MODULE VI

Syllabus:

- Introduction to AWT:
  - working with frames, graphics, color, font.
  - AWT Control fundamentals.
- Swing overview.
- Java database connectivity:
  - JDBC overview
  - creating and executing queries
  - dynamic queries.

Abstract Window Toolkit-AWT

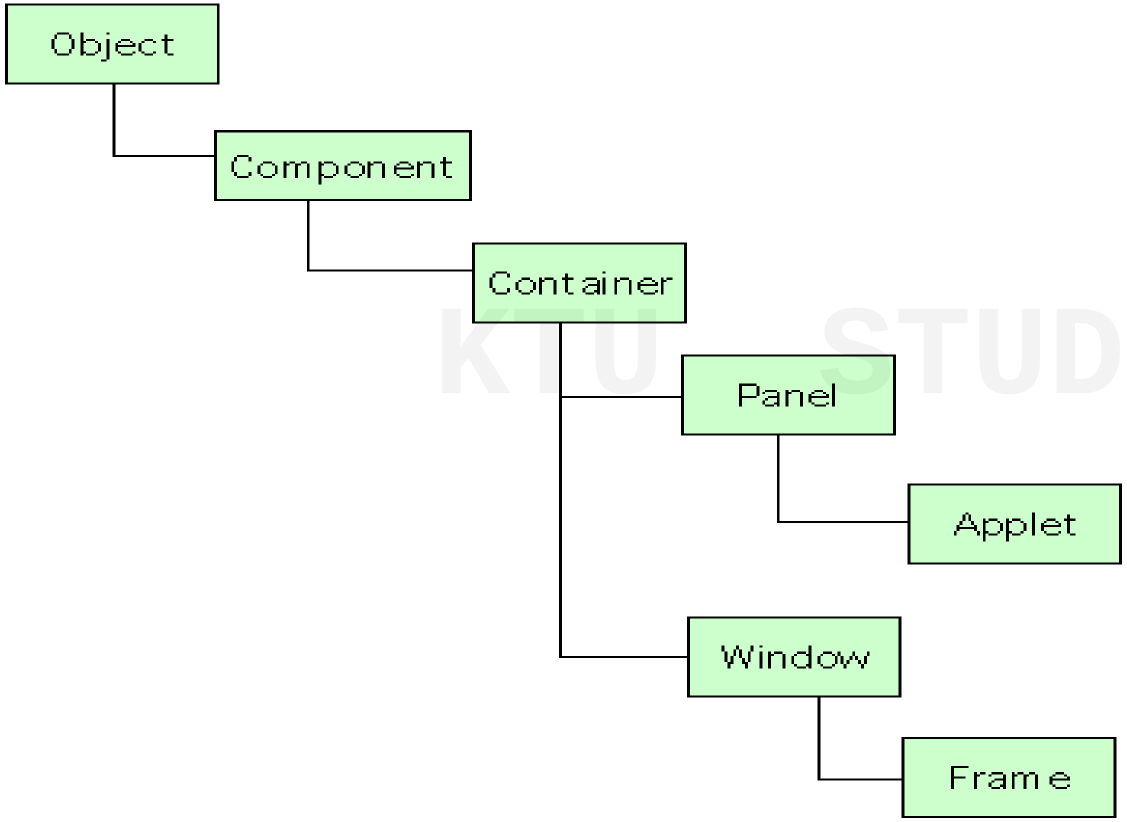
- The AWT contains numerous classes and methods that allow us to create and manage windows.

- AWT Classes :
  - The AWT classes are contained in the **java.awt** package.

Some AWT Classes	
Class	**Description**
AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five components: North, South, East, West, and Center.
Button	Creates a push button control.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index cards. Only the one on top is showing.
Checkbox	Creates a check box control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
Component	An abstract superclass for various AWT components.
Container	A subclass of **Component** that can hold other components.
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.
Dimension	Specifies the dimensions of an object. The width is stored in **width**, and the height is stored in h**eight**.
Event	Encapsulates events.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.
FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Font	Encapsulates a type font.
FontMetrics	Encapsulates various information related to a font. This information helps you display text in a window.
Frame	Creates a standard window that has a title bar, resize corners, and a menu bar.
Graphics	Encapsulates the graphics context. This context is used by the various output methods to display output in a window.
GraphicsDevice	Describes a graphics device such as a screen or printer.
GraphicsEnvironment	Describes the collection of available **Font** and **GraphicsDevice** objects.

Some AWT Classes(Cond..)	
Class	**Description**
GridBagConstraints	Defines various constraints relating to the **GridBagLayout** class.
GridBagLayout	The grid bag layout manager. Grid bag layout displays components subject to the constraints specified by **GridBagConstraints**.
GridLayout	The grid layout manager. Grid layout displays components in a two-dimensional grid.
Image	Encapsulates graphical images.
Insets	Encapsulates the borders of a container.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose. Similar to the standard Windows list box.
MediaTracker	Manages media objects.
Menu	Creates a pull-down menu.
MenuBar	Creates a menu bar.
MenuComponent	An abstract class implemented by various menu classes.
MenuItem	Creates a menu item.
MenuShortcut	Encapsulates a keyboard shortcut for a menu item.
Panel	The simplest concrete subclass of **Container**.
Point	Encapsulates a Cartesian coordinate pair, stored in **x** and **y**.
Polygon	Encapsulates a polygon.
PopupMenu	Encapsulates a pop-up menu.
PrintJob	An abstract class that represents a print job.
Rectangle	Encapsulates a rectangle.
Robot	Supports automated testing of AWT- based applications.
Scrollbar	Creates a scroll bar control.
ScrollPane	A container that provides horizontal and/or vertical scroll bars for another component.
SystemColor	Contains the colors of GUI widgets such as windows, scroll bars, text, and others.
TextArea	Creates a multiline edit control.
TextComponent	A superclass for **TextArea** and **TextField**.
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar, and no title.

Window Fundamentals



The two most common windows are:

- Those derived from **Panel**, which is used by applets
- Those derived from **Window**, which is used by Frame to create a window.

