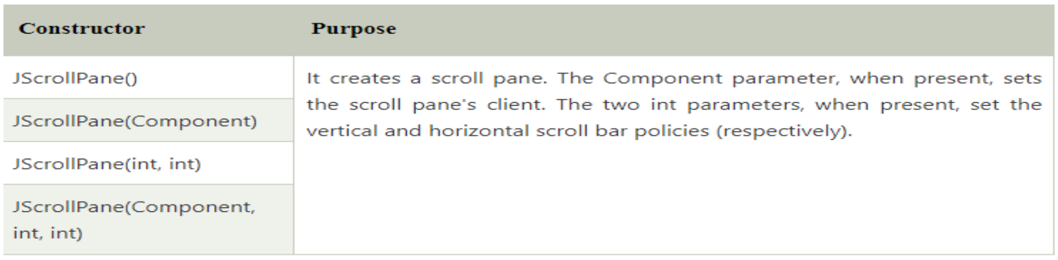
Unit V

Advanced Swing Control :

JScrollPane :

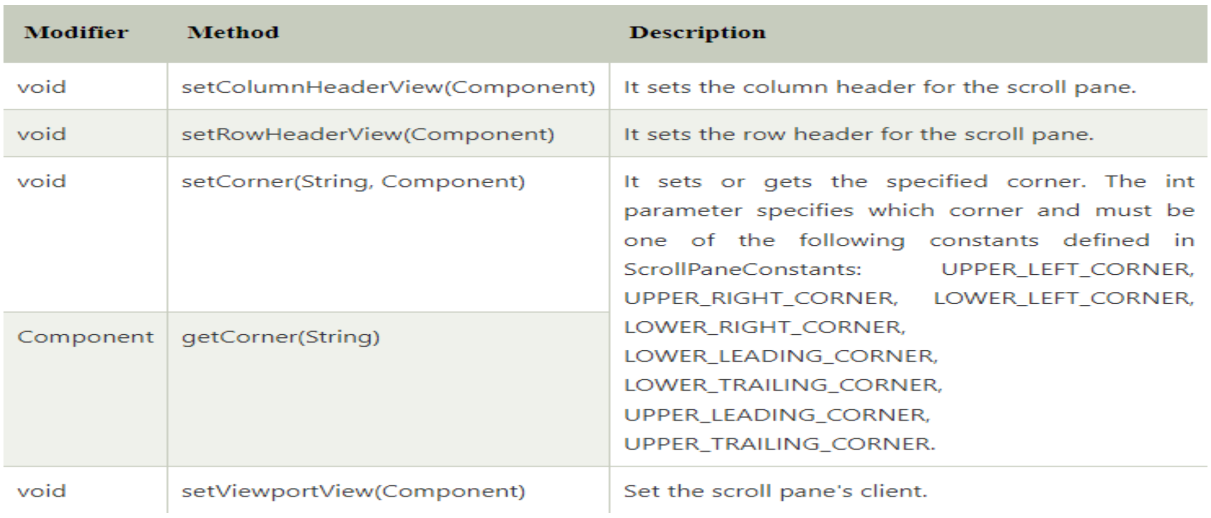
A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Constructors



Constructor	Purpose
JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).
JScrollPane(Component)	
JScrollPane(int, int)	
JScrollPane(Component, int, int)	

Useful Methods



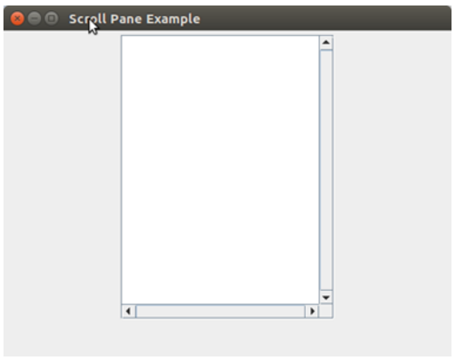
Modifier	Method	Description
void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void	setRowHeaderView(Component)	It sets the row header for the scroll pane.
void	setCorner(String, Component)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in ScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
Component	getCorner(String)	
void	setViewportView(Component)	Set the scroll pane's client.

