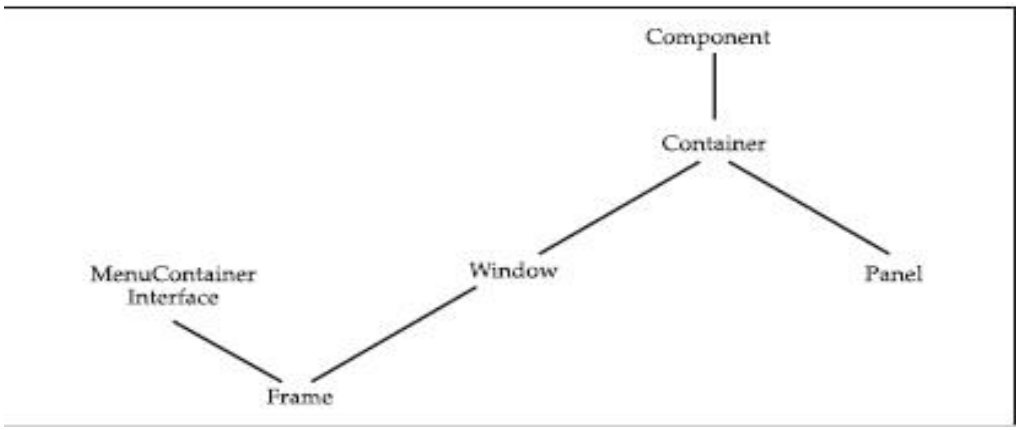
Unit 1. Abstract Window Toolkit

AWT: It is an **abstract window toolkit** use to create GUI application.

Package:java.awt.*;

Class hierarchy:



Component: At the top of the AWT hierarchy is the **Component** class. Component is an abstract class that encapsulates all of the attributes of a visual component. All user interface elements (Button, Checkbox, List, etc) that are displayed on the screen and that interact with the user are subclasses of **Component**.

Methods in Component class:1. setLayout(LayoutManager m)

2. setVisible(Boolean value)

3. add(Component c)

4 .remove(Component c) 5.removeAll()

Container: **Container** is a component in AWT that contains another component like button, text field, tables etc. **Container** is a subclass of component class. **Conatiner** class keeps track of components that are added to another component.

Window: The **Window** class creates a top-level window. A *top-level window* is not contained within any other object; it sits directly on the desktop.

Frame: It is a subclass of **Window** and has a title bar, menu bar, borders, and resizing corners.

Constructor: 1. **Frame**() :Creates Frame without title
2. Frame(String *title*): Creates Frame with title

Methods: 1. setTitle(String title) 2. setSize(int width, int height)
3. setSize(Dimension d) 4. setVisible(true)

Panel: A **Panel** is a window that does not contain a title bar, menu bar, or border.

Canvas: A canvas is subclass of Window . It is a rectangular window where you can draw.

❖ Features of AWT Component:

1. **AWT Components are platform dependent:**
   The look and feel of component changes according to Operating system
2. **AWT Component are heavy weighted:**
   AWT component creates process and hence they uses system resources like memory,cpu time,etc.

❖ Steps to add Component to Container

1. Declare object of component.
   e.g Button b;
2. initialize this object using constructor.
   e.g b=new Button("submit");
3. add object to container using add(Component obj)
   e.g add(b);

❖ Creating Frame Windows:

1.By extending Frame class(inheritance)

```
import java.awt.*;
class Demo extends Frame
{
        Button b;
      Demo(String title)
      {
         setLayout(new FlowLayout());
         setSize(300,300);
         setVisible(true);
       b=new Button("submit");
         add(b);
 }
public static void main(String args[])
{
    Demo d=new Demo("myFrame");
}}
```

By creating Frame Object: (Association):

```
import java.awt.*;
class Demo
{
        Button b;
      Demo()
      {
         Frame f=new Frame();
         f.setTitle("MyFrame");
         f.setLayout(new FlowLayout());
         f.setSize(300,300);
        f. setVisible(true);
         b=new Button("submit");
           f.add(b);
       }
     public static void main(String args[])
     {
        Demo d=new Demo();
     }
}
```

❖ Dialog Box:

Dialog boxes are primarily used to obtain user input. They are similar to frame windows, except that dialog boxes are always child windows of a top-level window. Dialog boxes don't have menu bars.

