Java JDBC

JDBC stands for Java Database connectivity. It is a standard API provided by oracle for java application to interact with different set of database i.e Oracle, Microsoft Access, MySQL, SQL Server etc.

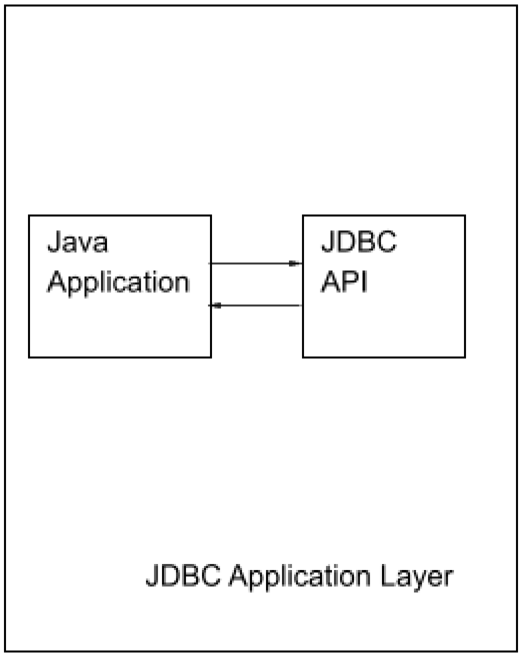
JDBC is a API to connect and execute query with the database. JDBC API uses jdbc database drivers to connect with the database.

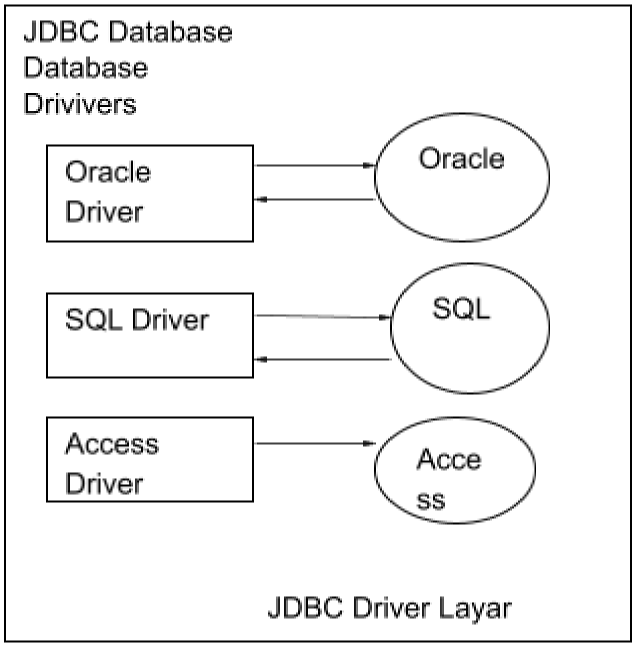
Why JDBC?

```
Class Test
{
public static void main(String[] args) {
        int eid=1001;
        String ename="Manjeet";
        double salary=20000;
        System.out.println(eid+"\t"+ename+"\t"+salary);
   }
}
```

JDBC is used to store data permanently into the database.

How JDBC Work (Architecture of JDBC)





JDBC Drivers

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

JDBC API

- java.sql.DriverManager
- java.sql.Connection
- java.sql.Statement
- java.sql.ResultSet
- java.sql.PreparedStatment
- java.sql.SQLException

Steps to :