JScrollPane Example

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JtextArea;
```

```java

public class JScrollPaneExample {
    private static final long serialVersionUID = 1L;

    private static void createAndShowGUI() {

        // Create and set up the window.
        final JFrame frame = new JFrame("Scroll Pane Example");

        // Display the window.
        frame.setSize(500, 500);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // set flow layout for the frame
        frame.getContentPane().setLayout(new FlowLayout());

        JTextArea textArea = new JTextArea(20, 20);
        JScrollPane scrollableTextArea = new JScrollPane(textArea);

        scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        frame.getContentPane().add(scrollableTextArea);
    }
    public static void main(String[] args) {


        javax.swing.SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                createAndShowGUI();
            }
        });
    }
40.}
```

Output :



Java JColorChooser

The JColorChooser class is used to create a color chooser dialog box so that user can select any color. It inherits JComponent class.

JColorChooser class declaration

Let's see the declaration for javax.swing.JColorChooser class.

```
1. public class JColorChooser extends JComponent implements Accessible
```

Commonly used Constructors:

Constructor	Description
JColorChooser()	It is used to create a color chooser panel with white color initially.
JColorChooser(color initialcolor)	It is used to create a color chooser panel with the specified color initially.

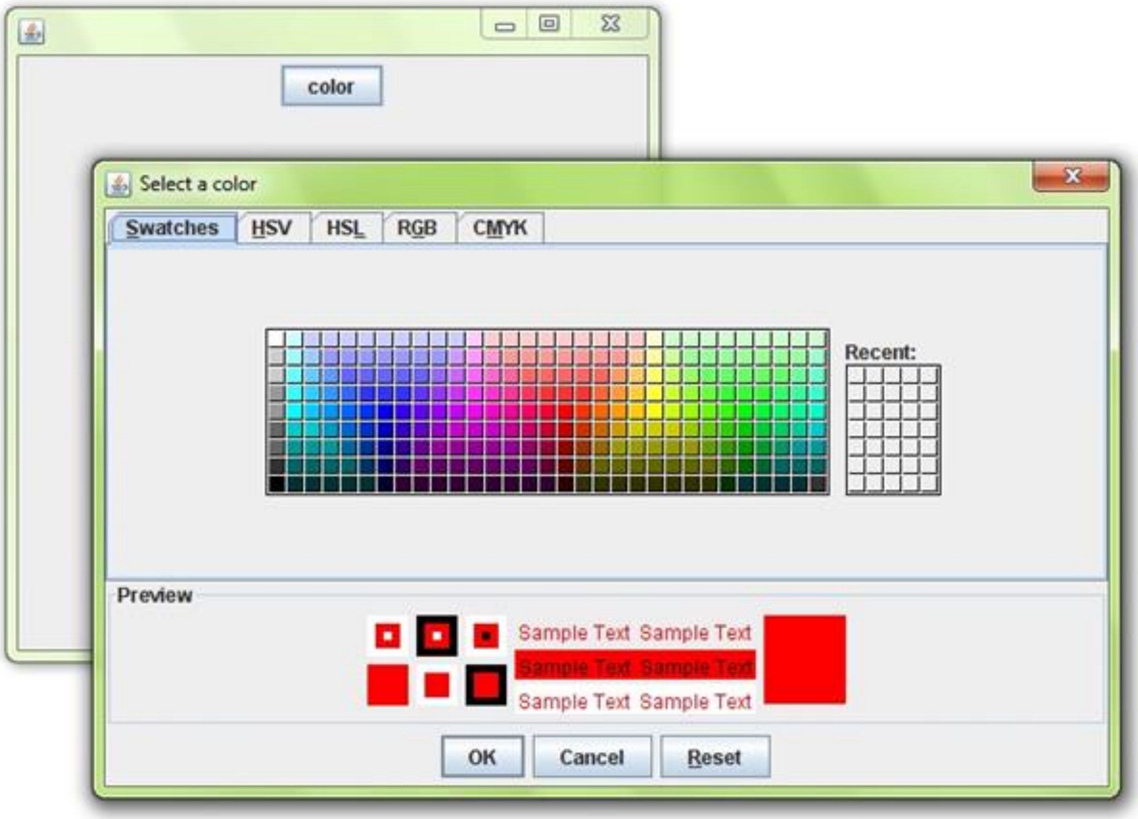
Commonly used Methods:

Method	Description
void addChooserPanel(AbstractColorChooserPanel panel)	It is used to add a color chooser panel to the color chooser.
static Color showDialog(Component c, String title, Color initialColor)	It is used to show the color chooser dialog box.

Java JColorChooser Example

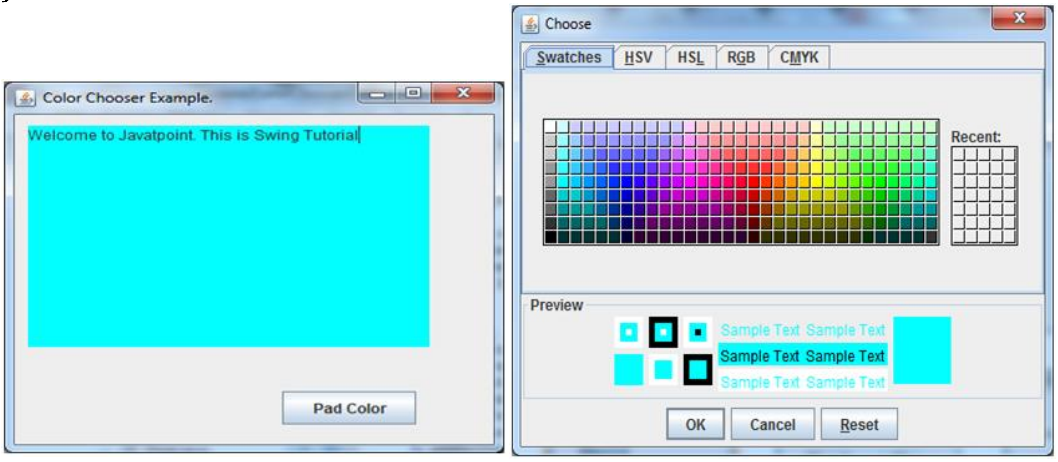
```
1. import java.awt.event.*;
```

