UNIT V

N. MANIKANDAN
A.P/CS

```
f.setLayout(null);

f.setSize(400, 400);

f.setVisible(true);
```

10. Image

There can be a single image or multiple images within a UI. There can be a button being associated with an image and when it is clicked it can produce some functionality.

Syntax:

```
Image i=t.getImage("pic2.gif");
```

11. Scroll Bar

The scroll bar like a normal one is used to scroll or move from a varied range of values. The user selects one value from those range of values.

ıtax:

```
rollbar s=new Scrollbar();
setBounds(100,100,
),100);
```

12. Dialog

: dialog is used to take some form of input from the user and ›duce it in a sequential manner.

ıtax:

```
= new Dialog(f , "Hello World", true);
```

13. File Dialog

m a file dialog, a user can select a file which he/she wishes to ›.

ıtax:

```
leDialog(Dialog parent)
```

UNIT-V

ng Controls in Java

is article, I am going to discuss **Swing Controls in Java** Examples. Please read our previous article, where we ıssed **Swings in Java**. At the end of this article, you will rstand the following swing controls in Java in detail with ples.

1. **JLabel**
2. **JRadioButton**
3. **ButtonGroup**
4. **JCheckBox**
5. **JTextField**
6. **JTextArea**
7. **JButton**
8. **Border**
9. **JComboBox**
10. **JTabbedPane**
11. **JPasswordField**
12. **Look and Feel Management in Java Swing**

JLabel

The object of the **JLabel** class may be a component for puttin text in a container. It's used to display one line of read-only tex The text is often changed by an application but a user cann edit it directly. It inherits the **JComponent** class.

Declaration: public class JLabel extends JComponent implements SwingConstants, Accessible

Syntax: JLabel jl = new JLabel();

JLabel Constructors

1. **JLabel():** It is used to create a JLabel instan with no image and with an empty string for t title.
2. **JLabel(String s):** It is used to create a JLal instance with the specified text.
3. **JLabel(Icon i):** It is used to create a JLal instance with the specified image.
4. **JLabel(String s, Icon I, horizontalAlignment):** It is used to create JLabel instance with the specified text, ima and horizontal alignment.

Example to understand JLabel Swing Control in Java

Scanned with AltaScanner

```
import javax.swing.*; import
java.awt.*; public class JLabelDemo
extends JFrame
{
JLabel jl;
JLabelDemo ()
{
jl = new JLabel ("Good Morning");
Container c = this.getContentPane ();
c.setLayout (new FlowLayout ());
c.setBackground (Color.blue);
Font f = new Font ("arial", Font.BOLD, 34);  jl.setFont (f);
jl.setBackground (Color.white);  c.add (jl);  this.setVisible
(true);  this.setSize (400, 400);  this.setTitle ("Label");
this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
}  public static void main
(String[]args)
{
new JLabelDemo ();
}
}
```

ut

and Swing in Java

and Swing are used to develop window-based applications in Awt is an abstract window toolkit that provides various ponent classes like Label, Button, TextField, etc., to show window ponents on the screen. All these classes are part of the Java.awt ge.

e other hand, Swing is the part of JFC (Java Foundation Classes) on the top of AWT and written entirely in Java. The javax.swing provides all the component classes like JButton, JTextField, ckbox, JMenu, etc. The components of Swing are platform-endent, i.e., swing doesn't depend on the operating system to the components. Also, the Swing's components are lightweight. main differences between AWT and Swing are given in the ving table.

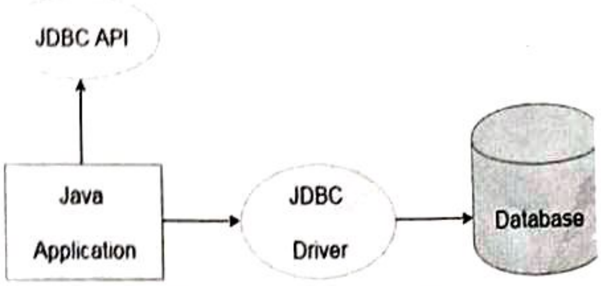
JDBC stands for Java Database Connectivity. JDBC is a Java API t connect and execute the query with the database. It is a part of JavaS (Java Standard Edition). JDBC API uses JDBC drivers to connect wi the database. There are four types of JDBC drivers:

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

o Thin Driver

We have discussed the above four drivers in the next chapter.

