UNIT V

UNIT V

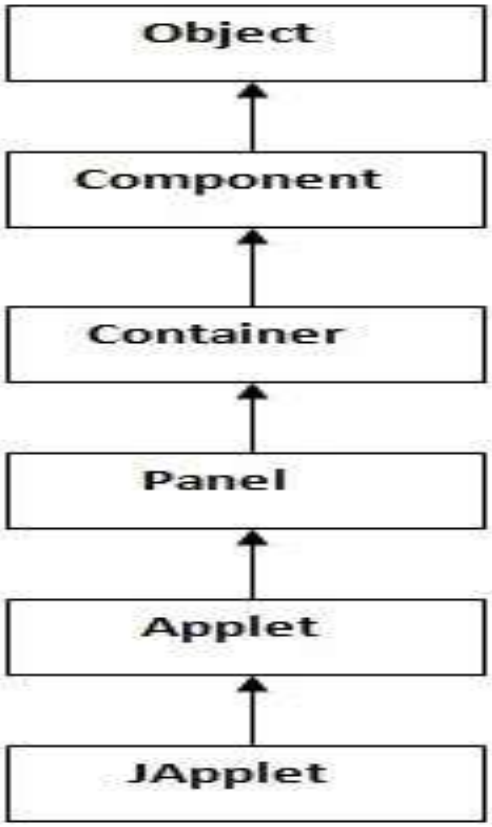
APPLETS, AWT AND SWINGS :

Applet class, Applet structure, an example Applet program, Applet life cycle. Event Handling- Introduction, Event Delegation Model, Java.awt.event Description, Adapter classes, Innerclasses. Abstract Window Toolkit: Why AWT?,java.awt package, components and containers, Button, Label, Checkbox, Radio buttons, List boxes, choice boxes, Text field and Text area, container classes, Layout Managers. Swing: Introduction, JFrame, JApplet, JPanel, Components in swings, JList and JScroll Pane, Split Pane, JTabbed Pane, Dialog Box, Pluggable Look andfeel.

Java Applet

- Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Sample Code for a Java Applet

```
import java.applet.Applet;
import java.awt.Graphics;
// importing necessary packages
/* <applet code="printapplet" width=500 height=500>
   </applet>*/
public class printapplet extends Applet
{
   public void paint(Graphics g)
   {
      // printing the message using drawString() method
      // other parameters are of the position
      g.drawString("Java Applet", 250, 250);
   }
}
```

we need to embed it into an HTML file. We will use the applet tags.

```
<html>
   <applet code="printapplet" width=500 height=500>
   </applet>
</html>
```

We will run the following command :

javac printapplet.java

```
//First.java
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome",150,150);
}
}
```

To compile and Execute the Applet We will run the following command :

```
javac First.java
appletviewer First.java
```

Lifecycle of Java Applet

- Applet is initialized.
- Applet is started.
- Applet is painted.
- Applet is stopped.
- Applet is destroyed.

- **Lifecycle methods for Applet (5 methods):**
- The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.
- For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.
- **public void init():** is used to initialized the Applet. It is invoked only once.
- **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
- **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- **public void destroy():** is used to destroy the Applet. It is invoked only once.

- The Component class provides 1 life cycle method of applet.
- **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.
- **How to run an Applet?**
- There are two ways to run an applet
- By html file.
- By appletViewer tool