Working with Frames

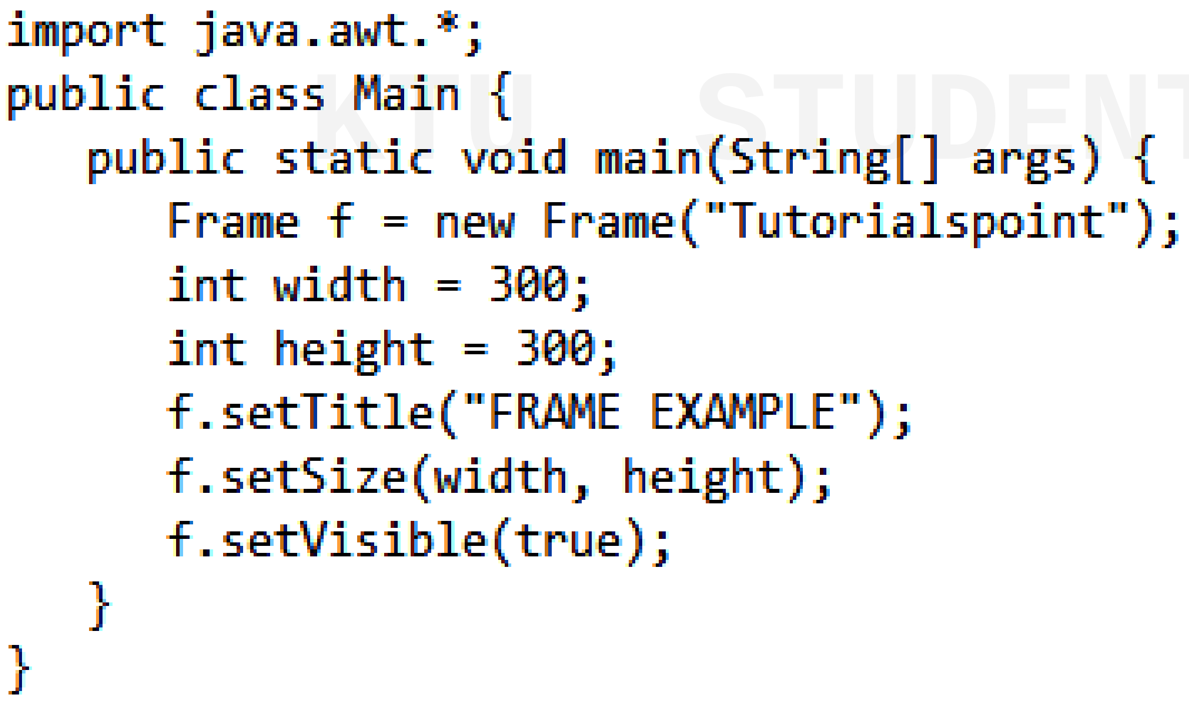
- **Frame( )** : Creates a standard window with title and border.
- **windowFrame(String *title*)** : Creates a window with title
- **void setSize(int *newWidth*, int *newHeight*)** : To set the dimensions of the window.

- **void setSize(Dimension *newSize*)** : Dimension object is passed. It contains width and height fields.
- **Dimension getSize( )** : To obtain the current size of a window
- **void setVisible(boolean *visibleFlag*)** : To make a created window visible.

- **void setTitle(String *newTitle*)** : To change window title.

- Closing a Frame Window : Call **setVisible(false)** followed by **windowClosing**( ).

- <u>**Example: To Create a Frame**</u>



```
import java.awt.*;
public class Main {
   public static void main(String[] args) {
      Frame f = new Frame("Tutorialspoint");
      int width = 300;
      int height = 300;
      f.setTitle("FRAME EXAMPLE");
      f.setSize(width, height);
      f.setVisible(true);
   }
}
```

- **<u>Example to create a frame by extending Frame class</u>**

```
import java.awt.*;
class Myframe extends Frame
{    Myframe(String title)
     {  super(title);
        setSize(350,150);
        setVisible(true);
     }
}
public class Main
{    public static void main(String args[])
     {     new Myframe("FRAME EXAMPLE");
     }
}
```

- **<u>Example: To create Frame in applet window</u>**

Refer appletframe.java

Graphics Class

- Java’s Graphics class include methods for drawing many different types of shapes, from simple lines to polygons to text in variety of fonts and colors.
- Methods of Graphics Class

Method	Description
drawString()	Displays a text String
drawLine()	Draws a straight Line
drawArc()	Draws a hollow Arc
drawRect()	Draws a hollow Rectangle
drawOval()	Draws a hollow Oval
drawRoundRect()	Draws a hollow rectangle with rounded corners
drawPolygon()	Draws a hollow polygon
fillArc()	Draws a filled Arc
fillPolygon()	Draws a filled Polygon
fillRect()	Draws a filled Rectangle
fillRoundRect()	Draws a filled rectangle with rounded corners
getColor()	Retrieves the current drawing color
getFont()	Retrieves the currently used font
setColor()	Sets the drawing color
setFont()	Sets the font

- **drawString(String s, int x, int y)**
  where (x,y) position from top left corner of window.

- **drawLine(int x1,int y1, int x2, int y2)**
  where (x1,y1) and (x2,y2) are pairs of coordinates.

- **drawRect(int x, int y, int width, int height)**
  where (x,y) represents top left corner of rectangle and width, height represents width and height of rectangle respectively.

- **fillRect(int x, int y, int width, int height)**

Same meaning for parameters as that of drawRect(). A solid Rectangle will be drawn.

- drawRoundRect() and fillRoundRect() are similar to drawRect() and fillRect() respectively except that they take additional two parameters representing width and height of angle of corners.

drawRoundRect(int x, int y, int width, int height,int wth_c, int hght_c)

fillRoundRect(int x, int y, int width, int height,int wth_c, int hght_c)

- **drawOval(int x, int y, int width, int height)**

Where (x,y) represents top left corner and width, height represents width and height of oval.

- **fillOval(int x,int y, int width, int height)**

Draws a solid oval with same meaning for parameters.

- **drawArc(int x, int y, int width, int height,int startangle, int nodegree)**

Where (x,y) represents top left corner and width, height represents width and height of arc, startangle represents starting angle of arc , nodegree represents number of degrees around the arc.

- **fillArc(int x, int y, int width, int height,int startangle, int nodegree) –** fills the arc drawn

- **drawPolygon(int xpoints[], int ypoints[], int npoints)**

xpoints[] represent array of integers containing x coordinates,

ypoints[] represent array of integers containing y coordinates, npoints represent total number of points.

Eg:

```
public void paint(Graphics g)
{
   int xpoints[]={10,170,80,10};
   int ypoints[]={20,40,140,20};
   int npoints=xpoints.length;
  g.drawPolygon(xpoints, ypoints, npoints);
}
```

- **fillPolygon(int xpoints[], int ypoints[], int npoints)** – fills the polygon drawn.

Color and Font

- **Color Class**

  The following are three constructors for Color class.

  Color(int red, int green, int blue)

  Color(int rgbvalue)

  Color(float red, float green, float blue)

- **Color Methods**

  Color getColor() returns the current color

  setColor(Color newcolor): Change the foreground color.

- set the background color and foreground colors using the following methods:

  void setBackground(Color *newColor*)

  void setForeground(Color *newColor*)

- You can obtain the current settings for the background and foreground colors by calling **getBackground( ) and getForeground( ).**

  Color.getBackground()

  Color getForeground()

- **Font class**

  Font(String fontName, int fontStyle, int pointStyle)

Where fontName specifies the name of the desired font.

The style of the font is given by fontStyle. fontStyle consists of one or more of these three constants: Font.PLAIN, Font.BOLD, Font.ITALIC.

The size of the font is specified by pointSize

- **Font Methods**

  Font getFont() : obtain information of currently selected font.

  String getName() : get the name of current Font

  int getSize() : get the size of current font.

  int getStyle() : get the style of current font.

  setFont(Font f) : set the new font

```
import java.awt.*;
import java.applet.*;
/*      <applet code="Sample" width=2000 height=500>
          </applet>
*/
public class Sample extends Applet
{
    public void init()
    {
        Font f=new Font("Times New Roman",Font.ITALIC,25);
        setFont(f);
        setForeground(Color.red);
    }
    public void paint(Graphics g)
    {
        g.drawString("hello", 10, 30);
        g.drawRect(10,50,70,35);
        g.fillOval(90,70,50,90);
        g.drawRoundRect(200,300,100,50,70,45);
    }
}
```

AWT Controls

- *Controls* are components that allow a user to interact with the application in various ways.

