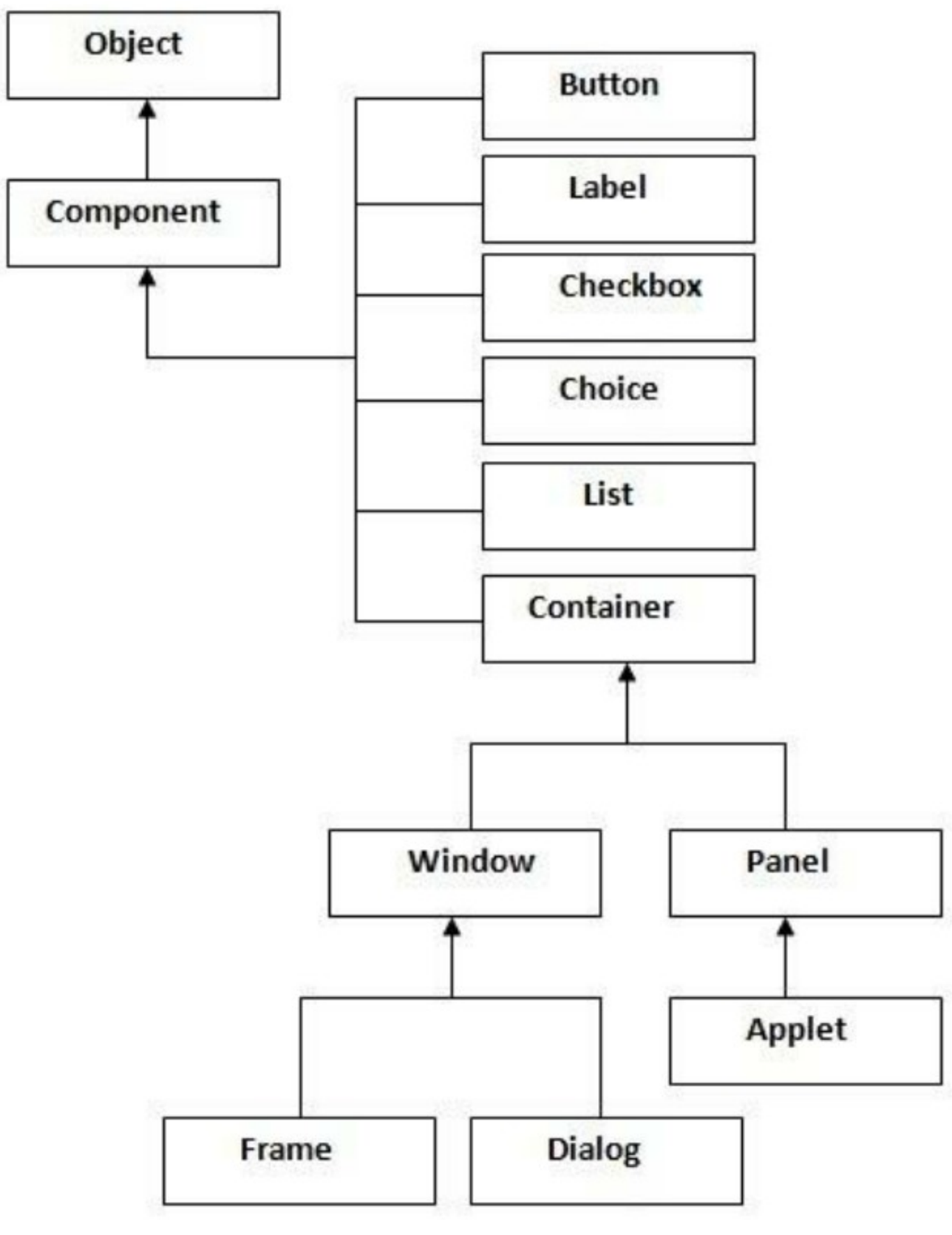
Abstract Window Toolkit: (AWT)

- **Java AWT** is an API to develop GUI or window-based application in java.
- Java AWT provides many of user interface objects are called **"Components"** of the Java AWT.
- Java AWT components are **platform-dependent** that means components are displayed according to the view of operating system.
- AWT is heavyweight. Its components use the resources of system.
- The **java.awt** package provides classes for AWT API such as **TextField**, **Label**, **TextArea**, **RadioButton**, **CheckBox**, **List** etc.

Java AWT Classes Hierarchy



Class	Description
Component	○ Component class is at the **top of AWT hierarchy**. ○ Component is an **abstract** class that encapsulates all attribute of **visual component**. ○ Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. **Examples:** buttons, checkboxes, list, scrollbars etc.

Container	○ Container is a component that can contain other components like **buttons**, **textfields**, **labels** etc. in a specific layout. ○ It is a **subclass** of **component** class. ○ It keeps track of components that are added to another component. ○ In **"Front to back"** order components are listed within the container. ○ The classes that extend Container class are known as container such as **Frame**, **Dialog** and **Panel**.
Window	○ Window is the container that has **no borders** and **menu bars**. ○ It is a rectangular area which is displayed on the screen. ○ You must use **frame**, **dialog** or **another window** for creating a window.
Panel	○ Panel is the container that doesn't contain **title bar and menu bars**. ○ It can have other components like **button**, **textfield** etc. ○ It is concrete **subclass** of **Container**. ○ The default **LayoutManager** of Panel is **Flow Layout**.
Frame	○ Frame is the container that has **title bar, menu bar, borders, and resizing corners.** ○ It is a **top-level** window & **subclass** of the **Window** class. ○ It can have other components like **button**, **textfield** etc.
Dialog	○ Dialog is a window that takes input from the user. ○ It is used to display **message, list of options** etc.

Basic Components available in AWT

Frame Window

- A Frame provides the "**main window**" for the GUI application, which has title bar (containing an icon, a title, minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area.
- When a frame object is created, by **default** it is **invisible**. You must call **setVisible()** method to make the frame appear on the screen.
- Then, you must set the size of the frame using **setSize()** or **setBound()** method.