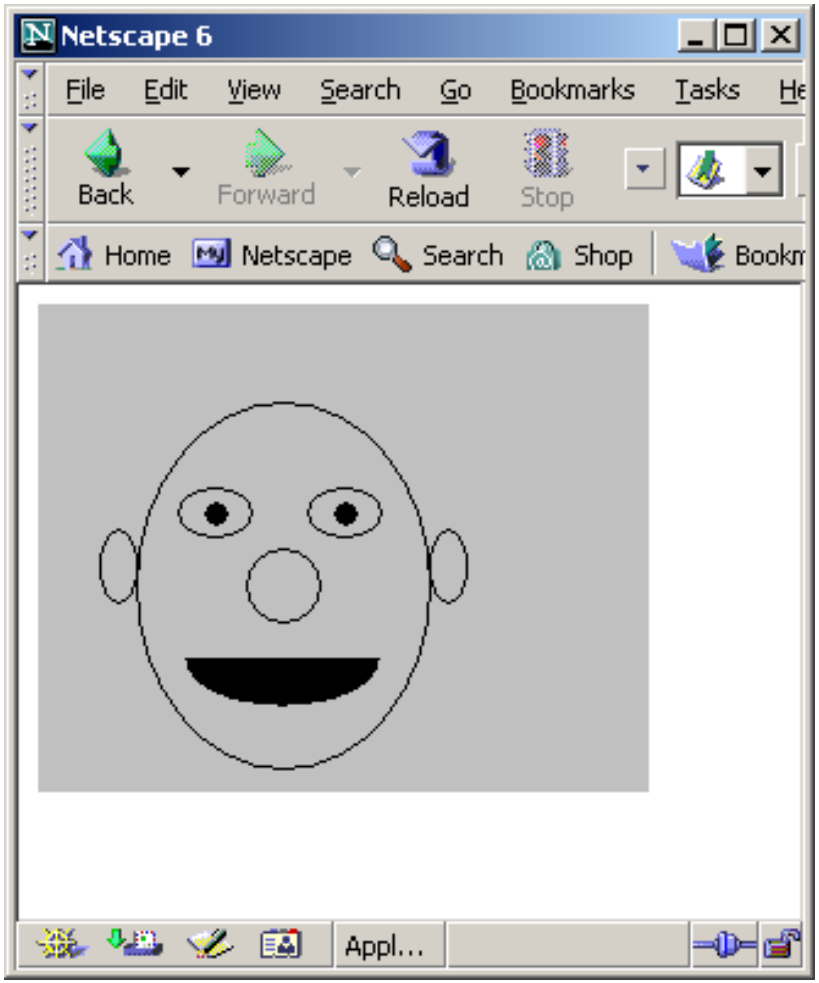
Displaying Graphics in Applet

- java.awt.Graphics class provides many methods for graphics programming.
- Commonly used methods of Graphics class:
- **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
- **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
- **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
- **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
- **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
- **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
- **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
- **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
- **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
- **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
- **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Drawing a Happy Face Applet

```
import java.awt.*;
import java.applet.*;
/* <applet code=" Face.class" width="500" height="500">
</applet> */
public class Face extends Applet
{
    public void paint(Graphics g)
    {
        g.drawOval(40,40,120,150);              //Head
        g.drawOval(57,75,30,20);                //Left eye
        g.drawOval(110,75,30,20);               //Right eye
        g.fillOval(68,81,10,10);                //Pupil (left)
        g.fillOval(121,81,10,10);               //Pupil (right)
        g.drawOval(85,100,30,30);               //Nose
        g.fillArc(60,125,80,40,180,180);        //Mouth
        g.drawOval(25,92,15,30);                //Left ear
        g.drawOval(160,92,15,30);               //Right ear
    }
}
```

☺ Output!


Netscape 6
File
Edit
View
Search
Go
Bookmarks
Tasks
Back
Forward
Reload
Stop
Home
Netscape
Search
Shop
Appl...

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.

The Delegation Event Model

- The modern approach to handling events is based on the *delegation event model, which defines* standard and consistent mechanisms to generate and process events.
- a *source generates an event and sends it to one or more listeners.*
- In this scheme,
- The listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns.
- In *delegation event model, listeners must register with a source in order to receive an event notification.*

Events

- In the delegation model, an *event is an object that describes a state change in a source*.
- *It can* be generated as a consequence of a person interacting with the elements in a graphical user interface.
- Ex: Pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse. Event may also occur that are not directly caused by interactions with a user interface. Ex: A counter exceeds a value, software or hardware failure occurs, or an operation is completed.

Event Sources

- A *source is an object that generates an event. This occurs when the internal state of that object* changes in some way. Sources may generate more than one type of event.
- A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method. Here is the
- general form:
- public void add*TypeListener(TypeListener el)*

Event Sources:

Event Source	Description
Button	Generates action events when the button is pressed.
Checkbox	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu Item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scrollbar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listeners

- A *listener is an object that is notified when an event occurs. It has two major requirements.*
- First, it must have been registered with one or more sources to receive notifications about specific types of events.
- Second, it must implement methods to receive and process these notifications.