- The AWT supports the following types of controls:
  - Labels
  - Push buttons
  - Check boxes
  - Check box groups(Radio buttons)
  - Choice lists
  - Lists
  - Scroll bars
  - Text Field
  - Text Area

- **Adding Controls**
  - Create an instance of the desired control
  - Add it to a window by calling **add( )**

    Component add(Component *compObj*)

    compObj is an instance of the control that we want to add

- **Removing Controls**
  - Call **remove( )**

    void remove(Component *obj*)

    obj is a reference to the control that we want to remove.
  - Remove all controls by calling **removeAll( )**.

- **Responding to Controls**
  - Labels are passive controls. Except for labels, all controls generate events when they are accessed by the user.
  - The program simply implements the appropriate interface and then registers an event listener for each control that we need to monitor.

Labels

- A *label* contains a string, which it displays.
- Labels are passive controls that do not support any interaction with the user.

- **Label** defines the following constructors:
  - Label( ) : Creates a blank label
  - Label(String *str*) :Creates a label that contains the string str
  - Label(String *str*, int *how*) : Creates a label that contains string str
    how - **Label.LEFT**, **Label.RIGHT**, or **Label.CENTER**

- void setText(String *str*) : To set or change the text in a label.
- String getText( ) : To obtain the current label
- void setAlignment(int *how*) : To set the alignment of the string
- int getAlignment( ) : To obtain the current alignment

```
import java.awt.*;
import java.applet.*;
/*
<applet code="LabelDemo" width=300 height=200>
</applet>
*/
public class LabelDemo extends Applet
{
public void init()
{
Label l1 = new Label("One");
Label l2 = new Label("Two");
Label l3 = new Label("Three");
add(l1);
add(l2);
add(l3);
}
}
```

Push Buttons

- A *push button* contains a label that generates an event when it is pressed.
- This is sent to any listeners that previously registered an interest in receiving action event notifications from that component.
- 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** defines two constructors:
  - Button( ) :Creates an empty button
  - Button(String *str*) :Creates a button that contains *str* as a label
- void setLabel(String *str*) :To set label of the button
- String getLabel( ) : To retrieve the label

```
importjava.awt.*;
importjava.awt.event.*;
importjava.applet.*;
/*
        <applet    code="ButtonDemo"    width=250
        height=150>
        </applet>
*/
public    class    ButtonDemo    extends    Applet
implements ActionListener
{       String msg = "";
        Button b1,b2,b3;
        public void init()
        {       b1 = new Button("Yes");
                b2 = new Button("No");
                b3 = new Button("Undecided");
                add(b1);
                add(b2);
                add(b3);
                b1.addActionListener(this);
                b2.addActionListener(this);
                b3.addActionListener(this);
        }
```

```
public void actionPerformed(ActionEvent ae)
{
    String str = ae.getActionCommand();
    if(str.equals("Yes"))
   {       msg = "You pressed Yes.";    }
    else if(str.equals("No"))
    {       msg = "You pressed No.";     }
    else
    {     msg = "You pressed Undecided.";   }
          repaint();
}
public void paint(Graphics g)
{      g.drawString(msg, 6, 100);
}
}
```

Check Boxes

- It consists of a small box that can either contain a check mark or not.
- A label is associated with each checkbox
- A *check box* is a control that is used to turn an option on or off.
- Check boxes can be used individually or as part of a group
- **Checkbox** supports these constructors:
  - Checkbox( ) : label - initially blank, state - unchecked
  - Checkbox(String *str*) : label – *str,* state - unchecked
  - Checkbox(String *str,* Boolean *on*) : label – *str, on* is **true**- checked , *on* is **false**- unchecked
  - Checkbox(String *str,* Boolean *on,* CheckboxGroup *cbGroup*) : label – *str,* group – *cbGroup(*If this check box is not part of a group, then *cbGroup*must be **null)**
  - Checkbox(String *str,* CheckboxGroup *cbGroup,* Boolean *on*)

- boolean getState( ) :To retrieve the current state of a check box
- void setState(boolean *on*) :To set the current state of a check box
- String getLabel( ) : To obtain the current label
- void setLabel(String *str*) : To set the label

- 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( )** method. An **ItemEvent** object is supplied as the argument to this method.

```
importjava.awt.*;
importjava.awt.event.*;
importjava.applet.*;
/*<applet code="CheckboxDemo" width=250
height=200>
</applet>
*/
public class CheckboxDemo extends Applet
implements ItemListener
{  String msg = "";
   Checkbox c1, c2,c3,c4;
    public void init()
   {
     c1=new Checkbox("Windows 98/XP",null, true);
     c2 = new Checkbox("Windows NT/2000");
     c3= new Checkbox("Solaris");
     c4 = new Checkbox("MacOS");
     add(c1); add(c2);
     add(c3); add(c4);
     c1.addItemListener(this);
     c2.addItemListener(this);
     c3.addItemListener(this);
     c4.addItemListener(this);
   }
```

```
public void itemStateChanged(ItemEvent ie)
{      repaint();      }
public void paint(Graphics g)
{       msg = "Current state: ";
        g.drawString(msg, 6, 80);
        msg = " Windows 98/XP: " + c1.getState();
        g.drawString(msg, 6, 100);
       msg = " Windows NT/2000:"+
c2.getState();
       g.drawString(msg, 6, 120);
       msg = " Solaris: " + c3.getState();
        g.drawString(msg, 6, 140);
        msg = " MacOS: " + c4.getState();
        g.drawString(msg, 6, 160);
}
}
```

CheckboxGroup

- Used 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.
  - First define the group to which they will belong
  - Then specify that group when you construct the check boxes

- These check boxes are often called *radio buttons.*

- Checkbox getSelectedCheckbox( ) : Determine which check box in a group is currently selected

- void setSelectedCheckbox(Checkbox *which*) :To set a check box. Here, *which* is the check box that you want to be selected. The previously selected checkbox will be turned off.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*      <applet code="CBGroup" width=250 height=200>
</applet>
*/
public class CBGroup extends Applet implements ItemListener
{       String msg = "";
   Checkbox c1,c2,c3,c4;
   CheckboxGroup cbg;
    public void init()
   {    cbg = new CheckboxGroup();
        c1 = new Checkbox("Windows 98/XP", cbg, true);
        c2 = new Checkbox("Windows NT/2000", cbg, false);
        c3 = new Checkbox("Solaris", cbg, false);
        c4 = new Checkbox("MacOS", cbg, false);
        add(c1);
        add(c2);
        add(c3);
        add(c4);
        c1.addItemListener(this);
        c2.addItemListener(this);
        c3.addItemListener(this);
        c4.addItemListener(this);
   }
        public void itemStateChanged(ItemEvent ie)
        {
                repaint();
        }
        // Display current state of the check boxes.
        public void paint(Graphics g)
        {
                msg = "Current selection: ";
                msg                                         +=
                cbg.getSelectedCheckbox().getLabel();
                g.drawString(msg, 6, 100);
        }
}
```

Choice Lists

- The **Choice** class is used to create a *pop-up list* of items from which the user may choose.

- When the user clicks on choice, the whole list of choices pops up, and a new selection can be made.

- **Choice** constructor creates an empty list
- void add(String *name*) : To add a selection to the list
- String getSelectedItem( ) :Returns currently selected string
- int getSelectedIndex( ) : Returns the index of the currently selected item. The first item is at index 0
- getItemCount( ) :To obtain the number of items in the list
- void select(int *index*) : Set the currently selected item
- void select(String *name*) : Set the currently selected item
- String getItem(int *index*) : To obtain the name associated with the item

- Each time a choice is selected, 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( )** method. An **ItemEvent** object is supplied as the argument to this method.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*      <applet code="ChoiceDemo" width=300 height=180>
</applet>
*/
public class ChoiceDemo extends Applet implements
ItemListener
{       Choice os, browser;
String msg = "";
public void init()
{       os = new Choice();
        browser = new Choice();
        os.add("Windows 98/XP");
        os.add("Windows NT/2000");
        os.add("Solaris");    os.add("MacOS");
        browser.add("Netscape 3.x");
        browser.add("Netscape 4.x");
        browser.add("Netscape 5.x");
        browser.add("Netscape 6.x");
        browser.add("Internet Explorer 4.0");
        browser.add("Internet Explorer 5.0");
        browser.add("Internet Explorer 6.0");
        browser.add("Lynx 2.4");
        browser.select("Netscape 4.x");
                add(os);      add(browser);
                os.addItemListener(this);
                browser.addItemListener(this);
        }
        public void itemStateChanged(ItemEvent ie)
        {    repaint();     }
        public void paint(Graphics g)
        {       msg = "Current OS: ";
                msg += os.getSelectedItem();
                g.drawString(msg, 6, 120);
                msg = "Current Browser: ";
                msg += browser.getSelectedItem();
                g.drawString(msg, 6, 140);
        }
}
```

