java Checkbox: checked

☐ C++

☑ Java

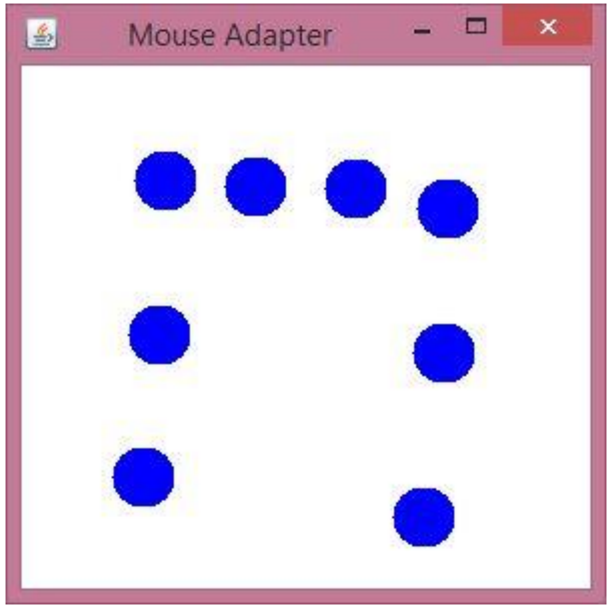
Java Adapter Classes

- Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.
- In an adapter class, there is no need for implementation of all the methods presented in an interface. It is used when only some methods of defined by its interface have to be overridden.
- In a listener, all the methods that have been declared in its interface have to be implemented. This is because these methods are

Adapter class	**Listener interface**
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

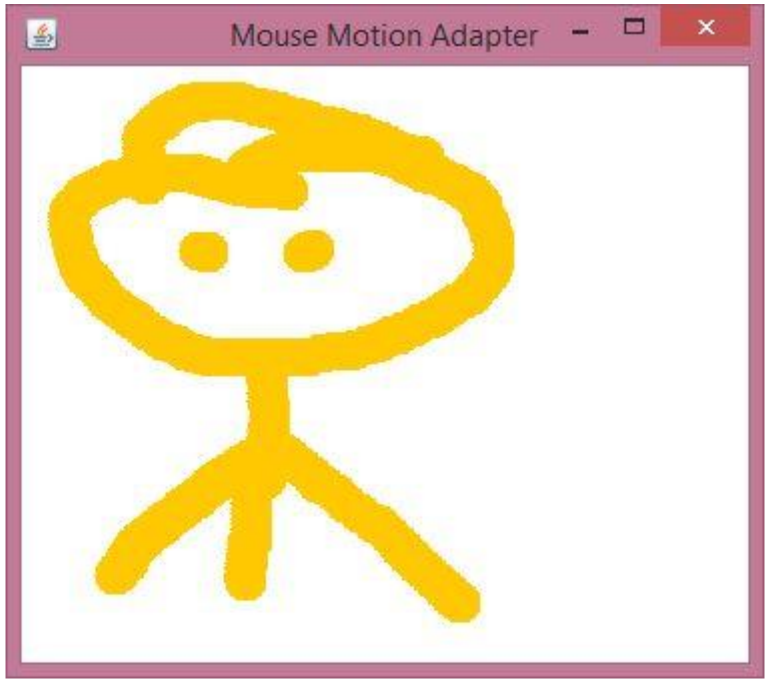
Java MouseAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
    Frame f;
    MouseAdapterExample(){
        f=new Frame("Mouse Adapter");
        f.addMouseListener(this);

