Jspiders
Training & Development Center

Author:-Shishira Bhat
Technical Architect

JDBC

→ When the method returntype is superclass then that method can return superclass object.
→ When the method parameter is of superclass, then the argument must be an subclass object.

JAR Files :

→ Jar file means a java archive file ie it's a compressed folder which contains java and related files.
→ Generally the jar file contains .class files and other configuration files.
→ .xml and .properties are general configuration files.

Java Archives :

→ Generally we dont share .java files in jar file.
→ Command to creat a jar is given below

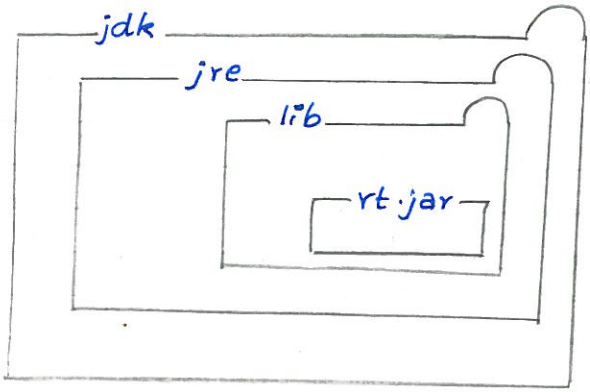
Adding libraries into the project (Build Path)

→ It is always a good practice to copy the libraries into the project.

Steps to be followed :

1. Right click on the project → select properties
2. Select java build path → Select libraries tab
3. Click on add jars
4. Browse the jar file → click on OK

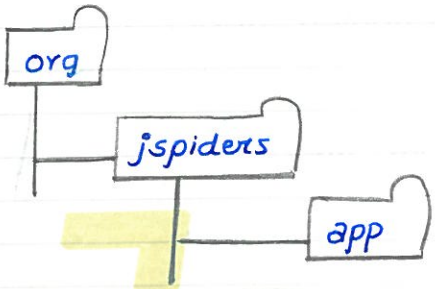
→ If the member of the jar file is public, then only we can access them.
→ Before accessing members of jar file we need to import the class.
→ all 3000+ inbuilt classes of java are present in rt.jar with appropriate package structure.



Author:-Shishira Bhat
Technical Architect

Class Loading :-

- When any of the member of the class is called, then that class is loaded by class loader.
- Bringing the class into JVM memory from the hard disk can be considered as class loading.
- Members of class are
  a. methods
  b. variables
  c. constructors
  d. Blocks
- It is possible to load a class without calling any of its members.



- By using a static method called forName
- forName method is present in java.lang.Class
- Syntax of the method is

```
public static class forName (String fullyQualifiedClassName);
```

- The forName method throws a checked exception called ClassNotFoundException.

Program1 :

```
package com.jspiders.demoapp;
public class LoadTest
{
 public static void main (String [] args)
  {
   try
     {
       Class.forName ("com.jspiders.demoapp.Fish");
     }
   catch (ClassNotFoundException e)
   {
    e.printStackTrace ();
    System.out.println ("Class not loaded");
   }
  }
}
```

```
package com.jspiders.demo
public class Fish
{
 static
  {
  S.o.p("9m loaded");
  }
}
```


o/p :-

9m loaded.

Static Blocks :-

* A static block is the one which has some logic as like any other method.
* Static block is executed only ~~at~~ once when a class is loaded.
* Static blocks are executed with high priority ie static block is executed before the execution of any other member.
* When we have any no. of static blocks & in this case, execution of blocks are sequential.
* Static blocks cannot be inherited.
* Static blocks cannot be called explicitly indeed they are called by JVM when a class is loaded.

JDBC Configurations :

→ All JDBC configurations data must be global constant.
→ All examples of JDBC constants
1. DB username
2. DB password
3. DB URL

Design Principles :

"Program an interface, but not an implementation"

Ex:
```
List lst = new ArrayList();
lst.add("shishira");
```

Ex:
```
ISwitch sw = new LedLight9mpl();
sw.switchOn();
sw.switchOff();
```

→ The subclass of an interface is called as implementation class.
→ The object of subclass or implementation class is called as implementation object

Advantages of interfaces :-
1. Loose coupling
2. abstraction.

Factory Design Pattern :
Program 2 :-

```
package org.jspiders.demoapp;
public interface ISwitch
{
  void switchOn();
  void switchOff();
}
```



```
package org.jspiders.demoapp;
 /*
  implementation class
 */
public class LedLightImpl implements ISwitch
{
  @Override
   public void switchOn()
   {
    S.o.p(" LED light switched on ");
   }
 @Override
  public void switchOff()
  {
   S.o.p(" LED light switched off ");
  }
}
```

```
// Write SolarLightImpl & MercuryLightImpl classes same as LedLightImpl

package org.jspiders.demoapp;
public class LightFactory
{
 /* Helper method or factory method
 * @author Shishira Bhat
 * @param typeOfLight
 * Helper method
 */
 public static ISwitch getLight(String typeOfLight)
 {
  if (typeOfLight.equals("led"))
```

```
    {
     return new LedLightImpl();
    }
    else if ( typeOfLight.equals ("solar");
    {
     return new SolarLightImpl();
    }
    else
    {
     return new MercuryLightImpl();
    }
  }
}
```



```
package org.jspiders.demoapp;
import java.util.Scanner;
public class Test
{
  public static void main(String[] args)
  {
    User user = new User();
     user.read();
  }
}
```

```
package org.jspiders.demoapp;
import java.util.Scanner;
public class User
{
  public void read()
   {
    Scanner scan = new Scanner(System.in);
    S.o.p("Enter light of your choice");
    String typeOfLight = scan.next();
    ISwitch sw = LightFactory.getLight( typeOfLight);
     sw.switchOn();
     sw.switchOff();
   }
 }
}
```

Output:-

```
Enter light of your choice
led
LED light switched on.
LED light switched off.
```

Object Constants and Global Constants:-

→ Object constants are the constants which are applicable for an object. where as global constant is the one which is applicable for the entire application.

→ Ex: `public final String CARMODEL = "alto"; // object constant`
`public static final double PIE = 3.142; // global constant`

→ Object constant has one copy for one object. [instance]

→ It is a good practice to define the global constants in an interface. because all the data members in an interface are public static final.

→ It's again a good practice that final data members must be in uppercase.

→ Data members of an interface can be accessed by using interface name. because it is static.