Event Classes

- The classes that represent events.
- At the root of the Java event class hierarchy is **EventObject, which is in java.util. It is the** superclass for all events. Its one constructor is shown here:
- EventObject(Object *src)*
- Here, *src is the object that generates this event.*
- **EventObject contains two methods:**
- **getSource( ) and toString( ).**
- **The getSource( ) method** returns the source of the event.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu Item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

TABLE 22-2 Event Source Examples

Event Listener Interfaces

- The delegation event model has two parts: sources and listeners.
- Listeners are created by implementing one or more of the interfaces defined by the **java.awt.event package.**
- When an event occurs, the event source invokes the appropriate method defined by the listener

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

TABLE 22-3 Commonly Used Event Listener Interfaces

Mouse event handlers

- The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.
- **public abstract void** mouseClicked(MouseEvent e);
- **public abstract void** mouseEntered(MouseEvent e);
- **public abstract void** mouseExited(MouseEvent e);
- **public abstract void** mousePressed(MouseEvent e);
- **public abstract void** mouseReleased(MouseEvent e);

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Applet implements MouseListener{
        public void init() {
addMouseListener(this);
}
 public void mouseClicked(MouseEvent e)  {       showStatus("Mouse Clicked");     }
public void mouseEntered(MouseEvent e) {        showStatus("Mouse Entered");      }
 public void mouseExited(MouseEvent e) {        showStatus("Mouse Exited");     }
 public void mousePressed(MouseEvent e) {        showStatus("Mouse Pressed");     }
  public void mouseReleased(MouseEvent e) {        showStatus("Mouse Released");     }

}
```

Java KeyListener Interface

- The **Java KeyListener is notified whenever you change the state of key.** It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package, and it has three methods.

Sr. no.	Method name	Description
1.	public abstract void keyPressed (KeyEvent e);	It is invoked when a key has been pressed.
2.	public abstract void keyReleased (KeyEvent e);	It is invoked when a key has been released.
3.	public abstract void keyTyped (KeyEvent e);	It is invoked when a key has been typed.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/* <applet code="SimpleKey" width=300 height=100> </applet> */
public class SimpleKey extends Applet implements KeyListener {
String msg = "";
int X = 10,Y = 20; // output coordinates
public void init() {
addKeyListener(this);
requestFocus(); // request input focus }
public void keyPressed(KeyEvent ke) {       showStatus("Key Down");          }
public void keyReleased(KeyEvent ke) {      showStatus("Key Up");                          }
public void keyTyped(KeyEvent ke) {         msg += ke.getKeyChar();//showStatus("Key
typed");
repaint(); } // Display key that is typed.
public void paint(Graphics g) { g.drawString(msg, X,Y); }
}
```

Adapter Classes

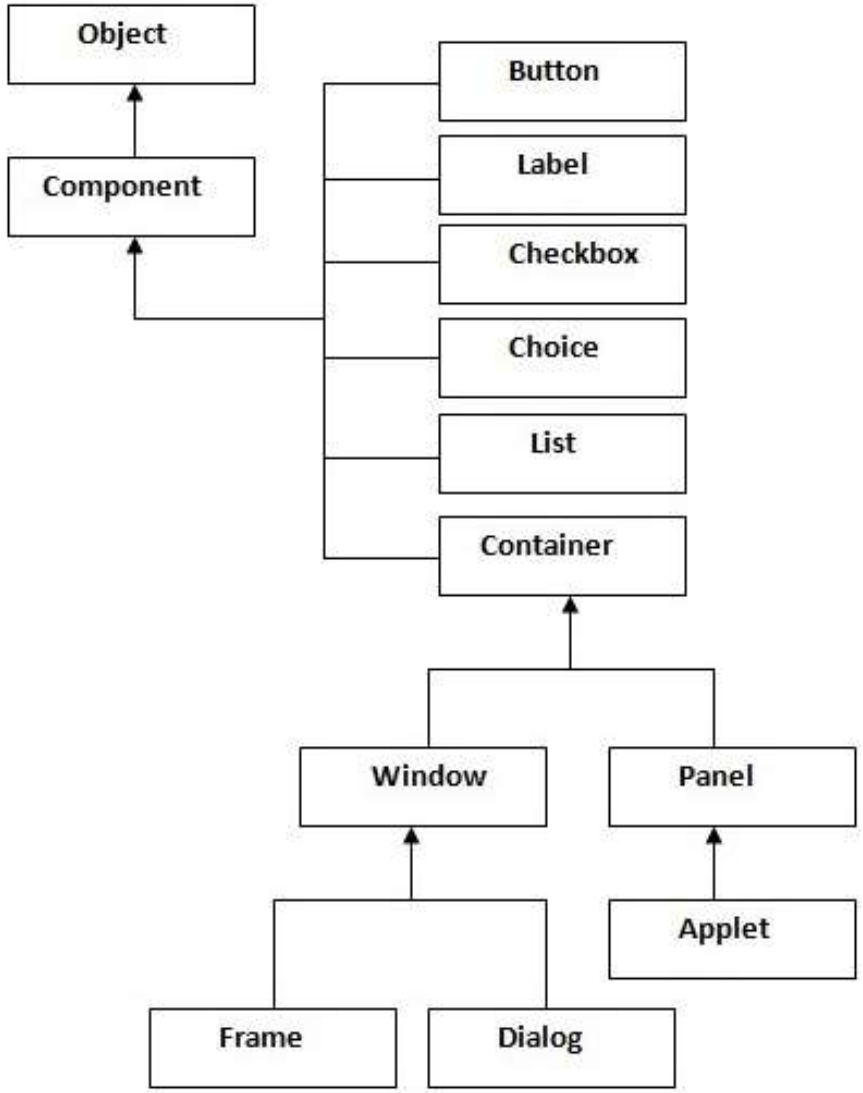
- An *adapter class, that can simplify the creation of event* handlers in certain situations.
- An adapter class provides an empty implementation of all methods in an event listener interface.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.