TYPES - **1.modal** : When a *modal* dialog box is active, all input is directed to it untill it is closed.Modal dialog blocks all the user input to all other windows in same application

2.modeless:When a *modeless* dialog box is active, input focus can be directed to another window in your program

Constructors:

Dialog(Frame *parentWindow*, boolean *mode*)

Dialog(Frame *parentWindow*, String *title*, boolean *mode*)

❖ FileDialog:

A built-in dialog box provided by Java that lets the user to specify a file. It is used to save or load a file.

Constructors:

FileDialog(Frame *parent*, String *boxName*)

FileDialog(Frame *parent*, String *boxName*, int *how*)

FileDialog(Frame *parent*)

Methods: 1.String getDirectory() 2.String getFile()

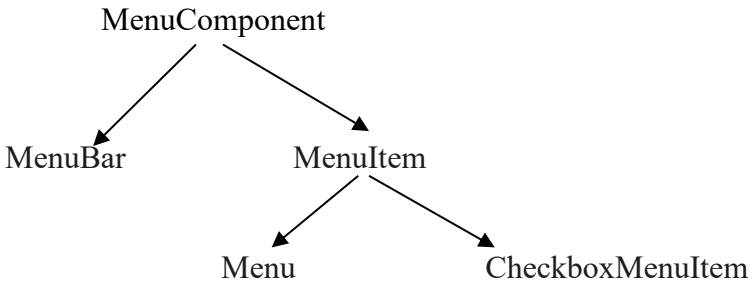
❖ AWT Components/Controls:

Control Name	Constructor and its Description	Methods
Label: Labels are passive controls that do not support any interaction with the user.	1.Label():Creates empty label 2.Label(String str):Creates label with given string 3.Label(String str, int align):Creates label with given text and align Align has following constant value: Label.LEFT, Label.RIGHT, Label.Center	**1.setText(String str)** **2.String getText()** **3.setAlignment(int how)** **4.int getAlignment()**
Button: A *push button* is a component that contains a label and that generates an event when it is pressed.	1.Button( ): Creates Button without label on it. 2.Button(String *str*) : Creates Button with given label on it	**1.String getLabel()** **2.Void getLabel()**
Checkbox: A *check box* is a control that is used to turn an option on or off. **CheckboxGroup** object is used to make radiobutton.	1.Checkbox( ) 2.Checkbox(String *str*) 3.Checkbox(String *str*, boolean *on*) 4.Checkbox(String *str*, boolean *on*, CheckboxGroup *cbg*) 5.Checkbox(String str , CheckboxGroup *cbg*, boolean *on*)	**1.boolean getState()** **2.void setState(Boolean on)** **3.String getLabel()** **4.void setLabel(String str)**
Choice **It** displays a pop-up list from which one item is selected	Choice()	**1.add(String item)** **2.String getSelectedItem()** **3.int getSelectedIndex()** **4.int getItemCount()** **5.String getItem(int index)**

List It is scrolling selection list which allows to select multiple items.	**List()** **List(int numrows)** **List(int rows, Boolean multiple)**	**1.add(String item)** **2.add(String item,int index)** **3. String getSelectedItem()** **4.int getSelectedIndex()** **5.int getItemCount()** **6.String getItem(int index)**
Scroll bar: The slider box in the scroll bar can be dragged by the user to a new position. It can be horizontal or vertical	**1.Scrollbar( ):** creates vertical scrollbar **2. Scrollbar(int *style*):** creates scrollbar with given style. Scrollbar.HORIZONTAL Scrollbar.VERTICAL **3.Scrollbar(int *style*, int *initialValue*, int *thumbSize*, int *min*, int *max*)**	**1.getMaximum()** **2.int getMinimum()** **3.setBlockIncrement(int n)**
TextField Text Field is single line area used to take user input when a text such as name,username, password etc is entered in GUI.	**1.TextField( ):** creates empty textfield. **2.TextField(int *numChars*):** creates textfield with given width. **3.TextField(String *str*):** **4.TextField(String *str*, int *numChars*)**	**1.String getText()** **2.setText(String str)** **3.setEchoChar(Char ch)** **4.char getEchoChar()** **5.setEditable(Boolean value)** **6. Boolean isEditable()**
TextArea It is multiline area used to take user input in GUI	**1.TextArea( )** **2.TextArea(**int *Lines,* int *numChars*) **3.TextArea(**String *str*) **4.TextArea(**String *str*, int *numLines*, int *numChars*) **5.TextArea(**String *str*, int *numLines*, int *numChars*, int *sBar)*	**1.getText()** **2.setEditable(Boolean value)** **3.isEditable()** **4.append(String str)** **5.insert(String str,int index)** **6.setText(String str)**