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



The current version of JDBC is 4.3. It is the stable release since 2 September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and rfaces for JDBC API. A list of popular *interfaces* of JDBC   Ora

are given   2) Native-API driver   does

○ Driver interface   The Native API driver uses the client-side libraries of the datab ○   Connection interface   database   supp

API. It is not written entirely in java. ○ Statement interface ○   PreparedStatement interface ○   CallableStatement

interface ○ ResultSet interface ○ ResultSetMetaData   JDE

interface ○ DatabaseMetaData interface   OD

○ RowSet interface   Bri

fr

Java

Ora

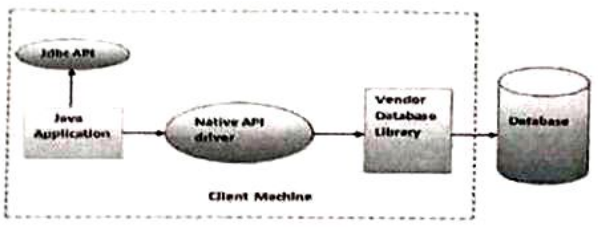


Figure- Native API Driver

st of popular *classes* of JDBC API are given below:

Advantage:

○ DriverManager class ○ performance upgraded than JDBC-ODBC bridge
○ Blob class driver.
○ Clob class

Disadvantage:

○ Types class ○   The Native driver needs to be installed on the each

client machine.

3C Driver ○ The Vendor client library needs to be installed on

3C Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers: client machine.

1. JDBC-ODBC bridge driver

3) Network Protocol driver

2. Native-API driver (partially java driver)

The Network Protocol driver uses middleware (application

3. Network Protocol driver (fully java driver) server) that converts JDBC calls directly or indirectly into the
4. Thin driver (fully java driver) vendor-specific database protocol. It is fully written in java.

JDBC-ODBC bridge driver

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

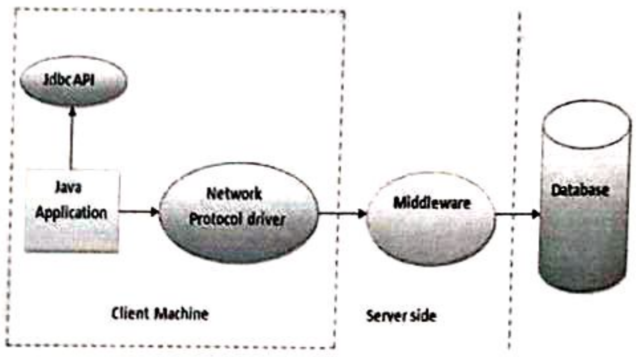


Figure- Network Protocol Driver

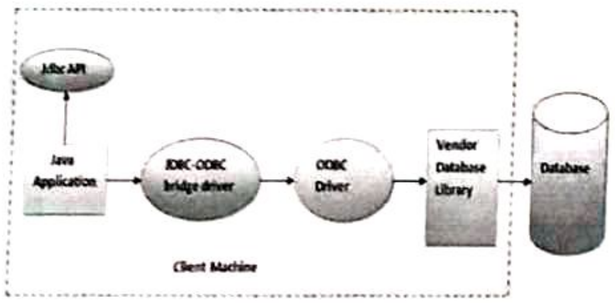


Figure- JDBC-ODBC Bridge Driver

recommends that you use JDBC drivers provided by vendor of your database instead of the JDBC-ODBC Brid

Advantages: ○

easy to use.

○ can be easily connected to any database.

Disadvantages:

○ Performance degraded because JDBC method cal converted into the ODBC function calls.