```java
import java.awt.*;
import javax.swing.*;
public class ColorChooserExample extends JFrame implements ActionListener {
JButton b;
Container c;
ColorChooserExample(){
    c=getContentPane();
    c.setLayout(new FlowLayout());
    b=new JButton("color");
    b.addActionListener(this);
    c.add(b);
}
public void actionPerformed(ActionEvent e) {
Color initialcolor=Color.RED;
Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
c.setBackground(color);
}

public static void main(String[] args) {
    ColorChooserExample ch=new ColorChooserExample();
    ch.setSize(400,400);
    ch.setVisible(true);
    ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```



Java JColorChooser Example with ActionListener

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ColorChooserExample extends JFrame implements ActionListener{
JFrame f;
JButton b;
JTextArea ta;
ColorChooserExample(){
    f=new JFrame("Color Chooser Example.");
    b=new JButton("Pad Color");
    b.setBounds(200,250,100,30);
    ta=new JTextArea();
    ta.setBounds(10,10,300,200);
    b.addActionListener(this);
    f.add(b);f.add(ta);
    f.setLayout(null);
    f.setSize(400,400);
    f.setVisible(true);
}
public void actionPerformed(ActionEvent e){
    Color c=JColorChooser.showDialog(this,"Choose",Color.CYAN);
    ta.setBackground(c);
}
public static void main(String[] args) {
    new ColorChooserExample();
}
}
```



Welcome to Javatpoint. This is Swing Tutorial

Java JFileChooser

JFileChooser is a class that is present in the java Swing package. The java Swing package is essential for JavaTM Foundation Classes(JFC). JFileChooser contains many elements that assist in building a graphical user Interface in java. Java Swing gives components like buttons, panels, dialogs, etc. JFileChooser is a simple and successful method for inciting the client to pick a file or a directory. JFileChooser inherited the properties of JComponent class and implemented them with an Accessible interface.

Constructors Present in Java JFileChooser Class

1. JFileChooser():

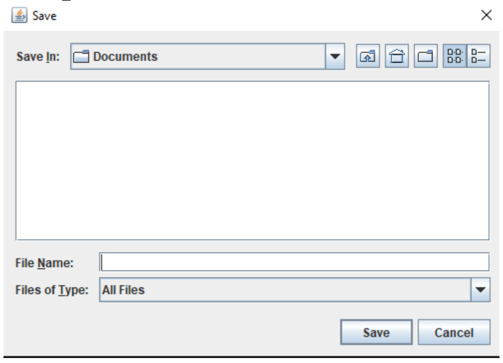
Constructs a JFileChooser highlighting the client's default directory.

Example program on JFileChooser()