Creating a Frame

- There are two ways to create a Frame. They are,
  1) **By Instantiating Frame class**
  2) **By extending Frame class**
- **Creating Frame Window by Instantiating Frame class**

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class FrameDemo
{
        FrameDemo()
        {
                Frame fm=new Frame();       //Creating a frame.
                Label lb = new Label("welcome to java ");    //Creating a label
```

```
        fm.add(lb);                         //adding label to the frame.
        fm.setSize(300, 300);               //setting frame size.
        fm.setVisible(true);                //set frame visibilty true.
        fm.setTitle("Frame Example");

    //To close the window
        fm.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                                }
                        });
        }
        public static void main(String args[])
        {
                FrameDemo ta = new FrameDemo();
        }
}
```

- **Creating Frame window by extending Frame class**

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
class DemoFrameExample extends Frame
{
        DemoFrameExample()
        {
                super("Frame Example");
                Label lb = new Label("Welcome to java world");
                lb.setBounds(30,100,150,30);  // setting button position
                add(lb);  //adding button into frame
                setSize(300,300);  //frame size 300 width and 300 height
                setLayout(null);  //no layout manager
                setVisible(true);  //now frame will be visible, by default not visible

                //To close the window
                addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                        }
                });
        }
        public static void main(String args[])
        {
                DemoFrameExample f = new DemoFrameExample ();
        }
}
```

Creating a Frame Window in Applet

- **The following steps used to create a frame in applet :**
  - Create a subclass of **Frame.**
  - Override any of the standard window methods such as **init(), start(), stop() and paint()**.
  - Implement the **windowClosing()** method of the **windowlistener** interface, calling **setVisible(false)** when the **window** is **closed**.
  - Once you have defined a **Frame subclass**, you can create an object of that class. But it will not be initially **visible**.
  - When created, the window is given a **default height and width**.
  - You can set the size of the window explicitly by calling the **setSize()** method.

Example:

```
import java.awt.*;
import java.awt.Label;
import java.awt.event.*;
import java.applet.*;

public class DemoFrameApplet extends Applet
{
        Frame f;
        public void init()
        {
                f = new Frame("A Frame Window");
                f.setSize(300, 300);
                Label lb = new Label("You are in a frame ");
                f.add(lb);
                f.setVisible(true);
}
public void start()
{
        f.setVisible(true);
}
public void stop()
{
        f.setVisible(false);
}
        public void paint(Graphics g)
        {
                g.drawString("** You are in Applet **", 15, 30);
        }
}
```

Canvas

- Canvas component represents a **rectangular area** where application can **draw** something or can **receive** inputs created by user.
- Drawing is not implemented on the canvas itself, but on the **Graphics object** provided by the canvas.

- The Canvas is a section of a window to draw graphics or display images.

Constructors of Canvas

Constructor	Description
Canvas()	This constructor creates instance of a Canvas.
Canvas(GraphicsConfiguration config)	This constructor creates instance of a Canvas with the given object of Graphics Configuration.

Methods of Canvas

Method	Description
Paint(Graphics g)	Paint the Canvas.
Update(Graphics g)	Update the Canvas.

Example:

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class DemoCanvas extends Frame
{
public DemoCanvas()
{
        super("Canvas Example");
        Canvascls cx = new Canvascls();
        cx.setSize(125, 100);
        cx.setBackground(Color.black);
        add(cx,"North");
        setSize(300, 200);
        setVisible(true);

        addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent we){
                        System.exit(0);
                }
        });
        }
        public static void main(String args[])
        {
                DemoCanvas dc = new DemoCanvas();
        }
}
class Canvascls extends Canvas
{
        public void paint(Graphics g)
```

```
        {
                g.setColor(Color.blue);
                g.fillOval(75, 10, 150, 75);
        }
}
```

Working With Graphics: AWT Controls

- AWT provides many ready- made and reusable GUI components.
- All the AWT controls inherit from **java.awt.*** where * represents all class like label, button etc..
- The frequently used are: Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice, as given below table.

Component	Description
Label	○ It is used to display information that should **not be modified** by users.
TextField	○ It is useful for obtaining information from users. It considers only **single line** input.
TextArea	○ It is a text component that allows for the editing of a **multiple lines** of text.
Button	○ It creates a **labeled** button.
CheckBox	○ It is a graphical component that can be in either an **on** (true) or **off** (false) state.
CheckboxGroup	○ It is used to **group** the set of checkbox.
List	○ It presents the user with a **scrolling list of text items**.
Choice	○ It is used to show **drop-down menu** of choices. Selected choice is shown on the **top** of the menu.
ScrollBar	○ A Scrollbar control represents a scroll bar component in order to enable user to select from **range of values**.
Canvas	○ It represents a **rectangular area** where application can draw something or can receive inputs created by user.

Labels

- It is used to display information that should **not be modified** by users.

Constructor of Label

Constructor	Description
Label()	Create an empty label.
Label(String)	Create a new label with the specified string of text, which is left justified.
Label(String, int)	Create a label with specified text and alignment indicated by the **int** Argument: **Label.Right, Label.Left, Label.Center**

Method of Label

Method	Description
int getAlignment()	Gets the current alignment of the label.
String getText()	Gets the text of the label.
void setAlignment(int align)	Set the alignment for the label to specified alignment. Possible values are **Label.LEFT, Label.RIGHT and Label.CENTER.**
void setText(String label)	Sets the text for the label to the specified text.

Syntax:

```
Label L1 = new Label("UserName");
add(L1);
```

TextField

- It is useful for obtaining information from users. It considers only **single line** input.

Constructor of TextField

Constructor	Description
TextField()	Create a default text field.
TextField(int numChar)	Create a new empty text field with the specified number of character.
TextField(String text)	Create a new text field initialized with the specified text.
TextField(String text, int numChar)	Create a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

Method of TextField

Method	Description
getText()	Return the text this text field contains (as a string).
setText()	Puts the given text string into the field.
getColumns()	Returns the width of this text field.
select(int, int)	Select the text between the two integer positions (position start from 0).
getSelectedText( )	Get the currently selected text.
isEditable()	Return true or false based on whether the text is editable.
setEditable(boolean)	True (the default) enables text to be edited, False not editable.

Syntax:

```
TextField t1 = new TextField (12);
Add(t1);
```

TextArea

- It is a text component that allows for the editing of a **multiple lines** of text.
- We can set **number of rows** and **columns** of the **TextArea.**
- The text in a **TextArea** appears left justified, and the justification is not **customizable**.

Constructors of TextArea

Constructor	Description
TextArea()	Creates an empty text area.
TextArea(int, int)	Creates an empty text area with the given number of rows and columns.
TextArea(String)	Creates a text area displaying the given string.
TextArea(String, int, int)	Creates a text area displaying the given string and with the given number of rows and columns.
TextArea(String, int , int , int)	Creates a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified. **SCROLLBARS_BOTH, SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_NONE** and **SCROLLBARS_VERTICAL_ONLY.**

Method of TextArea

Method	Description
getColumns()	Returns the number of columns of the text area.
setColumns(int columns)	Sets the number of columns for the text area.
getRows()	Returns the number of rows in the text area.
setRows(int rows)	Sets the number of rows for the text area.
insert(String text, int pos)	Inserts the specified text at the specified position in the text area.
append(String str)	Appends the given text at the end of the text area's current text.
replaceRange (String text, int start, int end)	Replaces text between the indicated start and end positions with the specified replacement text.

Syntax :

