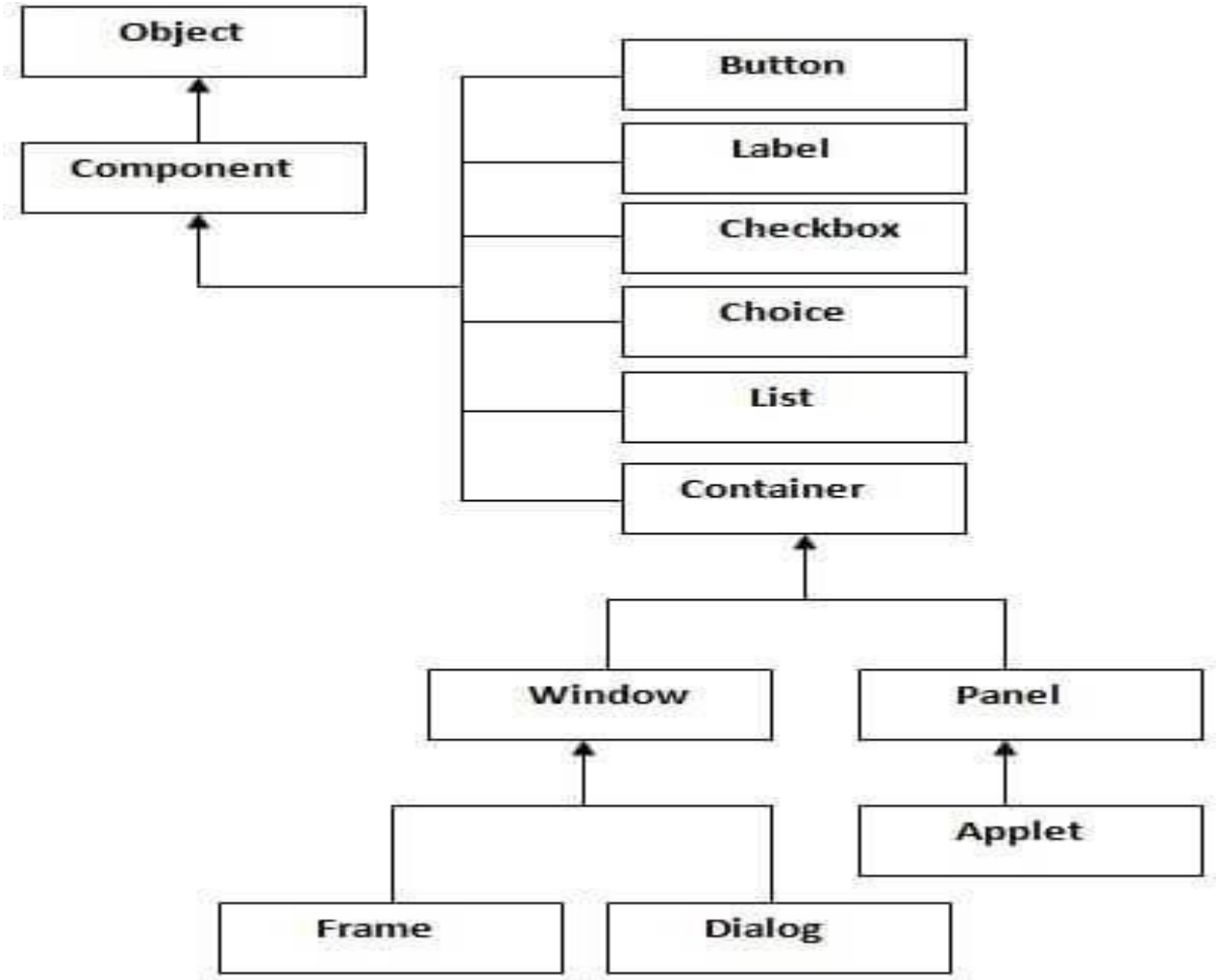
Unit -V
AWT Controls



AWT Controls:

Controls are components that allow a user to interact with our application.

The AWT supports the following types of controls:

- Labels
- Buttons
- Check boxes
- Choice lists
- Lists
- Scroll bars
- Text editing

All these controls are subclasses of Component class.

Adding Controls:

Steps to add a control to a container (Applet or Frame):

1. Create an instance of the desired control.
2. Add it to the container by calling the add method defined in Container class.

Component add(Component obj)

obj is the instance of the control.

Once a control is added, it will automatically visible on the container.

Removing Controls:

To remove a control, Container class defined a method:

void remove(Component obj)

Label:

Labels are passive controls that do not support any interaction with the user.

Label class Constructors:

Label( )

Label(String str)

Label(String str, int how)

str is the text of the Label

how specifies the alignment

Label.LEFT

Label.RIGHT

Label.CENTER

Methods:

void setText(String str)

String getText( )

void setAlignment(int how)

int getAlignment( )

Button:

A button (push button) is a component that contains a label and that generates an event when it is pressed.

Button class Constructors:

Button( )

Button(String str)

Methods:

void setLabel(String str)

String getLabel( )

Checkbox:

A check box is a control that is used to turn an option on or off (check mark).

Each time a check box is selected or deselected, an item event is generated.

Checkbox class Constructors:

Checkbox( )

Checkbox(String str)

Checkbox(String str, boolean on)

Checkbox(String str, boolean on, CheckboxGroup cbg)

Checkbox(String str, CheckboxGroup cbg, boolean on)

Checkbox methods:

void setLabel(String str)

String getLabel( )

void setState(boolean on)
boolean getState( )

CheckboxGroup methods:

Checkbox getSelectedCheckbox( )
void setSelectedCheckbox(Checkbox cb)

Choice:

The Choice is used to create a pop-up list of items.
Each time a choice is selected, an item event is generated.
Choice class contains only the default constructor.

Choice methods:

void addItem(String name)
void add(String name)
String getSelectedItem( )
int getSelectedIndex( )
int getItemCount( )
void select(int index)
void select(String name)
String getItem(int index)

List:

The List provides a compact, multiple-choice, scrolling selection list.
A list shows a number of choices in the visible widow. It also allow multiple selections.

List constructors:

List( )
List(int numRows)
List(int numRows, boolean multipleSelect)

List methods:

void add(String name)
void add(String name, int index)
String getSelectedItem( )
int getSelectedIndex( )
String[ ] getSelectedItems( )
int[ ] getSelectedIndexes( )
int getItemCount( )
void select(int index)
String getItem(int index)

In List:

Each time an item in the list is selected or deselected with a single click, an item event is generated.
When a list item is double-clicked, an action event is generated.

Scrollbar:

Scrollbars are used to select continuous values between a specified minimum and maximum.

