UNIT - V

Applet

An applet is a java program that is embedded in a Web page to generate dynamic content.

The life cycle of an applet involves the following states

* Born State
* Running State
* Idle State
* Dead State

Born State

An applet enters the born state when it is first loaded This is achieved by calling the init() method of Applet class

```
public void init( )
{

}
```

Running State

An applet enters the running state when the system calls the start() method of Applet class

```
public void start( )
{

}
```

Idle state

An applet becomes idle when it is stopped from running.

An applet can be stopped by calling the stop( ) method explicitly.

```
public void stop( )
{

}
```

Dead state

An applet is said to be dead when it is removed from memory.

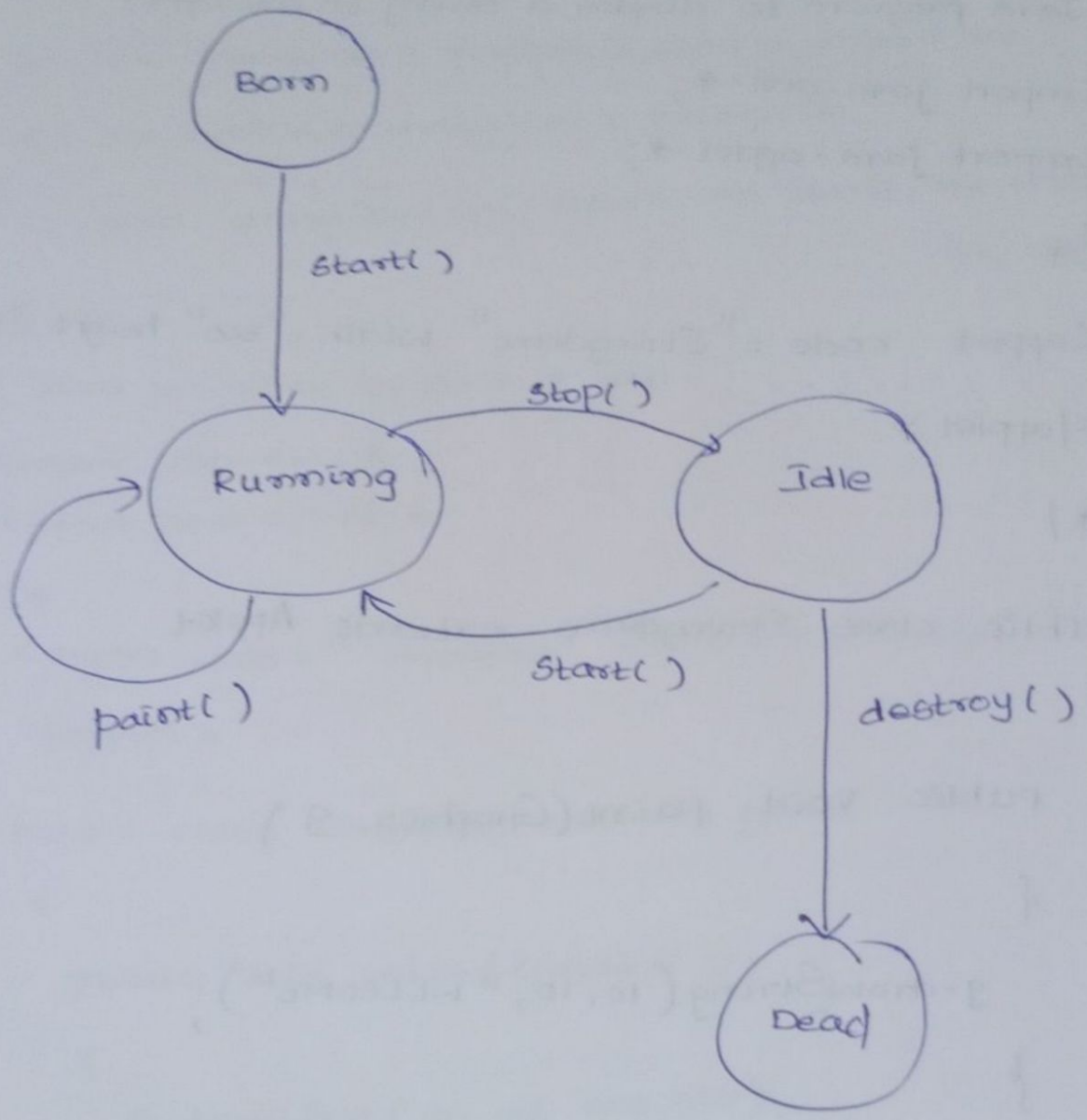
It occurs automatically by invoking the destroy( ) method when we quit the browser.

```
public void destroy( )
{

}
```