```
TextArea T2 = new TextArea (10,20);
add(T2);
```

Button (Push Button)

- The Button component is rectangular button that has label and generate event when pressed.

Constructors of Button

Constructor	Description
Button()	Creates a button with an empty string for its label.
Button(String text)	Creates a button with the given string as a label.

Method of Button

Method	Description
getLabel()	Get the label of the Button.
setLabel(string text)	Set the label of the Button with given text.
setEnable(boolean)	Enable or disable this Button. Disabled Button cannot be clicked.

Syntax :

```
Button  b = new Button (“Hello”);
add (b);
```

CheckBox

- The Checkbox class is used to display checkbox controls.
- The Checkbox has a label to indicate its meaning. Checkbox component is **toggle box** that can be either **selected** or **deselected** indicating presence or absence of choice.
- If a Checkbox object is not in a **CheckboxGroup** object, it is implemented as a **simple checkbox**.
- If a Checkbox object is with a **CheckboxGroup** object, it is implemented as a **radio button**.

Constructors of Checkbox

Constructor	Description
Checkbox()	Creates a check box with an empty string for its label.
Checkbox(String text)	Creates a checkbox with the given string as a label.
Checkbox(String text, boolean state)	Creates a check box with the given string as a label and sets the specified state (Selected/Deselected by True/False).
Checkbox(String text, CheckboxGroup group, boolean state)	Creates a check box with the given string as a label, in the specified check box group, and set to the specified state. The null is used for a group argument. Only radio button have groups.

Method of Checkbox

Method	Description
getLabel()	Returns the label of the check box.
setLabel(String text)	Sets the check box's label to by given text.
getState()	Returns true or false, based on whether the check box is selected or not.
setState(boolean)	Sets the state of the check box to the specified state (True/False).
setCheckboxGroup(CheckboxGroup grp)	Sets the check box's group to the specified check box group.
getCheckboxGroup()	Return the check box's group.

Syntax :

```
Checkbox C = new Checkbox ("Java");
add (C);
```

CheckboxGroup

- To create a group of checkboxes, you use the **CheckboxGroup** class.
- The CheckboxGroup class is used with the Checkbox class to implement **radio buttons**.
- All Checkbox that are associated with a **CheckboxGroup** are treated as a single radio button.
- It allows only one button in group to be set at a time.

Constructors of CheckboxGroup

Constructor	Description
CheckboxGroup()	Creates a new instance of CheckboxGroup.

Method of CheckboxGroup

Method	Description
getSelectedCheckbox()	Returns the current choice from the check box group.
setSelectedCheckbox(Checkbox box)	The method of CheckboxGroup is used to make one of the box selected among all the check boxes.

Choice

- It is used to show **drop-down list** of choices. Selected choice is shown on the **top** of the menu
- Form this lists a single choice can be selected, similar to a group of checkboxes.

Constructors of Choice

Constructor	Description
Choice()	Creates a new choice List.

Method of ChoiceList

Method	Description
add(String text)	Add an item to the choice list.
getItem(int index)	Returns the string at the specified index from the Choice list.
getItemCount()	Returns the number of items in the choice list.
getSelectedIndex()	Return the index of the currently selected item.
getSelectedItem()	Returns the currently selected item as a string.
insert(String item, int index)	Inserts the item into the choice list at the specified position.
remove(int position)	Removes an item from the choice list at the specified position.
remove(String item)	Removes the first occurrence of item from the Choice list.

removeAll()	Removes all items from the choice list.
select(int)	Selects the item at the given position.
select(String)	Selects the item with the given string.

Lists

- The List component is a Scrolling list of string from which one or more string can be selected.
- The List class is use for creating single and multiple selection list.
- The List class provides facility to set display size (number of elements) and also allow selecting multiple items from the list.

Constructors of List

Constructor	Description
List()	Creates an empty scrolling list.
List(int rows)	Creates a new scrolling list initialized with the specified number of visible lines.
List(int rows, boolean multipleMode)	Creates a scrolling list with the given number of visible lines on the screen. The Boolean indicates whether this list enables multiple selection (true) or not (false).

Method of List

Method	Description
add()	Used to build the scrolling list.
add(String item)	Adds the specified item to the end of scrolling list.
add(String item, int index)	Adds the specified item to the scrolling list at the position indicated by the index.
getItem(int index)	Returns the item at given index.
getItemCount()	Returns the number of items in the list.
String[] getItems()	Gets the items in the list.
getSelectedIndex()	Gets the index of the selected item on the list (used for lists that allow only single selections).
int[] getSelectedIndexes()	Returns a integer array of selected indexes positions (used for lists that allow multiple selections).
getSelectedItem()	Returns the currently selected item on the list.
String[] getSelectedItems()	Returns a string array of selected items on the list. (used for lists that allow multiple selections).
deselect(int index)	Deselects the item at the specified index.
remove(int position)	Removes an item from the list at the specified position.
remove(String item)	Removes the first occurrence of item from the list.

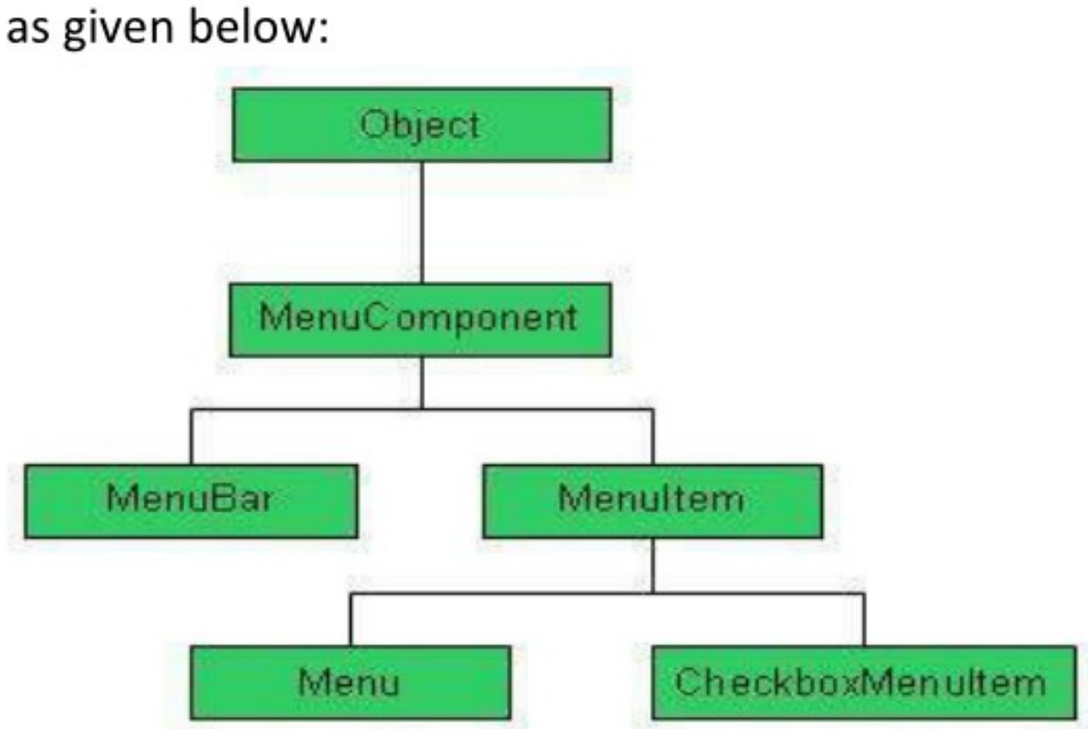
removeAll()	Removes all items from the list.
select(int index)	Select the item at the given index.
replaceItem(String newValue, int index)	Replaces the item at the specified index in the list with the new string.
isIndexSelected(int index)	Determines if the specified item in this scrolling list is selected or not.
isMultipleMode()	Determines whether this list allows multiple selections or not.
setMultipleMode(boolean b)	Sets the flag that determines whether this list allows multiple selections.

Syntax:

```
List L1 = new List(2,true);
L1.add("Java");
L1.add("DWSL");
add(L1);
```

Menu and Menu Bar

- The abstract class **MenuComponent** is the super class of all menu-related components.
- Menu can be added only to a menu container.
- Menu Component hierarchy as given below:



- The **MenuBar** class provides menu bar bound to a **frame** and is **platform specific**.
- **MenuBar** contains one or more **Menu** objects.
- **Menu** are used to **display and control** menu items.
- Each **Menu object** contains a list of **MenuItem** objects.
- **An ActionListener** can be added to a **MenuItem** object**.**
- First create a menu bar by creating an **instance of menuBar**.

Constructor of Menu and Menubar

Constructor	Description
MenuBar()	Creates a new menu bar.

Menu()	To create a default menu.
Menu(String str)	str specifies the name of the Menu selection
Menu(String str, Boolean flag)	str specifies the name of the Menu selection flag represents the popup menu if set true.
MenuItem()	To create a default MenuItem.
MenuItem(String str)	Str is the name shown in the Menu.
MenuItem(String str, MenuShortcut key)	Key is the short cut key for that Menu Item.

CheckboxMenuItem

- **CheckboxMenuItem** is checkable menu item, which provides selection (on or off) listed in menus.
- **CheckboxMenuItem** can be controlled by the **ItemListener** interface.

Constructor of CheckboxMenuItem

Constructor	Description
CheckboxMenuItem()	To create a default CheckBoxMenuItem.
CheckboxMenuItem(String str)	Str is the name shown in the menu.
CheckboxMenuItem(String str, boolean flag)	Flag can be set on for the Item to be checkable.

Method of Menu

Method	Description
setEnabled(boolean flag)	To enable or disable menu item
isEnabled()	To retrieve the status of the menu item.
setLabel(String str)	To change the name of the menu item.
String getLabel()	To retrieve the current name of the menu item.
boolean getState()	Return true if the item is checked otherwise false.
setState(Boolean flag)	To check an item, pass **true** and to clear an item, pass **false**.
add (MenuItem m)	Used to add menu item in Menu.
add(Menu menu)	Add the specified menu to the menu bar.
getItem(int index)	Return the menu item at the given index
getItemCount()	Return the number of item inside the menu.
remove(MenuComponent item)	Remove menu item from menu.
remove(int index), removeAll()	Remove the menu located at the specified index and remove all menu from the menu bar.

Example:

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class MenuDemo extends Frame
{
        MenuBar mbar;
        Menu m;
        MenuDemo()
        {
                super("Menu Demo");
                setSize(400,400);    // Set size to the frame
                setLayout(new FlowLayout());   // Set the layout
                setVisible(true);

                mbar = new MenuBar();

                m = new Menu("File");
                m.add(new MenuItem("New"));
                m.add(new MenuItem("Open"));
                m.add(new MenuItem("Save"));
                m.add(new MenuItem("Save As"));
                m.add(new MenuItem("Print"));

                m.addSeparator();
                m.add(new MenuItem("Quit"));

                mbar.add(m);

                m = new Menu("Help");
                m.add(new MenuItem("Help"));
                m.addSeparator();
                m.add(new MenuItem("About"));

                mbar.add(m);

                setMenuBar(mbar);

                addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                        System.exit(0);
                        }
                });
        }
```

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

Layout Managers

- The **Layout Mangers** are used to **arrange components** in particular manner.
- **Layout Manager** is an interface that is implemented by all the classes of layout managers.
- The layout manager set by the SetLayout() method. If we don't use this method then **default layout manager** is used.
- There are following classes that represents the layout managers:
  1) java.awt.BorderLayout
  2) java.awt.FlowLayout
  3) java.awt.GridLayout
  4) java.awt.CardLayout

Layout Manager Classes

- Some possible Layout Managers are listed in the table below:

Layout Manager	Description	Constants
BorderLayout	Arranges components to the top, bottom, left, right, and center of a container.	NORTH, SOUTH, EAST, WEST, CENTER –positions in the container.
FlowLayout	It is used to arrange the components in a line, one after another (in a flow) from left to right.	LEFT, CENTER, RIGHT – these tell it how to align the components in each row.
GridLayout	Divide the container into equal-sized rectangles and arrange each component into one of these cells.	GridLayout(int rows, int columns) To specify the number of rows and columns in the grid.
CardLayout	It manages the components in such a way that only one component is visible at a time.	**CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

FlowLayout

- The **FlowLayout** is used to arrange the components in a line, one after another (in a flow).
- It is the default layout of applet or panel.
- **FlowLayout** arranges swing component from left to right until there's no more space available.
- Then it begins a new row below it and moves from left to right again.
- **LEFT, CENTER, RIGHT** – these tell it how to align the components in each row.
- It can be used like **FlowLayout.LEFT.**