Introducing the AWT: Working with Windows, Graphics, and Text

- The AWT contains numerous classes and methods that allow you to create and manage windows.
- Although a common use of the AWT is in applets, it is also used to create stand-alone windows that run in a GUI environment, such as Windows.

AWT Classes

- The AWT classes are contained in the **java.awt package. It is one of Java's largest packages.**
- **Window Fundamentals**


Object
Component
Button
Label
Checkbox
Choice
List
Container
Window
Panel
Applet
Frame
Dialog

Using AWT Controls

- **Control Fundamentals**
- The AWT supports the following types of controls:
- • Labels
- • Push buttons
- • Check boxes
- • Choice lists
- • Lists
- • Scroll bars
- • Text editing
- These controls are subclasses of **Component.**

Adding and Removing Controls

- To include a control in a window, you must add it to the window. To do this, you must first
- create an instance of the desired control and then add it to a window by calling **add( )**
- Component add(Component *compObj)*
- Here, *compObj is an instance of the control that you want to add. A reference to compObj is*
- returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed.
- Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call **remove( ). This method is also defined by Container. It has**
- this general form:
- void remove(Component *obj)*

Buttons

- A push button is a component that contains a label and that generates an event when it is pressed.
- Push buttons are objects of type Button. Button defines these two **constructors:**
- Button() -> creates an empty button
- Button(String str) -> creates a button that contains str as a label

Methods:

- Void setLabel(String str) -> set its label
- String getLabel() -> retrieve its label

- Handling Buttons Each time a button is pressed, an action event is generated.
- This is sent to any listeners that previously registered. Each listener implements the ActionListener interface.
- That interface defines the actionPerformed() method, which is called when an event occurs.
- An ActionEvent object is supplied as the argument to this method.
- Button ob=new Button("hello");
- add(ob);
- ob.addActionListener(this);

