GUI Programming with java

The AWT Class hierarchy

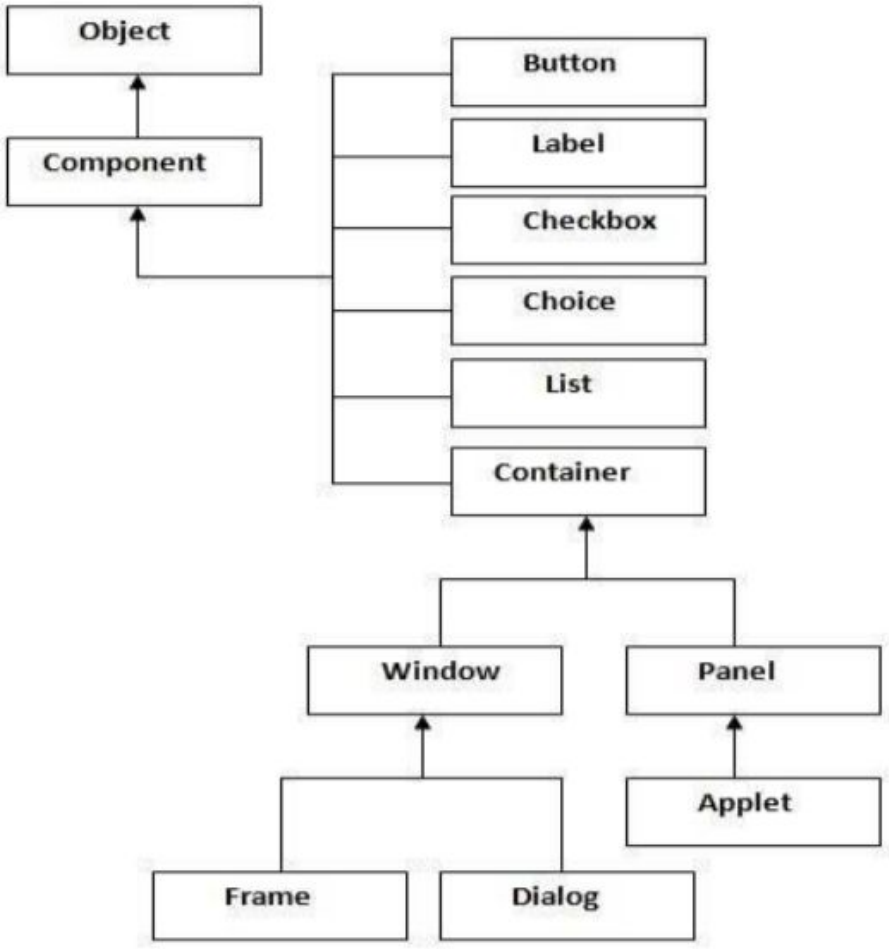
Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

Applets

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

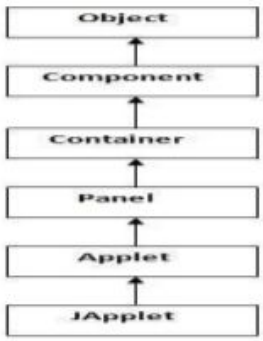
Drawback of Applet

- Plugin is required at client browser to execute applet.

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Hierarchy of Applet



Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
1.  //First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("welcome",150,150);
}
}
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
1.  //First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}
}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Difference between Applet and Application programming

	Java Applet	Java Application
User graphics	Inherently graphical	Optional
Memory requirements	Java application requirements plus web browser requirements	Minimal java application requirements
Distribution	Linked via HTML and transported via HTTP	Loaded from the file system or by a custom class loading process
Environmental input	Browser client location and size; parameters embedded in the host HTML document	command-line parameters
Method expected by the virtual Machine	init- initialization method start-startup method stop pause/ deactive method destroy-termination method paint-drawing method	Main - startup method
Typical applications	public-access order-entry systems for the web, online multimedia persentations, web page animation	Network server, multimedia kiosks, developer tools, appliance and consumer electronics control and navigation.