Set JDBC Driver class-path for Notepad

1. **Download the driver for database**

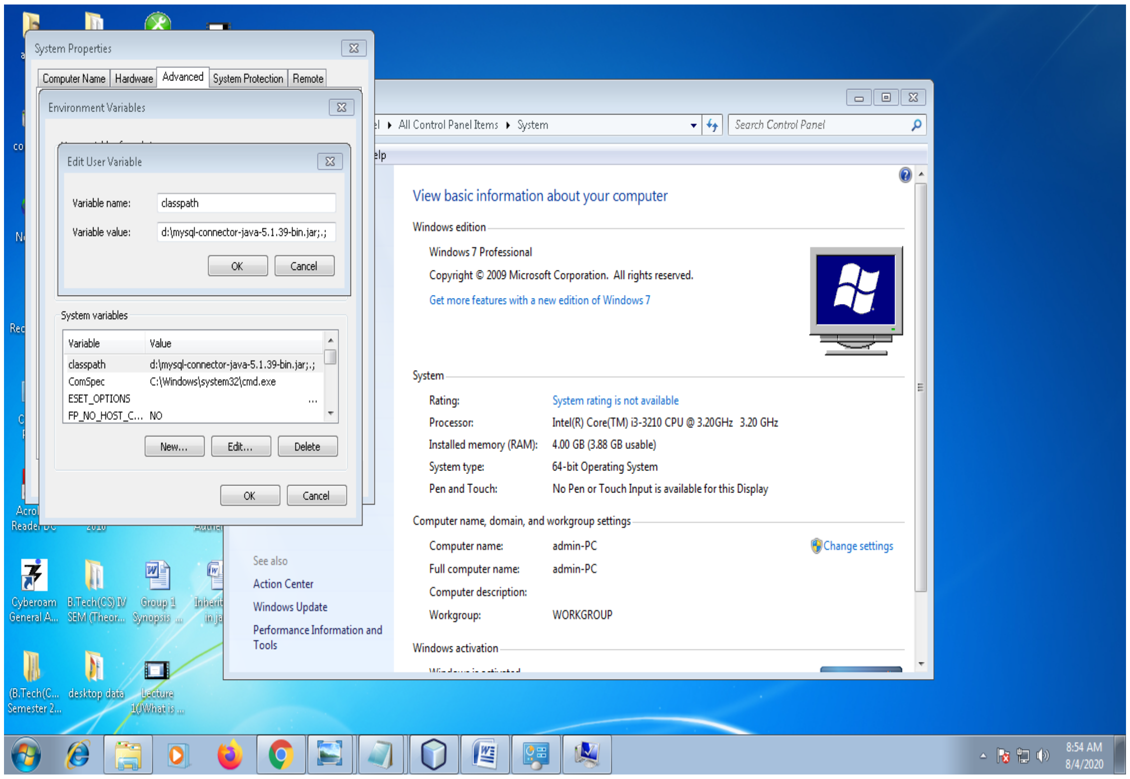
   Eg: for MySQL

   Mysql-connector.jar

2. **Set the MySQL connector jar file to the classpath:**

   Computer -> Advanced System Settings -> Environment Variable

   Click on *New button*


Edit User Variable

Variable name: classpath

Variable value: d:\mysql-connector-java-5.1.39-bin.jar;.;

System variables

Variable	Value
classpath	d:\mysql-connector-java-5.1.39-bin.jar;.;
ComSpec	C:\Windows\system32\cmd.exe
ESET_OPTIONS	...
FP_NO_HOST_C...	NO

View basic information about your computer

Windows edition

Windows 7 Professional

Copyright © 2009 Microsoft Corporation. All rights reserved.

System

Rating:	System rating is not available
Processor:	Intel(R) Core(TM) i3-3210 CPU @ 3.20GHz 3.20 GHz
Installed memory (RAM):	4.00 GB (3.88 GB usable)
System type:	64-bit Operating System
Pen and Touch:	No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name:	admin-PC
Full computer name:	admin-PC
Computer description:	
Workgroup:	WORKGROUP

Steps to :

Set JDBC Driver for NetBeans IDE/Add jar file to the application

1. **Right click on Project Name i.e JavaApplication1 and select properties**

   Libraries

   Click on Add Jar/Folder then a dialog box applear

   Then go to c:\program files\java\jdk\db\lib and then select

   derby and derbyclient JAR file from lib folder

   And clikc on OK button.

Five Steps to connect to the database in java

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing 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

public static void forName(String className)throws ClassNotFoundException

Driver for Oracle

Class.forName("oracle.jdbc.driver.OracleDriver");

Driver for MS-ACCESS

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Driver for MySQL

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

Driver for derby Database

Class.forName("org.apache.derby.jdbc.ClientDriver");

2) Create the connection object

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

Syntax of getConnection() method

1) public static Connection getConnection(String url)throws SQLException

2) public static Connection getConnection(String url,String name,String password)

throws SQLException

Example to establish connection with the Oracle database

Connection conn=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","password");

Example to establish connection with the MS-ACCESS database

Connection conn=DriverManager.getConnection("jdbc:odbc:mydsn");

Example to establish connection with the MySQL database

Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/manjeet","root","");

Example to establish connection with the deyby database

Connection

conn=DriverManager.getConnection("jdbc:derby://localhost:1527/bns","bns","bns");

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(), executeUpdata() method of Statement interface is used to execute queries to the database.

executeQuery() method is used to execute SELECT query. It returns the object of ResultSet that can be used to get all the records of a table.

executeUpdate() is used to execute specified query, it may be create, drop, insert, update, delete etc.