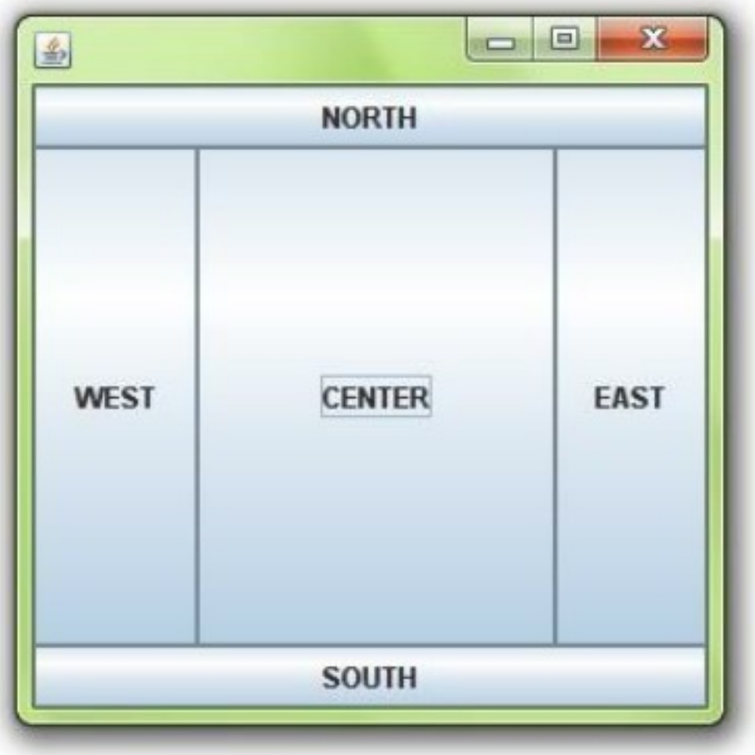
Constructor of FlowLayout

Constructor	Description
FlowLayout()	Create a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align)	Create a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align, int hgap, int vgap)	Create a flow layout with the given alignment and given horizontal and vertical gap.

Border layout

- The **BorderLayout** is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only.
- It is the default layout of frame or window.
- The BorderLayout provides five constants for each region: **NORTH, SOUTH, EAST, WEST, CENTER.**
- These constants are used like **BorderLayout.NORTH.**
- While adding the component these constants are used by using following form of **add()** method.

add (Component compObj**, Object** region**);**



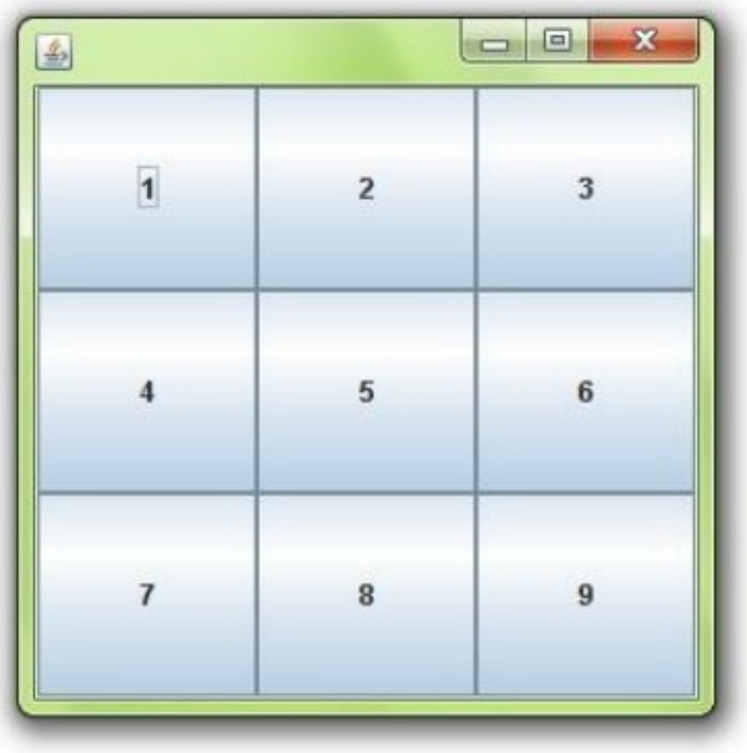
Constructors of BorderLayout

Constructor	Description
BorderLayout()	Creates a border layout with no gaps between the components.
BorderLayout(int hgap, int vgap)	Creates a border layout with the given horizontal and vertical gaps between the components.

Grid Layout

- The **GridLayout** is used to arrange the components in rectangular grid inside the container.
- One component is displayed in each rectangle.

- Components added to the container with the **GridLayout** manager are arranged in order from **left to right**.



Constructors of GridLayout

Constructor	Description
GridLayout ()	Creates a grid layout with **one column per component** in a row.
GridLayout (int rows, int columns)	Creates a grid layout with the given **rows** and **columns** but **no gaps** between the components.
GridLayout (int rows, int columns, int hgap, int vgap)	Creates a grid layout with the given rows and columns along with given **horizontal and vertical gaps**.

CardLayout

- The CardLayout manager treats each component as a card. Only **one card/component visible** at a time. So it is known as **CardLayout**.

Constructors of CardLayout

Constructor	Description
CardLayout()	Creates a card layout with zero horizontal and vertical gap.
CardLayout(int hgap, int vgap)	Creates a card layout with the given horizontal and vertical gap.

Methods of the CardLayout

Methods	Description
void first(Container a)	It is used to flip to the first card of the given container.
void last(Container a)	It is used to flip to the last card of the given container.
void next(Container a)	It is used to flip to the next card of the given container.
void previous(Container a)	It is used to flip to the previous card of the given container.
void show(Container a, String cardName)	It is used to flip to the specified card with the given name.

- **'a'** is a reference to the container (usually a panel) that holds the cards, and **cardName** is the name of a card.