Creating the Frame:

- We can create a Frame by creating Frame class object Frame obj = new Frame ();

(or)

Create a class that extends Frame class then create an object to that class

```
class MyClass extends Frame
MyClass obj = new MyClass ();
```

After creating the Frame we need to set Frame width and height using setSize () method as: fsetSize (400, 350);

We can display the frame using setVisible () method as:

```
fsetVisible (true);
```

Program 1: Write a program to create a Frame without extending Frame class

```
//creating a Frame
import javaawt*;
class MyFrame


{
        public static void main(String args[])
        {Frame  f1  =  new  Frame  ();
                f1setSize (500,150);
                f1setTitle ("GUI World");
                f1setVisible (true);
        }
}
```

Output:

Program 2: Write a program to create a Frame by extending Frame class

```
//creating a Frame
import javaawt*;
class MyFrame extends Frame
{public static void main(String args[])
        {MyFrame  f1  =  new  MyFrame  ();
                f1setSize (500,200);
                f1setTitle ("GUI World");
                f1setVisible (true);
        }
}
```

Output:

The frame can be minimized, maximized and resized but cannot be closed Even if we click on close button of the frame, it will not perform any closing action Closing a frame means attaching action to the component To attach actions to the components, we need 'Event Delegation Model'

Event-Delegation-Model: Graphical representation of an object is called a component User interaction with the component is called event When an event is generated by the user on the component, component will delegate (hand over) the event to the listener The listener will delegate the event to one of its method The method is finally executed and the event is handled This is called Event-Delegation-Model Event-Delegation-Model is used in awt to provide actions for components In Event Delegation Model:

Attach the Listener to the component
Implement all the methods of the Listener
When an Event is generated one of these methods performs the required action

Closing the Frame: We know Frame is also a component We want to close the frame by clicking on its close button Let us follow these steps to see how to use event delegation model to do this:

We should attach a listener to the frame component Remember, all listeners are available in javaawtevent package The most suitable listener to the frame is 'WindowListener' It can be attached using addWindowListener () method as:

faddWindowListener (WindowListener obj);

Please note that the addWindowListener () method has a parameter that is expecting object of WindowListener interface Since it is not possible to create an object to an interface, we should create an object to the implementation class of the interface and pass it to the method

Implement all the methods of the WindowListener interface The following methods are found in WindowListener interface:

```
public void windowActivated (WindowEvent e)
public void windowClosed (WindowEvent e)
public void windowClosing (WindowEvent e)
public void windowDeactivated (WindowEvent
e)
public void windowDeiconified (WindowEvent e)
public void windowIconified (WindowEvent e)
public void windowOpened (WindowEvent e)
```

In all the preceding methods, WindowListener interface calls public void windowClosing ( )
method when the frame is being closed So, implementing this method alone is enough, as:

```
public void windowClosing (WindowEvent e)
{               //Close the application
                Systemexit (0);
}
```

Displaying text in the Frame: We need paint () method whenever we want to display some new drawing or text or images in the Frame The paint () method is automatically called when a frame is created and displayed The paint () method refreshes the frame contents automatically when a drawing is displayed The paint () method takes Graphics Class object as parameter Gaphics class is present in javaawt package

To display some text or strings in the frame, we can take the help of drawstring () method of Graphics class as:

gdrawString ("Hai Readers", x, y);

Here, the string "Hai Readers" will be displayed starting from the coordinates (x, y)

If we want to set some color for the text, we can use setColor () method of Graphics class as: gsetColor (Colorred);

The first way is by directly mentioning the needed standard color name from Color class as Colorblack, Colorblue, Colorcyan, Colorpink, Colorred, Colororange, Colormagenta, ColordarkGray, Colorgray, ColorlightGray, Colorgreen, Coloryellow and Colorwhite etc

The second way to mention any color is by combining the three primary colors: red, green and blue while creating Color class object as:

Color c = new Color (r, g, b);

Here, r, g, b values can change from 0 to 255 0 represents no color 10 represent low intensity whereas 200 represent high intensity of color

eg: Color c = new Color (255, 0, 0); //red color

To set some font to the text, we can use setFont () method of Graphics class, as:

gsetFont (Font object);

This method taked Font class object, which can be created as:

Font f = new Font ("SansSerif", FontBOLD, 30);