Display state

An applet enters into display state when the paint( ) method is called.

```
Public void paint(Graphics g)
{

}
```



Applet State transition Diagram.

```
// Java program to display a string in an applet
import java.awt.*;
import java.applet.*;
/*
<applet code = "Stringdemo" width = "500" height = "500">
</applet>
*/
public class Stringdemo extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString(10, 10, "WELCOME");
    }
}
```

Drawing a line

* The drawLine( ) method is used to draw a line.
* The syntax of drawLine( ) method is

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

E.g.

```
// Java program to draw a line
import java.awt.*;
import java.applet.*;
/*
<applet code = "Linedemo" width = "500" height = "500">
</applet>
public class Linedemo extends Applet
 {
   public void paint(Graphics g)
   {
     g.drawLine(100, 100, 400, 400);
   }
}
```

Drawing a Rectangle

The drawRect() method is used to draw a rectangle.

The syntax of drawRect() method is

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

E.g.

```
// Java program to draw a Rectangle
import java.awt.*;
import java.applet.*;
/*
<applet code="Rectdemo" width="500" height="500">
</applet>
*/
public class Rectdemo extends Applet
{
  public void paint(Graphics g)
  {
    g.drawRect(10, 10, 60, 50);
  }
}
```

Drawing a Filled Rectangle

* The fillRect() method is used to draw a filled Rectangle.
* The syntax of fillRect() method is

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

E.g.

```
// Java program for drawing a filled rectangle.
import java.awt.*;
import java.applet.*;
/*
<applet code = "Fillrectdemo" width = "500"
                                height = "500">

</applet>
*/
public class Fillrectdemo extends Applet
{
  public void paint(Graphics g)
  {
    g.  Rect(10, 10, 60, 50);
  }
}
```

Drawing a Rounded Rectangle

The drawRoundRect( ) method is used to draw a Rounded Rectangle.

The syntax of drawRoundRect( ) method is

```
Void drawRoundRect (int top, int left, int width, int heigt,
                    int xDiam, int yDiam)
```

E.g.

```
// Java program to draw a Rounded Rectangle.

import java.awt.*;
import java.applet.*;
/*
<applet code="RoundrectDemo" width="500" height="500">
</applet>
*/
public class RoundrectDemo extends Applet
{
  public void paint(Graphics g)
  {
    g.drawRoundRect(190, 10, 60, 50, 15, 15);
  }
}
```

Drawing a filled Rounded Rectangle

The fillRoundRect( ) method is used to draw a filled Rounded Rectangle.

The Syntax of fillRoundRect( ) method is

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

E.g.

```
// Java program to draw a filled Rounded Rectangle.

import java.awt.*;
import java.applet.*;

/*
<applet code = "Fillroundrectdemo" width = "500"
                                   height = "500" >

</applet>
*/
public class Fillroundrectdemo extends Applet
{
   public void paint(Graphics g)
   {
      g.fillRoundRect(190,10,60,50,15,15);
   }
}
```

Drawing an Ellipse

The drawOval() method is used to draw an ellipse.

The Syntax of drawOval() method is

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

E.g.

```
// Java program for drawing an Oval
import java.awt.*;
import java.applet.*;
/*
<applet code="Ellipsedemo" width="500" height="500">
</applet>
*/
public class Ellipsedemo extends Applet
{
  Public void paint(Graphics g)
  {
    g.drawOval(190, 10, 90, 30);
  }
}
```

Drawing a filled ellipse

The fillOval() method is used to draw a filled ellipse.