Scrollbar constructors:

Scrollbar( )
Scrollbar(int style)
Scrollbar(int style, int initialValue, int thumbSize, int min, int max)

Style can be,
Scrollbar.VERTICAL
Scrollbar.HORIZONTAL

Scrollbar methods:

void setValues(int initialValue, int thumbSize, int min, int max)
int getValue( )
void setValue(int newValue)
int getMinimum( )
int getMaximum( )
void setUnitIncrement(int newIncr)
void setBlockIncrement(int newIncr)

TextField:

TextField is a single line text entry area.
TexField is subclass of TextComponent.

TextField constructors:

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

TextField methods:

String getText( )
void setText(String str)
String getSelectedText( )
void select(int startIndex, int endIndex)
boolean isEditable( )
void setEditable(boolean canEdit)
int getSelectionStart( )
int getSelectionEnd( )
void setEchoChar(char ch)
boolean echoCharIsSet( )
char getEchoChar( )

TextArea:

TextArea is a multiline editor.
TextArea is subclass of TextComponent.

TextArea constructors:

TextArea( )
TextArea(int numLines, int numChars)
TextArea(String str)
TextArea(String str, int numLines, int numChars)
TextArea(String str, int numLines, int numChars, int scrBars)

scrBars can be

SCROLLBARS_BOTH
SCROLLBARS_NONE
SCROLLBARS_HORIZONTAL_ONLY
SCROLLBARS_VERTICAL_ONLY

TextArea methods:

String getText( )
void setText(String str)
String getSelectedText( )
void select(int startIndex, int endIndex)
boolean isEditable( )
void setEditable(boolean canEdit)
int getSelectionStart( )
int getSelectionEnd( )
void append(String str)
void insert(String str, int index)
void replaceRange(String str, int startIndex, ine endIndex)

Menu Bars and Menus:

Menus in Java are implemented by the use of the following classes:

MenuBar
Menu
MenuItem
CheckboxMenuItem

To create a menu bar, just create an instance on MenuBar class.
MenuBar class defines only the default constructor.

Menu:

Constructors:

Menu( )
Menu(String optionName)
Menu(String optionName, boolean removable)

MenuItem:

Constructors:

MenuItem( )
MenuItem(String itemName)
MenuItem(String itemName, MenuShortcut keyAccel)

CheckboxMenuItem:

Constructors:

CheckboxMenuItem( )
CheckboxMenuItem(String itemName)
CheckboxMenuItem(String itemName, boolean on)

MenuShortcut:

Constructors:

MenuShortcut(int key)

MenuShortcut(int key, boolean useShiftMidifier)

Event and Listener (Java Event Handling)

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

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener

KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

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

- public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java event handling by implementing ActionListener

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

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

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

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);

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


}
public void actionPerformed(ActionEvent e){
tf.setText("ANURAG UNIVERSITY");
```

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

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.

Java AWT Button

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

AWT Button Class declaration

public class Button **extends** Component **implements** Accessible

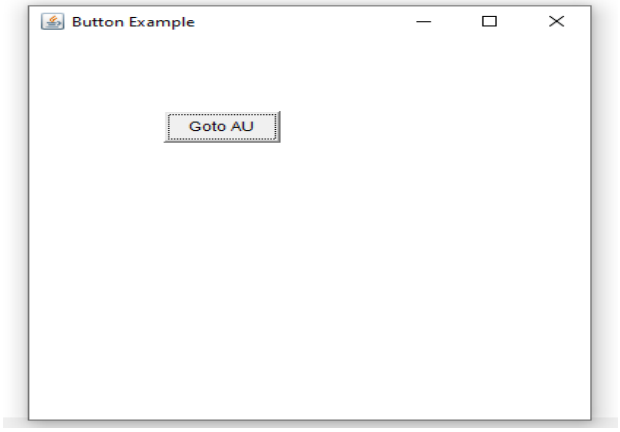
Java AWT Button Example

```
//AWT Button Class declaration

import java.awt.*;
import java.awt.event.*;

public class ButtonExample {
public static void main(String[] args) {
   Frame f=new Frame("Button Example");
   Button b=new Button("Goto AU");
   b.setBounds(100,100,80,30);
   f.add(b);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
         f.addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e){
         System.exit(0);
         }
         });
}
}
```

OUTPUT



Java AWT Label

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

AWT Label Class Declaration

public class Label **extends** Component **implements** Accessible

Java Label Example

//AWT Label Class declaration

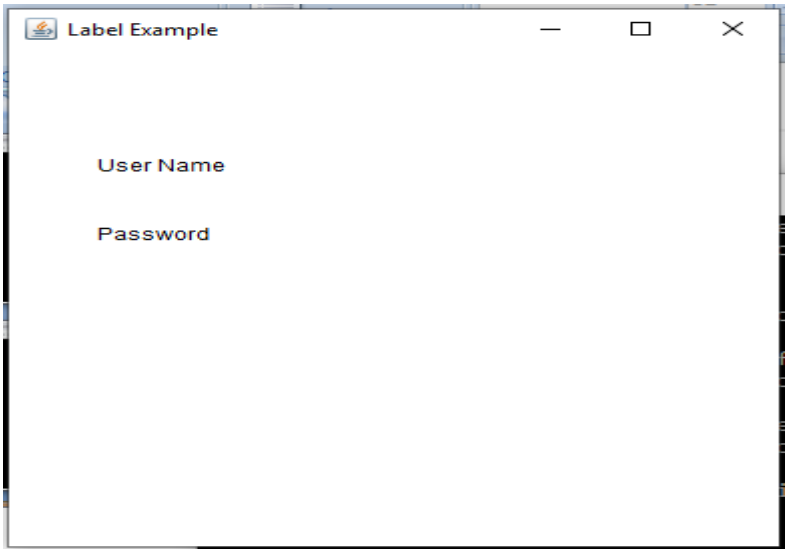
```
import java.awt.*;
import java.awt.event.*;

class LabelExample{
public static void main(String args[]){
   Frame f= new Frame("Label Example");
   Label l1,l2;
   l1=new Label("User Name");
   l1.setBounds(50,100, 100,30);
   l2=new Label("Password");
   l2.setBounds(50,150, 100,30);
   f.add(l1); f.add(l2);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);

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

}
}
```

Output:



Java AWT TextField

The object of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

AWT TextField Class Declaration

public class TextField **extends** TextComponent

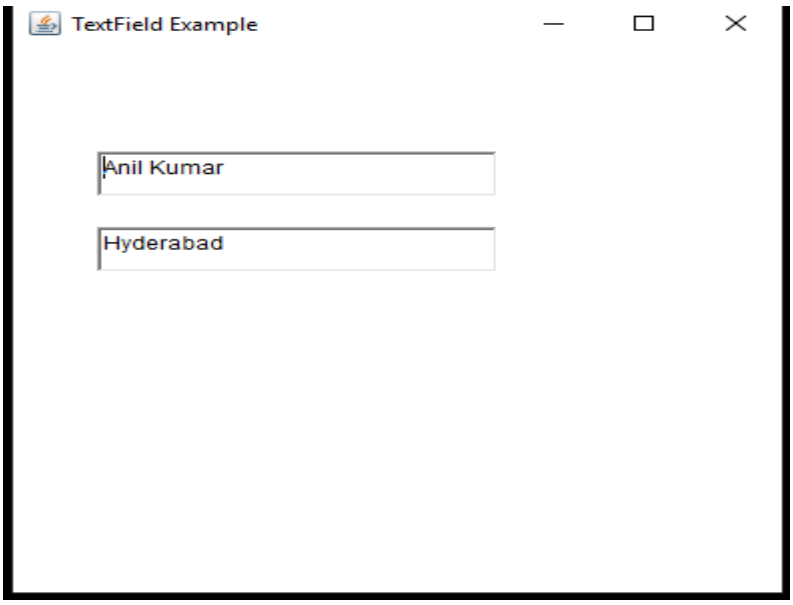
Java AWT TextField Example