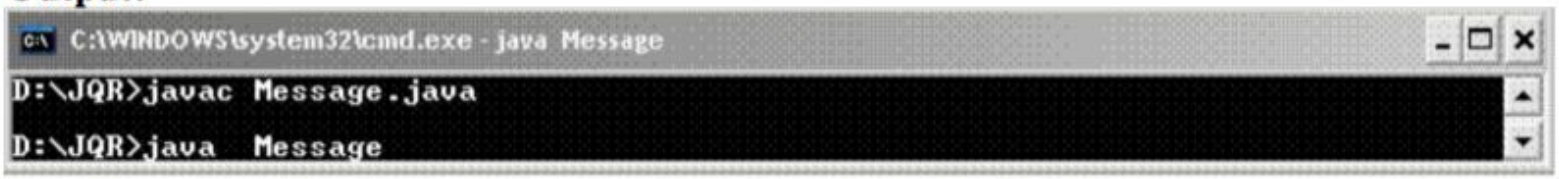
Here, "SansSerif" represents the font name; FontBOLD represents the font style and 30 represents the font size in pixels

Program 4: Write a program to display a message in the frame

```
//Displaying a message in the frame
import javaawt*;
import javaawtevent*;
class Message extends Frame
{Message ()
        {//code to close the Frame thisaddWindowListener
                (new WindowAdapter ()
                {public void windowClosing (WindowEvent we)
                        {
```

```
                    Systemexit (0);
                }
            });
        }
        public static void main (String args[])
        {Message m = new Message ();
            msetTitle ("Simple
            Message"); msetSize
            (650,150); msetVisible (true);
        }
        public void paint (Graphics g)
        {thissetBackground (Colorgreen);
            gsetColor (Colorred);
            Font f = new Font ("Times New Roman", FontBOLD+FontITALIC,
            60); gsetFont (f);
            gdrawString ("Hello Readers!", 50, 100);
        }
}
```

Output:



Component Class Methods: A component is a graphical representation of an object on the screen For example, push buttons, radio buttons, menus etc are components Even frame is also a component There is Component class available in javaawt package which contains the following methods which are applicable to any component

Method	Description
Font getFont ()	This method returns the font of the component
void setFont (Font f)	This method sets a particular font f for the text of the component
Color getForeground ()	This method gives the foreground color of the component
void setForeground (Color c)	This method sets a foreground color c to the component
Color getBackground ()	Gets the background color of the component
void setBackground (Color c)	Sets the background color c for the component
String getName ()	Returns the name of the component
void setName (String name)	Sets a new name for the component
int getHeight ()	Returns the height of the component in pixels as an integer
int getWidth ()	Returns the width of the component in pixels as an integer
Dimension getSize ()	Returns the size of the component as an object of Dimension class Dimensionwidth and Dimensionheight will provide the width and height of the component
int getX ()	Returns the current x coordinate of the components origin
int getY ()	Returns the current y coordinate of the components origin
Point getLocation ()	Gets the locationof the component in the form of a point specifying the components top-left corner
void setLocation (int x, int y)	Moves the component to a new location specified by (x, y)
void setSize(int width, int height)	Resizes the component so that it has new width and height as passed to setSize () method
void setVisible (boolean b)	Shows the component if the value of b is true or hides the component if the value of parameter b is false
void setEnabled (boolean b)	Enables the component if the value of b is true or disables the component if the value of parameter b is false
void setBounds (int x, int y, int w, int h)	This method allots a rectangular area starting at (x ,y) coordinates and with width w and height h The component is resized to this area before its display This method is useful to specify the location of the component in the frame

After creating a component, we should add the component to the frame For this purpose, add () method is used fadd (component); where f is frame class object

Similarly, to remove a component from the frame, we can use remove () method as:
fremove (component); where f is frame class object

Frame class contains a method called setLayout () setLayout () is useful to set a layout for the frame A layout represents a manner of arranging components in the frame All layouts are represented as implementation classes of LayoutManager interface For example, the following layouts are available in AWT:

- o FlowLayout: FlowLayout is useful to arrange the components in a line after the other When a line is filled with components, they are automatically placed in the next line
- o BorderLayout: BorderLayout is useful to arrange the components in the 4 borders of the frame as well as in the center The borders are specified as South, North, East, West and Center
- o CardLayout: A cardLayout treats each component as a card Only one card is visible at a time and it arranges the components as a stack of cards

GridLayout: It is useful to divide the display area into a two dimensional grid form that contains several rows and columns The display area is divided into equal sized rectangles and one component is placed in each rectangle

GridBagLayout: This layout is more flexible as compared to other layouts since in this layout the components can span more than one row or column and the size of the components can be adjusted to fit the display area