The syntax of fillOval() method is

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

E.g.

```
// Java program for drawing a filled ellipse

import java.awt.*;
import java.applet.*;

/*
<applet code = "Fillellipsedemo" width = "500" height = "500"
>

</applet>
*/

public class Fillellipsedemo extends Applet
{

   public void paint(Graphics g)
   {
      g.fillOval(190, 10, 90, 30);
   }
}
```

Drawing a Polygon

The drawPolygon() method is used to draw a polygon.

The syntax of drawPolygon() method is

```
void drawPolygon(int x[], int y[], int numpoints)
```

E.g.

```
// Java program for drawing a polygon.
import java.awt.*;
import java.applet.*;
/*
<applet code = "Polygondemo" width = "500" height = "500">
</applet>
*/
public class Polygondemo extends Applet
 {
  public void paint(Graphics g)
  {
    int x[] = { 30, 200, 30, 200, 30 };
    int y[] = { 30, 30, 200, 200, 30 };
    int n = 5;
    g.drawPolygon(x, y, n);
  }
}
```

Drawing a filled polygon

The fillPolygon() method is used to draw a filled polygon.

The syntax of fillPolygon() method is

```
void fillPolygon(int x[], int y[], int numpoints)
```

E.g.

```
// Java program for drawing a filled polygon
import java.awt.*;
import java.applet.*;
/*
<applet code="Fillpolygondemo" width="500" height="500">
</applet>
*/
public class Fillpolygondemo extends Applet
{
  public void paint(Graphics g)
  {
    int x[] = { 30, 200, 30, 200, 30};
    int y[] = { 30, 30, 200, 200, 30};
    int n = 5;
    g.fillPolygon(x, y, n);
  }
}
```

Drawing an Arc

the drawArc( ) method is used to draw an Arc.

The syntax of drawArc( ) method is

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

E.g.

```
// Java program for drawing an arc
import java.awt.*;
import java.applet.*;
/*
<applet code = "Arcdemo" width = "500" height = "500">
</applet>
*/
Public class Arcdemo extends Applet
{
  Public void paint ( Graphics g )
  {
     g.drawArc( 10, 40, 70, 70, 0, 75);
  }
}
```

Drawing a filled arc

The fillArc() method is used to draw a filled arc.

The syntax of fillArc() method is

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

E.g.

```
// Java program for drawing a filled Arc
import java.awt.*;
import java.applet.*;
/*
<applet code = "fillarcdemo" width = "500" height = "500">
</applet>
*/
public class fillarcdemo extends Applet
{
    public void paint(Graphics g)
    {
        g.fillArc(10, 40, 70, 70, 0, 75);
    }
}
```

The repaint() method

It is used to call the update() method internally that calls the paint() method to repaint the component.

The repaint() method cannot be overriden.

The repaint() method is available in java.applet.Applet class.

E.g.