Scanned with AltaScanner

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

antage:

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

advantages:

- 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.

hin driver

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.

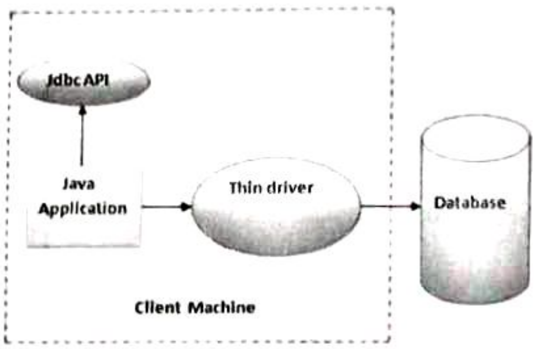


Figure- Thin Driver

Advantage:

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

Disadvantage:

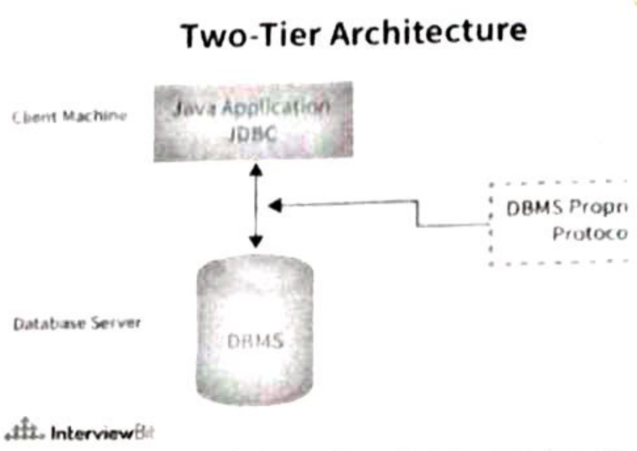
- Drivers depend on the Database.

JDBC Architecture

There are two architectures of JDBC:

Two-Tier Architecture

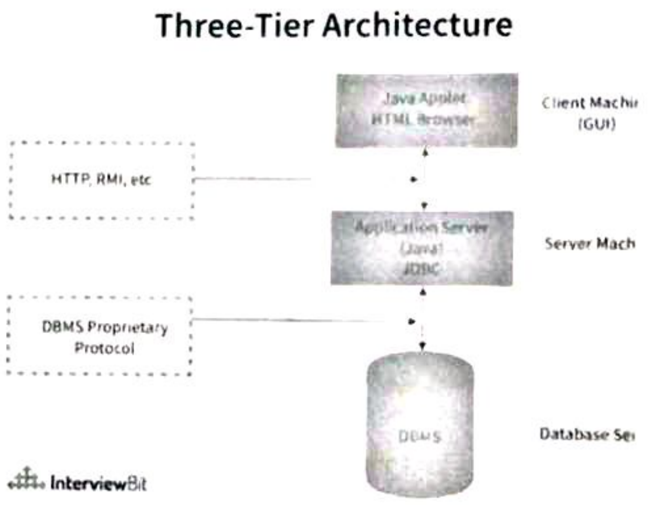
A Java applet or application communicates directly with the data source in the two-tier paradigm. This necessitates the use of a JDBC driver that can interface with the data source in question. The user's commands are transmitted to the database or other data source, and the statements' results are returned to the user. The data source could be on another machine to which the user has a network connection. A client/server configuration is one in which the user's machine acts



as the client and the system that houses the da source acts as the server. An intranet, for example, ca connect people within a company, or the Internet ca be used as the network.

Three Tier Architecture

Commands are sent to a "middle tier" of services in the three-tier paradigm, which subsequently transmits the commands to the data source. The data source interprets the commands an provides the results to the middle tier, which ultimately passes them on to the user. The threetier architecture appeals to MIS directors because the intermediate tier allows them to maintai control over access and the types of changes that can be mad to company data. Another benefit is that it makes application deployment easier. Finally, the three-tier architecture can brin performance benefits in many circumstances.