```
// program to demonstrate the JFileChooser() constructor
// importing the required packages
import java.io.*;
importjavax.swing.*;
```

```
importjava.awt.event.*;
importjavax.swing.filechooser.*;
classHelloWorld
{
public static void main(String[] args) {
     // creating object to the JFileChooser class
   JFileChooserjf = new JFileChooser(); // default constructor JFileChooser is called.
     jf.showSaveDialog(null);
  }
}
```

Output:



Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

JTable class declaration

Let's see the declaration for javax.swing.JTable class.

Commonly used Constructors:

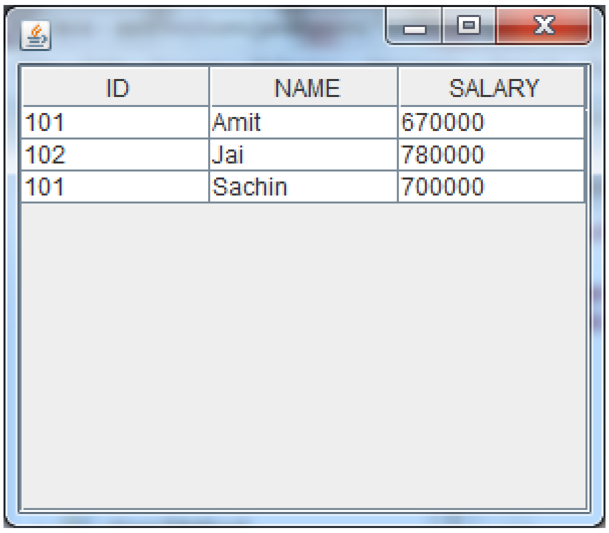
Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JTable Example

```
import javax.swing.*;
public class TableExample {
  JFrame f;
```

```
    TableExample(){
    f=new JFrame();
    String data[][]={ {"101","Amit","670000"},
                          {"102","Jai","780000"},
                          {"101","Sachin","700000"}};
    String column[]={"ID","NAME","SALARY"};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
    JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TableExample();
}
}
```

Output :



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration

Let's see the declaration for javax.swing.JTabbedPane class.

```java
public class JTabbedPane extends JComponent implements Serializable, Accessible, SwingConstants
```

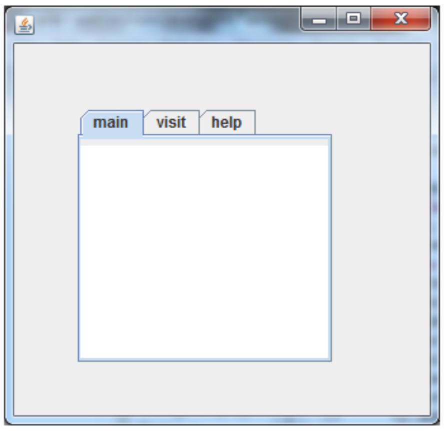
Commonly used Constructors:

Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.

Java JTabbedPane Example