        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
        Graphics g=f.getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(e.getX(),e.getY(),30,30);
    }

public static void main(String[] args) {
    new MouseAdapterExample();
}
}
```

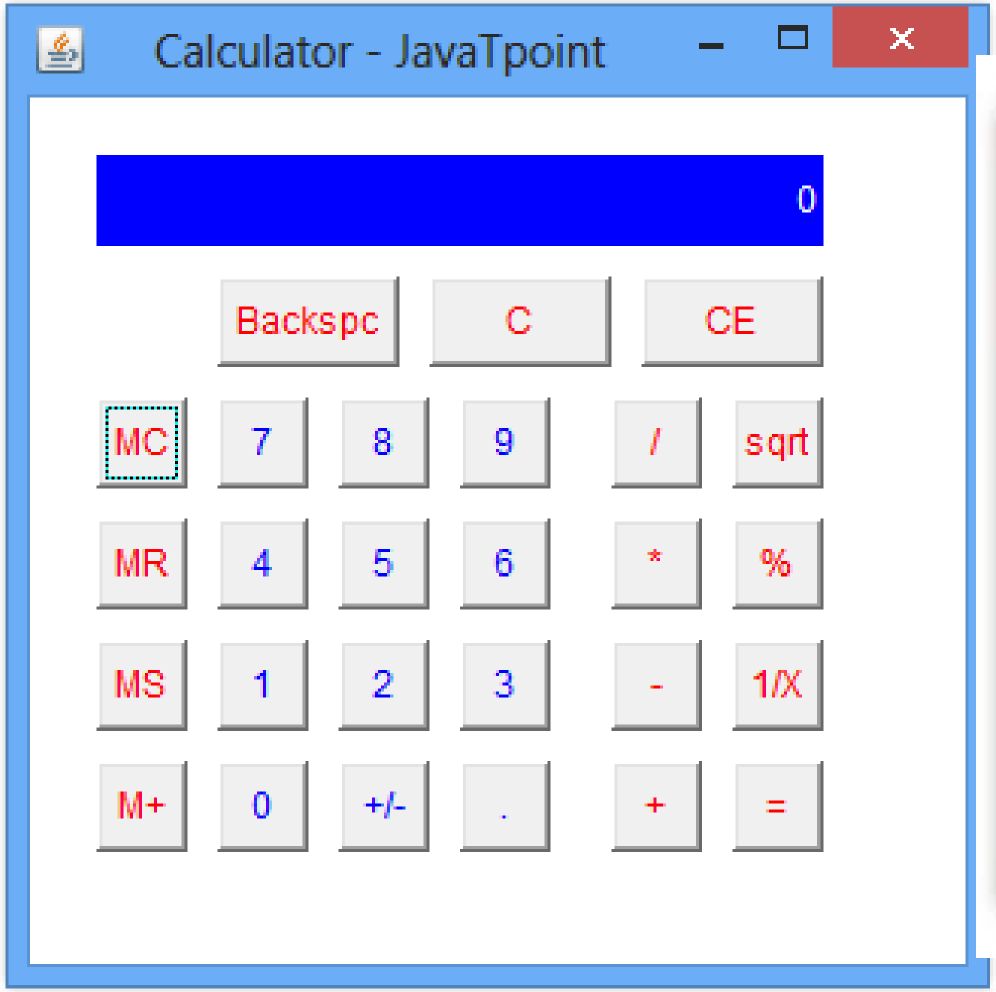


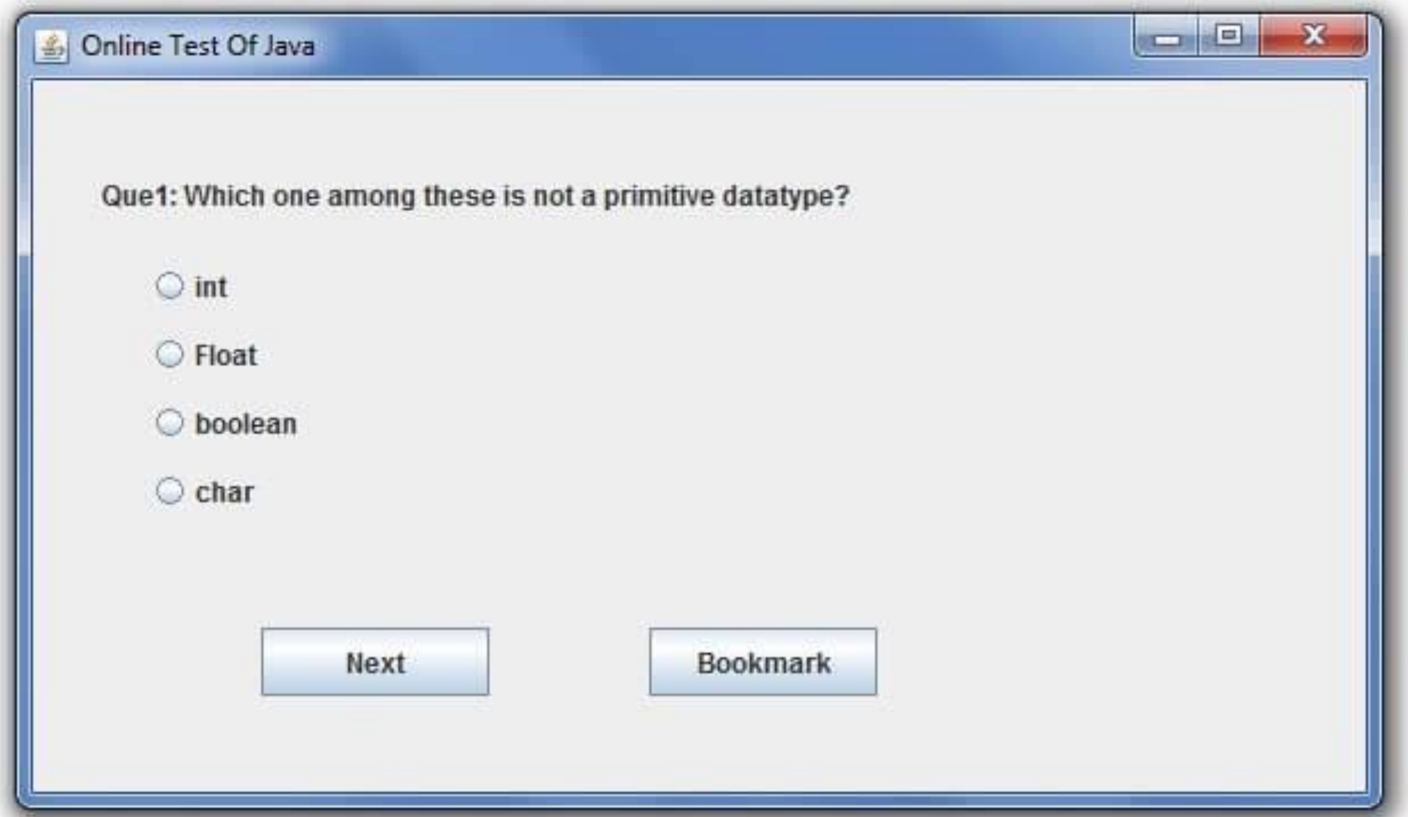
MouseMotionAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionAdapterExample extends MouseMotionAdapter{
    Frame f;
    MouseMotionAdapterExample(){
        f=new Frame("Mouse Motion Adapter");
        f.addMouseMotionListener(this);

        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
public void mouseDragged(MouseEvent e) {
    Graphics g=f.getGraphics();
    g.setColor(Color.ORANGE);
    g.fillOval(e.getX(),e.getY(),20,20);
}
public static void main(String[] args) {
    new MouseMotionAdapterExample();
}
}
```