Program 3:

```
package org.jspiders.demoapp;
public interface AppConstants
{
 String USER_NAME = "scott";
 String PASS = "tiger";
 double PIE = 3.142;
 int AGE = 16;
}
```

```
package org.jspiders.demoapp;
 public class DbUser
 {
   public static void main(String[] args)
    {
     S.o.p("AppConstants.USERNAME);
     S.o.p(AppConstants.PASS);
    }
}
```


Output:-

```
scott
tiger
```

Jspiders
Training & Development Center

Author:-Shishira Bhat
Technical Architect

Static Import :

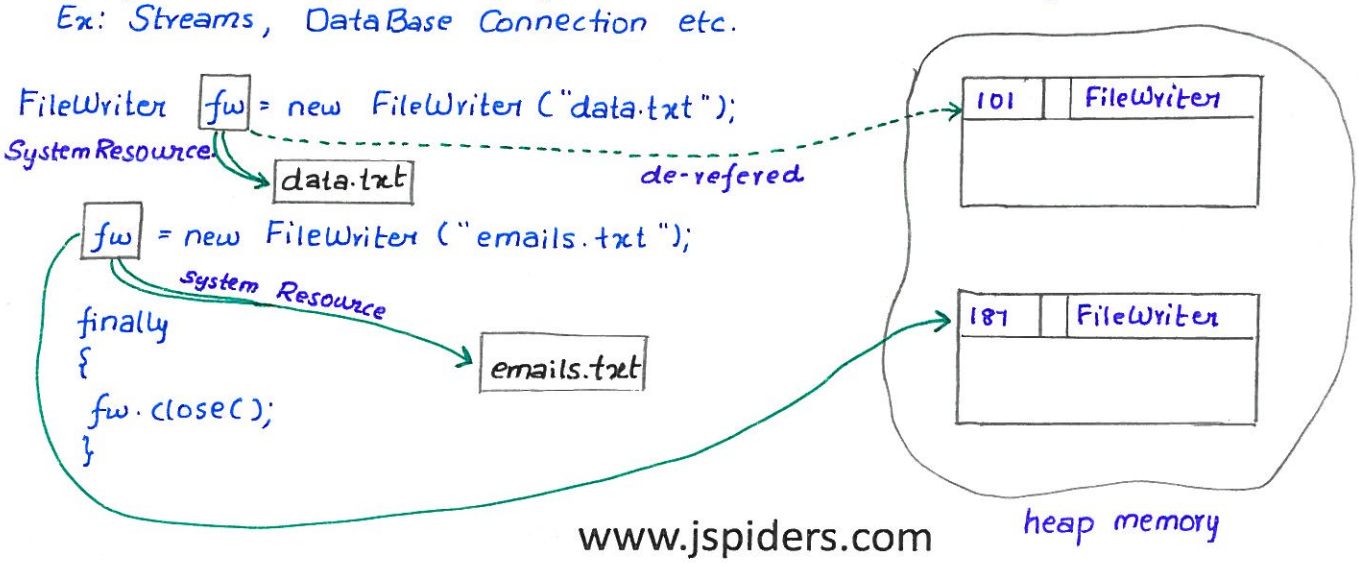
> Static import is a feature of java which is introduced in jdk1.5
> This feature imports all static members either from an interface or abstract class or a class
> If we use static import then the static members can be accessed without using the classname or interface name.

Program 4 :

```
package org.jspiders.demoapp;
import static org.jspiders.demoapp.AppConstants.*;
public class DbUser
{
  public static void main(String[] args)
  {
    S.o.p(PASS);
    S.o.p(USER_NAME);
  }
}
```


Costly Resources or Heavy Weight Object :

→ Generally most of the logical objects are lightweight objects. ie once an object is created in the heap memory, it may not use the system resources hence once the object is derefered, the logical object which is present in heap memory is destroyed.
→ Sometimes when we creat the logical objects, they start using the system resources and when the logical object is derefered, logical objects are garbage collected but system resources are not released.
→ When we have such more and more objects which uses system resources the application performance is reduced. Hence it is very important to close such costly resources and release the system resource.
→ Costly resources can be closed using the close method [close()]
→ We generally close the costly resources inside the finally block
Ex: Streams, DataBase Connection etc.

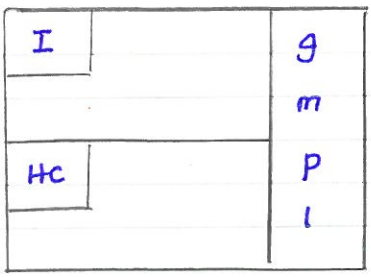


Author:-Shishira Bhat
Technical Architect

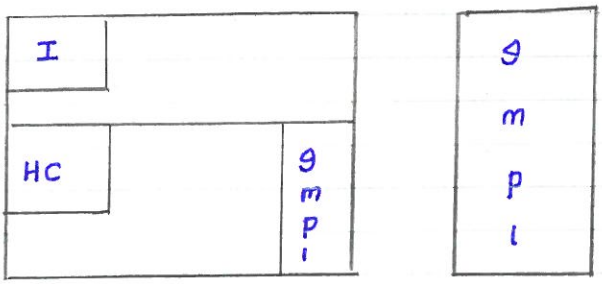
Application Programming Interface :-

→ API is a collection of interfaces, helper classes and implementation classes which is used for inter application communication in a loosely coupled way.
→ If java based API then it is generally given in the form jar file.
→ API is the physical form of abstraction
→ API's are generally given by application providers
→ There are many categories of API's.

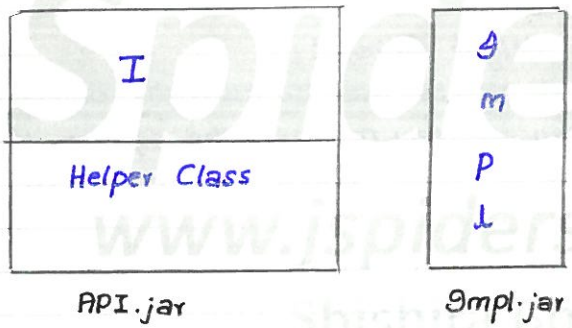
1)



2)



3)



JDBC :-

Drivers :-

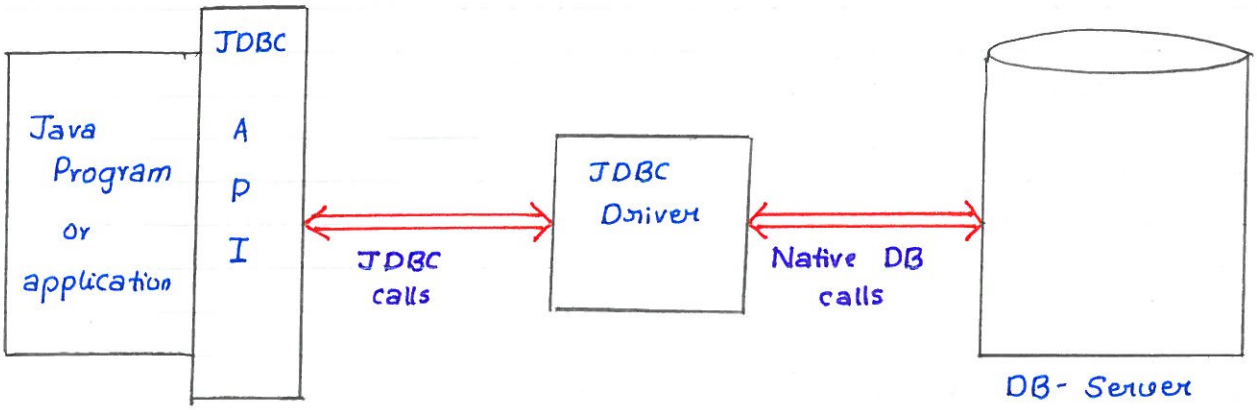
* Driver is the piece of code or set of instructions which helps to communicate between 2 different n/w systems, h/w and a software and b/w multiple software.
* A driver can be written in any of the programming language like C, C++, java etc.
* Driver works like a translator.
  Examples : Printer Driver, Camera Driver etc.
* The drivers which helps to communicate with the devices are called as device drivers.