```java
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
   f=new JFrame();
   JTextArea ta=new JTextArea(200,200);
   JPanel p1=new JPanel();
   p1.add(ta);
   JPanel p2=new JPanel();
   JPanel p3=new JPanel();
   JTabbedPane tp=new JTabbedPane();
   tp.setBounds(50,50,200,200);
   tp.add("main",p1);
   tp.add("visit",p2);
   tp.add("help",p3);
   f.add(tp);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
public static void main(String[] args) {
   new TabbedPaneExample();
}}
```

Output :

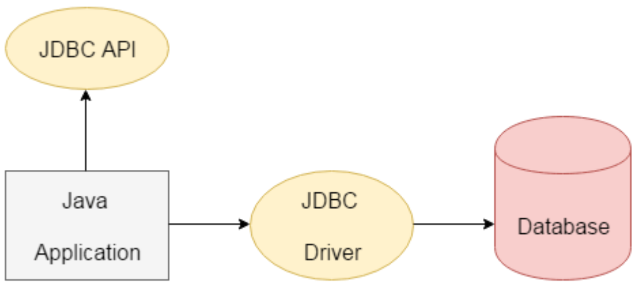


Java JDBC

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

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.

What is API

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

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.

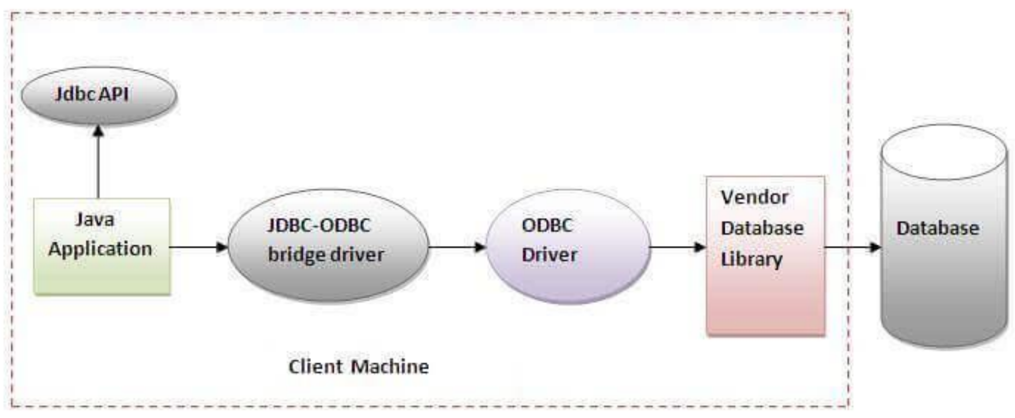


Figure- JDBC-ODBC Bridge Driver

In Java 8, the JDBC-ODBC Bridge has been removed.

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.

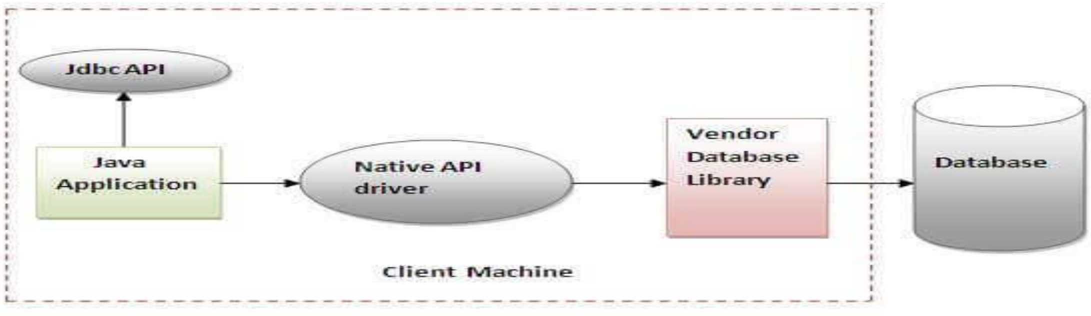


Figure- Native API Driver

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.

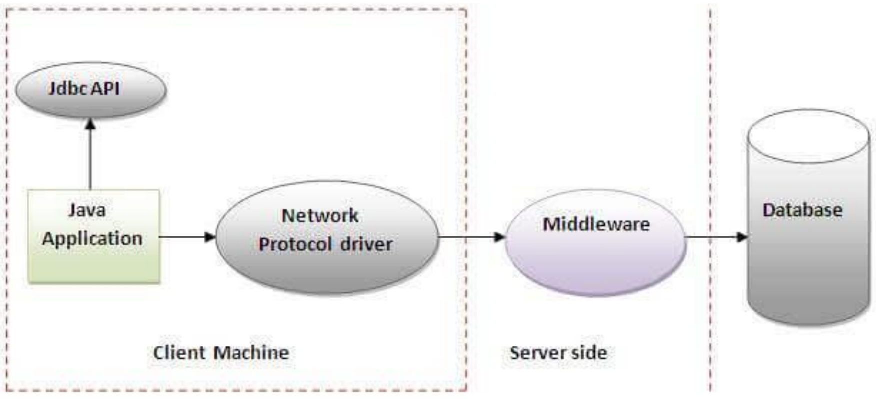


Figure- Network Protocol Driver

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.

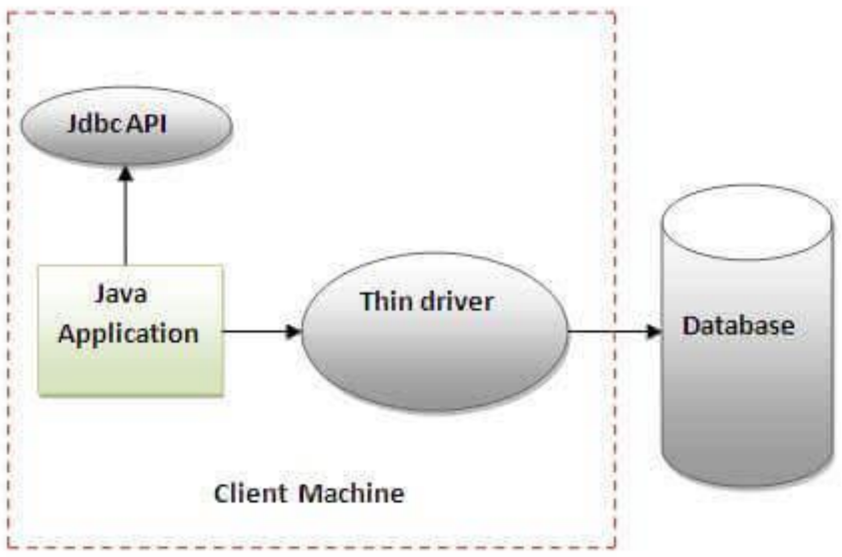


Figure- Thin Driver

Advantage:

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

Disadvantage:

- Drivers depend on the Database.