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

/*
<applet code = "classname" width="500" height="500>
</applet>
*/
public class <classname> extends Applet
{
    public  <methodname> (<arguments>)
    {
        repaint();
    }
}
```

Using the Status Window

We can print text in the status window of an applet through showStatus() method

Delegation Event Model

Delegation Event Model

It is a mechanism to generate and process events

Source

A source is an object that generates an event.

Listener

A listener is an object that is notified when an event occurs.

Event

An event is an object that describes a state change in a source.

Advantage

The application logic that processes events is cleanly separated from the user interface logic that generates those events

All events are represented as classes.

All listeners are represented as interfaces.

S.No.	Event class	Description	Event Listener	Method
1.	ActionEvent	Generated when a button is pressed, a list item is double clicked or a menu item is selected	ActionListener	actionPerformed( )
2.	AdjustmentEvent	Generated when a Scrollbar is manipulated	AdjustmentListener	adjustmentValueChanged( )
3.	ComponentEvent	Generated when a component is hidden, moved, resized or becomes visible	ComponentListener	componentResized( ) componentMoved( ) componentShown( ) componentHidden( )

S.No	Event Class	Description	Event Listener	Method
4.	ContainerEvent	Generated when a component is added to or removed from a container	ContainerListener	componentAdded( ) componentRemoved( )
5.	FocusEvent	Generated when a component gains or loses keyboard-focus	FocusListener	focusGained( ) focusLost( )
6.	ItemEvent	Generated when a checkbox or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.	ItemListener	itemStateChanged( )

S.NO	Event class	Description	Event Listener	Method
7.	KeyEvent	Generated when input is received from the keyboard	KeyListener	keyPressed( ) keyReleased( ) keyTyped( )
8.	MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed or released; also generated when the mouse enters or exits a component	MouseListener	mousePressed( ) mouseReleased( ) mouseClicked( ) mouseEntered( ) mouseExited( )
			MouseMotionListener	mouseDragged( ) mouseMoved( )

S.No.	Event	Description	Listener	Methods
9.	MouseWheelEvent	Generated when the mouse wheel is moved.	MouseWheelListener	mouseWheelMoved( )
10.	TextEvent	Generated when the value of a text area or text field is changed.	TextListener	textChanged( )
11.	WindowEvent	Generated when a window is activated, closed, deactivated, iconified, deiconified opened or quit	WindowListener	windowActivated( ) windowClosed( ) windowDeactivated( ) windowOpened( ) windowIconified( ) windowDeiconified( ) windowClosing( )

	AWT Components	Swings
1.	Label	JLabel
2.	TextField	JTextField
3.	TextArea	JTextArea
4.	Button	JButton
5.	CheckBox	JCheckBox
6.	List	JList
7.	Choice	JChoice
8.	ScrollBar	JScrollBar
9.	Menu	JMenu
10.	MenuBar	JMenuBar
11.	MenuItem	JMenuItem

Event classes

The ActionEvent class

The AdjustmentEvent class

The ComponentEvent class

The ContainerEvent class

The FocusEvent class

The ItemEvent class

The KeyEvent class

The MouseEvent class

The MouseWheelEvent class

The TextEvent class

The WindowEvent class.

Event Listener interfaces

The ActionListener interface.

The AdjustmentListener interface.

The ComponentListener interface.

The ContainerListener interface.

The FocusListener interface.

The ItemListener interface.

The KeyListener interface.

The MouseListener interface.

The MouseMotionListener interface.

The MouseWheelListener interface.

The TextListener interface.

The WindowListener interface.

Dialog box

A dialog is a Graphical user interface object that displays a message to the user, prompts the user for an input (or) asks for confirmation from the user

Dialog boxes are of two types, they are,

1. Modal dialog box
2. Modeless dialog box

Modal dialog box :- A modal dialog box is a dialog box that prevents the user from interacting with the rest of the application until the user dismisses the dialog box.

Modeless dialog box:- A modeless dialog box is a dialog box that allows the user to interact with the rest of the application.

* Dialog boxes in java can be classified into three types
  1. Message Dialog box
  2. Input Dialog box
  3. Confirmation Dialog box

All the dialog boxes are available in JOptionPane class of javax.swing package.

JOptionPane.showMessageDialog()

JOptionPane.showInputDialog()

JOptionPane.showConfirmDialog()

// Java program to illustrate Message DialogBox

```
import javax.swing.*;
Class Mdemo
{
```

```
    public static void main (String args[]) throws Exception
    {
      JOptionPane.showMessageDialog(null," WELCOME");
    }
}

// Java program to illustrate InPut Dialog box
  import javax.swing.*;
  class Idemo
  {
    Public Static void main (String args[]) throws Exception
    {
       String S;
       S = JOptionPane.showInputDialog ("ENTER A STRING");
       System.out.print("THE GIVEN STRING IS---"+S);
    }
}