JDBC Drivers :-

→ JDBC driver is a piece of code which helps to communicate from a java program or application with a database server.
→ JDBC driver works as a translator b/w java application and DB-Server
→ JDBC driver translates JDBC instructions (JDBC calls) into native database instructions (native DB Calls).
→ There are different types of JDBC drivers called type1, type2, type3, type 4 drivers.

→ Type 4 drivers are written in java language.

JDBC Architecture :-



Ex: * MS - SQL Server
* IBM DB2
* Derby
* Oracle
* MySQL
* Sybase

→ We have to write java application or program by using JDBC API
→ JDBC API is used to achieve loose-coupling b/w java application and database servers.
→ When we make JDBC calls which are translated to native DB calls by JDBC drivers. Hence from java application to communicate with any DB-Server we have to use API as well as Driver.
→ Conclusion:
- Purpose of API is loose-coupling.
- Purpose of Driver is translation.

Servers :
* A server is a s/w which serves the client request
* Server does not create any resources on its own. Indeed it serves only if the resources are available.
* Different types of server
1. DB Server
2. Web Server
3. Application Server.
* Example for DB Server is oracle, sybase, My-SQL.
* Example for web Server Apache and Tomcat, jetty, Sun, GlassFish
* Example for app Server is JBoss, Weblogic, WebSphere, Dynamo.

DB - Client :-

→ DB - Client is a lightWeight or thin client Application using which we can communicate to local or remote server.
→ Example Toad, Squirrel or SQL - yog for MySQL, SQL Work Bench
→ The information which is required to access the remotely hosted server is called as host information.
→ Below are the host information
  a. IP address of the Server (DB)
  b. Port Number
  c. User Name
  d. Password.

Installation of MySQL and SQLYog Softwares :-

Double click on .exe file → click Yes → click on complete → Next → skip signup → click on next → check the option "configure mysql server" → click on Finish → next → select standard configuration → next → check the option "install as windows service" → check "Launch MySQL server automatically" → check "Include bin directory in windows path" → next → enter root password and confirm → check enable root access → next → next → Finish.

SQL - Yog :-

Double click → next → licence agreement → next → next → Finish → Connection name → OK → localhost → click on connect [test connection] → save changes.

Program 5 :-

```
package org.jspiders.app;
/* import classes from jar file */
import com.shishira.pdfapp.PDFGenerator;
import com.shishira.pdfapp.PDFHelper;
public class PDFUse
{
 public static void main(String[] args)
 {
  S.o.p(" App start ");
  /* get hidden implementation object from helper factory */
  PDFGenerator pdfgtr = PDFHelper.genPDF();
  /* abstraction */
   pdfgtr.generatePDF();
  S.o.p(" App end ");
 }
}
```

Jspiders
Training & Development Center

Author:-Shishira Bhat
Technical Architect

Properties :-

→ Properties is a type of data which stores the data in the form of key-value pair.
→ Properties file is platform and technology independant.
→ To read the data from a properties file, we have to use the
  * java.io.FileReader;
  * java.util.properties;
→ While reading the data from properties file we get checked exceptions
  * java.io.FileNotFoundException.
  * java.io.IOException.
→ FileReader is a costly resource hence it must be closed ideally inside the finally block.
→ To read the data we use 2 overloaded methods of java.util.properties class.
  1. public String getProperty(String key)
     > The method takes key as an argument
     > If the key is present then the method returns the associated value.
     > If the key is not present, then it returns null, but we dont get any exception or error.
  2. public String getProperty(String key, String message)
     > The method returns the associated value if the key is present
     > If the key is not present the message could be returned.

Program 6 :

```
package org.jspiders.app;
import java.io.*;
import java.util.properties;
public class PropertyReader
{
  public static void main(String[] args)
  {
    Properties props = new Properties();
    FileReader reader = null;
    try{
    reader = new FileReader("config/dbconfig.properties"); //FileNotFoundException.
    props.load(reader); //IOException.
    /* Read/get the property */
    String oracleUser = props.getProperty("ouser");
    String oraclePass = props.getProperty("opass");
    S.o.p(oracleUser + "      " + oraclePass);
    String mysqlUser = props.getProperty("usr" , "key not found");
    S.o.p(mysqlUser);}
    catch(Exception e)
    {
      e.printStackTrace();
```

```
    finally {
     try {
     /* close costly resources */
      reader.close();
     }
     catch (IOException e)
     {
      e.printStackTrace();
     }
   }
}
```


Output :-

scott tiger

key not Found.

JDBC API :-

→ JDBC API contains many interfaces, many implementations, classes and helper classes [Helper or Factory Implementations]
→ All these members are modularized into a package.
  * java.sql
  * javax.sql.
→ The basic features of JDBC are given in the form of API which is present in java.sql.package.
→ The extended feature of JDBC which are in the form of API are present javax.sql package.

Note :-

JDBC members are the part of JRE library. Hence they need not to be added to the project buildpath.

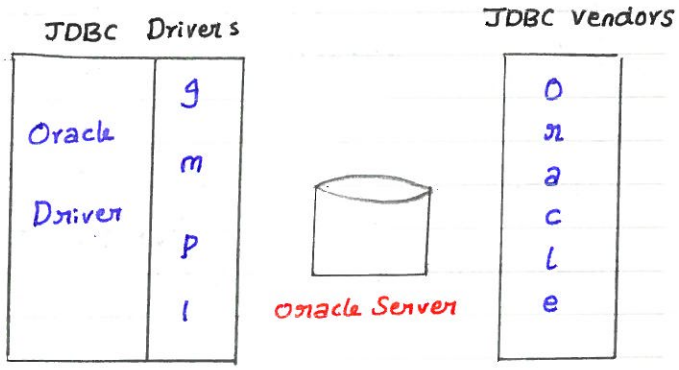
→ The JDBC API is given by the sun microsystems.
→ API's are exposed to two categories of users
  * Category 1 : Developers or programmers who use the interfaces or helper classes of an API (usage logic)