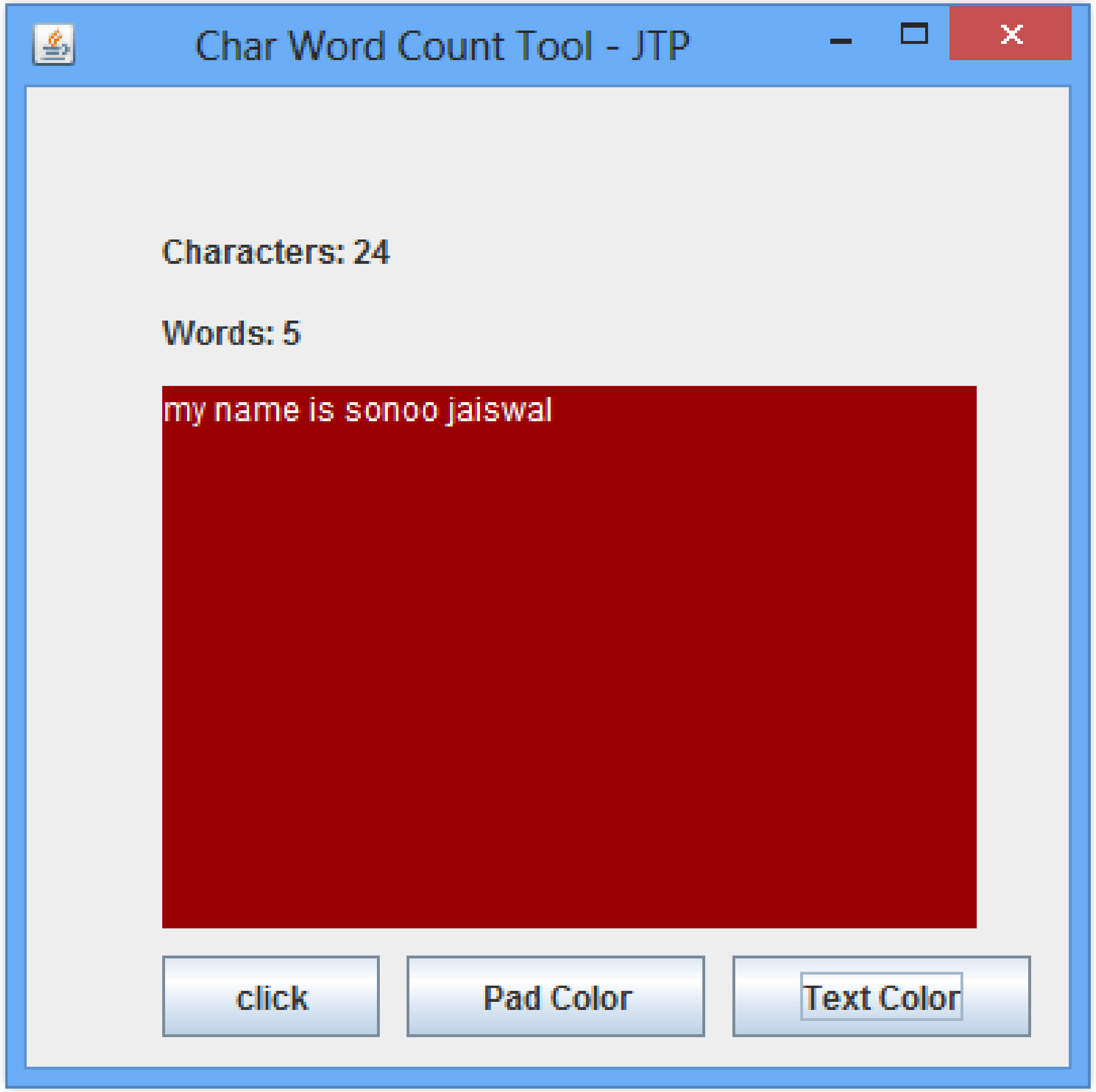
Classworks (Some Applications)