We will discuss about layout manager in later chapter The following points are helpful to understand how to work with a layout manager:

To set a layout for out components, we can pass the layout class object to the setLayout () method as: setLayout (new FlowLayout ());

Suppose, we do not want to set any layout, then we should pass null to the setLayout () method as: setLayout (null);

Suppose, we do not use setLayout () method at all, then the Java compiler assumes a default layout manager The default layout in case of a frame is BorderLayout

Listeners and Listener Methods: Listeners are available for components A Listener is an interface that listens to an event from a component Listeners are available in javaawtevent package The methods in the listener interface are to be implemented, when using that listener

Component	Listener	Listener methods
Button	ActionListener	public void actionPerformed (ActionEvent e)
Checkbox	ItemListener	public void itemStateChanged (ItemEvent e)
CheckboxGroup	ItemListener	public void itemStateChanged (ItemEvent e)
TextField	ActionListener FocusListener	public void actionPerformed (ActionEvent e) public void focusGained (FocusEvent e) public void focusLost (FocusEvent e)
TextArea	ActionListener FocusListener	public void actionPerformed (ActionEvent e) public void focusGained (FocusEvent e) public void focusLost (FocusEvent e)
Choice	ActionListener ItemListener	public void actionPerformed (ActionEvent e) public void itemStateChanged (ItemEvent e)
List	ActionListener ItemListener	public void actionPerformed (ActionEvent e) public void itemStateChanged (ItemEvent e)
Scrollbar	AdjustmentListener MouseMotionListener	public void adjustmentValueChanged (AdjustmentEvent e) public void mouseDragged (MouseEvent e) public void mouseMoved (MouseEvent e)
Label	No listener is needed	

Connceting to DB

Whatis JDBCDriver?

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

The *Java.sql* package that ships with JDK, contains various classes with their behaviours defined and their actual implementaions are done in third-party drivers. Third party vendors implements the *java.sql.Driver* interface in their database driver.

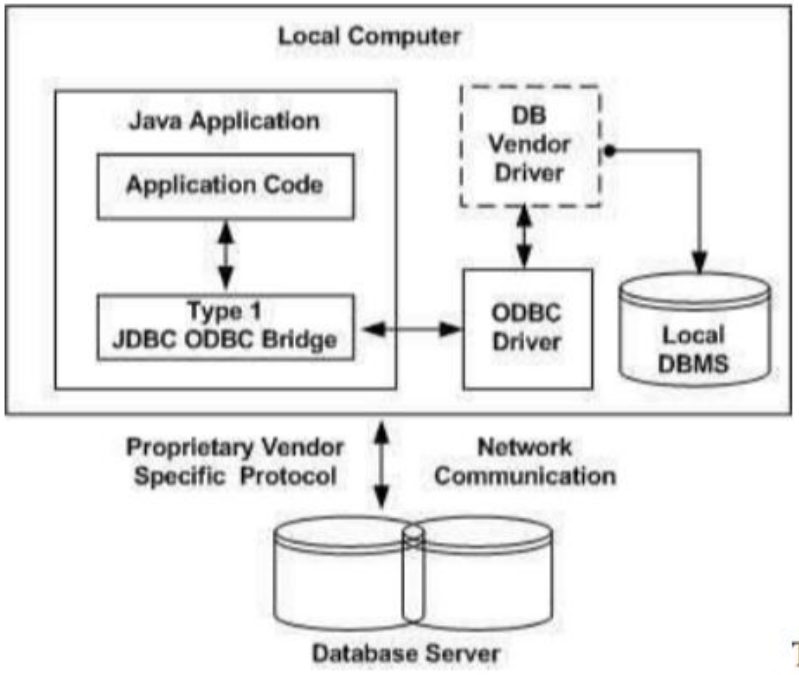
JDBC Drivers Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below −