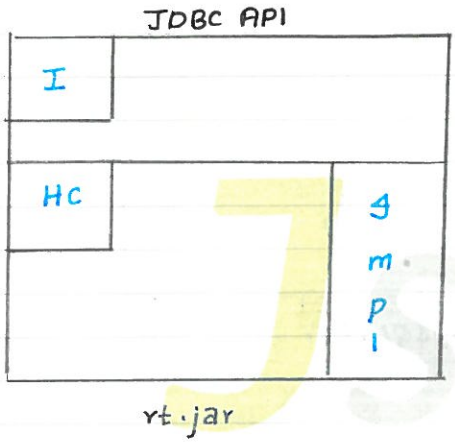
  * Category 2 : For implementation provider ie the one who gives an implementation (DB-vendors)

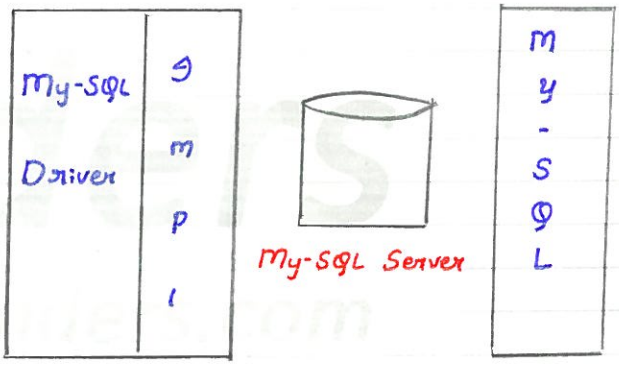
JDBC - Drivers :-

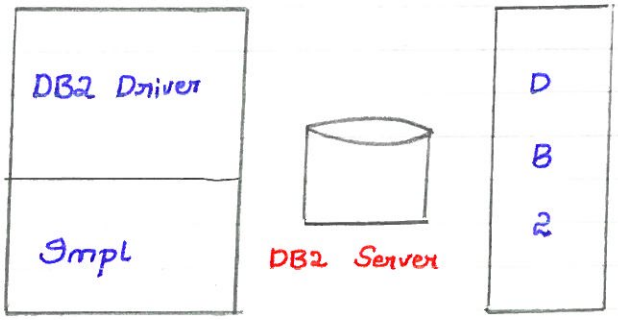
→ JDBC driver is an implementation of JDBC API.
→ JDBC API drivers which contains implementation is given by DB-providers

who are typically called as DB-Vendors.
→ The JDBC Driver is vendor specific ie it is DataBase specific.
→ The driver given by MySQL can be used to connect with MySQL DB server and cannot be used to connect with some other DB Server.
→ JDBC drivers are given in the form of jar files. Hence this driver jar file has to be added to project buildpath.







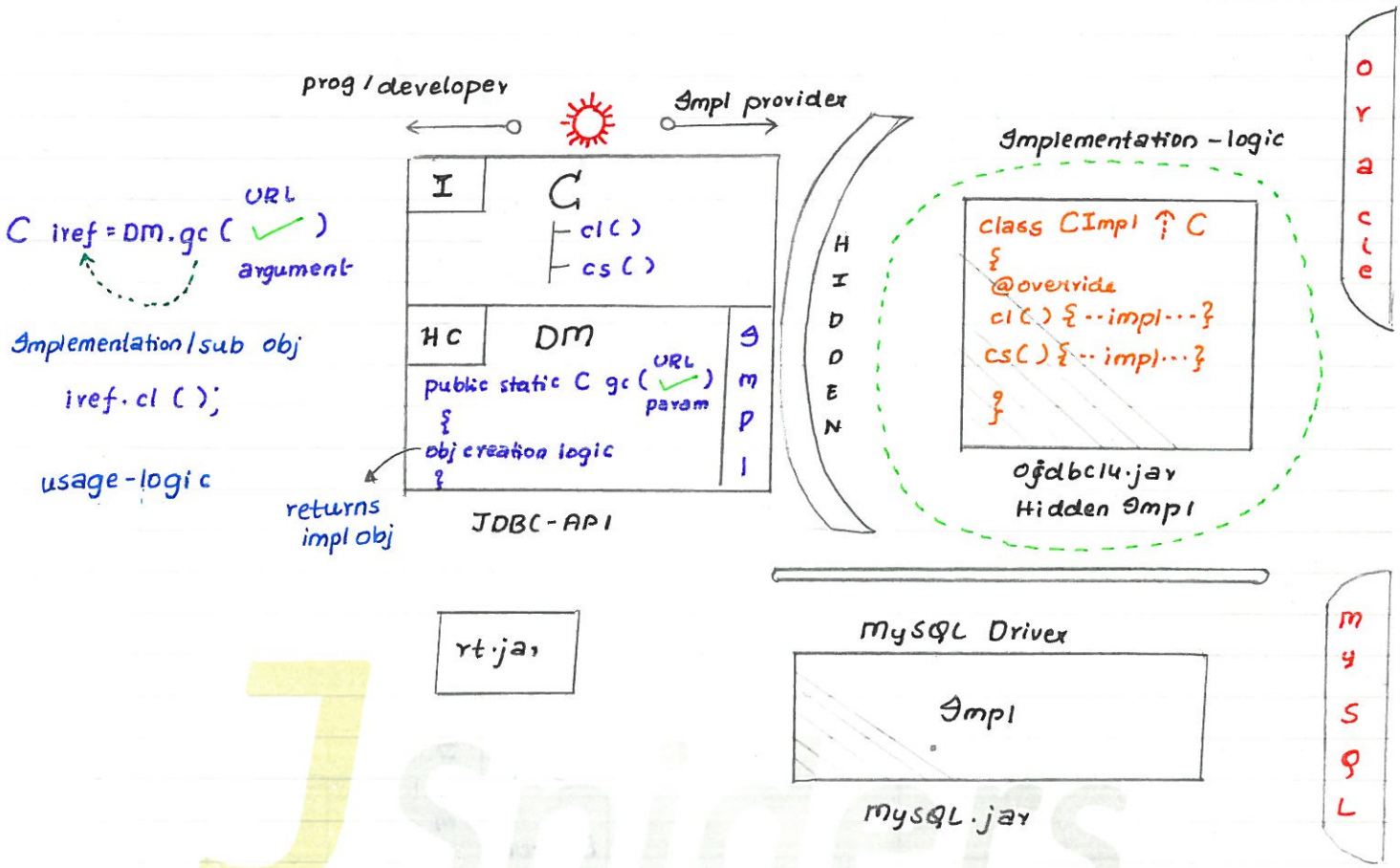


Driver Jar Files :-

* mysql-connector-version.jar
  Ex: mysql-connector-3.1.jar

* ojdbc version.jar
  Ex: ojdbc 1.4.jar

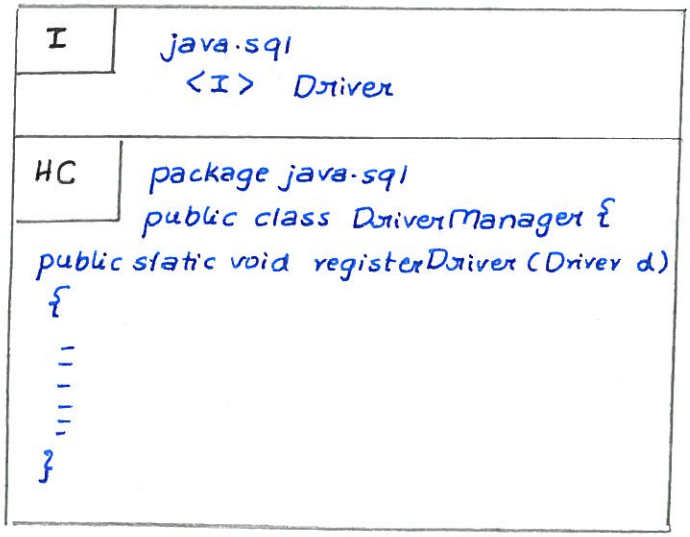
Author:-Shishira Bhat
Technical Architect