Graphical User Interface with Swing

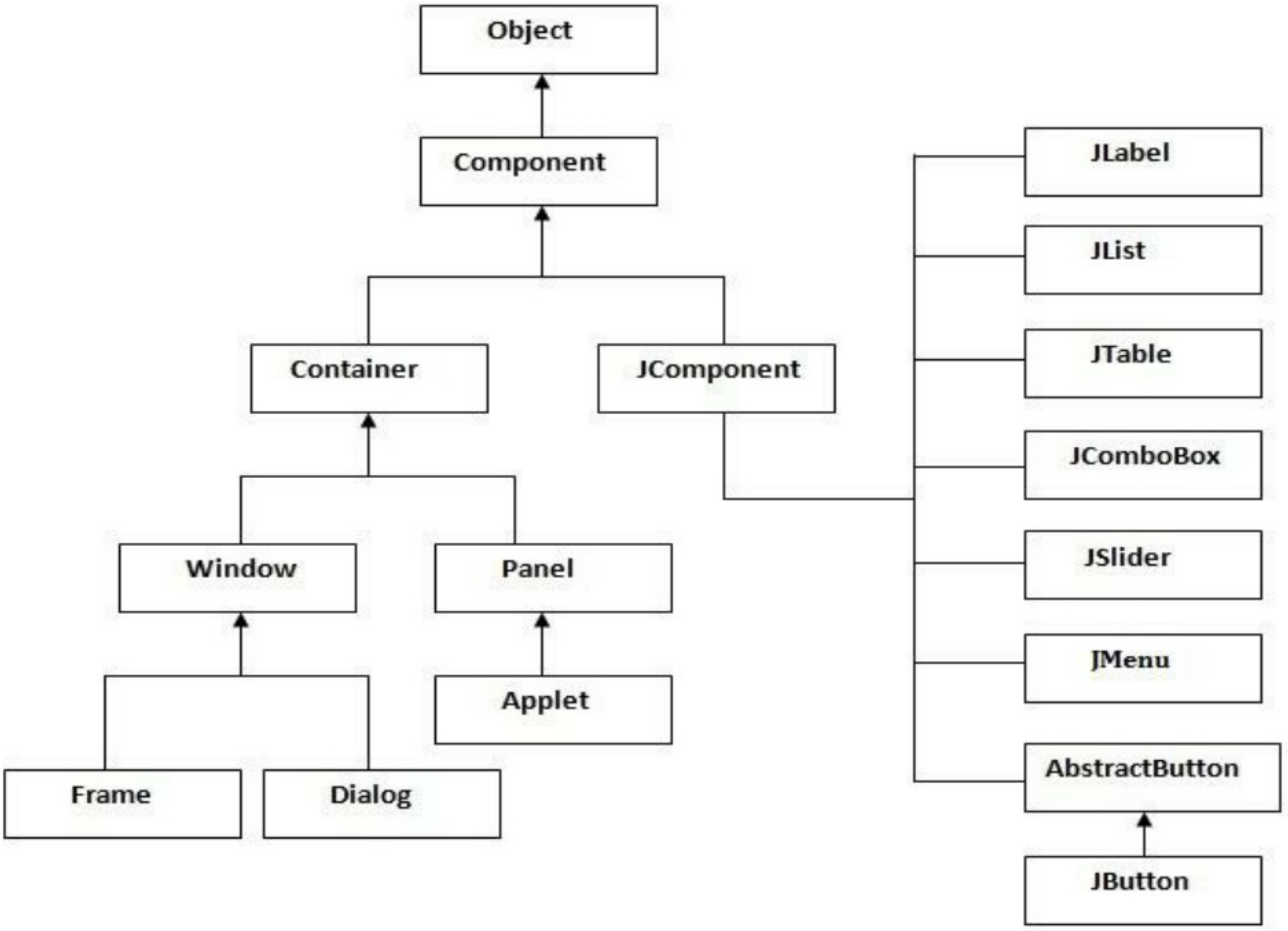
- It is part of **Java Foundation Classes** (JFC) that is used to create window-based applications. Java Swing provides **platform-independent** and **lightweight** components.
- The Swing components are **100% pure java**. This means that they don't depend on the native window implementation to support them.
- Swing provides programmer the facility to change the **look and feel of components** being displayed on any system. This is called **'plaf' (pluggable look and feel).**
  - **Look** refers to the appearance
  - **Feel** represents how the user can interact with the component.
- **There are 3 types of Look & Feel available in Swing :**
  - Metal Look & Feel
  - Motif Look & Feel
  - Windows Look & Feel
- By default, Swing programmer use '**metal look and feel'**.
- The **javax.swing** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

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

Java AWT	Java Swing
AWT stands for **Abstract windows toolkit**.	Swing is also called as **JFC**.
AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
AWT components are **heavyweight**.	Swing components are **lightweight**.
AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
AWT components require **java.awt** package.	Swing components require **javax.swing** package.
AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
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**.

Hierarchy of Java Swing classes



- JFrame, JPanel, JWindow is the Swing's version of Frame, Panel and Windows respectively.

JApplet

- The JApplet class extends the **Applet** class.
- We can use **JApplet** that can have all the controls of swing.

JLabel

- A JLabel is a single line label similar to **java.awt.Label**.

Constructor of JLabel

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

Methods of JLabel

Constructor	Description
getText()	Returns the string that associate with the label.
setText(String text)	Set the specified text to label.
getIcon()	Returns the image that displays the label.
setIcon(Icon image)	Set the image to the label.
getHorizontalAlignment()	Returns the alignment of the label along the X axis.
setHorizontalAlignment(int alignment)	Sets the alignment of the label along the X axis
getVerticalAlignment()	Returns the alignment of the label along the Y axis.
setVerticalAlignment(int alignment)	Sets the alignment of the label along the Y axis.

JTextField

- The class **JTextField** is a component which allow the editing of a single line of text.

Constructor of JLabel

Constructor	Description
JTextField()	Creates a default text field.

JTextField(int columns)	Creates a new empty text field with the specified number of character.
JTextField(String text)	Creates a new text field initialized with the specified text.
JTextField(String text, int columns)	Creates a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

Methods of JTextField

Method	Description
getText()	Return the text this text field contains (as a string).
setText()	Puts the given text string into the field.
getColumns()	Returns the width of this text field.
getSelectedText( )	Get the currently selected text.

JButton

- A JButton can be used in a GUI just like a java.awt.Button. It behaves like an AWT Button, notifying ActionListener list elements when pushed.

Constructors of JButton

Constructor	Description
JButton()	Creates a button with an empty string for its label.
JButton(String text)	Creates a button with the given string as a label.
JButton(Icon image)	Creates a button with an specified image.
JButton(String text, Icon icon)	Creates a button with specified text and an image.

Methods of JButton

Method	Description
getLabel()	Get the label of the Button.
setLabel(string text)	Set the label of the Button with given text.
setEnable(boolean)	Enable or disable this Button. Disabled Button cannot be clicked.

JCheckBox

- A JCheckBox is similar to an AWT Checkbox that is not in a CheckboxGroup.
- Purpose of the check box - an item that can be selected or deselected, and which displays its state to the user.

Constructors of the checkbox

Constructor	Description
JCheckbox()	Creates a check box with no image and text
JCheckbox(String text)	Creates a checkbox with the given string as a label.
JCheckBox(Icon image)	Creates a check box with an image.
JCheckbox(String text, boolean state)	Creates a check box with the given string as a label and sets the specified state (Selected/Deselected by True/False).
JCheckBox(String text, Icon image)	Creates a checkbox with specified text and image
Checkbox(String text, Icon image, boolean state)	Creates a check box with the given string as a label, image, and set to the specified state.

Methods of JCheckBox

Method	Description
getTextl()	Returns the label of the check box.
setText(String text)	Sets the check box's label by given text.
isSelected()	Check whether checkbox is selected or not.

JRadioButton

- The JRadioButton class is used to create a radio button.
- It is used to choose one option from multiple options.
- It should be added in **Group** to select one radio button only by using **ButtonGroup** class.
- By default radio button is not selected.