Lists

- It provides a compact, multiple-choice, scrolling selection list.
- It can be created to allow multiple selections.

- **List** constructors are:
  - List( ) : Allows to select only one item at a time
  - List(int *numRows*) : *numRows* specifies the number of entries in the list that will always be visible
  - List(int *numRows*, boolean *multipleSelect*) : If *multipleSelect* is **true**, then the user may select two or more items at a time. If it is **false**, then only one item may be selected

- void add(String *name*) : Adds items to the end of the list.
- void add(String *name*, int *index*) : Adds the item at the index specified by *index.* Indexing begins at zero. You can specify –1 to add the item to the end of the list.

- For lists that allow only single selection:
  - String getSelectedItem( ) : Returns a string containing the name of the selected item.
  - int getSelectedIndex( ) : Returns the index of the selected item.

- For lists that allow multiple selection:
  - String[ ] getSelectedItems( ) : Returns an array containing the names of the currently selected items
  - int[ ] getSelectedIndexes( ) : Returns an array containing the indexes of the currently selected items.

- int getItemCount( ) : To obtain the number of items in the list

- void select(int *index*) : To set the currently selected item

- String getItem(int *index*) : To obtain the name associated with the item at that index

- Need to implement the **ActionListener** interface.

- Each time a **List** item is double-clicked, an **ActionEvent** object is generated.

- Its **getActionCommand( )** method can be used to retrieve the name of the newly selected item.

- Also, each time an item is selected or deselected with a single click, an **ItemEvent** object is generated.

- Its **getStateChange( )** method can be used to determine whether a selection or deselection triggered this event.

- **getItemSelectable( )** returns a reference to the object that triggered this event.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="ListDemo" width=300 height=180>
   </applet>
*/
public class ListDemo extends Applet implements
ActionListener
{       List os, browser;
String msg = "";
public void init()
{       os = new List(4, true);
        browser = new List(4, false);
        os.add("Windows 98/XP");
       os.add("Windows NT/2000");
       os.add("Solaris");    os.add("MacOS");
        browser.add("Netscape 3.x");
        browser.add("Netscape 4.x");
        browser.add("Netscape 5.x");
        browser.add("Netscape 6.x");
       browser.add("Internet Explorer 4.0");
       browser.add("Internet Explorer 5.0");
       browser.add("Internet Explorer 6.0");
       browser.add("Lynx 2.4");
      browser.select(1);
               add(os);
               add(browser);
               os.addActionListener(this);
               browser.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae)
        {    repaint();     }
        public void paint(Graphics g)
        {       int idx[];
               msg = "Current OS: ";
               idx = os.getSelectedIndexes();
               for(int i=0; i<idx.length; i++)
                    msg += os.getItem(idx[i]) + " ";
               g.drawString(msg, 6, 120);
               msg = "Current Browser: ";
               msg += browser.getSelectedItem();
               g.drawString(msg, 6, 140);
        }
}
```

Scroll Bars

- *Scroll bars* are used to select continuous values between a specified minimum and maximum.

- Scroll bars may be oriented horizontally or vertically.

- The current value of the scroll bar relative to its minimum and maximum values is indicated by the *slider box* (or *thumb*) for the scroll bar.

- **Scrollbar** defines the following constructors:
  - Scrollbar( ) : Creates a vertical scroll bar
  - Scrollbar(int *style*) :

    style - **Scrollbar.VERTICAL, Scrollbar.HORIZONTAL**

- Scrollbar(int *style,* int *initialValue,* int *thumbSize,* int *min,* int *max*) :

  initialValue - initial value of the scroll bar

  thumbSize - height of the thumb

  min and max - minimum and maximum values of the scroll bar

- void setValues(int *initialValue*, int *thumbSize*, int *min*, int *max*) : to set parameters
- int getValue( ) : To obtain the current value of the scroll bar
- void setValue(int *newValue*) : To set the current value
- int getMinimum( ) : Retrieve the minimum value
- int getMaximum( ) : Retrieve the maximum value
- void setUnitIncrement(int *newIncr*): scrolled up/down one line
- void setBlockIncrement(int *newIncr*): page-up/page-down increments are 10

- Implement the **AdjustmentListener** interface.

- Each time a user interacts with a scroll bar, an **AdjustmentEvent** object is generated.

- Its **getAdjustmentType()** method can be used to determine the type of the adjustment.

  - BLOCK_DECREMENT: A page-down event has been generated.
  - BLOCK_INCREMENT: A page-up event has been generated.
  - TRACK : An absolute tracking event has been generated.
  - UNIT_DECREMENT: The line-down button in a scroll bar has been pressed.
  - UNIT_INCREMENT : The line-up button in a scroll bar has been pressed.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*      <applet code="SBDemo" width=300 height=200>
</applet>
*/
public class SBDemo extends Applet implements
AdjustmentListener, MouseMotionListener
{       String msg = "";
Scrollbar vertSB, horzSB;
public void init()
{
vertSB = new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 200);
horzSB = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0,
300);
add(vertSB);
add(horzSB);
vertSB.addAdjustmentListener(this);
horzSB.addAdjustmentListener(this);
addMouseMotionListener(this);
}
public void adjustmentValueChanged
(AdjustmentEvent ae)
{       repaint();      }
public void mouseDragged(MouseEvent
me)
{       int x = me.getX();
int y = me.getY();
vertSB.setValue(y);
horzSB.setValue(x);
repaint();
}
public void mouseMoved(MouseEvent me)
{   }
public void paint(Graphics g)
{       msg = "Vertical: " + vertSB.getValue();
msg += ", Horizontal: " + horzSB.getValue();
g.drawString(msg, 6, 160);
g.drawString("*",           horzSB.getValue(),
vertSB.getValue());
}
}
```

TextField

- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.