Driver Classes :-

* Oracle :
  oracle.jdbc.driver.OracleDriver
* MySQL :
  com.mysql.jdbc.Driver
* Microsoft (MS-SQL) :
  com.microsoft.sqlserver.jdbc.SQLServerDriver

JDBC - Specifications :-



```
package oracle.jdbc.driver
        ──────────────────────→ Specification1
                                                (I)
public class OracleDriver  ↑  java.sql.Driver
{
 static
 ──────→ Specification2
{
  OracleDriver od = new OracleDriver();
  DriverManager.registerDriver(od);

  ──────────→ Specification3
 }

}
```

O
R
A
C
L
E

Specifications :-

→ The driver class given by DB Vendors must implement an interface called java.sql.Driver, which is the part of JDBC API

→ All the driver class provided by DB Vendor must have static block.

→ Inside the static block driver object must be created (Same class object)

→ Created driver object must be registered with a static method called "registerDriver" which is present in java.sql.DriverManager which is the part of JDBC API.

→ Below is the syntax of the method.

`public static void registerDriver (java.sql.Driver dr)`

JDBC Definition :-

JDBC is the specification given in the form of abstraction API to communicate with DB Server in an independant way.

Advantages :-

1. loose coupling b/w DB Server and java application
2. Flexibility to connect to different DB ie DB independant.
3. Platform independant.

JDBC Driver Classes :-

class com.mysql.jdbc.Driver implements java.sql.Driver
class oracle.jdbc.driver.OracleDriver implements java.sql.Driver
class com.microsoft.sqlserver.jdbc.SqlServerDriver implements java.sql.Driver

class — implements — java.sql.Driver

- * Driver class
- * DB Vendors
- * part of JDBC drivers
- * Driver jar

- * Part of API
- * provided by sun
- * interface
- * JRE (rt.jar)

→ To communicate from a java program with any DB Server we need an object of Driver class.
→ This Driver classes are provided by database vendors like oracle, mysql, DB2 etc in the form of jar file
→ It is very important to create an object of the driver class

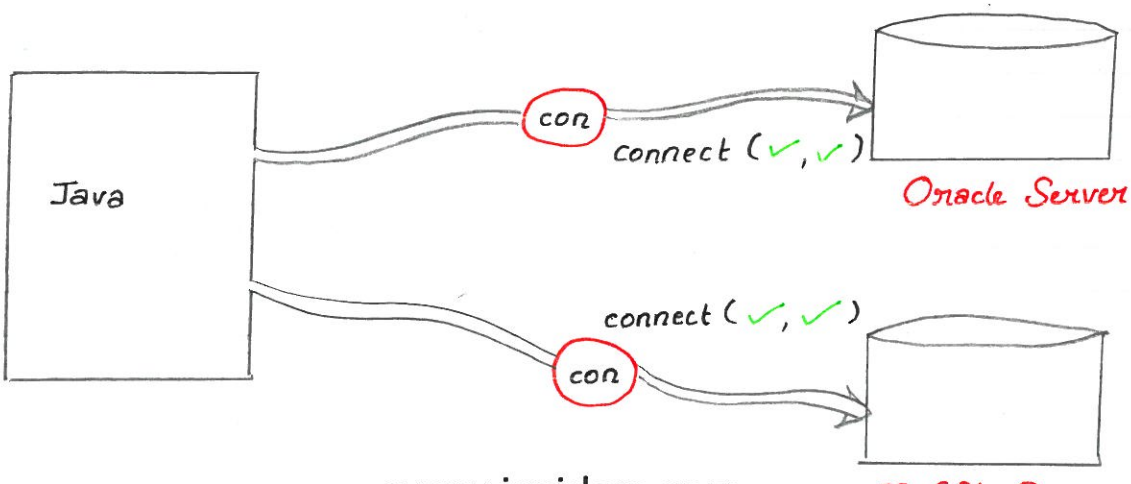
Creating an object of Driver Class :-
→ java.sql.Driver interface has an abstract method.
→ Syntax of method is
public Connection connect(String url, Properties props) throws SQLException