Type 1: JDBC-ODBCBridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

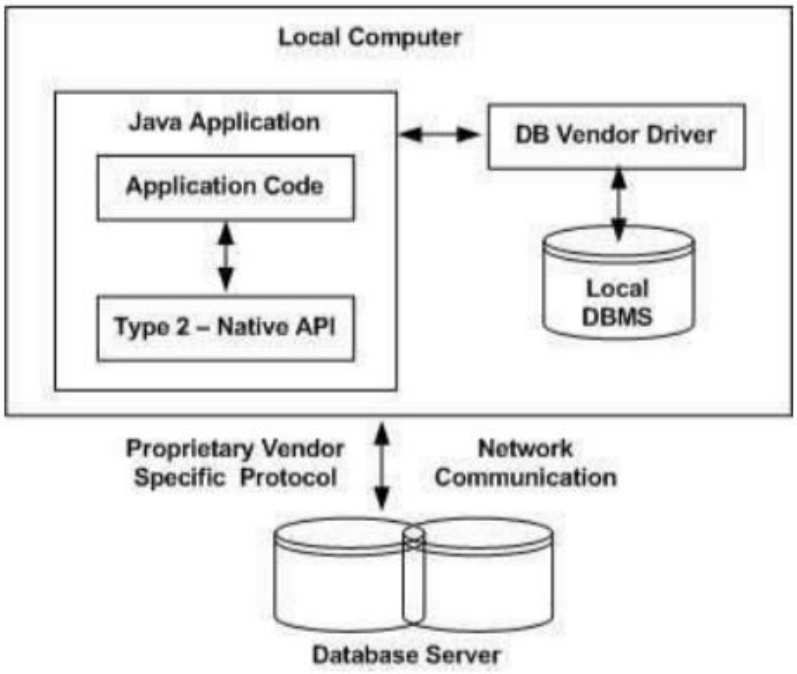


The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

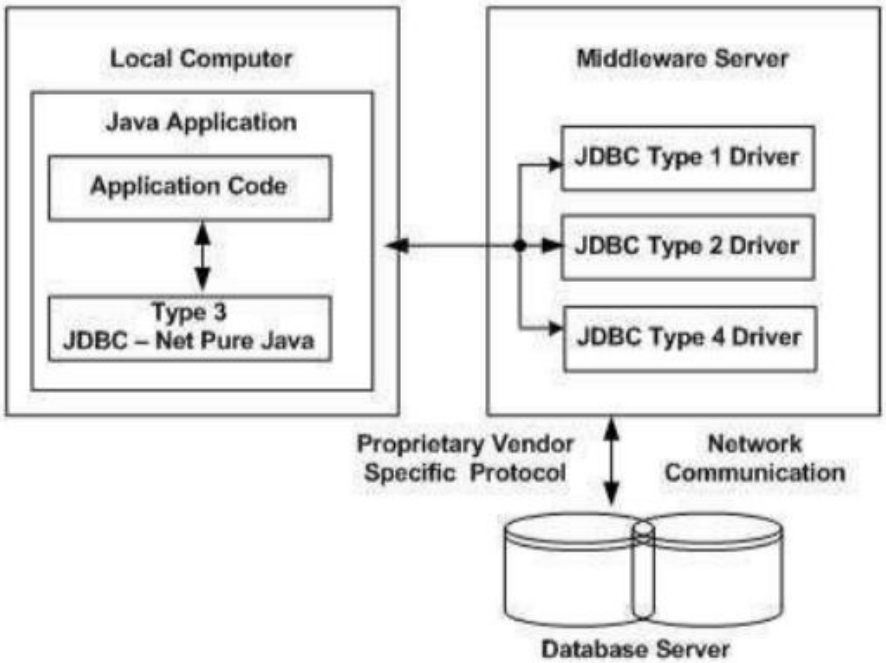


The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



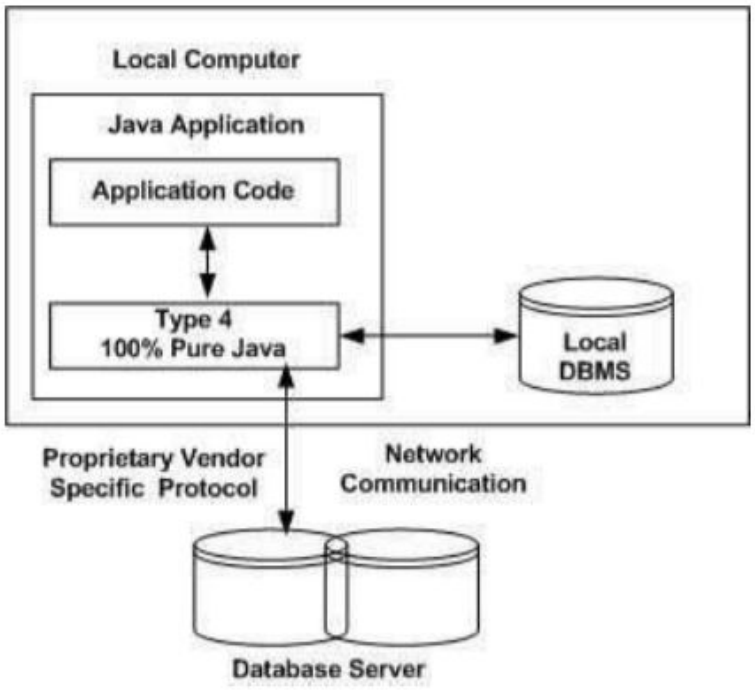
You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Which Driver should be Used?