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*<applet code="SimpleButtonDemo.class" width=200 height=300>
</applet>*/
public class SimpleButtonDemo extends Applet implements ActionListener
{
public void init()
{
Button b=new Button("ok");
add(b);
b.addActionListener(this);
}
public void actionPerformed(ActionEvent j)
{          showStatus("Button Clicked");    }          }
```

Using a TextField

- The **TextField class implements a single-line text-entry area, usually called an *edit control*.**
- Text fields allow the user to enter strings and to modify the text

Constructors

- TextField( )
- TextField(int *numChars)*
- TextField(String *str)*
- TextField(String *str, int numChars)*

Methods

- String getText( )
- void setText(String *str)*
- String getSelectedText( )
- void select(int *startIndex, int endIndex)*

Using a TextArea

- Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT include TextArea

Constructors

- TextArea( )
- TextArea(int *numLines, int numChars)*
- TextArea(String *str)*
- TextArea(String *str, int numLines, int numChars)*

- It supports the folloing methods
- **getText( ), setText( ),**
- **getSelectedText( ),**
- **select( ),**
- **isEditable( ), and setEditable( ) methods**
- void append(String *str)*
- void insert(String *str, int index)*
- void replaceRange(String *str, int startIndex, int endIndex)*

CheckBoxes

- A *check box is a control that is used to turn an option on or off. It consists of a small box that* can either contain a check mark or not. There is a label associated with each check box that describes what option the box represents. You change the state of a check box by clicking on it. Check boxes can be used individually or as part of a group. Check boxes are objects of the **Checkbox class.**
- **Checkbox supports these constructors:**
- Checkbox( )
- Checkbox(String *str)*
- Checkbox(String *str, boolean on)*
- Checkbox(String *str, boolean on, CheckboxGroup cbGroup)*
- Checkbox(String *str, CheckboxGroup cbGroup, boolean on)*

CheckBoxes

- boolean getState( ) : To retrieve the current state of a check box
- void setState(boolean *on) :* **To set its state**
- String getLabel( ) **You can obtain the current label associated with a check box**
- void setLabel(String *str)* **To set the label**
- Here, if *on is* ***true, the box is checked. If it is false, the box is cleared. The string passed in str*** becomes the new label associated with the invoking check box.

Handling Check Boxes

- Each time a check box is selected or deselected, an item event is generated. This is sent to any listeners that previously registered an interest in receiving item event notifications from that component.