The components of JDBC are listed below. These elements assist us in interacting with a database. The following are the JDBC components:

Scanned with AltaScanner

1. **JDBC Driver Manager:** In a JDBC application, the Driver Manager loads database-specific drivers. This driver manager makes a database connection. To handle the user request, it additionally makes a database-specific call to the database.
2. **Driver:** A driver is an interface that manages database server connectivity. Communication is handled using DriverManager objects.
3. **JDBC-ODBC Bridge Drivers:** They are used to link database drivers to the database. The JDBC method calls are translated into ODBC method calls by the bridge. To access the ODBC (Open Database Connectivity) characteristics, it uses the sun jdbc odbc package, which includes the native library.
4. **JDBC API:** Sun Microsystem has provided JDBC API, which allows you to write a Java program that talks with any database without modifying the code. The JDBC API is implemented by the JDBC Driver.
5. **JDBC Test Suite:** The JDBC Test Suite aids in the testing of JDBC Driver operations such as insertion, deletion, and updating. It aids in determining whether or not the JDBC Drivers will run the program. It ensures that the program will be run by JDBC Drivers with confidence and conformity.
6. **Database Server:** This is the database server that the JDBC client wants to communicate with, such as Oracle, MySQL, SQL Server, and so on.
7. **Statement:** To send SQL statements to the database, you use objects built using this interface. In addition to performing stored procedures, certainly derived interfaces accept parameters.
8. **RuleSet:** These objects retain data retrieved from a database when you use Statement objects to conduct a SQL query. It functions as an iterator, allowing you to cycle through the data it contains.
9. **SQL Exception:** This class is responsible for any errors that occur in a database application.

JDBC CLASSES & INTERFACES

JDBC API is available in two packages java.sql, core API and javax.sql JDBC optional packages. Following are the important classes and interfaces of JDBC.

Steps for developing JDBC Application

1. Load and register Driver Class
2. Establish Connection between Java Application and Database
3. Create Statement Object
4. Send and execute SQL Query
5. Process Result from ResultSet
6. Close Connection

1. Load and register Driver Class

JDBC API is a Set of Interfaces defined by Java Vendor.

Database Vendor is responsible to provide

Implementation. This Group of Implementation Class

is nothing but "Driver Software". We have to make t

Driver Software available to our Java Program. For t

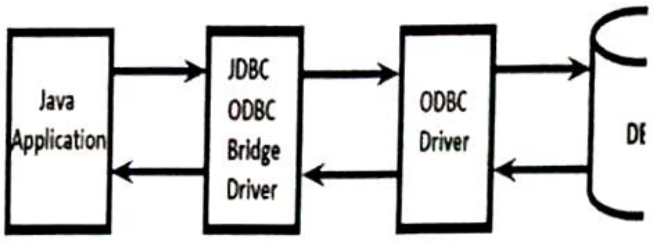
we have to place corresponding Jar File in the Cla

Path.

2. Establish Connection between Java Application and Databa

Once we loaded and registered Driver, by using that we c

establish Connection to the Database. For this



DriverManager Class contains

ODBC Driver needs Database Name & its Location to conn

with Database. ODBC Driver collect this

Information from DSN i.e. internally ODBC Driver will use DSN

get Database Information (DSN Concept

applicable only for Type-1 Driver). There are 3 Types of

DSN

1. User DSN It is the non-sharable DSN and available only for

Current User.

2. System DSN

It is the sharable DSN and it is available for all Users who c

access that System. It is also known as Global DSN.

3. File DSN

It is exactly same as User DSN but will be stored in a File w

.dsn Extension.

According to Database Specification, all SQL Commands a

divided into following Types

1. DDL (Data Definition Language) Commands

Scanned with AltaScanner

Create Table, Alter Table, Drop Table Etc

2. DML (Data Manipulation Language) Commands

E.g. Insert, Delete, Update

3. DQL (Data Query Language) Commands

E.g. Select

4. DCL (Data Control Language) Commands E.g.

Alter Password, Grant Access Etc.