Que1: Which one among these is not a primitive datatype?

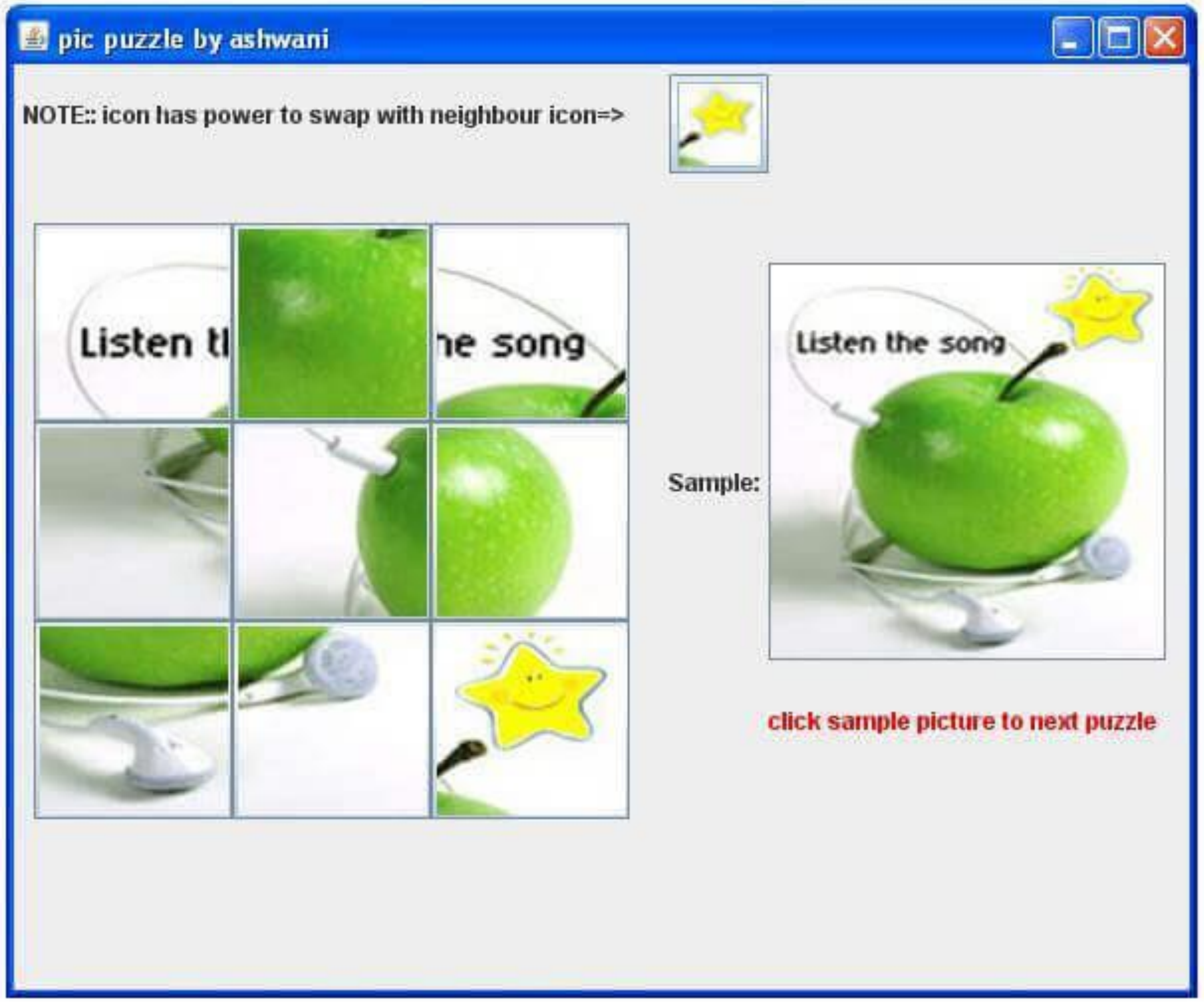
- ◯ int
- ◯ Float
- ◯ boolean
- ◯ char



Characters: 24

Words: 5

my name is sonoo jaiswal



NOTE:: icon has power to swap with neighbour icon=>

click sample picture to next puzzle

What is Java Beans?