- Each listener implements the **ItemListener interface. That interface defines** the **itemStateChanged(ItemEvent object) method.**
- It contains information about the event (for example, whether it was a selection or deselection).

CheckboxGroup

- It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are often called *radio buttons*,
- To create a set of mutually exclusive check boxes, you must first define the group to which they will belong and then specify that group when you construct the check boxes.
- **CheckboxGroup cbg= new CheckboxGroup();**

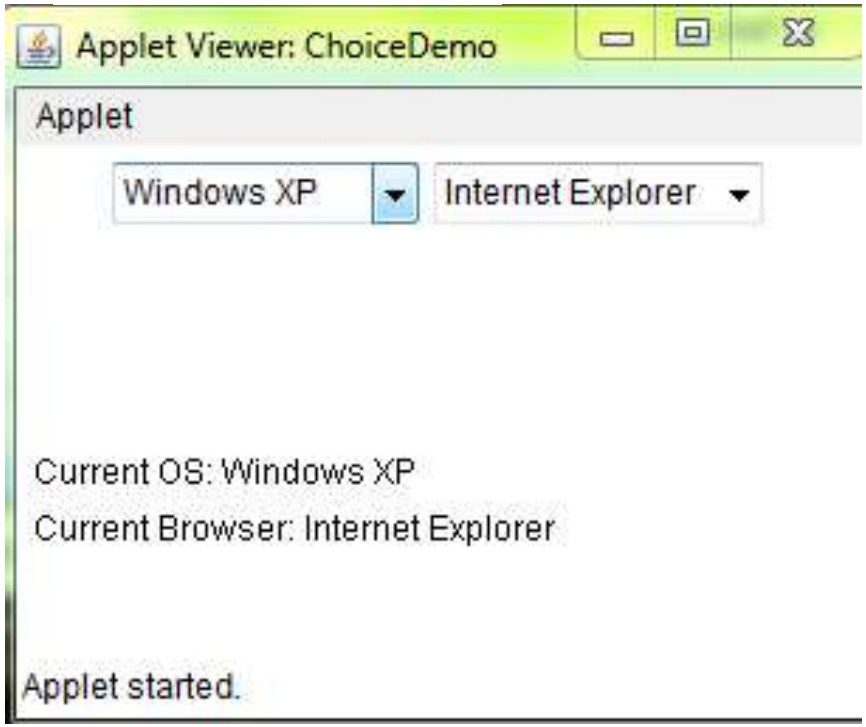
Checkbox winXP = new Checkbox("Windows XP", cbg, true);

- You can determine which check box in a group is currently selected by calling **getSelectedCheckbox( ).**
- **You can set a check box by calling setSelectedCheckbox(**Checkbox *which*).

Choice Controls

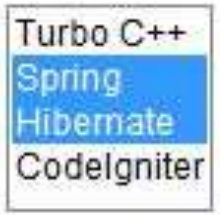
- The **Choice class is used to create a *pop-up list of items from which the user may choose*.**
- Thus, a **Choice control is a form of menu. When inactive, a Choice component takes up** only enough space to show the currently selected item.
- When the user clicks on it, the whole list of choices pops up, and a new selection can be made.
- To add a selection to the list, call **add( ). It has this general form:** void add(String *name)*
- To determine which item is currently selected, you may call either **getSelectedItem( )** or **getSelectedIndex( ). These methods are shown here:**
- String getSelectedItem( )
- int getSelectedIndex( )


Item 4
Item 1
Item 2
Item 3
Item 4
Item 5


Applet Viewer: ChoiceDemo
Applet
Windows XP
Internet Explorer
Current OS: Windows XP
Current Browser: Internet Explorer
Applet started.

Using Lists

- The **List class provides a compact, multiple-choice, scrolling selection list. Unlike the Choice object, which shows only the single selected item in the menu, a List object can be** constructed to show any number of choices in the visible window. It can also be created to allow multiple selections.
- **List provides these constructors:**
- List( )
- List(int *numRows)*
- List(int *numRows, boolean multipleSelect)*



To add a selection to the list, call **add( ).**

It has the following two forms:

- void add(String *name)*
- void add(String *name, int index)*
- String getSelectedItem( )
- int getSelectedIndex( )
- The **getSelectedItem( ) method returns a string containing the name of the item. If more than** one item is selected, or if no selection has yet been made, **null is returned.**
- **getSelectedIndex( )** returns an array containing the indexes of the currently selected items.