5. Data Administration Commands

E.g. Start Audit

Stop Audit

6. Transactional Control Commands

Commit, Rollback, Savepoint Etc

According to Java Developer Point of View, all SQL

Operations are divided into 2 Types

1. Select Operations (DQL)
2. Non-Select Operations (DML, DDL Etc)

Once we create Statement Object, we can call the

following Methods on that Object to execute our

Queries

1. executeQuery()
2. executeUpdate()
3. execute()

1. executeQuery() Method

We can use this Method for Select Operations. Because

of this Method Execution, we will get a Group of

Records, which are represented by ResultSet Object.

Hence the

Return Type of this Method is ResultSet

2. executeUpdate() Method

We can use this Method for Non-Select Operations

(Insert|Delete|Update). Because of this Method

Execution, we won't get a Group of Records and we will

get a Numeric Value represents the Number of Rows

effected. Hence Return Type of this Method is int

3. execute() method

We can use this Method for both Select and Non-Select

6

Operations. If we don't know the Type of Query at

beginning and it is available dynamically at run time th

we should use this execute() Method

executeQuery() Vs executeUpdate() Vs execute()

- If we know the Type of Query at th beginning and it is always Select Quer then we should use "executeQuery Method"
- If we know the Type of Query at th beginning and it is always Non-Sele Query then we should us executeUpdate() Method
- If we don't know the Type of SQL Query a the beginning and it is availabl dynamically at Runtime (May be fro Properties File OR From Comman Prompt Etc) then we should go fo execute() Method

5. Process Result from ResultSet

After executing Select Query, Database Engine will send Resu

back to Java Application. This Result is available in the form o

ResultSet

i.e., ResultSet holds Result of executeQuery() Method, whic

contains a Group of Records. By using ResultSet we can g

Results. ResultSet is a Cursor always locating Before Fir

Record (BFR). To check whether the next Record is availabl

OR not, we have to use rs.next() Method. This Method Return

True if the next

Record is available, otherwise returns False.

To create a table in a database using JDBC API you need to:

- **Register the driver**: Register the driver class using th **registerDriver()** method of the **DriverManager** clas. Pass the driver class name to it, as parameter.
- **Establish a connection**: Connect ot the database usin the **getConnection()** method of the **DriverManage** class. Passing URL (String), username (String password (String) as parameters to it.
- **Create Statement**: Create a Statement object using th **createStatement()** method of the **Connection** interfac
- **Execute the Query**: Execute the query using th execute() method of the Statement interface.

Example

Following JDBC program establishes connection with MySQL a creates a table named customers in the database nam **SampleDB**

Scanned with AltaScanner

```
ıort  java.sql.Connection;  import  java.sql.DriverManager;
ıort  java.sql.SQLException;  import  java.sql.Statement;
ılic  class  CreateTableExample  {      public  static  void
n(String args[]) throws SQLException {
//Registering the Driver
DriverManager.registerDriver(new
ı.mysql.jdbc.Driver());
//Getting the connection
String mysqlUrl = "jdbc:mysql://localhost/SampleDB";
                         Connection        con        =
verManager.getConnection(mysqlUrl, "root", "password");
System.out.println("Connection established......");
//Creating the Statement
Statement stmt = con.createStatement();
//Query to create a table
String query = "CREATE TABLE CUSTOMERS("
  + "ID INT NOT NULL, "
  + "NAME VARCHAR (20) NOT NULL, "
  + "AGE INT NOT NULL, "
  + "SALARY DECIMAL (18, 2), "
  + "ADDRESS CHAR (25) , "
PRIMARY KEY (ID))";
t.execute(query);
System.out.println("Table Created......");
```

```
:put mysql> show tables;
-----------------+
bles_in_sampledb |
-----------------+
cles          |
tomers        |
ɔatches       |
ınologies     |
ırial         |
-----------------+
vs in set (0.00 sec)
```

Scanned with AltaScanner