- **TextField** constructors:
  - TextField( ) : Creates a default text field
  - TextField(int *numChars*) : *numChars – size of text field*
  - TextField(String *str*) : Initializes the text field with the string
  - TextField(String *str*, int *numChars*) : Initializes a text field and sets its width

- String getText( ) : To obtain the current string in the text field
- void setText(String *str*) : To set the text
- String getSelectedText( ) : Obtain the currently selected text
- void select(int *startIndex*, int *endIndex*): Select a portion of text

- boolean isEditable( ) : Determine editability of the text field.
- void setEditable(boolean *canEdit*): To set the editability
- void setEchoChar(char *ch*) : disable the echoing of the characters. *ch* - character to be echoed
- boolean echoCharIsSet( ) : Echo set mode is activated or not

- char getEchoChar( ) : Retrieve the echo character
- Text fields respond when the user presses ENTER.
- When this occurs, an action event is generated.

```
import java.awt.*;
Import java.awt.event.*;
import java.applet.*;
/*
<applet        code="TextFieldDemo"        width=380
height=150>
</applet>
*/
public    class    TextFieldDemo    extends    Applet
implements ActionListener
{       TextField t1,t2;
    public void init()
   {  Label l1 = new Label("Name: ", Label.RIGHT);
      Label l2 = new Label("Password: ", Label.RIGHT);
      t1 = new TextField(12);
      t2 = new TextField(8);
      t2.setEchoChar('?');
     add(l1);
     add(t1);
     add(l2);
     add(t2);
     t1.addActionListener(this);
     t2.addActionListener(this);
   }
```

```
// User pressed Enter.
public void actionPerformed(ActionEvent ae)
{
    repaint();
}
public void paint(Graphics g)
{
g.drawString("Name: " + t1.getText(), 6, 60);
g.drawString("Selected   text   in   name:   "   +
t1.getSelectedText(), 6, 80);
g.drawString("Password: " + t2.getText(), 6, 100);
}
}
```

TextArea

- **TextArea** is a multiline editor.

- Constructors for **TextArea** are:
  - TextArea( )
  - TextArea(int *numLines,* int *numChars*) : *numLines* – height, *numChars* - width
  - TextArea(String *str*) : str - Initial text
  - TextArea(String *str*, int *numLines,* int *numChars*)
  - TextArea(String *str*, int *numLines,* int *numChars,* int *sBars*) :

    sBars - SCROLLBARS_BOTH SCROLLBARS_NONE
    SCROLLBARS_HORIZONTAL_ONLY
    SCROLLBARS_VERTICAL_ONLY

- String getText( )
- void setText(String str)
- String getSelectedText( )
- void select(int startindex,int endindex)
- boolean isEditable()
- void setEditable(boolean canedit)
- void append(String *str*)
- void insert(String *str*, int *index*)
- void replaceRange(String *str*, int *startIndex*, int *endIndex*)

```
importjava.awt.*;
importjava.applet.*;
/*
        <applet code="TextAreaDemo" width=300 height=250>
        </applet>
*/
public class TextAreaDemo extends Applet
{
        public void init()
        {
                String val = "There are two ways of constructing " + "a software design.\n" + "One way is to
                make it so simple\n" + "that there are obviously no deficiencies.\n" + "And the other way is to
                make it so complicated\n" + "that there are no obvious deficiencies.\n\n" + " -C.A.R. Hoare\n\n"
                + "There's an old story about the person who wished\n" + "his computer were as easy to use as
                his telephone.\n" + "That wish has come true,\n" + "since I no longer know how to use my
                telephone.\n\n" + " -BjarneStroustrup, AT&T, (inventor of C++)";

                TextArea text = new TextArea(val, 10, 30);
                add(text);
        }
}
```

TUTORIAL 10

Q1. Write an applet program that has 2 Buttons. On clicking the first button The font has to be changed to “Aharoni” with size 32” and display the text “hello” with the new font and on clicking second button a rounded filled rectangle has to be displayed.

Q2. Write an applet program to create 4 RadioButtons. Display the text of radiobutton clicked in a new Font Arial with size 40.

Q3. Design a java program to create a login page with username, password and a submit button. On clicking submit the text that was entered in the text fields has to be displayed.

Q1
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*   <applet code="MouseEvents" width=250 height=150>
</applet>  */
public class MouseEvents extends Applet implements ActionListener
{       String msg = "";      int c=0;
        Button b1,b2;
        public void init()
        {   b1 = new Button("Yes");    b2 = new Button("No");
             add(b1);    add(b2);
             b1.addActionListener(this);  b2.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae)
        {    String str = ae.getActionCommand();
             if(str.equals("Yes"))
             {        Font f=new Font("Aharoni",Font.BOLD,32);
                      c=1;  setFont(f);
             }
             else
                    c=2;
             repaint();
        }
        public void paint(Graphics g)
        {    if(c==1)
                    g.drawString("hello",10,30);
             if(c==2)
                   g.fillRoundRect(20,50,70,30,60,30);
        }
}
```

Q2.