```
import  java.awt.*;
import java.awt.event.*;
class TextFieldExample{
public static void main(String args[]){
   Frame f= new Frame("TextField Example");
   TextField t1,t2;
   t1=new TextField("Anil Kumar");
   t1.setBounds(50,100, 200,30);
   t2=new TextField("Hyderabad");
   t2.setBounds(50,150, 200,30);
   f.add(t1); f.add(t2);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);

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

}
}
```

Output:



Java AWT TextArea

The object of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

AWT TextArea Class Declaration

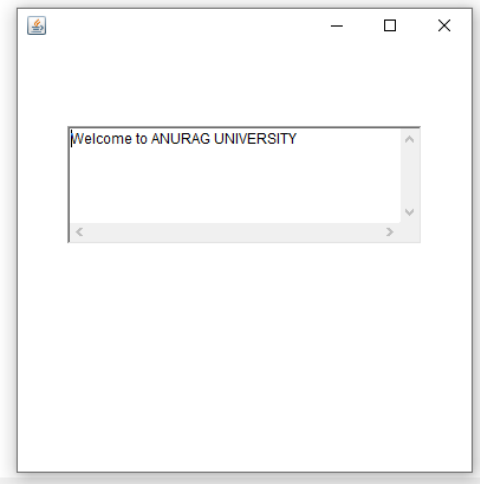
public class TextArea **extends** TextComponent

Java AWT TextArea Example

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

public class TextAreaExample
{
    TextAreaExample(){
      Frame f= new Frame();
      TextArea area=new TextArea("Welcome to ANURAG UNIVERSITY");
        area.setBounds(50,100, 300,100);
      f.add(area);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent e){
System.exit(0);
}
});
    }
public static void main(String args[])
{
  new TextAreaExample();
}
}
```

OUTPUT:



Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

AWT Checkbox Class Declaration

public class Checkbox **extends** Component **implements** ItemSelectable, Accessible

Java AWT Checkbox Example

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

public class CheckboxExample
{
    CheckboxExample(){
      Frame f= new Frame("Checkbox Example");
       Checkbox checkbox1 = new Checkbox("C++");
       checkbox1.setBounds(100,100, 50,50);
       Checkbox checkbox2 = new Checkbox("Java", true);
       checkbox2.setBounds(100,150, 50,50);
       Checkbox checkbox3 = new Checkbox("Python");
       checkbox3.setBounds(100,200, 70,50);
       Checkbox checkbox4 = new Checkbox("Haskell", true);
       checkbox4.setBounds(100,250,70,50);
       f.add(checkbox1);
       f.add(checkbox2);
       f.add(checkbox3);
       f.add(checkbox4);
       f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
          f.addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
          System.exit(0);
          }
          });
    }
public static void main(String args[])
{
  new CheckboxExample();
}
}
```

Output:

Java AWT CheckboxGroup

The object of CheckboxGroup class is used to group together a set of **Checkbox**. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the **object class**.

Note: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

AWT CheckboxGroup Class Declaration

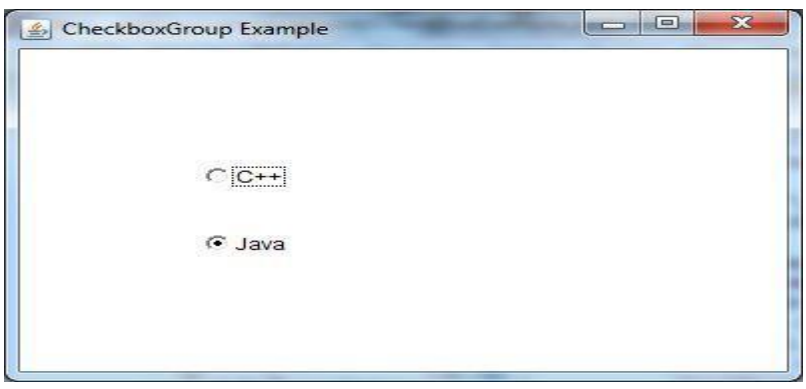
public class CheckboxGroup **extends** Object **implements** Serializable

Java AWT CheckboxGroup Example

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

public class CheckboxGroupExample
{
      CheckboxGroupExample(){
      Frame f= new Frame("CheckboxGroup Example");
       CheckboxGroup cbg = new CheckboxGroup();
       Checkbox checkBox1 = new Checkbox("C++", cbg, false);
       checkBox1.setBounds(100,100, 50,50);
       Checkbox checkBox2 = new Checkbox("Java", cbg, true);
       checkBox2.setBounds(100,150, 50,50);
       f.add(checkBox1);
       f.add(checkBox2);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
       f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent e){
System.exit(0);
            }
            });
    }
public static void main(String args[])
{
 new CheckboxGroupExample();
}
}
```

Output:



Java AWT Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

AWT Choice Class Declaration

public class Choice **extends** Component **implements** ItemSelectable, Accessible

Java AWT Choice Example

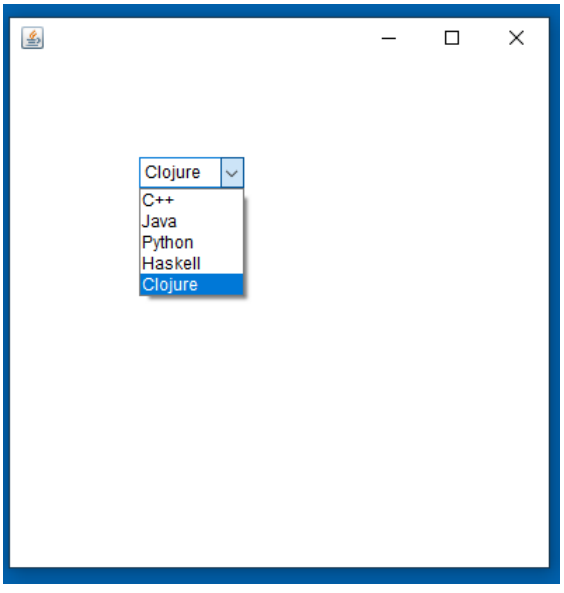
```
import java.awt.*;
import java.awt.event.*;

public class ChoiceExample
{
      ChoiceExample(){
      Frame f= new Frame();
      Choice c=new Choice();
      c.setBounds(100,100, 75,75);
      c.add("C++");
      c.add("Java");
      c.add("Python");
      c.add("Haskell");
      c.add("Clojure");
      f.add(c);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);

         f.addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e){
         System.exit(0);
         }
         });
    }
public static void main(String args[])

{
new ChoiceExample();
}
}
```