- A **Java Bean** is a software component that has been designed to be **reusable** in a variety of different environments.
- A lot of Objects are encapsulated into one object called Beans.
- Sun Microsystem introduced JavaBeans in the year 1996
- Written in Java
- There is no restriction on the capability of a Bean. It may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio.
- A JavaBean is just a standard. It's just a convention. Lots of libraries depend on it though.

Properties of Java Beans

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why Java Beans have to be serializable?

Serializable interface can have their state saved and restored. This state includes data from all of the classes in the object's class hierarchy. So an object need not worry about the serialization of data from superclasses, as this is handled automatically. The Serializable interface has no methods.

Advantages of Java Beans?

1. Bean obtains all the benefits of Java's "write-once, run-anywhere" paradigm.
2. Easy to Configure
3. Platform Independent
4. The properties, events, and methods of a Bean that are exposed to an application builder tool can be controlled.
5. A Bean may be designed to operate correctly in different locales, which makes it useful in global markets.
6. Auxiliary software can be provided to help a person configure a Bean. This software is only needed when the design-time parameters for that component are being set. It does not need to be included in the run-time environment.
7. The configuration settings of a Bean can be saved in persistent storage and restored at a later time.
8. A Bean may register to receive events from other objects and can generate events that are sent to other objects.

Is this class Java Bean? Why?

```
public class Employee implements java.io.Serializable{

        private int id;
        private String name;

 public Employee(){}

 public void setId(int id){this.id=id;}

 public int getId(){return id;}

 public void setName(String name){this.name=name;}

 public String getName(){return name;}

}
```

How to access Java Bean Class?

```
public class Test{
public static void main(String args[]){

        Employee e=new Employee(); //object is created

        e.setName("Arjun"); //setting value to the object

        System.out.println(e.getName());

}}
```

Note: There are two ways to provide values to the object,
one way is by constructor and second is by setter method.

So, What’s difference between Java Bean Class and basic java class?

A JavaBean is just a normal Java class that conforms to a number of conventions:

- it should have a public default constructor (i.e. a constructor that takes no arguments)
- it should be serializable
- it has getter and setter methods to get and set its properties

A class is a group of objects which have common properties.

- It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.
- A class in Java can contain:
    - fields
    - methods
    - constructors

BDK and Bean Box

- BeanBox is a test container in BDK. We can use BeanBox to create Application, Applet, and New Beans.
- We can lay out, edit and interconnect beans visually. That is one of the benefits of Javabeans is the capability for the beans to be used in visual application builder tools.
- The BeanBox allows you to:
    - Drop beans onto a composition window
    - Resize and move beans around
    - Edit the exported properties of a bean
    - Run a customizer to configure a bean
    - Connect a bean event source to an event handler method
    - Connect together bound properties on different beans
    - Save and restore sets of beans
    - Making applets from beans
    - Get an introspection report on a bean
    - Add new beans from JAR files

Working with Manifest Files

- JAR files support a wide range of functionality, including electronic signing, version control, package sealing, and others. What gives a JAR file this versatility? The answer is the JAR file's *manifest*.

- The manifest is a special file that can contain information about the files packaged in a JAR file. By tailoring this "meta" information that the manifest contains, you enable the JAR file to serve a variety of purposes.

- When you create a JAR file, it automatically receives a default manifest file. There can be only one manifest file in an archive, and it always has the pathname

- META-INF/MANIFEST.MF
- When you create a JAR file, the default manifest file simply contains the following:

- Manifest-Version: 1.0
- Created-By: 1.7.0_06 (Oracle Corporation)

Java Beans with JSP

```
package com.tutorialspoint;

public class StudentsBean implements java.io.Serializable
{
   private String firstName = null;
   private String lastName = null;
   private int age = 0;

   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(Integer age){
      this.age = age;
   }
}
```

```
<html>   <head>     <title>get and set properties Example</title>   </head>

  <body>
    <jsp:useBean id = "students" class = "com.tutorialspoint.StudentsBean">
      <jsp:setProperty name = "students" property = "firstName" value = “Sandeep"/>
      <jsp:setProperty name = "students" property = "lastName" value = “Lamichhane"/>
      <jsp:setProperty name = "students" property = "age" value = “24"/>
    </jsp:useBean>

    <p>Student First Name:
            <jsp:getProperty name = "students" property = "firstName"/>
    </p>
    <p>Student Last Name:
              <jsp:getProperty name = "students" property = "lastName"/>
    </p>
    <p>Student Age:
              <jsp:getProperty name = "students" property = "age"/>
    </p>

  </body>
</html>
```