```
import java.awt.*;    import java.awt.event.*;    import java.applet.*;
/*      <applet code="CBGroup" width=250 height=200>  </applet>  */
public class CBGroup extends Applet implements ItemListener
{      String msg = "";    Checkbox c1,c2,c3,c4;  CheckboxGroup cbg;
     public void init()
    {     cbg = new CheckboxGroup();
          c1 = new Checkbox("Windows 98/XP", cbg, true);
          c2 = new Checkbox("Windows NT/2000", cbg, false);
          c3 = new Checkbox("Solaris", cbg, false);
          c4 = new Checkbox("MacOS", cbg, false);
         add(c1);  add(c2);   add(c3);   add(c4);
         c1.addItemListener(this);  c2.addItemListener(this);
         c3.addItemListener(this);  c4.addItemListener(this);
     }
    public void itemStateChanged(ItemEvent ie)
    {     Font f=new Font("Arial",Font.BOLD,40);
          setFont(f);   repaint();
    }
    public void paint(Graphics g)
    {  msg = "Current selection: ";
       msg += cbg.getSelectedCheckbox().getLabel();
       g.drawString(msg, 6, 100);
    }
}
```

Swings

- The Swing-related classes are contained in **javax.swing**.

- Swing is a set of classes that provides more powerful and flexible GUI components than are possible with the AWT.

- All components have more capabilities in Swing.

  Example: A button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

Features of Swing

- **Swing Components Are Lightweight**

  This means that they are written entirely in Java and do not map directly to platform-specific peers.

- **Swing Supports a Pluggable Look and Feel**

  SWING based GUI Application look and feel can be changed at run-time, based on available values.

- **Swing uses MVC Architecture**

Java's Swing components have been implemented using the model-view controller (MVC) model. Any Swing component can be viewed in terms of three independent aspects: what state it is in (its model), how it looks (its view), and what it does (its controller). Suppose the user clicks on a button. This action is detected by the controller. The controller tells the model to change into the pressed state. The model in turn generates an event that is passed to the view. The event tells the view that the button needs to be redrawn to reflect its change in state.

Advantages of Swings over AWT

- Swing is the latest GUI toolkit, and provides a richer set of interface components than the AWT.

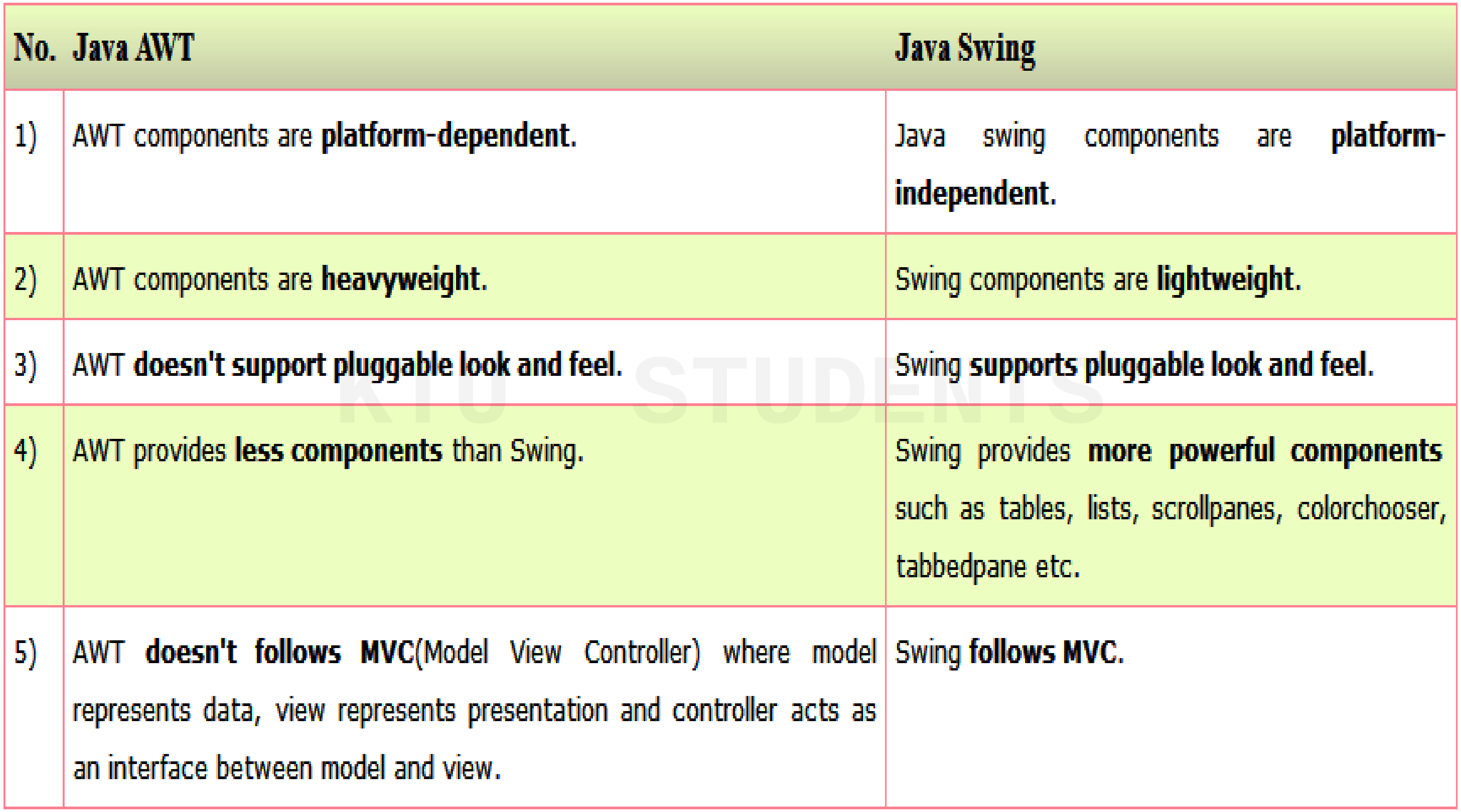
- The behavior and appearance of Swing components is consistent across platforms, whereas AWT components will differ from platform to platform. Thus Swing is platform independent whereas AWT is platform dependent.

  Reason: AWT translates its various visual components into their corresponding, platform-specific equivalents, or *peers. This means that the look* and feel of a component is defined by the platform, not by Java. Because the AWT components use native code resources, they are referred to as *heavyweight.*

- Swing components can be given their own "look and feel“.

  Example: A button may have both an image and a text string associated with it. Also, the image can be changed as the state of the button changes.

Difference between AWT and Swing



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.**

- Some Swing classes are:
  - JFrame allows to use Swing components in a frame.
  - JLabel
  - JButton
  - JTextField
  - JApplet

JFrame and JLabel

```
import javax.swing.*;
class swingdemo
{   swingdemo()
    {
       JFrame jf=new JFrame("Swing Appl");
       jf.setSize(200,300);
       JLabel jl=new JLabel("Name",JLabel.CENTER);
       jf.add(jl);
       jf.setVisible(true);
       jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //closing window will end the application
    }
}
public class Main
{   public static void main(String args[])
    {
       new swingdemo();
    }
}
```

JLabel Constructors used are:

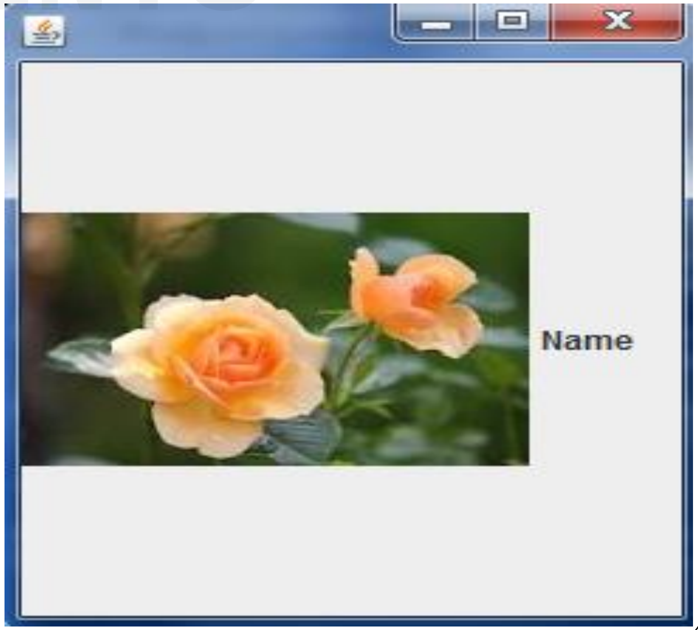
- JLabel(Icon ic)
- JLabel(String str)
- JLabel(String str,Icon ic,int align)

- Here Icon is abstract class that cannot be instantiated. ImageIcon is a class that extends Icon. So to load images the following statement can be used:

```
ImageIcon ic=new ImageIcon("filename");
```

where filename is a string quantity.