Output:



Java AWT List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

AWT List class Declaration

public class List **extends** Component **implements** ItemSelectable, Accessible

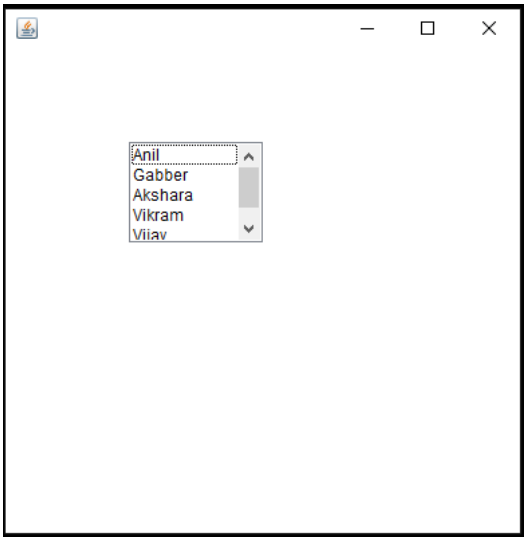
Java AWT List Example

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

public class ListExample
{
    ListExample(){
      Frame f= new Frame();
      List l1=new List(5);
      l1.setBounds(100,100, 100,75);
      l1.add("Anil");
      l1.add("Gabber");
      l1.add("Akshara");
      l1.add("Vikram");
      l1.add("Vijay");
      f.add(l1);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);


          f.addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
          System.exit(0);
          }
          });
    }
public static void main(String args[])
{
new ListExample();
}
}
```

Output:



Java AWT Canvas

The Canvas control represents a blank rectangular area where the application can draw or trap input events from the user. It inherits the Component class.

AWT Canvas class Declaration

public class Canvas **extends** Component **implements** Accessible

Java AWT Canvas Example

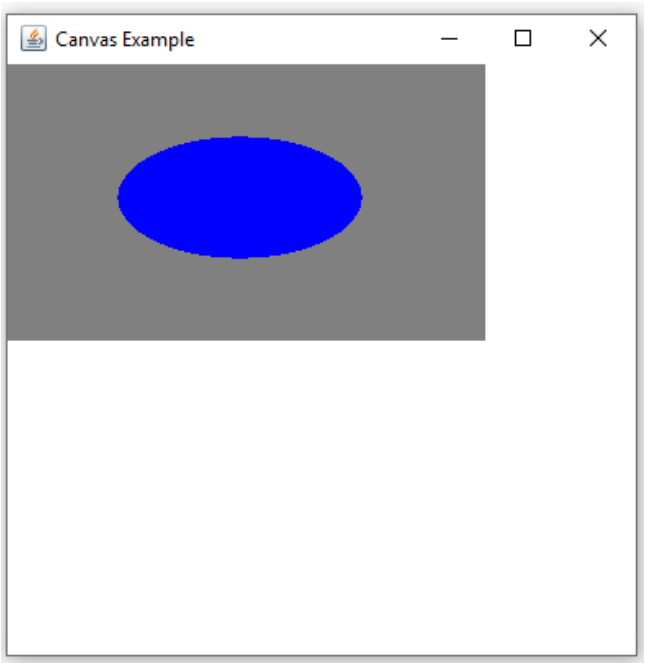
```
import java.awt.*;
import java.awt.event.*;

public class CanvasExample
{
  public CanvasExample()
  {
    Frame f= new Frame("Canvas Example");
    f.add(new MyCanvas());
    f.setLayout(null);
    f.setSize(400, 400);
    f.setVisible(true);

            f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
            System.exit(0);
            }
            });
     }
public static void main(String args[])
{
new CanvasExample();
}
}

class MyCanvas extends Canvas
{
        public MyCanvas() {
        setBackground (Color.GRAY);
        setSize(300, 200);
     }
  public void paint(Graphics g)
  {
    g.setColor(Color.blue);
    g.fillOval(75, 75, 150, 75);
  }
}
```

Output:



Java AWT Scrollbar

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

AWT Scrollbar class declaration

public class Scrollbar **extends** Component **implements** Adjustable, Accessible

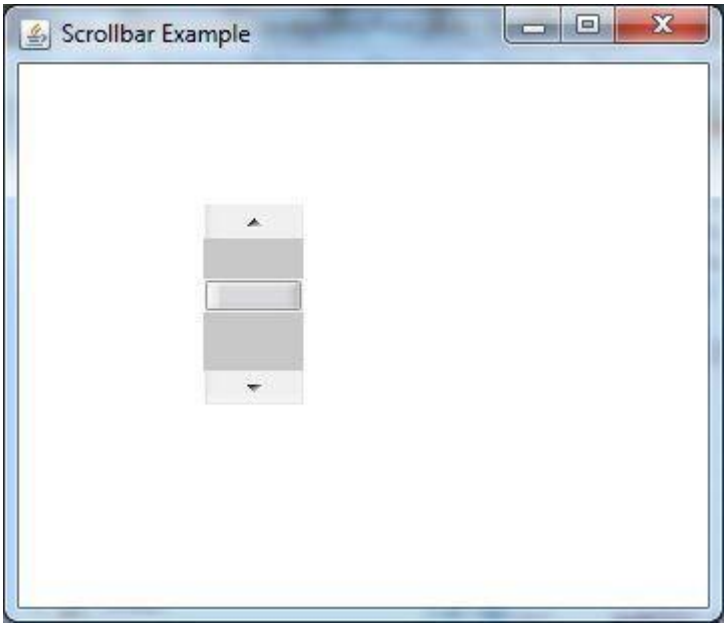
Java AWT Scrollbar Example

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

class ScrollbarExample{
ScrollbarExample(){
        Frame f= new Frame("Scrollbar Example");
        Scrollbar s=new Scrollbar();
        s.setBounds(100,100, 50,100);
        f.add(s);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
        System.exit(0);
        }
        });
    }
public static void main(String args[])
{
new ScrollbarExample();

}
}
```

Output:



Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

AWT MenuItem class declaration

public class MenuItem **extends** MenuComponent **implements** Accessible

AWT Menu class declaration

public class Menu **extends** MenuItem **implements** MenuContainer, Accessible

Java AWT MenuItem and Menu Example

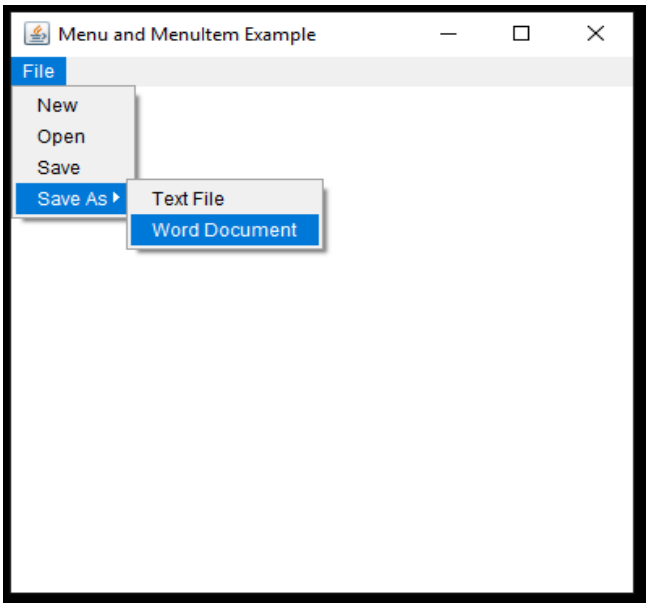
```
import java.awt.*;
import java.awt.event.*;

class MenuExample
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("File");

        Menu submenu=new Menu("Save As");
        MenuItem i1=new MenuItem("New");
        MenuItem i2=new MenuItem("Open");
        MenuItem i3=new MenuItem("Save");
        MenuItem i4=new MenuItem("Text File");
        MenuItem i5=new MenuItem("Word Document");
        menu.add(i1);
        menu.add(i2);
```

```
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
          f.addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
          System.exit(0);
          }
          });
    }
public static void main(String args[])
{
new MenuExample();
}
}
```

Output:



Java AWT PopupMenu

PopupMenu can be dynamically popped up at specific position within a component. It inherits the Menu class.

AWT PopupMenu class declaration

public class PopupMenu **extends** Menu **implements** MenuContainer, Accessible

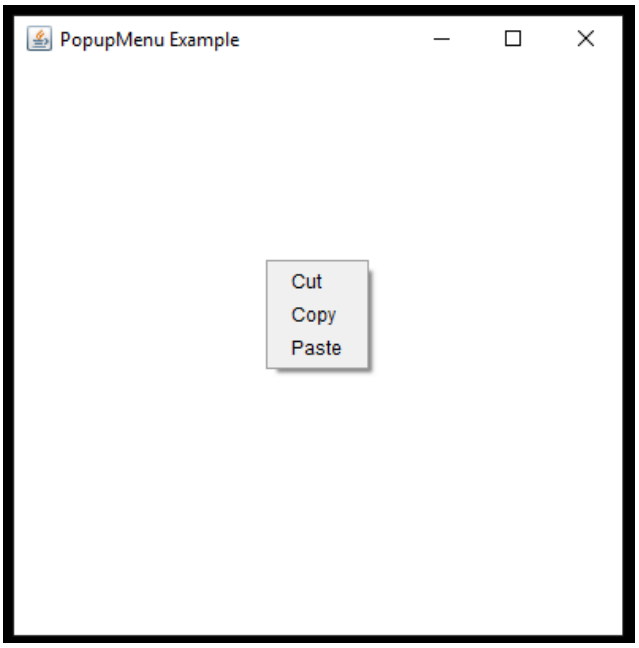
Java AWT PopupMenu Example

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

class PopupMenuExample
{
    PopupMenuExample(){
        final Frame f= new Frame("PopupMenu Example");
        final PopupMenu popupmenu = new PopupMenu("Edit");
        MenuItem cut = new MenuItem("Cut");
        cut.setActionCommand("Cut");
        MenuItem copy = new MenuItem("Copy");
        copy.setActionCommand("Copy");
        MenuItem paste = new MenuItem("Paste");
        paste.setActionCommand("Paste");
        popupmenu.add(cut);
        popupmenu.add(copy);
        popupmenu.add(paste);
        f.addMouseListener(new MouseAdapter() {
          public void mouseClicked(MouseEvent e) {
             popupmenu.show(f , e.getX(), e.getY());
          }
        });
        f.add(popupmenu);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);


          f.addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
          System.exit(0);
          }
          });
    }
public static void main(String args[])
{
new PopupMenuExample();
}
}
```

Output:



APPLET

Applet Definition:

- Applets are small Java applications that can be accessed on an Internet server, transported over Internet, and can be automatically installed and run as a part of a web document.
- After a user receives an applet, the applet can produce a graphical user interface. It has limited access to resources so that it can run complex computations without introducing the risk of viruses or breaching data integrity.
- Any applet in Java is a class that extends the java.applet.Applet class.
- An Applet class does not have any main() method. It is viewed using JVM. The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.
- JVM creates an instance of the applet class and invokes **init()** method to initialize an Applet.

An applet is a Java program designed to be included in an HTML Web document. You can write your Java applet and include it in an HTML page. When you use a Java-enabled browser to view an HTML page that contains an applet, the applet's code is transferred to your system and is run by the browser's Java virtual machine.

The HTML document contains tags, which specify the name of the Java applet and its Uniform Resource Locator (URL). The URL is the location at which the applet bytecodes reside on the Internet. When an HTML document containing a Java applet tag is displayed, a Java-enabled Web browser downloads the Javabytecodes from the Internet and uses the Java virtual machine to process the code from within the Web document. These Java applets are what enable Web pages to contain animated graphics or interactive content.

Difference between Applet and Application in Java:

Sr.No.	Characteristics	Java Application	Java Applet
1.	Definition	An application is a standalone Java program that can be run independently on a client/server without the need for a web browser.	An applet is a form of Java program which is embedded with an HTML page and loaded by a web server to be run on a web browser.
2.	main() method	The execution of the program starts from the main() method.	There is no requirement of main() method for the execution of the program.
3.	Access Restrictions	The application can access local disk files/ folders and the network system.	The applet doesn't have access to the local network files and folders.
4.	GUI	It doesn't require any Graphical User Interface (GUI).	It must run within a GUI.
5.	Security	It is a trusted application and doesn't require much security.	It requires high-security constraints as applets are untrusted.
6.	Environment for Execution	It requires Java Runtime Environment (JRE) for its successful execution.	It requires a web browser like Chrome, Firefox, etc for its successful execution.
7.	Installation	It is explicitly run and installed on a local system. An applet doesn't have access to local files and so it cannot perform read and write operations on files stored on the local disk.	It doesn't require any explicit installation to be done.
8.	Read/ Write Operation	An application can perform read and write operations on files stored on the local disk.	An applet doesn't have access to local files so you cannot perform read and write operations on files stored on the local disk.

Applications of Java Applet

The **Applets** are used to provide interactive features to web **applications** that cannot be provided by HTML alone.
They can capture mouse input and also have controls like buttons or check boxes. In response to user actions,
an **applet** can change the provided graphic content.

Advantages of Applets

1. It takes very less response time as it works on the client side.
2. It can be run on any browser which has JVM running in it.

Applet class

Applet class provides all necessary support for applet execution, such as initializing and destroying of applet.It also provide methods that load and display images and methods that load and play audio clips

Lifecycle/ An Applet Skeleton

Most applets override these four methods. These four methods forms Applet lifecycle.

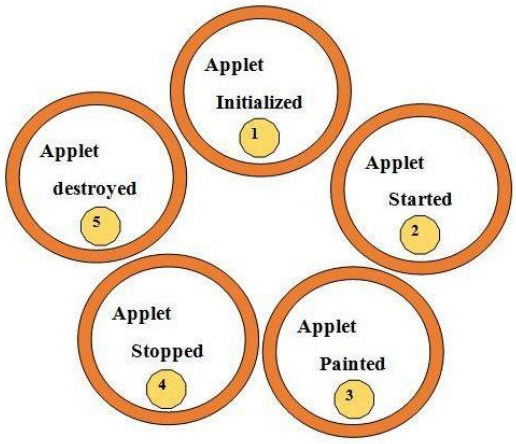
- **init() :** init() is the first method to be called. This is where variable are initialized.

  This method iscalled only once during the runtime of applet.
- **start() :** start() method is called after init(). This method is called to restart an applet

  after it has beenstopped.
- **stop() :** stop() method is called to suspend thread that does not need to run when applet is not visible.
- **destroy() :** destroy() method is called when your applet needs to be removed

  completely frommemory.

Note: The stop() method is always called before destroy() method.



Applet and AWT:
To create an applet first write the program and Save the file with name
Sample Applet Program:

```
import java.applet.*;
import java.awt.*;
/*
<applet code="Sample1" width=500 height=500>
</applet>
*/
public class Sample1 extends Applet {
  String msg;
  public void init( ) {
    msg="Hello";
  }
  public void start( ) {
    msg+="Start";
  }
  public void paint(Graphics g) {
    msg+="paint";
    g.drawString(msg,50,50);
  }
}
```

HTML program to run an Applet on Webbrowser:

```
<!– Simple html Program -->
<html>
  <head>
    <title> The Life Cycle of an Applet </title>
  </head>
  <body>
    <applet code=“Sample1" width="500“  height=“500">
    </applet>
  </body>
</html>
```

Alternate way to run an Applet:

C:\...\>appletviewer Sample1.java

The <APPLET> Tag:

```
< APPLET
[CODEBASE = codebaseURL]
CODE = appletFile
[ALT = alternateText]
[NAME = appletInstanceName]
```

WIDTH = pixels
HEIGHT = pixels
[ALIGN = alignment]
[VSPACE = pixels]
[HSPACE = pixels]
>
[< PARAM NAME = appletParameter1 VALUE = value >]
[< PARAM NAME = appletParameter2 VALUE = value >]
. . .
[alternateHTML]
</APPLET>

Methods of Applet class:

Applet class is in java.applet .*; package

Some methods:

void init()

void start()

void stop()

void destroy()

URL getCodeBase()

URL getDocumentBase()

String getAppletInfo()

String getParameter(String pname)

String [ ] [ ] getParameterInfo()

boolean isActive()

void showStatus(String str)

void resize(int width, int height)

void resize(Dimension d)

Passing parameters to applet

write a program to illustrate passing parameters to applet.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletDemo" width="400" height="200">
<param name="Name" value="Anurag University">
<param name="Dept" value="AIML">
<param name="Year" value="2">
<param name="Sem" value="1">
<param name="Place" value="Hyderabad">
<param name="State" value="Telangana">
</applet>*/
```

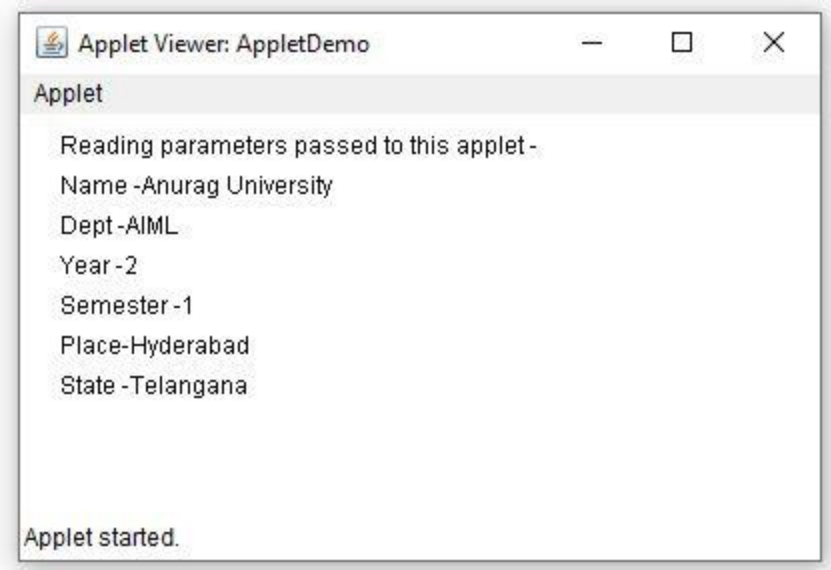
```
public class AppletDemo extends Applet
{
String name;
String dept;
String year;
String sem;
String place;
String st;
public void init()
{
name = getParameter("Name");
dept = getParameter("Dept");
year = getParameter("Year");
sem = getParameter("Sem");
place = getParameter("Place");
st = getParameter("State");
}
public void paint(Graphics g)
{
g.drawString("Reading parameters passed to this applet -", 20, 20);
g.drawString("Name -" + name, 20, 40);
g.drawString("Dept -" + dept, 20, 60);
g.drawString("Year -" + year, 20, 80);
g.drawString("Semester -" + sem, 20, 100);
g.drawString("Place-" + place, 20, 120);
g.drawString("State -" +st, 20, 140);
}
}
```

:

Output:

1. Compile: javac AppletDemo.java

2. Run: appletviewer AppletDemo.java



Working with colors:

To set colors for a component, Component class have defined two methods:

void setBackground(Color newColor)

void setForeground(Color newColor)

Color class defines constants and constructors to work with colors.

Color class:

Constants defined in Color class are:

Color.black

Color.blue

Color.cyan

Color.gray

Color.darkGray

Color.lightGray

Color.green

Color.magenta

Color.orange

Color.pink

Color.red

Color.white

Color.yellow

Constructors defined in Color class are

Color(int red, int green, int blue)

Color(int rgbValue)

Color(float red, float green, float blue)

Methods defined in Color class are:

int getRed()

int getGreen()

int getBlue()

int getRGB()

Color.yellow

Graphics class:

Graphics class is in java.awt package.

