JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks commonly associated with database usage:

* Making a connection to a database.
* Creating SQL statements.
* Executing that SQL queries in the database.
* Viewing & Modifying the resulting records.

JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as:

* Java Applications
* Java Applets
* Java Servlets
* Java ServerPages (JSPs)
* Enterprise JavaBeans (EJBs)

All of these different executables can use a JDBC driver to access a database and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

JDBC Architecture:

The JDBC API supports both two-tier and three-tier processing models for database access but in general JDBC Architecture consists of two layers:

* JDBC API: This provides the application-to-JDBC Manager connection.
* JDBC Driver API: This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Common JDBC Components:

The JDBC API provides the following interfaces and classes:

* DriverManager: This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication subprotocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database connection.
* Driver: This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manage objects of this type. It also abstracts the details associated with working with Driver objects
* Connection: This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
* Statement: You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
* ResultSet: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
* SQLException: This class handles any errors that occur in a database application.

The java.sql and javax.sql are the primary packages for JDBC. It offers the main classes for interacting with your data sources.

## Creating JDBC Application:

Steps involved in building a JDBC application:

* Import the packages. Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.\** will suffice.
* Register the JDBC driver. Requires that you initialize a driver so you can open a communications channel with the database.
* Open a connection. Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database.
* Execute a query. Requires using an object of type Statement for building and submitting an SQL statement to the database.
* Extract data from the result set. Requires that you use the appropriate *ResultSet.getXXX()* method to retrieve the data from the result set.
* Clean up the environment. Requires explicitly closing all database resources versus.

ORACLE🡪 ojdbc.jar

MYSQL🡪 MySQL-connection.5.18.jar

Class.forName(“com.mysql.jdbc.Driver”) 🡪 MySql

Class.forName(“oracle.jdbc.driver.OracleDriver”); 🡪 Oracle

Driver Manager:

Static connection getConnection(url,username,password) 🡪 is a factory method with 3 parameters which returns a connection objects. It is a static method.

String url = “jdbc:mysql://localhost:3306/sampledb”;

String username = “root”;

String password = ”root”;

Or

String url =”jdbc:oracle:thin:@localhost:1521:xe”;

String username = “system”;

String password =”manager”;

Conenction con = DriverManager.getConnection(url,username,password);

Statement: it is an interface ( we can create reference)

Int executeUpdate(String) //insert /update/delete

ResultSet executeQuery(String) // select

Connection:

Statement createStatement(); // Static Factory Method();

ORACLE🡪 ojdbc.jar

MYSQL🡪 MySQL-connection.5.18.jar

Class.forName(“com.mysql.jdbc.Driver”) 🡪 MySql

Class.forName(“oracle.jdbc.driver.OracleDriver”); 🡪 Oracle

Driver Manager:

Static connection getConnection(url,username,password) 🡪 is a factory method with 3 parameters which returns a connection objects. It is a static method.

String url = “jdbc:mysql://localhost:3306/sampledb”;

String username = “root”;

String password = ”root”;

Or

String url =”jdbc:oracle:thin:@localhost:1521:xe”;

String username = “system”;

String password =”manager”;

Conenction con = DriverManager.getConnection(url,username,password);

Statement: it is an interface ( we can create reference)

Int executeUpdate(String) //insert /update/delete

ResultSet executeQuery(String) // select

Connection:

Statement createStatement(); // Static Factory Method();