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

Example to connect to the mysql database in java

For connecting java application with the mysql database, you need to follow 5 steps to perform database connectivity.

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, you need to replace the sonoo with your database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** Password is 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 sonoo;
2. use sonoo;
3. create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password.

```
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/sonoo","root","root");
//here sonoo 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.getString(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.

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:

1.temporary 2.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;

JDBC-Result Sets

The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The *java.sql.ResultSet* interface represents the result set of a database query.

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface can be broken down into three categories −

- **Navigational methods:** Used to move the cursor around.
- **Get methods:** Used to view the data in the columns of the current row being pointed by the cursor.
- **Update methods:** Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

The cursor is movable based on the properties of the ResultSet. These properties are designated when the corresponding Statement that generates the ResultSet is created.

JDBC provides the following connection methods to create statements with desired ResultSet −

- **createStatement(int RSType, int RSConcurrency);**
- **prepareStatement(String SQL, int RSType, int RSConcurrency);**
- **prepareCall(String sql, int RSType, int RSConcurrency);**

The first argument indicates the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable.

Type of ResultSet

The possible RSType are given below. If you do not specify any ResultSet type, you will automatically get one that is TYPE_FORWARD_ONLY.

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.

ResultSet.TYPE_SCROLL_SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

Concurrencyof ResultSet

The possible RSConcurrency are given below. If you do not specify any Concurrency type, you will automatically get one that is CONCUR_READ_ONLY.

Concurrency	Description
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

Viewinga Result Set

The ResultSet interface contains dozens of methods for getting the data of the current row.

There is a get method for each of the possible data types, and each get method has two versions

- One that takes in a column name.
- One that takes in a column index.

For example, if the column you are interested in viewing contains an int, you need to use one of the getInt() methods of ResultSet −

S.N.	Methods & Description
1	**public int getInt(String columnName) throws SQLException** Returns the int in the current row in the column named columnName.
2	**public int getInt(int columnIndex) throws SQLException** Returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

Similarly, there are get methods in the ResultSet interface for each of the eight Java primitive types, as well as common types such as java.lang.String, java.lang.Object, and java.net.URL.

There are also methods for getting SQL data types java.sql.Date, java.sql.Time, java.sql.TimeStamp, java.sql.Clob, and java.sql.Blob. Check the documentation for more information about using these SQL data types.

For a better understanding, let us study Viewing - Example Code.

Updatinga Result Set

The ResultSet interface contains a collection of update methods for updating the data of a result set.

As with the get methods, there are two update methods for each data type −

- One that takes in a column name.
- One that takes in a column index.

For example, to update a String column of the current row of a result set, you would use one of the following updateString() methods −

S.N.	Methods & Description
1	**public void updateString(int columnIndex, String s) throws SQLException** Changes the String in the specified column to the value of s.
2	**public void updateString(String columnName, String s) throws SQLException** Similar to the previous method, except that the column is specified by its name instead of its index.

There are update methods for the eight primitive data types, as well as String, Object, URL, and the SQL data types in the java.sql package.

Updating a row in the result set changes the columns of the current row in the ResultSet object, but not in the underlying database. To update your changes to the row in the database, you need to invoke one of the following methods.

S.N.	Methods & Description
1	**public void updateRow()** Updates the current row by updating the corresponding row in the database.
2	**public void deleteRow()** Deletes the current row from the database
3	**public void refreshRow()** Refreshes the data in the result set to reflect any recent changes in the database.
4	**public void cancelRowUpdates()** Cancels any updates made on the current row.
5	**public void insertRow()** Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.