Methods:

void drawLine(int startX, int startY, int endX, int endY)

void drawRect(int top, int left, int width, int height)

void fillRect(int top, int left, int width, int height)

void drawRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)

void fillRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)

void drawOval(int top, int left, int width, int height)

void fillOval(int top, int left, int width, int height)

void drawArc(int top, int left, int width, int height, int startAngle, int sweepAngle)

void fillArc(int top, int left, int width, int height, int startAngle, int sweepAngle)

void drawPolygon(int x[ ], int y[ ], int numPoints)

void setColor(Color newColor)

void fillPolygon(int x[ ], int y[ ], int numPoints)

Color getColor( )

Working with fonts:

To set a different font for a component,

Component class have defined one method:

void setFont(Font fontObj)

Font class:

The general form of Font class constructor is:

Font(String fontName, int fontStyle, int pointStyle)

fontName can be Dialog, DialogInput, Sans Serif, Serif, Monospaced & Symbol.

fontStyle can be any of the constants: Font.PLAIN, Font.BOLD, Font.ITALIC.

fontStyle can be combine style in the form Font.PLAIN,Font.ITALIC.

Methods of Font class:

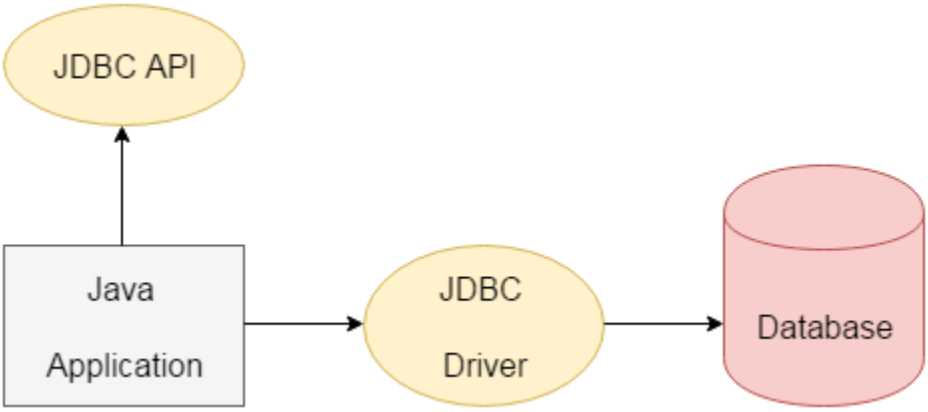
static Font decode(String str)
boolean equals(Object fontObj)
String getFamily()
static Font getFont(String pr)
static Font getFont(String pr,Font dfFont)
String getFontName()
String getName()
int getSize()
int getStyle()
int hashCode()
boolean isBold()
boolean isItalic()
boolean isPlain()
String toString()

JDBC Connectivity

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform

dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

JDBC Driver Types:

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

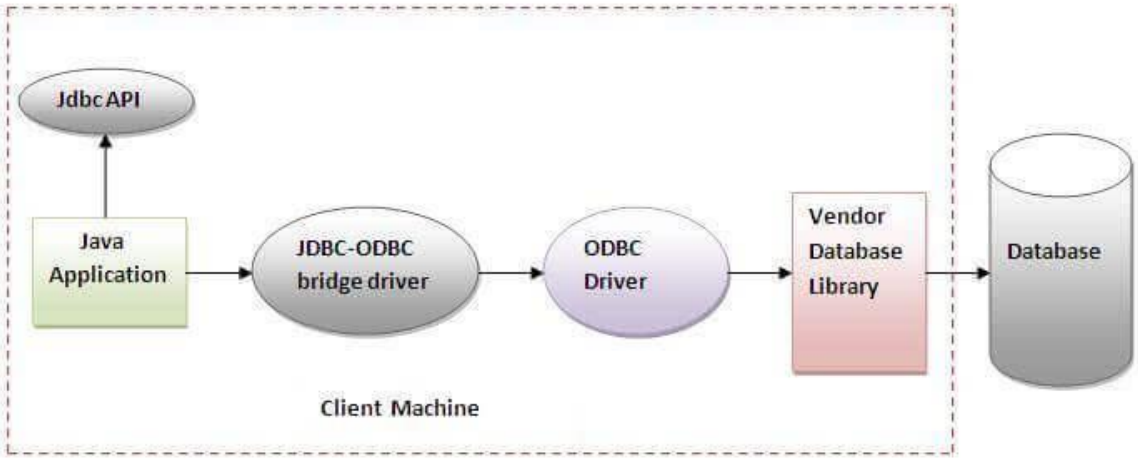


Figure- JDBC-ODBC Bridge Driver

NOTE: Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

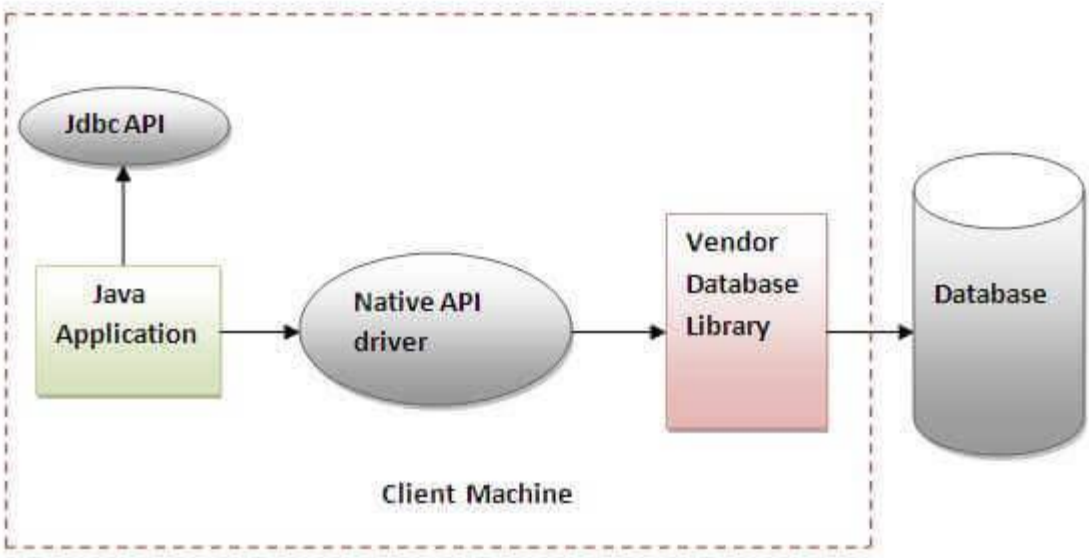


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.

- The Vendor client library needs to be installed on client machine.