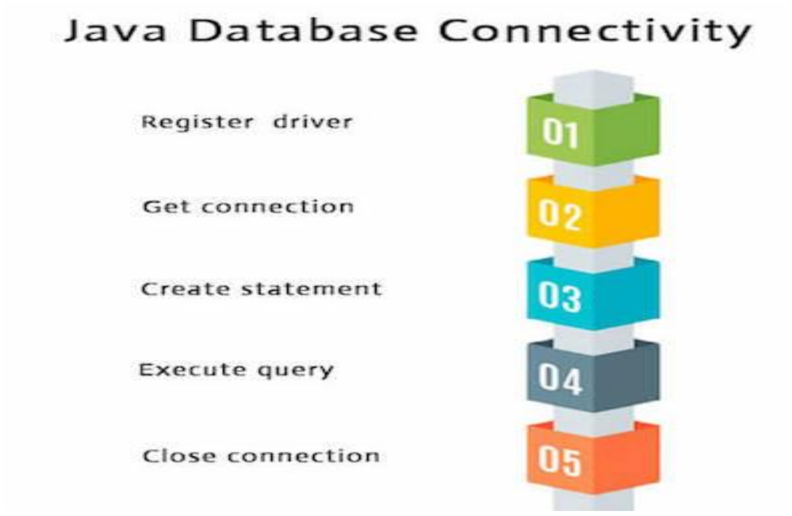
Java Database Connectivity with 5 Steps

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

Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.

Example to register the OracleDriver class

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

1. Class.forName("com.mysql.cj.jdbc.Driver ");

---

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) **throws** SQLException

Example to establish connection with the Oracle database

1. Connection con=DriverManager.getConnection(
2. " jdbc:mysql://localhost:3306/sonoo ","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

```
public Statement createStatement()throws SQLException
```

Example to create the statement object

```
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

```
public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

---

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

```
public void close()throws SQLException
```

Example to close connection

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

Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.

It avoids explicit connection closing step.

Example to Connect Java Application with mysql database

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

```
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);}
}
}
```

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;.;