- int getItemCount( )
- void select(int *index)*


Applet Viewer: ListDemo
Applet
Windows XP
Windows Vista
Solaris
Mac OS
Internet Explorer
Firefox
Opera
Current OS: Windows XP Windows Vista
Current Browser: Opera
Applet started.

Layout Managers

- When you add a component to an applet or a container, the container uses its *layout manager* to decide where to put the component. Different LayoutManager classes use different rules to place components.

Layout Managers

- The LayoutManagers are used to arrange components in a particular manner.
- a layout manager automatically arranges your controls within a window
- The layout manager is set by the **setLayout( ) method. If no call to setLayout( ) is made, then the default layout** manager is used.
- Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.
- void setLayout(LayoutManager *layoutObj)*

- java.awt.LayoutManager is an interface. Five classes in the java.awt package implement it:
- FlowLayout
- BorderLayout
- CardLayout
- GridLayout
- GridBagLayout

FlowLayout

- Java has several predefined **LayoutManager classes**
- **FlowLayout is the default layout manager.**
- which, by default, is left to right, top to bottom. Therefore, by default, components are laid out line-by-line beginning at the upper-left corner.
- FlowLayout( )
- FlowLayout(int *how)*
- FlowLayout(int *how, int horz, int vert)*

BorderLayout

- A BorderLayout places objects in the North, South, East, West and center of an applet.
- this.setLayout(new BorderLayout());
- There's no centering, left alignment, or right alignment in a BorderLayout.
- However, you can add horizontal and vertical gaps between the areas.
- this.setLayout(new BorderLayout(5, 10));
- To add components to a BorderLayout include the name of the section you wish to add them to like this
- this.add(new Button("Start"),BorderLayout.SOUTH);

CardLayout

- The CardLayout is unusual. It breaks the applet into a deck of cards, each of which will normally have a panel with its own LayoutManager.
- Only one card appears on the screen at a time. You then flip between cards, each of which shows a different set of components.
- The common analogy is with an installer wizard. This might be used for a series of data input screens, where more input is needed than can comfortably be fit on one screen. Conversely, you can use a CardLayout for a slide show, where there's more data to be presented than will fit on one screen.

GridLayout

- A GridLayout specifies the number of rows and columns into which components will be placed. The applet is broken up into a table of equal sized cells.

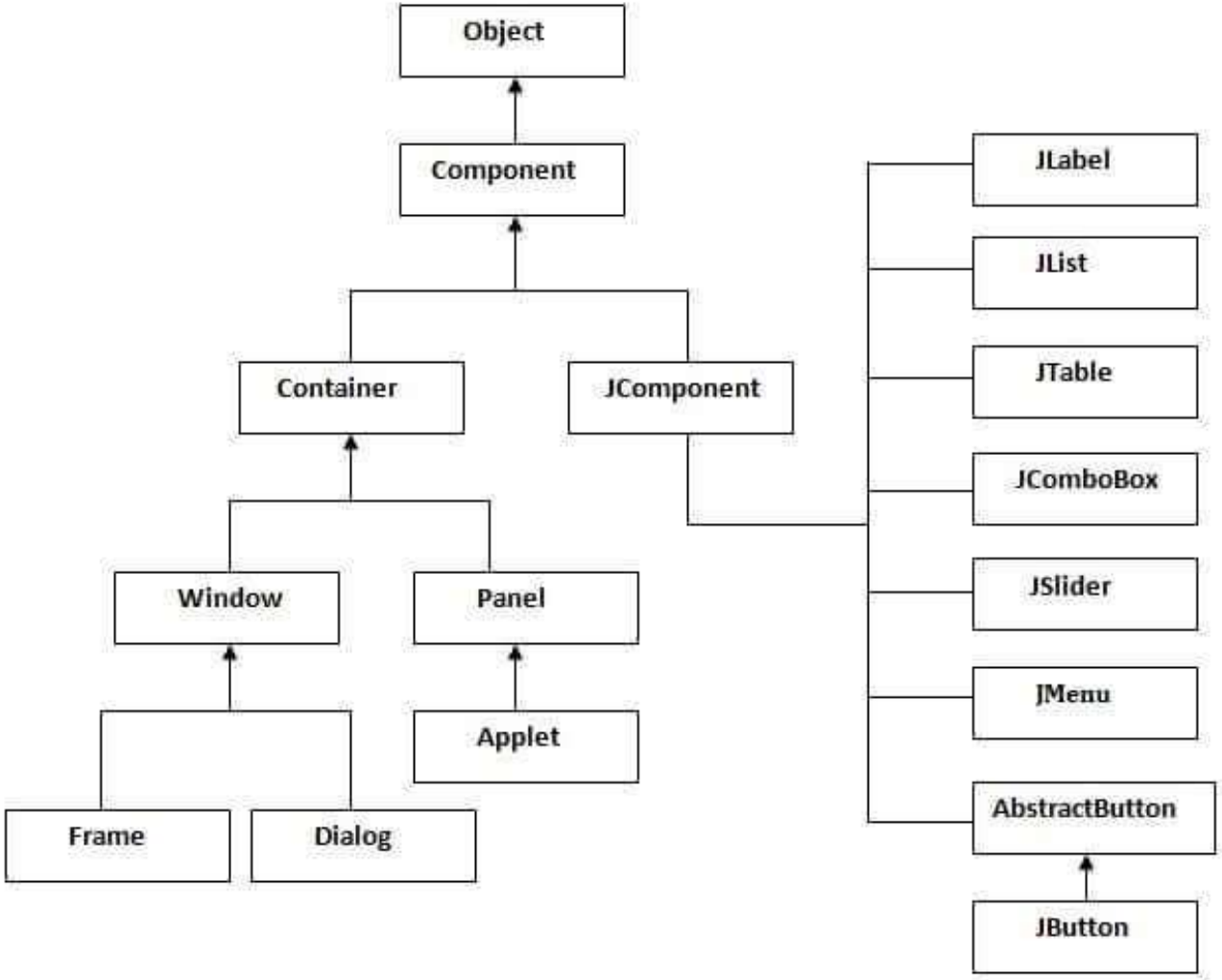
GridBagLayout