// Java program to illustrate Confirmation dialog box
  import javax.swing.*;
  class Cdemo
  {
    public static void main (String args[]) throws Exception
    {
      int n;
      n = JOptionPane.showConfirmDialog (null," DO YOU WANT TO EXIT?");
    }
}
```

JDBC :- JDBC stands for Java DataBase Connectivity.

- JDBC is an API which is used to Connect java application with a database.

API :

API stands for Application Programing Interface.

- It is a set of predefined functions.

Frontend :

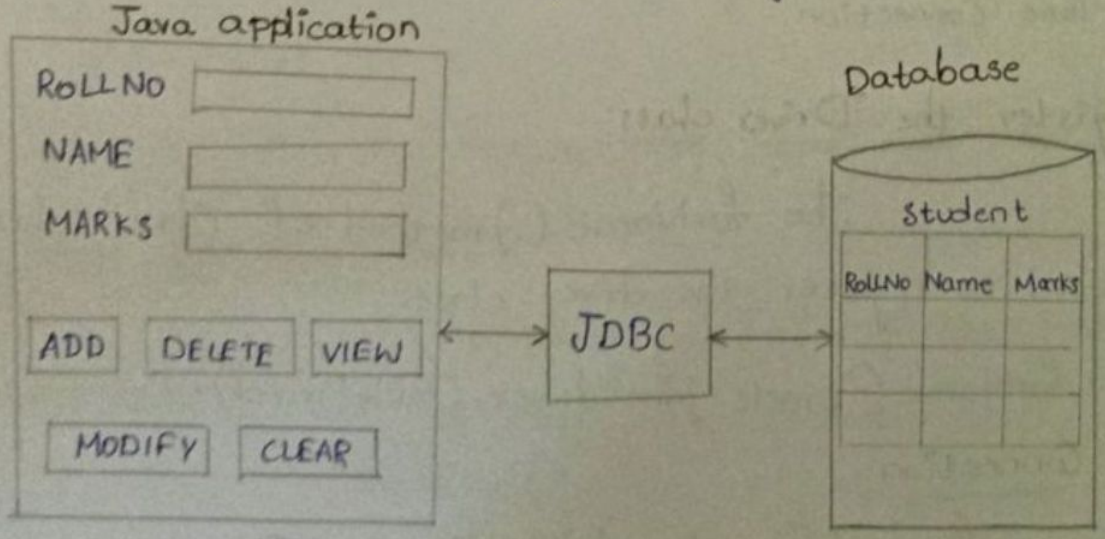
The software which is used to Create user interface is known as Frontend.

Ex :- HTML, CSS
JAVA.

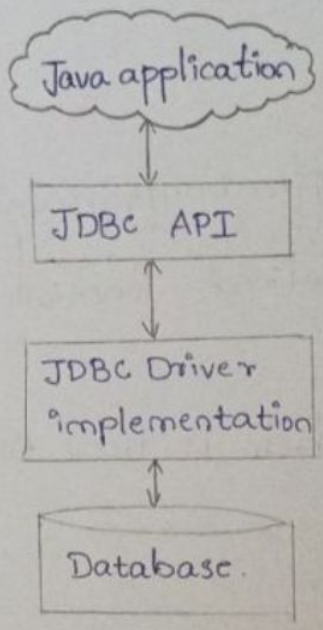
Backend :

The software which is used to store the data is called Backend.

Ex :- SQL Server, Oracle, MySQL, Mongo DB.



JDBC Architecture :-



Steps in JDBC :-

1. Register the Driver class.
2. Get Connection.
3. Create statement.
4. Execute Query.
5. close Connection.

⇒ Register the Driver class :-

The forName ( ) method of Class class is used to register the driver class.

```
class.forName("oracle.jdbc.driver.OracleDriver");
```

⇒ Get Connection :-

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

```
Connection con = DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
```

Connection URL — Username — password

jdbc ⇒ API
Oracle ⇒ Database
thin ⇒ Driver
@localhost ⇒ System on which Server is running
1521 ⇒ port
xe ⇒ Oracle Service

⇒ **Create the statement:**

The CreateStatement() method of Connection interface is used to create the statement object.

```
Statement stmt = con.createStatement();
```

⇒ **Execute the Query:**

The executeQuery() method is used to execute the query.

```
ResultSet rs = stmt.executeQuery();
```

⇒ **close the Connection:**

The close() method of Connection interface is used to close the connection.

```
con.close();
```

executeQuery()	executeUpdate()
1. It is used with Select Statement.	1. It is used with insert, delete and update statements.
2. A set of records are returned.	2. It returns the number of records affected.