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();
```

Example to connect to the MySQL database

```
import java.sql.*;

import java.util.*;

class query{

public static void main(String args[]){

try{

Class.forName("com.mysql.jdbc.Driver");    //driver for MySQL

Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/manjeet","root","");

Statement stat=conn.createStatement();
ResultSet rs=stat.executeQuery("select * from stud");
        while(rs.next()){
        //System.out.print(rs.getInt(1));
        System.out.println(rs.getInt(1)+" "+rs.getString(2));
        }
conn.close(); }//try close
catch(Exception e){
System.out.println(e);}
}}
```

DriverManager class:

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Commonly used methods of DriverManager class:

- **public static Connection getConnection(String url):-** is used to establish the connection with the specified url.
- **public static Connection getConnection(String url,String userName,String password):-** userName,String password): is used to establish the connection with the specified url, username and password.

Connection interface:

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement. The Connection interface provide many methods for transaction management like commit(),rollback() etc.

- **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.
- **public Statement createStatement(int resultSetType,int resultSetConcurrency**): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
- **public void commit():** saves the changes made since the previous commit/rollback permanent.
- **public void rollback():** Drops all changes made since the previous commit/rollback.
- **public void close():** closes the connection and Releases a JDBC resources immediately.

Statement interface

The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

- **public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
- **public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
- **public boolean execute(String sql):** is used to execute queries that may return multiple results.

Example

```
import java.sql.*;

import java.util.*;

class query{

public static void main(String args[]){

try{

Class.forName("com.mysql.jdbc.Driver");    //driver for MySQL

Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/manjeet","root","");

Statement stat=conn.createStatement();

int x=stat.executeUpdate("delete from stud where rno="+args[0]+"");

if(x>0)

System.out.println("Record deleted successfully.");

}//try close

catch(Exception e){
```

```
System.out.println(e);}}}
```

ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction.

Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,

ResultSet.CONCUR_UPDATABLE);

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

```
import java.sql.*;

class FetchRecord{

public static void main(String args[])throws Exception{


Class.forName("com.mysql.jdbc.Driver");    //driver for MySQL

Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/manjeet","root","");

Statement stmt=con.createStatement();

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


//getting the record of 3rd row

rs.absolute(3);

System.out.println(rs.getInt(1)+" "+rs.getString(2));


conn.close();

}}
```

PreparedStatement interface

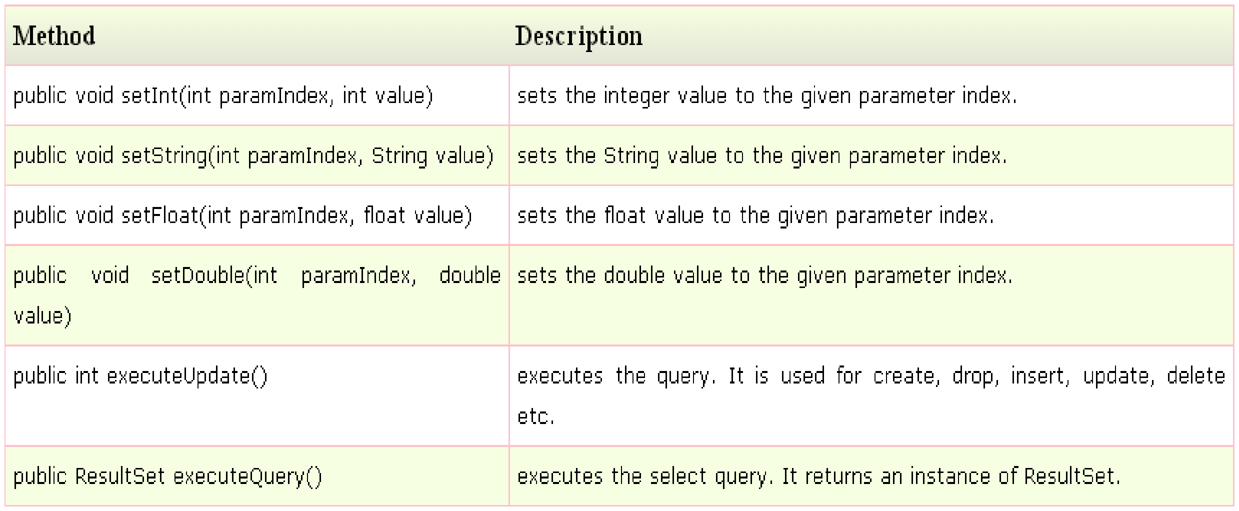
The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query. The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

example of parameterized query

```
String query="insert into emp values(?,?,?)";
```

Note: we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Methods of PreparedStatement interface



Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

```
import java.sql.*;

import java.util.*;

class query{

public static void main(String args[]){

try{

Class.forName("com.mysql.jdbc.Driver");    //driver for MySQL

Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/manjeet","root","");
Scanner scan=new Scanner(System.in);

System.out.println("Enter the first name to be deleted:");

String n=scan.next();

PreparedStatement stat=conn.prepareStatement("delete from stud where rno=? ");

stat.setString(1,n);

int x=stat.executeUpdate();

        if(x>0)

        {

        System.out.println("Record Deleted Successfully...");

        conn.close();

        stat.close();

        }

        else

        System.out.println("Record Not found..");

}

catch(Exception e){System.out.println(e);}

}

}
```