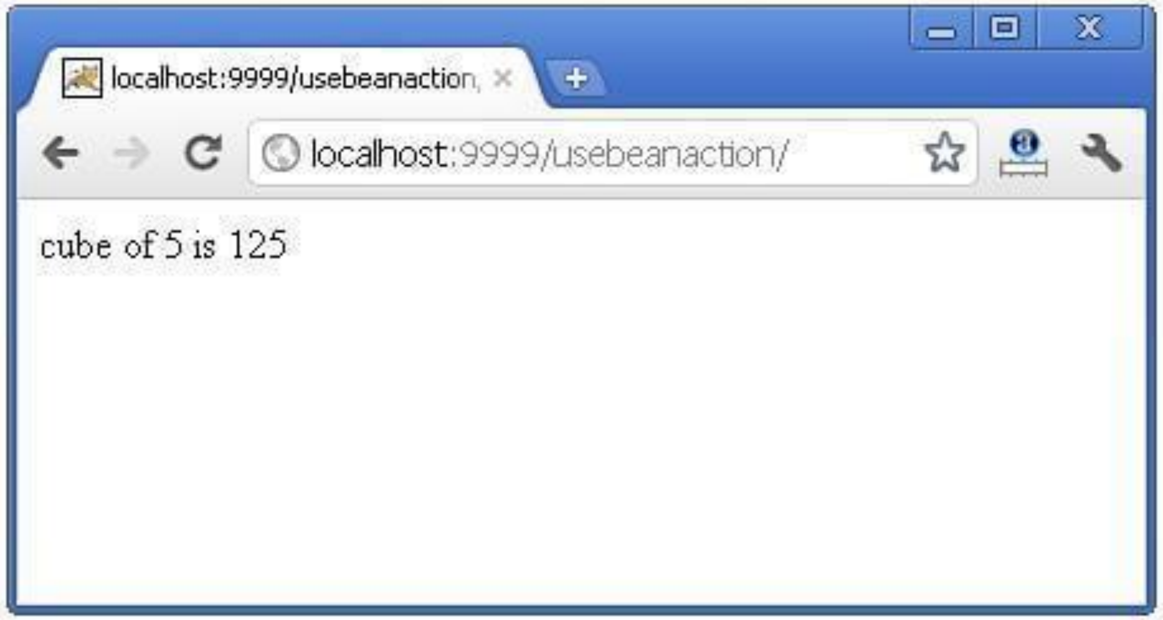
Output:

```
Student First Name: Sandeep

Student Last Name: Lamichhane

Student Age: 24
```

One More Simple Example

Calculator.java (a simple Bean class)

```
package com.javatpoint;
public class Calculator{

public int cube(int n){return n*n*n;}

}
```



```
cube of 5 is 125
```

index.jsp file

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

Video Tutorials (Java Beans)

- https://www.youtube.com/watch?v=6JZs8zJsyJE
- https://www.youtube.com/watch?v=u2QWxX7Iy3E

OLD TU Questions

- How java beans are different from java classes? Explain bean writing process. **[Model Question]**

- What is java beans? Differentiate it with java class. Discuss bean writing process with suitable examples. **(2+2+6) [2069]**

- What is java beans? How is it different from java class? **[2070]**

- What is java beans? Differentiate it with java classes. **(1+4) [2071]**

- What is Java bean? How is it different from other Java classes? **[2 + 3] [2072]**

- Discuss property design patterns for Java Beans. **[5] [2074]**