MenuBar A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down **Menu**	**MenuBar():creates menubar object**	**setMenuBar(MenuBar m)** **inserts menubar in frame window**
MenuItem	MenuItem() MenuItem(String item) MenuItem(String item, MenuShourtcut key)	setEnabled(Boolean value) Boolean isEnabled() getLabel() setLabel(String str)
CheckboxMenuItem	CheckboxMenuItem() CheckboxMenuItem(String item) CheckboxMenuItem(String item, boolean on)	Boolean getState() setState(Boolean value)
Menu	Menu() Menu(String item) Menu(String item, boolean remov)	**add(MenuItem obj)**

❖ Class hierarchy of Menu,MenuItem and MenuBar



❖ Layout Manager :

Layout manager automatically arranges your controls within a window using setLayout().

void setLayout(LayoutManager m)

We can set layout manually using setBound(int x, int y, int width, int height)

TYPES OF LAYOUT:

1.FlowLayout 2.BorderLayout 3.GridLayout

4.GridBagLayout 5.CardLayout

1. FlowLayout:

In this Layout components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right. By default a 5px gap is placed between each component in flowlayout.

Constructors:

1. FlowLayout( )- by default starts placing component form center
2. FlowLayout(int *how*)-how specifies alignment of components
3. FlowLayout(int *how*, int *horz*, int *vert*)- horz and vert are horizontal and vertical space between components.

how will have following values:

FlowLayout.LEFT

FlowLayout.CENTER

FlowLayout.RIGHT

2. BorderLayout:

It positions component in five region -North, South, West,East and Center

It has four narrow, fixed-width components at the edges and one large area in the center. The four sides are referred to as north, south, east, and west. The middle area is called the center.

Constructors:

BorderLayout( )

BorderLayout(int *horz*, int *vert)*

The add() method changes as **add(Component obj, BorderLayout region)**

3. GridLayout:

It lays out components in a two-dimensional grid. When you instantiate a GridLayout, you define the number of rows and columns. **It makes all components to have equal size**

Constructors

GridLayout( )

GridLayout(int *numRows*, int *numColumns*)

GridLayout(int *numRows*, int *numColumns*, int *horz*, int *vert*)

4.CardLayout:

The **CardLayout** class is unique among the other layout managers in that it stores several different layouts. Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time.

Constructors

CardLayout( )

CardLayout(int *horz*, int *vert*)

Methods: next(), previous(), first(), last(), show()

5.GridBagLayout

GridLayout has component size fixed.To remove this drawback GridBagLayout was introduced. 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.

Constructor: GridBagLayout()

Method:

add(Component obj, GridBagConstraints gbc)

GridBagConstraints

GridBagConstraints();

1. GridBagConstraints specifies how to display a specific component.
2. Every component added to a GridBagLayout container should have a GridBagConstraints object associated with it.
3. Without GridBagConstraints, the GridBagLayout is a blank slate.

Constrains of gridbag

int gridheight Specifies the number of cells in a column for the component's display area.

Int gridwidth Specifies the number of cells in a row for the component's display area.

int gridx Specifies the cell x co-ordinate in component's display area, where the first cell in a row has gridx=0.

int gridy Specifies the cell y co-ordinate of the component's display area, where the topmost cell has gridy=0.

insets(int top,int bottom,int left,int right)-This field specifies the minimum amount of space between the component and the edges of its display area.

fill-Used when the component's display area is larger than the component's requested size to determine whether and how to resize the component. It will have following values

GridBagConstraints.HORIZONTAL
GridBagConstraints.VERTICAL

❖ setBounds()

setBounds() is used to position each component manually in a container (Frame/Applet) and its syntax is as follow:

setBounds(int x, int y, int w, int h)

x and y are the x-coordinate and y-coordinate of component top-left corner
w and h are width and height of component respectively