Constructors of JRadioButton

Constructor	Description
JRadioButton()	Creates radio button with no text.
JRadioButton(Icon image)	Creates radio button with the specified image but no text
JRadioButton(String text)	Creates radio button with the specified text.
JRadioButton(String text, Icon image)	Creates radio button with the specified image and text.
JRadioButton(Icon image, boolean selected)	Creates a radio button with the specified image and selection state, but no text.
JRadioButton(String text, boolean selected)	Creates a radio button with the specified text and selection state.
JRadioButton(String text, Icon image, boolean selected)	Creates a radio button with the specified text, image, and selection state.

- Methods of JRadioButton is same as JCheckBox.

JComboBox

- The JComboBox class is used to create drop-down list. At a time only one item can be selected from the item list.
- The **JComboBox** offers an editable option.

Constructors of JcomboBox

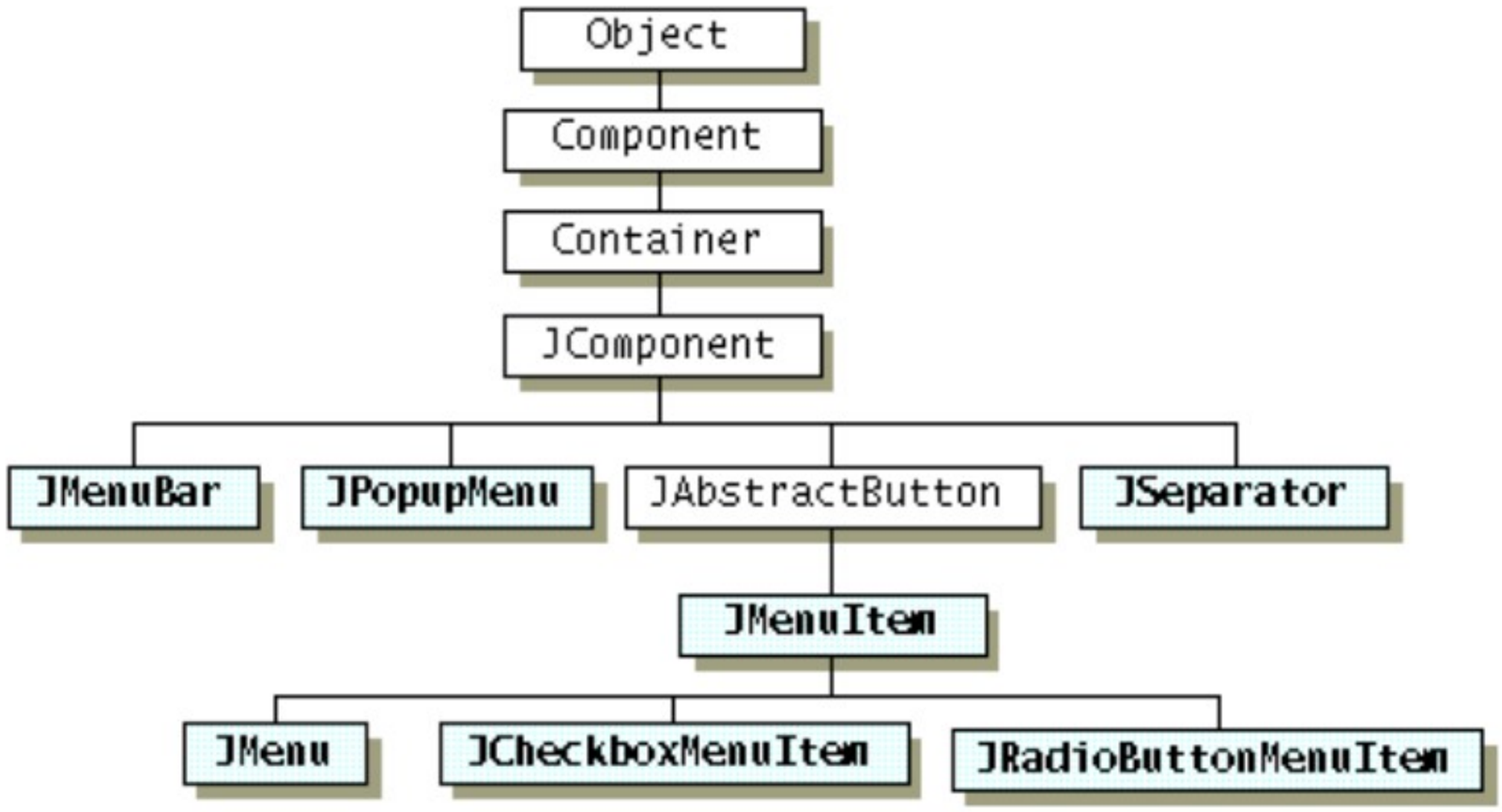
Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that takes its items from an existing array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Methods of JcomboBox

Method	Description
addItem(Object object)	It is used to add an item to the item list.
setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
removeItem(Object object)	It is used to delete an item from the item list.
removeAllItems()	It is used to remove all the items from the list.
getItemAt(int index)	Returns the list item at the specified index.
getItemCount()	Returns the number of items in the list.
getSelectedIndex()	Returns the index of the selected item.
setSelectedIndex(int index)	Selects the item at given index.
getSelectedItem()	Returns the selected item.
setSelectedItem(Object object)	Set the selected item specified as object.
isEditable()	Returns true if the JComboBox is editable.
addActionListener(ActionListener a)	It is used to add the ActionListener.
addItemListener(ItemListener i)	It is used to add the ItemListener.

Menus

- Every top-level window has a **menu bar** associated with it.
- This **menu bar** consists of various menu choices available to the end user.
- Further each choice contains list of options which is called **drop down menus**.
- **Menu** and **MenuItem** controls are subclass of **MenuComponent** class.
- **Menu hierarchy is as given below:**



JMenuBar

Constructor of JMenuBar

Constructor	Description
JMenuBar()	Creates a new menu bar.

Methods of JMenuBar

Method	Description
JMenu add(JMenu c)	Appends the specified menu to the end of the menu bar.
getComponentIndex(Component c)	Returns the index of the specified component.
Insets getMargin()	Returns the margin between the menubar's border and its menus.
void setMargin(Insets m)	Sets the margin between the menubar's border and its menus.
JMenu getMenu(int index)	Returns the menu at the specified position in the menu bar.
int getMenuCount()	Returns the number of items in the menu bar.
boolean isSelected()	Returns true if the menu bar currently has a component selected.

JMenuItem

- The **JMenuItem** class represents the actual item in a menu.
- All items in a menu should derive from class **JMenuItem**, or one of its subclasses.