```
import javax.swing.*;
public class SimpleLabel extends JFrame
{   SimpleLabel()
   {  ImageIcon ic=new ImageIcon("download.jpg");
      JLabel jl=new JLabel("Name",ic,JLabel.LEFT);
      setSize(250,300);
      setVisible(true);
       add(jl);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   public static void main(String args[])
   {
        new SimpleLabel();
   }
}
```



JButton

- The **JButton** class provides the functionality of a push button.
- **JButton** allows an icon, a string, or both to be associated with the push button.
- Some of its constructors are shown here:
  - JButton(Icon *i*)
  - JButton(String *s*)
  - JButton(String *s*, Icon *i*)

  Here, *s* and *i* are the string and icon used for the button.

JTextField

- Some of its constructors are shown here:
  - JTextField(int cols)
  - JTextField(String str,int cols)
  - JTextField(String str)

JApplet

- A **JApplet** is an **Applet** that supports the Swing graphics library
- Applets that use Swings must be subclasses of **JApplet**.
- Swing applet(i.e. JApplet also uses the same four life cycle methods: init(), start(), stop() and destroy().

- Difference between **Applet** and **JApplet :**
  - When adding a component to an instance of **JApplet**, call **add( )** for the *content pane* of the **JApplet** object.

- The content pane can be obtained by:

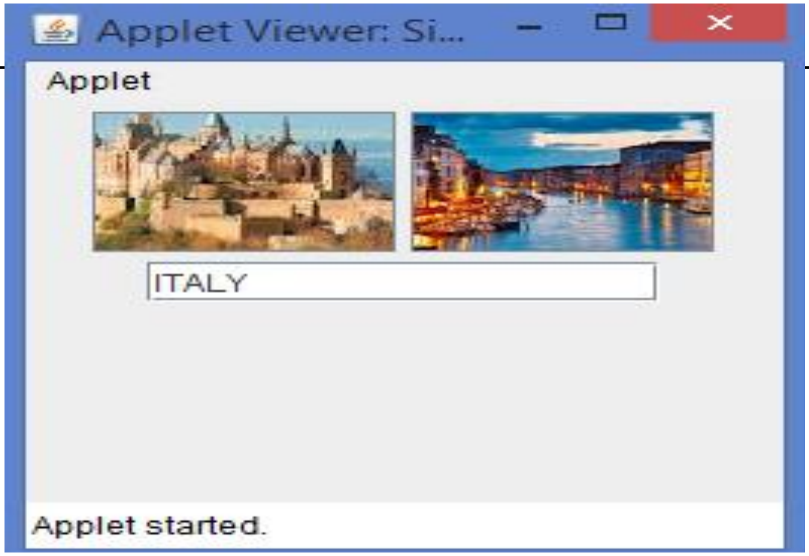
  Container getContentPane( )
- To add a component to a content pane:

  void add(*comp*)

comp - component to be added to the content pane

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*<applet          code="SimpleApplet"          width=250
height=300>
  </applet>
*/
public  class  SimpleApplet  extends  JApplet  implements
ActionListener
{      JTextField jtf;
       public void init()
       {         Container contentPane = getContentPane();
                 contentPane.setLayout(new FlowLayout());
                 ImageIcon ic1 = new ImageIcon("ger.jpg");
                 JButton jb = new JButton(ic1);
                  jb.setPreferredSize(new Dimension(80, 50));
                 jb.setActionCommand("GERMANY");
                 jb.addActionListener(this);
                 contentPane.add(jb);
                 ImageIcon ic2 = new ImageIcon("it.jpg");
                 jb = new JButton(ic2);
                 jb.setPreferredSize(new Dimension(80, 50));
                 jb.setActionCommand("ITALY");
                 jb.addActionListener(this);
                 contentPane.add(jb);

                 jtf = new JTextField(15);
                 contentPane.add(jtf);
        }
        public void actionPerformed(ActionEvent ae)
        {      jtf.setText(ae.getActionCommand());    }
}
```



```
ITALY
```

Applet started.

- Flow layouts are typically used to arrange buttons in a panel. It arranges buttons horizontally until no more buttons fit on the same line. The line alignment is determined by the align property. The possible values are:
- LEFT
- RIGHT
- CENTER

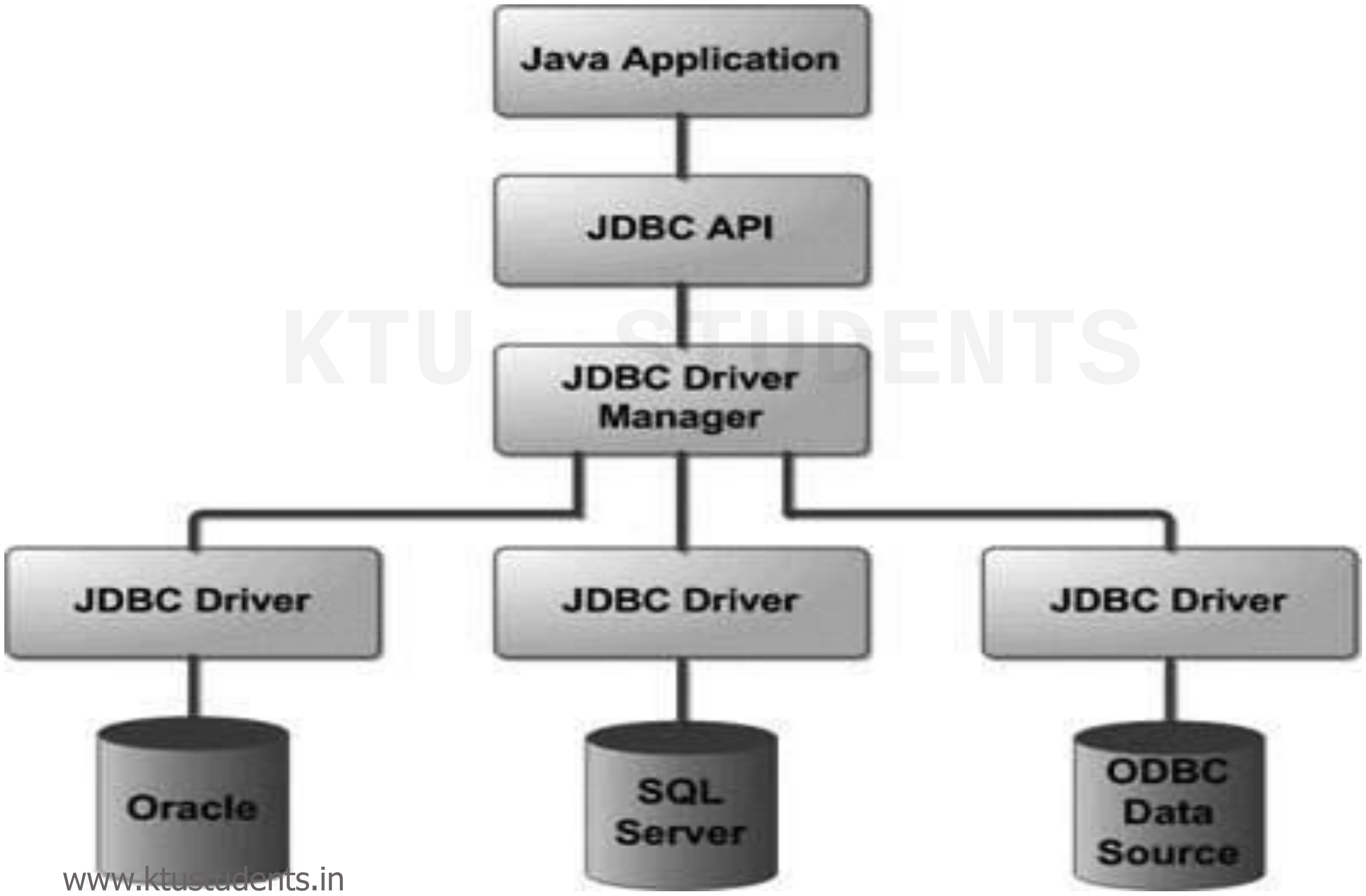
JAVA DATABASE CONNECTIVITY

- **J**ava **D**ata**b**ase **C**onnectivity : It is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

- The JDBC library includes APIs for each of the tasks commonly associated with database usage:
  - Making a connection to a database
  - Creating SQL or MySQL statements
  - Executing that SQL or MySQL queries in the database
  - Viewing & Modifying the resulting records

- JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

JDBC Architecture



- JDBC Architecture consists of two layers
  - **JDBC API:** This provides the application-to-JDBC Manager connection.
  - **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
- The JDBC API uses a driver manager to provide transparent connectivity to heterogeneous databases.
- The JDBC driver manager ensures that the correct driver is used to access each data source.