- The GridBagLayout provides a grid for components like GridLayout, but allows a single component element to occupy multiple cells of the grid. Each GridBagLayout uses a rectangular grid of cells, just like aGridLayout. However the cells are determined and shaped by the components placed in them rather than the components being shaped to fit the cells.

Java Swing

- Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT.
- Swing supplies several exciting additions, including tabbed panes, scroll panes, trees, and tables.
- Swing components are LightWeight Components and where as AWT Components are Heavyweight Components.

Hierarchy of Java Swing classes



Difference between AWT and Swing

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

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	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.	Swing **follows MVC**.

Java JLabel

- The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly.

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Methods

String getText()
void setText(String text)
Icon getIcon()

Java JButton

- The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```

Java JCheckBox

- The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on “.
- JCheckBox()
- JChechBox(String s)
- JCheckBox(String text, boolean selected)

Java JRadioButton

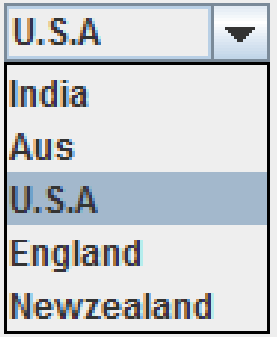
- The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.
- It should be added in ButtonGroup to select one radio button only.
- JRadioButton()
- JRadioButton(String s)
- JRadioButton(String s, boolean selected)

Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JComboBox

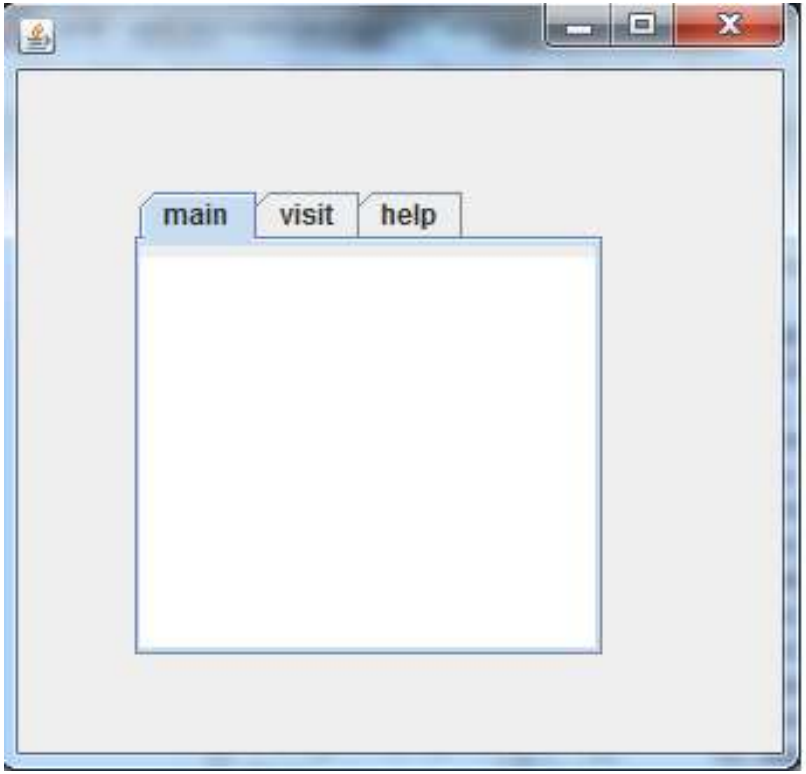
- The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu.
- JComboBox()
- JComboBox(Object[] items)



- void addItem(Object anObject)
- void removeItem(Object anObject)
- void removeAllItems()
- void setEditable(boolean b)

Java JTabbedPane

- The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon.
- JTabbedPane()

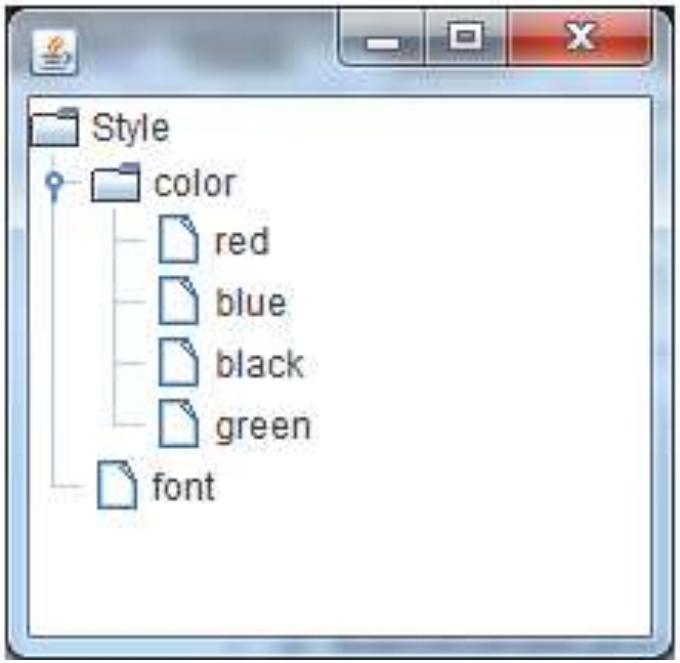


Java JScrollPane

- A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.
- JScrollPane()
- JScrollPane(Component)
- JScrollPane(int, int)
- JScrollPane(Component, int, int)

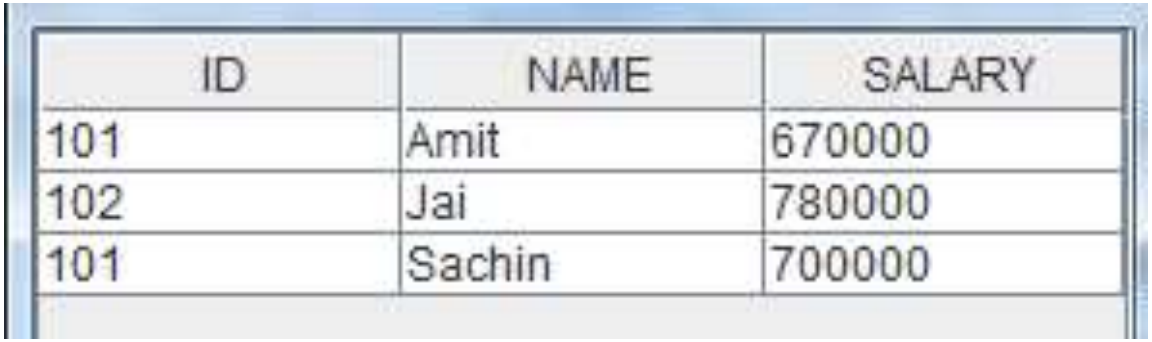
Java JTree

- The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree.



Java JTable

- The JTable class is used to display data in tabular form. It is composed of rows and columns
- JTable()
- JTable(Object[][] rows, Object[] columns)
- Creates a table with the specified data.



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

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();    }
}
```