Constructor of JMenuItem

Constructor	Description
JMenuItem()	Creates a JMenuItem with no set text or image.
JMenuItem(Icon image)	Creates a JMenuItem with the specified image.
JMenuItem(String text)	Creates a JMenuItem with the specified text.
JMenuItem(String text, Icon image)	Creates a JMenuItem with the specified text and icon.
JMenuItem(String text, int mnemonic)	Creates a JMenuItem with the specified text and keyboard mnemonic.

Methods of JMenuItem

Method	Description
addMenuKeyListener (MenuKeyListener l)	Adds a MenuKeyListener to the menu item.
getComponent()	Returns Component.
init(String text, Icon icon)	Initializes the menu item with the specified text and icon.
setEnabled(boolean b)	Enables or disables the menu item.

JMenu

- The **JMenu** class represents pull-down menu component which is deployed from a menu bar.

Constructor of JMenu

Constructor	Description
JMenu()	Creates a JMenu with no text.
JMenu(String text)	Creates a JMenu with the specified text.
JMenu(String s, boolean b)	Creates a JMenu with the specified text and specified as a tear-off menu or not.

Methods of JMenu

Method	Description
Component add(Component c)	Appends a component to the end of this menu.
Component add(Component c, int index)	Adds the specified component to this container at the given position.
JMenuItem add(JMenuItem menuItem)	Appends a menu item to the end of this menu.
JMenuItem add(String text)	Creates a menu item with the specified text and appends it to the end of this menu
void addSeparator()	Appends a new separator to the end of the menu.

JMenuItem getItem(int pos)	Returns the JMenuItem at the specified position.
int getItemCount()	Returns the number of items on the menu, including separators.
Component getMenuComponent(int n)	Returns the component at position n.
int getMenuComponentCount()	Returns the number of components on the menu.
JMenuItem insert(JMenuItem item, int pos)	Inserts the specified JMenuitem at a given position.
void insert(String text, int pos)	Inserts a new menu item with the specified text at a given position.
void insertSeparator(int index)	Inserts a separator at the specified position.
boolean isSelected()	Returns true if the menu is currently selected.
void remove(Component c)	Removes the component c from this menu.
void remove(int pos)	Removes the menu at the specified index from this menu
void remove(JMenuItem item)	Removes the specified menu item from this menu.
void removeAll()	Removes all menu items from this menu.

Event

- When the user interacts with a GUI application, an **event is generated.**
- **Example :** Pressing a button, Entering a character in Textbox., selecting an item or closing a window
- The super class of all event classes is **java.util.EventObject**.
- Event handling has three main components:
  - **Events :** An event is a change of state of an object.
  - **Events Source :** Event source is an object that generates an event.
  - **Listeners :** A listener is an object that listens the event. A listener gets notified when an event occurs.
- A source generates an Event and sends it to one or more **listeners** registered with the source.
- Once event is **received by the listener**, they process the event and then return.
- Events are supported by a number of Java packages, like **java.util, java.awt and java.awt.event**.
- **The list of Event classes are as follow:**

Event Class	Description	Listener Interface
ActionEvent	Generated when button is pressed, menu-item is selected, list-item is double clicked.	ActionListener
MouseEvent	Generated when mouse is dragged, moved, clicked, pressed or released also when the enters or exit a component.	MouseListener and MouseMotionListener

WindowEvent	Generated when window is activated, deactivated, opened or closed.	WindowListener
ItemEvent	Generated when check-box or list item is clicked.	ItemListener
KeyEvent	Generated when input is received from keyboard.	KeyListener
MouseWheelEvent	Generated when mouse wheel is moved.	MouseWheelListener
ComponentEvent	Generated when component is hidden, moved, resized or set visible.	ComponentEventListener
ContainerEvent	Generated when component is added or removed from container.	ContainerListener

ActionEvent class

- This class is defined in **java.awt.event** package.

Constructor of ActionEvent

Constructor	Description
ActionEvent (Object source, int id, String command)	Constructs an ActionEvent object
ActionEvent (Object source, int id, String command, int modifiers)	Constructs an ActionEvent object with modifier keys.

Method of ActionEvent

Method	Description
String getActionCommand()	Returns the command string associated with this action.
int getModifiers()	To return a value that indicates which modifier keys were pressed.

MouseEvent class

Method of MouseEvent

Method	Description
int getX() int getY()	To obtain X and Y co-ordinates of the mouse when an event occurred.
int getClickCount()	To obtain the number of mouse click for this event.
Point getPoint()	The location where the event happened.

WindowEvent class

Method of WindowEvent

Method	Action
Window getWindow()	To return the window object that generated the event.

Event Listener Interface

- The Event listener represents the interfaces **responsible to handle events**.
- It is a main interface which **every listener interface** has to **extend**.
- This class is defined in **java.util** package.

ActionListener

- The class which processes the **ActionEvent** should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the **addActionListener()** method.

Method of ActionListener

Method	Description
void actionPerformed (ActionEvent e)	Invoked when an action occurs.

MouseListener

- The class which processes the **MouseEvent** should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the **addMouseListener()** method.

Method of MouseListener

Method	Description
void mouseClicked (MouseEvent e)	Invoked when the mouse button has been clicked (pressed and released) on a component.
void mouseEntered (MouseEvent e)	Invoked when the mouse enters a component's coordinates space.
void mouseExited (MouseEvent e)	Invoked when the mouse exits a component's coordinates space.
void mousePressed (MouseEvent e)	Invoked when a mouse button has been pressed on a component.
void mouseReleased (MouseEvent e)	Invoked when a mouse button has been released on a component.

WindowListener

- The class which processes the **WindowEvent** should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the **addWindowListener()** method.

Method of MouseListener

Method	Description
void windowActivated (WindowEvent e)	Invoked when the Window is activated.
void windowClosed (WindowEvent e)	Invoked when a window has been closed.
void windowClosing (WindowEvent e)	Invoked when the user attempts to close the window from the window's system menu.
void windowDeactivated (WindowEvent e)	Invoked when a Window is no longer the active Window.
void windowOpened (WindowEvent e)	Invoked the first time a window is opened.
void windowIconified (WindowEvent e)	Invoked when a window is minimized.
void windowDeiconified (WindowEvent e)	Invoked when a window is changed from a minimized to a normal state.

KeyListener

- The class which processes the **KeyEvent** should implement this interface.
- The object of that class must be registered with a component.
- The object can be registered using the **addKeyListener()** method.

Method of KeyListener

Method	Description
void keyPressed(KeyEvent e)	Invoked when a key has been pressed.
void keyReleased(KeyEvent e)	Invoked when a key has been released.
void keyTyped(KeyEvent e)	Invoked when a key has been typed (pressed or released).