---

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

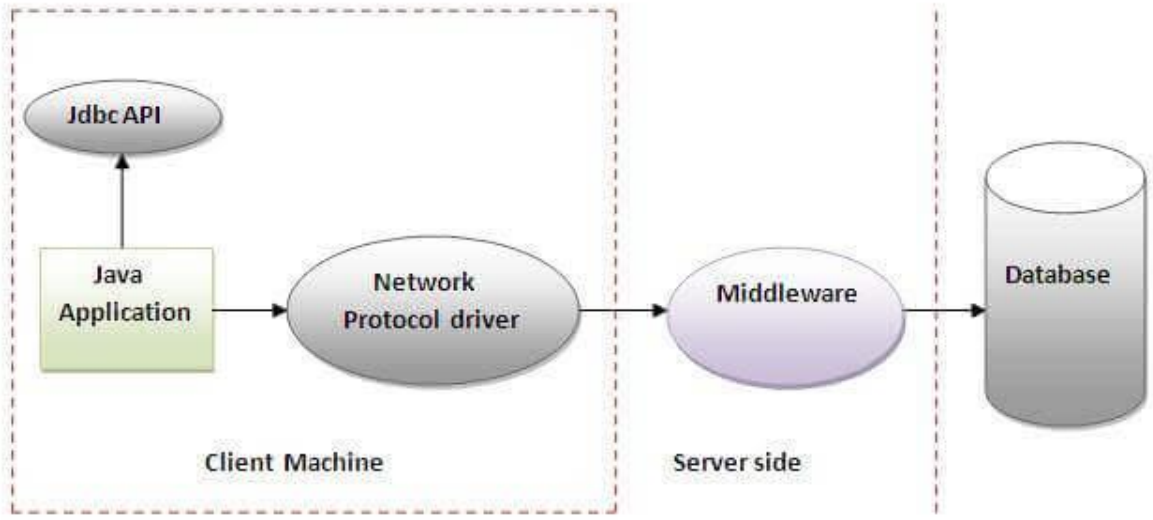


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

---

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known

as thin driver. It is fully written in Java language.

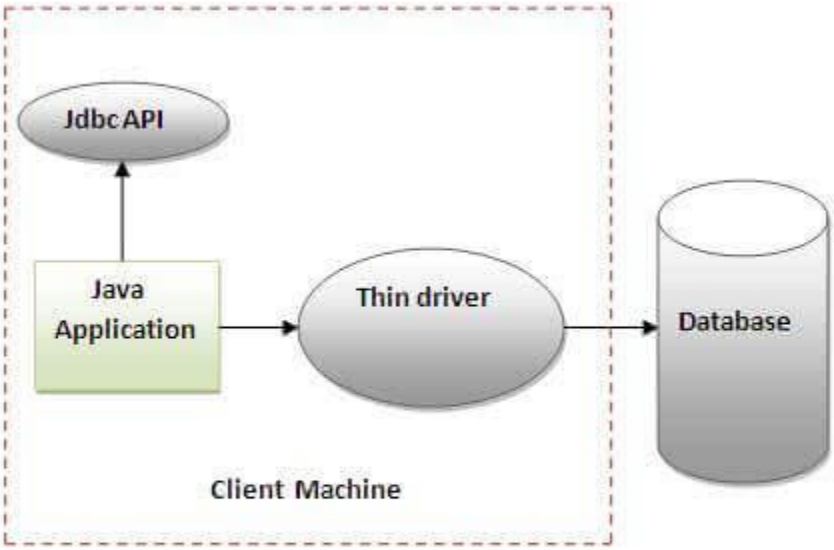


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

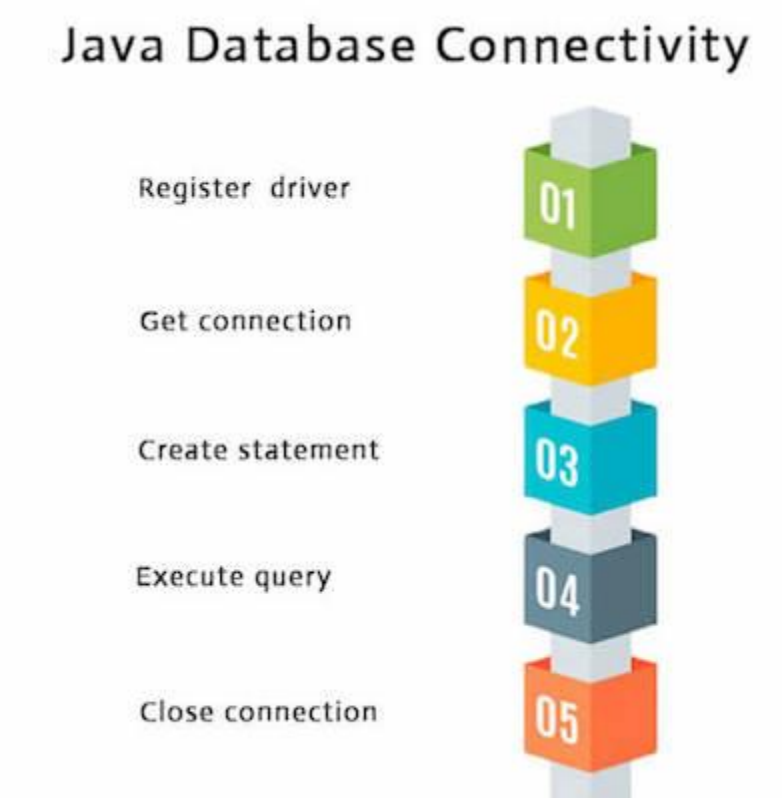
Disadvantage:

- Drivers depend on the Database.

Java Database Connectivity with 5 Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection



---

1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

1. **public static void** forName(String className)**throws** ClassNotFoundException

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

1. Class.forName("oracle.jdbc.driver.OracleDriver");

---

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1. 1) **public static** Connection getConnection(String url)**throws** SQLException
2. 2) **public static** Connection getConnection(String url,String name,String password)
3. **throws** SQLException

Example to establish connection with the Oracle database

1. Connection con=DriverManager.getConnection(

2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");

---

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

1. **public** Statement createStatement()**throws** SQLException

Example to create the statement object

1. Statement stmt=con.createStatement();

---

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute query

1. ResultSet rs=stmt.executeQuery("select * from emp");
2.
3. **while**(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }

---

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

1. **public void** close()**throws** SQLException

Example to close connection

con.close();

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.

2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database Anurag
2. use Anurag
3. create table emp(id **int**(10),name varchar(40),age **int**(3));
4.
   insert into emp values (1001, 'Smith', 56);
   insert into emp values (1001, 'Jhon', 45);
   insert into emp values (1001, 'Sunnah', 32);

5. select * from emp;

Example to Connect Java Application with mysql database

```
import java.sql.*;

class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection( "jdbc:mysql://localhost:3306/Anurag","root","r
oot");
        //here Anurag is database name, root is username and password
        Statement stmt=con.createStatement();
        ResultSet rs=stmt.executeQuery("select * from emp");
        while(rs.next())
        System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));
        con.close();
        }catch(Exception e){ System.out.println(e);}
```

```
}
}
```

The above example will fetch all the records of emp table.

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

download the jar file mysql-connector.jar

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2) Set classpath:

There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath

open command prompt and write:

1. C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;