user = root
password = dinga
.properties

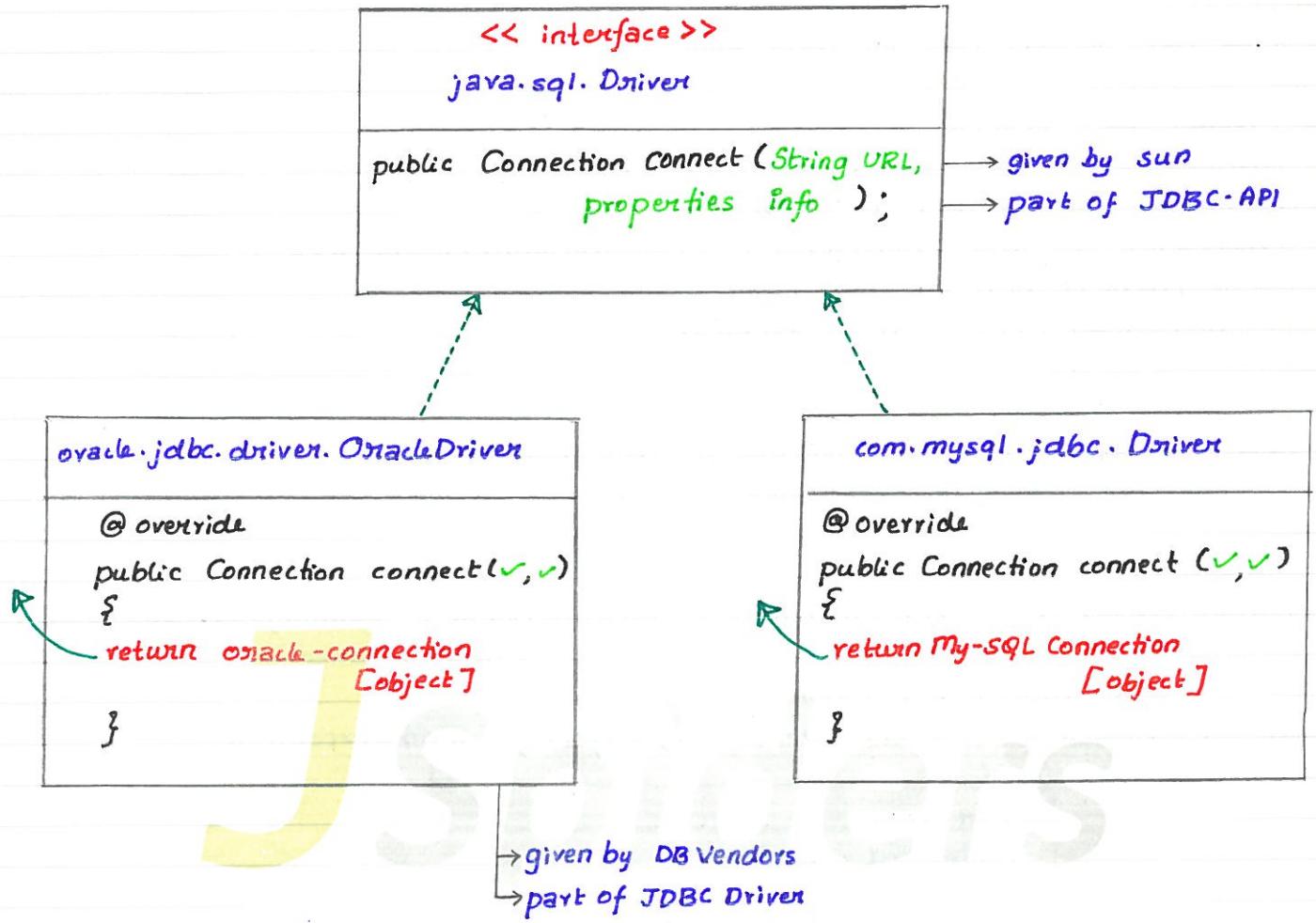
→ Connect() method is implemented in the Driver classes which are given by DB Vendors.
→ The connect() returns the connection object.

1. java.sql.Driver d = new Oracle.jdbc.driver.OracleDriver(); (I)
   Connection con = d.connect(✓, ✓);

2. java.sql.Driver d = (java.sql.Driver) Class.forName("oracle.jdbc.driver.OracleDriver")
   .newInstance();
   Connection con = d.connect(✓, ✓);



Author:-Shishira Bhat
Technical Architect



→ To establish the connection with DB Server we need connect method.
→ Connect method is given in Driver interface by Sun Microsystems and logic is given by DB Vendors in the Driver Class (Method is overridden)
→ To call the connect() method we need object of Driver class.

URL (Uniform Resource Locator):
→ URL is the path which is used to identify the resources over the network
→ The syntax of JDBC URL is

main protocol : sub protocol : Resource Path

mysql server
- jdbc : mysql : // localhost : 3306 → localhost
- jdbc : mysql : // 192.160.24.56 : 3306 → Remote host

Oracle Server
- jdbc : oracle : thin : // localhost : 1521 → localhost
- jdbc : oracle : thin : // 192.160.46.17 : 1521 → Remote host

→ The connect() method throws SQLException if the given URL is wrong
→ '@' can be used instead of '//' in the URL

Note :
On the URL we may pass the data also

Establishing the Connection with MySQL Server :

```
String URL = "jdbc:mysql://localhost:3306";
Properties props = new Properties();
FileReader fr = new FileReader("mysqldbconfig.properties");
   // FileNotFoundException
   props.load(fr);   // IOException
```

1st approach :

```
java.sql.Driver d = new com.mysql.jdbc.Driver();
java.sql.Connection con = d.connect(url, props);  // SQL Exception
```

2nd approach :

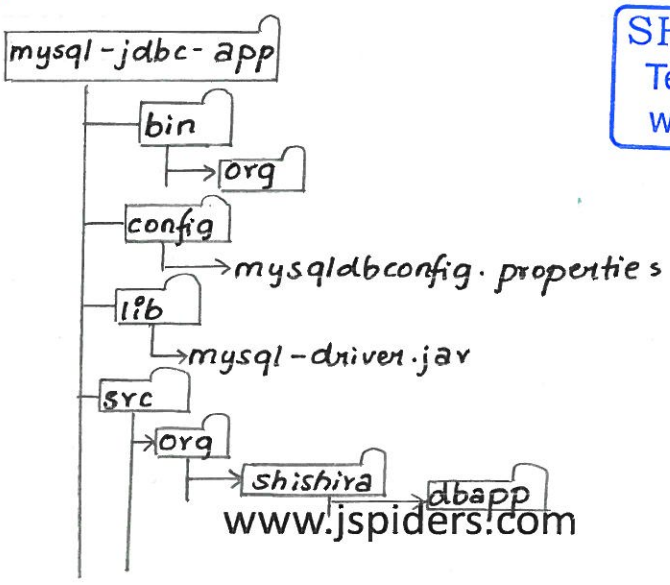
```
java.sql.Driver d = (java.sql.Driver) Class.forName("com.mysql.jdbc.Driver").
                                        newInstance();
//instantiationException
java.sql.Connection con = d.connect(url, props);  //SQLException
```

```
user = root
password = dinga
```

mysqldbconfig.properties

Setting up the project in Eclipse :-
→ Create a new java project in eclipse
→ Create appropriate package structure under src
→ Create a lib folder and copy driver jar file
→ Add the jar file into project build path
→ Create the properties file under config folder [optional step]

Directory Structure :



Author:-Shishira Bhat
Technical Architect

Program 7:

```
package org.shishira.dbapp;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IoException;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.SQLExeception;
import java.util.Properties;

public class MySqlConDemo
{
 public static void main(String[] args)
 {
  final String URL = "jdbc:mysql://localhost:3306";
  final String Driver = "com.mysql.jdbc.Driver";
  Properties props = new Properties();
  FileReader reader = null;
  try
  {
   reader = new FileReader("config/mysqldbconfig.properties");
   props.load(reader);
   Driver d = new com.mysql.jdbc.Driver();
   Connection con = d.connect(URL, props);
   S.o.p("Connection success"+con);
  }
 catch(FileNotFoundException e)
 {
  e.printStackTrace();
 }
 catch(IOException e)
{
 e.printStackTrace();
}
catch(SQLException e)
{
 e.printStackTrace();
}
finally
{
 if(reader!=null)
 {
  try{
   reader.close();
```

```
  }
  catch (IOException e)
   {
     e.printStackTrace();
    }
   }
  }
 }
}
```