- The JDBC API provides the following interfaces and classes:
  - **DriverManager:** This class manages a list of database drivers.
  - **Driver:** Handles the communications with the database server
    - A **JDBC driver** is a software component enabling a Java application to interact with a database.
    - To connect with individual databases, JDBC requires drivers for each database.
  - **Connection:** All communication with database is through connection interface object.
  - **Statement:** This interface object is used to submit the SQL statements to the database.
  - **ResultSet:** These objects hold data retrieved from a database
  - **SQLException:** This class handles any errors that occur in a database application.

Creating and Executing Queries

- **S**tructured **Q**uery **L**anguage (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.

- SQL is supported by almost any database that is used, and it allows to write database code independently of the underlying database.

- **Create Database**

```
CREATE DATABASE EMP;
```

- **Drop Database**

```
DROP DATABASE DATABASE_NAME;
```

- **Create Table**

  CREATE TABLE Employees (id INT NOT NULL, age INT NOT NULL, first VARCHAR(255), last VARCHAR(255), PRIMARY KEY (id));

- **Drop Table**

  DROP TABLE table_name;

- **INSERT Data**

  INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');

- **SELECT Data**

  SELECT first, last, age FROM Employees WHERE id = 100;

- **UPDATE Data**

  UPDATE Employees SET age=20 WHERE id=100;

- **DELETE Data**

  DELETE FROM Employee WHERE id=100;

- **Six steps in creating a JDBC application:**

1. **Import the packages:** *import java.sql.**

2. **Register the JDBC driver:**

   To open a communication channel with the database.

   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //MS Access

   [or Class.forName("org.apache.derby.jdbc.ClientDriver");]//netbeans

3. **Open a connection:**

   Connection connect = DriverManager.getConnection("jdbc:odbc:sql");

   [or Connection connect=DriverManager.getConnection("jdbc:derby://localhost:1527/Test1", "Test", "Test");]

4. **Execute a query:** build and submit an SQL statement

5. **Extract data from result set:** Use appropriate *ResultSet.getXXX()* to retrieve the data from the result set

6. **Clean up the environment:** closing all database resources

```
import java.sql.*;
public class FirstExample
{   public static void main(String[] args)
   {    Connection connect = null;
        try
        {    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
             connect = DriverManager.getConnection("jdbc:odbc:sql");
             PreparedStatement   pstm=connect.prepareStatement("SELECT id, first, last, age
FROM Employees");
             ResultSet rs =pstm.executeQuery();
             while(rs.next())
             {    int id = rs.getInt("id");                int age = rs.getInt("age");
                  String first = rs.getString("first");      String last = rs.getString("last");
                  System.out.print("ID: " +id+", Age: "+age+", First: "+first+", Last: "+ last);
             }
             rs.close();   pstm.close();   connect.close();
         }
         catch(SQLException se)
           {     System.out.println("SQL EXCEPTION OCCURRED");}

        }
}
```

Sample Code – select *

```
void viewall()
{
        try
        {
                PreparedStatement pstm=connect.prepareStatement("select * from Mytab");
                ResultSet rs=pstm.executeQuery ();
                while (rs.next())
                {     // Roll & Name are fields in database
                        System.out.println(rs.getInt("Roll")+ " "+rs.getString("Name"));
                }
        }
        catch(Exception e){}
}
```

Sample Code - Search by ID

```
void search()
{
        try
        {
                int r=12;
                PreparedStatement     pstm=connect.prepareStatement("select  *  from  Mytab
                where Roll=?");
                pstm.setInt(1,r);
                ResultSet rs=pstm.executeQuery ();
                while (rs.next())
                {
                        int roll = rs.getInt("Roll");
                        String sname = rs.getString("Name");
                        System.out.println(roll+ " "+sname);
                }
        }
        catch(Exception e){}
}
```

Sample Code - insert

```
void addstudent(int rollno, String name )
{
	try
	{
		PreparedStatement pstm=connect.prepareStatement("insert into Mytab
		(Roll,Name)values(?,?)");
		pstm.setInt(1,rollno);
		pstm.setString(2,name);
		pstm.executeUpdate();
	}
	catch(Exception e){}
}
```

Sample Code - update

```
void edit(int roll, int roll_edit, String name_edit)
{
	try
	{

		PreparedStatement   pstm  =connect.prepareStatement("update  Mytab  set
		Roll=?,Name=? where Roll=?");
		pstm.setInt(1,roll_edit);
		pstm.setString(2,name_edit);
		pstm.setInt(3,roll);
		pstm.executeUpdate();
	}
	catch(Exception e){}
}
```

Sample Code - delete

```
void delete(int roll)
{
        try
        {
                PreparedStatement pstm=connect.prepareStatement("delete from Mytab  where
                Roll=?");
                pstm.setString(1,roll);
                pstm.executeUpdate();
        }
        catch(Exception e){}
}
```

Example: emp table with id and name exists in database do the operations selection,insertion and updation

```
import java.sql.*;
public class S4CSE {
   public static void main(String[] args) {
      Connection conn = null;
      PreparedStatement st1=null,st2=null,st3=null,st4=null;
      try{
         Class.forName("org.apache.derby.jdbc.ClientDriver");
          conn                                                                      =
DriverManager.getConnection("jdbc:derby://localhost:1527/netb","cinita","cinita");
          st1=conn.prepareStatement("select * from cinita.emp");
         ResultSet rs=st1.executeQuery();
         while(rs.next())
         {
            System.out.println("ID : "+rs.getInt(1)+"  NAME : "+rs.getString(2));
         }
System.out.println("INSERTION INTO DATABASE");
        st2=conn.prepareStatement("insert into cinita.emp(id,name) values(?,?)");
        st2.setInt(1,17);
        st2.setString(2,“wini");
        st2.executeUpdate();
        System.out.println("UPDATION INTO DATABASE");
        st3=conn.prepareStatement("update cinita.emp set name=? where id=15");
        st3.setString(1,"hhhh");
        st3.executeUpdate();
        System.out.println("DELETION OF ROW");
        st4=conn.prepareStatement("delete from cinita.emp where id=16");
        st4.executeUpdate();
      }catch(Exception e)
     {
        System.out.println("Not connected“+e);
       }
  }
 }
```

Dynamic Queries

- Dynamic SQL or Dynamic Query is a programming technique that enables one to build SQL statements dynamically at runtime.

- One can create more general purpose, flexible applications by using dynamic SQL because the full text of a SQL statement may be unknown at compilation.
  - For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.
  - Applications that allow users to input or choose query search or sorting criteria at runtime

- It is basically like assembling SQL query segments together based on input entered.

<pseudocode>

```
select_clause = 'SELECT '
from_clause = 'FROM '
where_clause = 'WHERE '
orderby_clause = 'ORDER BY '

if [query on person] then
  select_clause += 'p.name, p.dob '
  from_clause += 'person p '
  orderby_clause += 'p.name '

  if [query on address] then
    select_clause += 'a.address_text '
    from_clause += ', address a '
    where_clause += 'p.address_id = a.id AND a.id=:p1 '
  else
    where_clause += 'p.id=:p1'
  end if

end if

sql_stmt = select_clause + from_clause + where_clause + orderby_clause + ';'
```

TUTORIAL 11

Q1 a. A table student exists in database that contain fields rollno, name and marks. Write a java program to do the following operations:

update name of student with rollno 12 to neethu,
delete(name="cini") and display students with marks>70.

- Q1 b: Calculate the rank of students based on total marks and display details with the rank.