```
//Java program to View all the records in a table.
import java.sql.*;
class Db1
{
 public static void main (String args[]) throws Exception
 {
  Class.forName ("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection (
"jdbc:oracle:thin:@localhost:1521:xe", "system", "oracle");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery ("Select * from Student");
   while (rs.next())
    {
     System.out.print(rs.getInt(1)+" "+rs.getString(2)+
                      " "+rs.getInt(3));
    }
     con.close();
}}
```

Java program to insert a record into a table.

```
import java.sql.*;
class Db2
{
public static void main (String args[])throws Exception
 {
 Class.forName ("oracle.jdbc.driver.oracleDriver");
 Connection con = DriverManager.getConnection (
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
 Statement stmt = con.CreateStatement();
int m = stmt.executeUpdate ("insert into student Values
                              (1001, 'Sree', 80"));
  if (m==1)
   {
   System.out.print("Record has been inserted");
   }
  else
   {
   System.out.print("Record has not been inserted");
   }
   con.close();
   }
}
```

JDBC Driver:

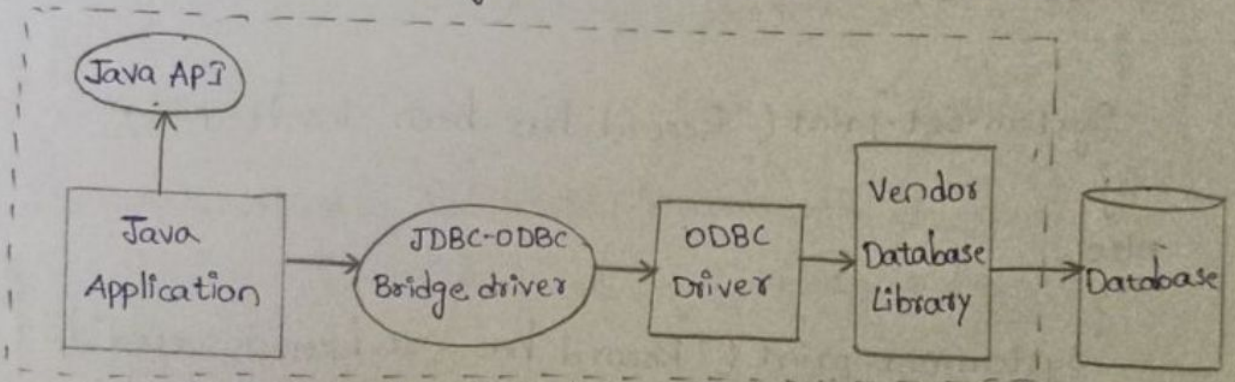
JDBC Driver is a software component that enables java application to interact with the database.

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.



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:

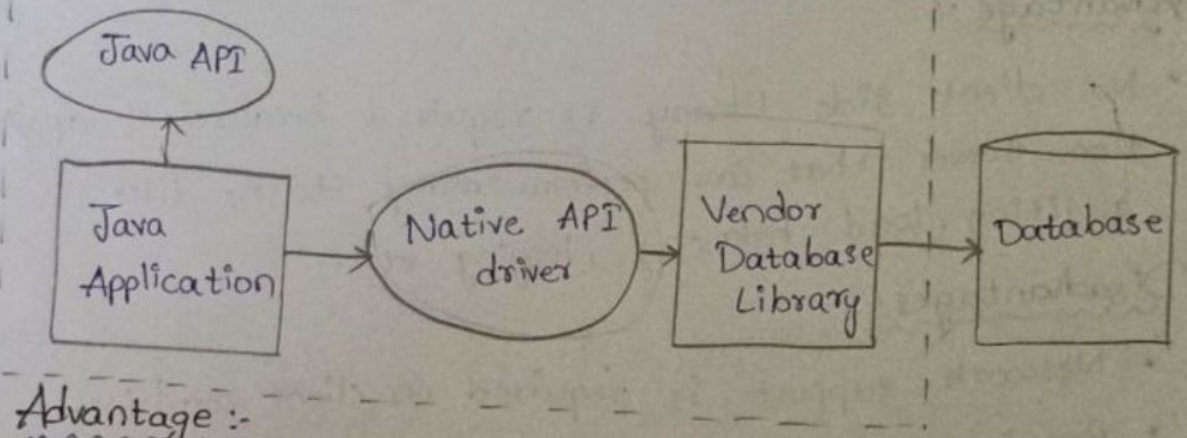
- easy to use.
- Can be easily connected to any database.

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.

2) 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.



Advantage:-

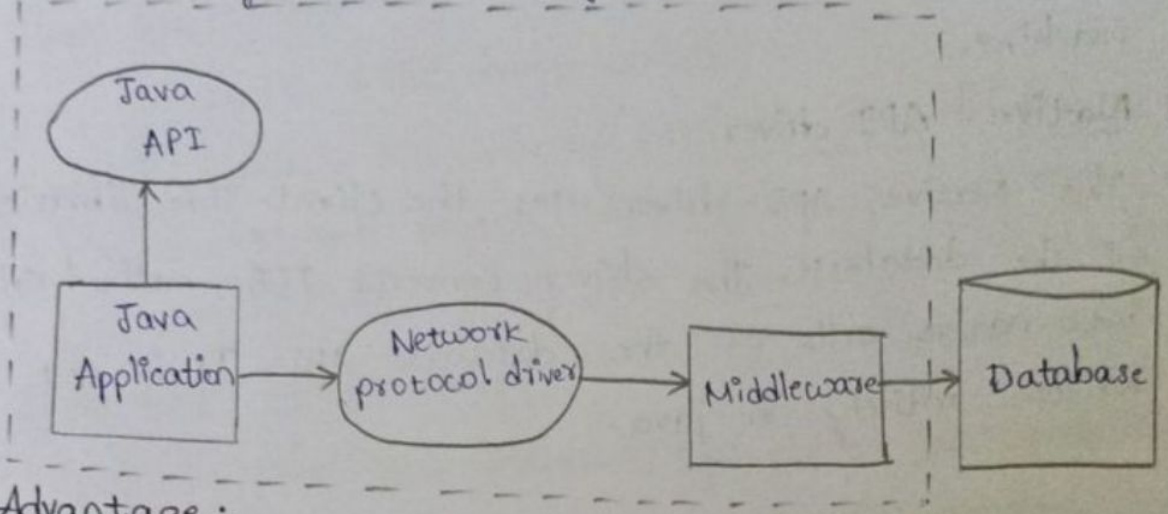
- performance upgraded than JDBC-ODBC bridge driver.

Disadvantages:-

- The Native drivers 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.



Advantage :-

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

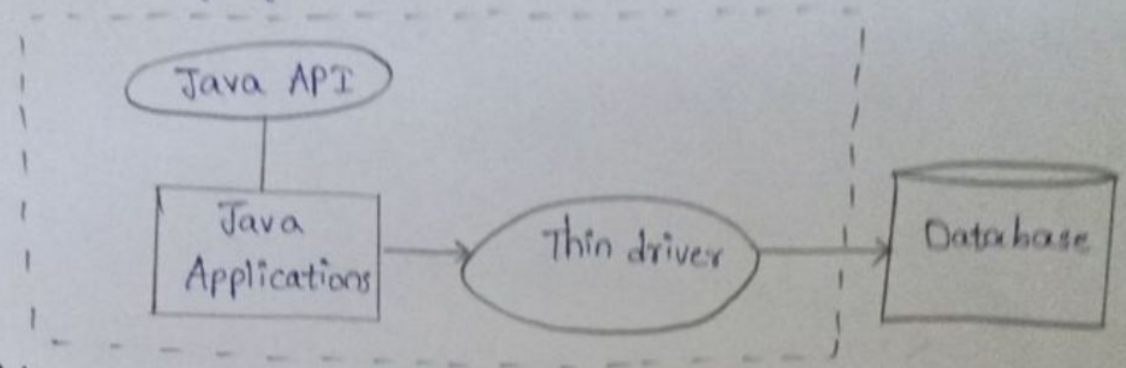
Disadvantages :-

- 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.



Advantages:-

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

Disadvantages:-

- Drivers depend on the Database.