Program 8:

```
package org.shishira.dbapp;
import java.io.*;
import java.sql.*;
import java.util.Properties;

public class MySqlConDemo1
{
 public static void main(String[] args)
 {
  final String URL = "jdbc:mysql://localhost:3306";
  final String DRIVER = "com.mysql.jdbc.Driver";
   Properties props = new Properties();
   FileReader fr = null;
    try
   {
    reader = new FileReader("config/mysqldbconfig.properties");
     props.load(reader);
     Driver d = (Driver) Class.forName(DRIVER).newInstance();
     Connection con = d.connect(URL, props);
     S.o.p("Connection success - " + con);
    }
catch(FileNotFoundException e)
 {
  e.printStackTrace();
 }
catch(IOException e)
 {
 e.printStackTrace();
 }
catch(SQLException e)
 {
 e.printStackTrace();
 }
```

```
    catch (InstantiationException  e)
    {
     e.printStackTrace();
     }
    catch (ClassNotFoundException e)
    {
     e.printStackTrace();
    }
    catch (IllegalAccessException e)
    {
     e.printStackTrace();
    }
    finally
    {
    if (reader!=null)
    {
      try
      {
        reader.close();
      }
    catch (IOException e)
     {
      e.printStackTrace();
     }
    }
   }
 }
}
```


Types of URL :

1. URL with host info :
   jdbc : mysql : // localhost : 3306

2. URL with host info ? Data :
   jdbc : mysql : // localhost : 3306 ? user = root & password = dinga.

JDBC Steps :-

i. load and register the driver
ii. Establish the connection
iii. Creat Statment
   3 types of statments
   - statment
   - prepared statement

• callable statement.

iv. Execute the query or SQL statement.
v. Optional step: Process the data
vi. Close all the JDBC Resources.

Step1 : Load and register the driver :

* The driver class is given by DB vendor can be loaded and registered with DriverManager by using below 2 ways

Option 1 :

Syntax : Class.forName ("Driver");

Ex : Class.forName ("oracle.jdbc.driver.OracleDriver");
Class.forName ("com.mysql.jdbc.Driver");
Class.forName ("com.microsoft.sqlserver.SqlServerDriver");

→ forName() is a static method which is present java.lang.Class
→ The forName() method loads the driver class which is the part of jdbc driver.
→ This possibly gives a checked exception called ClassNotFoundException.
→ When forName() method loads the given class the static block of Driver class is executed, which has the code to create its own object and register that object with DriverManager using the registerDriver() method.
→ registerDriver() is a static method which is present in java.sql.DriverManager class
→ syntax :

public static void registerDriver (java.sql.Driver dr);

Note :

This is highly prefered way of loading and registering the driver because developers dont have to create the dependancy on the Driver class.

Option 2 :

→ Creat an object of Driver class and register explicitly with DriverManager using registerDriver() method.

Ex :-

```
java.sql.Driver odriver = new oracle.jdbc.driver.OracleDriver();
DriverManager.registerDriver(odriver);   [for oracle]

java.sql.Driver mdriver = new com.mysql.jdbc.Driver();
DriverManager.registerDriver(mdriver);
```

Step2 : Establish the Connection with DB Server.

→ To establish the connection with DB Server we use the public static method of java.sql.DriverManager class called getConnection()
→ The getConnection method returns the connection object for an appropriate

database depending on the loaded driver.
→ getConnection() is an implementation of factory design pattern.
→ getConnection() possibly throws an checked exception called SQLException
→ Three overloaded versions of getConnection() method.
→ Syntax:
a. public static Connection getConnection(String url)
b. public static Connection getConnection(String url, properties info)
c. public static Connection getConnection(String url, String username, String password)

Demo for establishing connection with Mysql DB Server:
Program 9:

```
package com.shishira.conapp;
import java.sql.*;
public class ConDemo
{
  public static void main(String[] args)
  {
    final String URL = "jdbc:mysql://localhost:3306?user=root&password=dinga";
    try
    {
     Class.forName("com.mysql.jdbc.Driver");
     S.o.p("loaded and registered - success");
     Connection con = DriverManager.getConnection(URL);
     S.o.p("Connection success");
    }
  catch(ClassNotFoundException e)
  {
   e.printStackTrace();
  $.o.p("Failed to load & register the driver");
  }
 catch(SQLException e)
 {
  S.o.p("Connection failed");
 }
}
```


Sample program to insert a record to the database:

```
1. Class.forName("Driver");  //ClassNotFoundException
2. Connection con = DriverManager.getConnection(URL);  //SQLException
3. Statement stmt = con.createStatement();
4. int nori = stmt.executeUpdate("insert into $db.student values
                                  ('$s1', 'Manoj', 68.3)");
5. stmt.close();
   con.close();
```

Step3 : Creat Statement

Statements :-

→ Statement is an interface in java.sql. package
→ Statement is the part of JDBC API
→ Implementation of statement is given by DB Vendor as the part of JDBC Driver
→ To get the reference of statement, we use a method called creatStatment() which is defined in Connection interface.
→ The syntax of the method is

```
public Statement createStatement( )
```

→ Statement represent the platform or environment for execution of sql statements or queries like DML, DDL, etc.

Step 4 : Execution of query and SQL statements.

→ To execute SQL statements, we use the execute methods which is the defined in Statement interface.
→ We have 3 different versions of execute() method
→ Syntax

```
1. public boolean execute (String qry)
2. public ResultSet executeQuery (String qry)
3. public int executeUpdate (String dmlQry)
```


1. execute() :

→ execute() method is used to execute any type of SQL statement ie it is a generic method for SQL execution
→ execute() method is generally used for DDL statements.

2. executeQuery() :

→ executeQuery() method is used to execute SQL statement which generally returns the data.
→ ie if you want to query the DB then we can use executeQuery() method
→ executeQuery() method returns the ResultSet which represents the processed data [data returned from database]

3. executeUpdate();

→ executeUpdate() method is used to execute DB DML statements.
→ The returntype of executeUpdate() is an integer value which represents the no.of represent records affected in database.

Note :

We cannot use executeQuery() method for DML statement. If we use then at runtime we get SQLException.

→ All the execute() methods possibly throws a checked exception called SQLException.

Jspiders
Training & Development Center

Author:-Shishira Bhat
Technical Architect

Usage of two parameterized getConnection( ) method :

Syntax :

```
public static Connection getConnection (String url, Properties info )
```

Ex:-

```
final String URL = "jdbc: mysql: // localhost : 3306 ";
Properties props = new Properties ();
FileReader freader = new FileReader ("dbinfo. properties"); // FileNotFoundException
props.load ( freader); // IOException
1. Class.forName ("Driver"); // ClassNotFoundException
2. Connection con = DriverManager. getConnection (URL, props); //SQLException.
```

```
user = root
password = dinga
```

dbinfo. properties

Note :-

The getConnection( ) method internally has the method to read the properties (username & password) from properties file. Hence in the properties file keys must be user and password.

Usage of 3 parameterised getConnection( ) method :

Syntax :

```
public static Connection getConnection ( String url, String username,
                                         String password );
```

Ex:-

```
final String URL = "jdbc: mysql: // localhost : 3306 ";
final String USER = "root";
final String PASS = "dinga";
final String DRIVER = "com.mysql.jdbc. Driver";
1. Class.forName (Driver); // ClassNotFoundException
2. Connection con= DriverManager.getConnection ( URL, USER, PASS); //SQLException
```

Updating a record from DB :

```
String updateQry = "update sdb.student set perc = 71.8 where id = 'S4' ";
```

1. load & register the driver.
2. Establish the connection
3. Creat Statement
4. int noru = stmt. execute Updat (UpdateQry);
5. close stmt and connection

Deleting Record from Table :

```
String deleteQry = " delete from sdb.student where sid= 'S120' ";
```

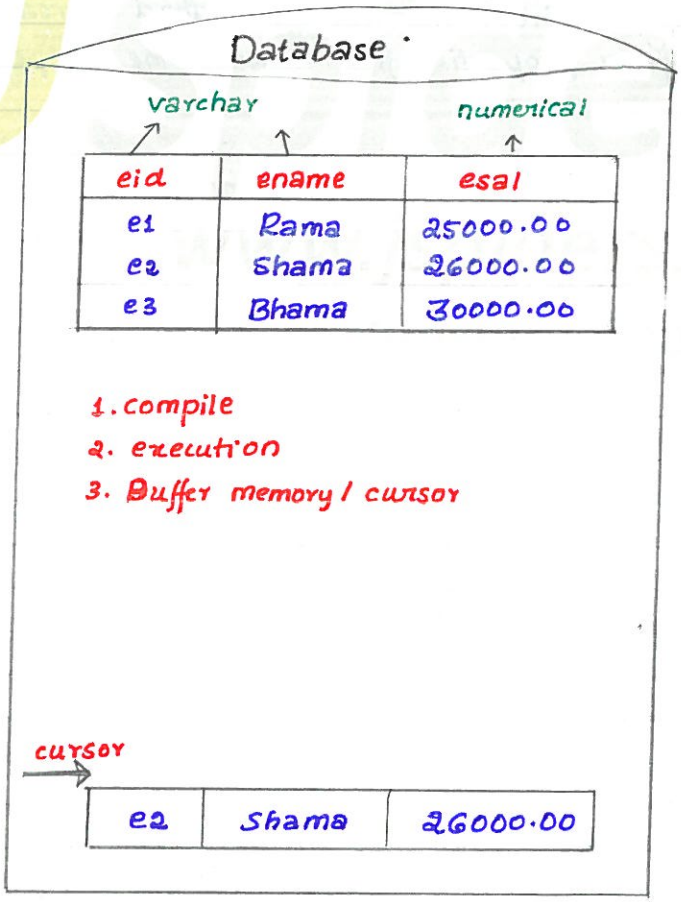
1. load & register the driver
2. establish the connection
3. create statement
4. int nord = stmt.executeUpdate(deleteQry);
5. close stmt and con.

Fetching the data from Database:

String selQry = "select * from empdb.employee where eid = 'e2' ";

1. load & register the driver
2. establish the connection
3. create statement
4. ResultSet rs = stmt.executeQuery(selQry);
   rs.next();
5. String name = rs.getString("ename");
   String id = rs.getString(1);
   double sal = rs.getDouble("esal");
6. close stmt and con

ResultSet:



→ ResultSet is an interface which is present in java.sql package.
→ Implementation for ResultSet is given by DBVendor as the part of JDBC Driver
→ ResultSet represents the processed data
→ Once query is fired from JDBC program and sent to the DB Server, the

DBServer does below things
1. Compile the SQL statement
2. Prepare execution plan
3. Execute query or sql statement
4. Stores the result or processed data in buffer memory (cursor)

→ To fetch the data from buffer memory & cursor and to iterate over the cursors we use the methods of ResultSet like absolute(), getFirst(), next() etc.
→ By default ResultSet does not point to any record. Hence we need to use the next() method.
→ The returntype of next() method is boolean ie next() moves to next actual record if it is present and returns true.
→ If the record is not present then it returns false.
→ To fetch the data from a particular column we use getXXX() method
Syntax :-
```
public XXX getXXX(String colname);
public XXX getXXX(int colnum);
```
→ The getXXX() method is overloaded.
→ The one method takes String columnname or columnlabel and the other method takes int columnNum or columnIndex
→ For all the datatypes we have getXXX() method
Ex:
```
getString();
getFloat();
getBoolean();
getInt();
```
→ Syntax of next() is,
```
public boolean next()
```
→ The absolute() method points to the exact record which is sent as an argument
→ The returntype of absolute() is boolean
Syntax: `public boolean absolute(int rownum);`
Ex:
```
rs.absolute(12);
rs.absolute(200);
```


Program 10 :
```
package org.jspiders.dbapp;
import java.sql.*;
public class DeleteProductDemo
{
 public static void main(String[] args)
 {
  String deleteQry = "delete from jjmll.product where category = 'household'";
  Connection con = null;
  Statement stmt = null;
```

```
final String URL = "jdbc:mysql://localhost:3306 ? user=root & password=dinga";
try
{
Class.forName ("com.mysql.jdbc.Driver");
con = DriverManager.getConnection (URL);
stmt = con.createStatement();
int nora = stmt.executeUpdate (deleteQry);
S.o.p(nora + " records are deleted");
}
catch (ClassNotFoundException e)
{
e.printStackTrace();
}
catch (SQLException e)
{
e.printStackTrace();
}
finally {
try
{
if (stmt != null)
{
stmt.close();
}
if (con != null)
{
con.close();
}
}
catch (SQLException sqle)
{
sqle.printStackTrace();
}
}
}
```


Note :

It is not a good practice to use "throws" along with main method.

Program II :

```
package org.jspiders.dbapp;
import java.sql.*;
import java.util.Scanner;
```

```
public class UpdateStudentDemo
{
public static void main(String[] args)
 {
  Scanner scan = new Scanner(System.in);
  S.o.p("Enter ID");
   String id = scan.next();
   S.o.p("Enter percentage");
   double perc = scan.nextDouble();
   scan.close();
  String updateQry = "update jjmll.student "+
                     " set perc = "+perc+" "+
                     "where sid = '"+id+"' ";
    Connection con=null;
    Statement stmt=null;
final String URL = "jdbc:mysql://localhost:3306?user=root & password=dinga";
  try
  {
   Class.forName("com.mysql.jdbc.Driver");
   con= DriverManager.getConnection(URL);
   stmt = con.createStatement();
   int noru = stmt.executeUpdate(updateQry);
   S.o.p(noru+" recordIs updated");
  }
catch(ClassNotFoundException | SQLException e)
 {
 e.printStackTrace();
}
finally
 {
 try
 {
 if(stmt!=null)
 {
  stmt.close();
 }
 if(con!=null)
 {
 con.close();
}
} catch(SQLException e)
 {
  e.printStackTrace();
 }
}
```

Commit :

→ When we establish the connection, by default the autoCommit() mode is enabled. ie when we execute queries the result is committed in the database

→ AutoCommit mode can be disabled by using

```
con.setAutoCommit(false);
```

→ When we disable autoCommit mode, then we have to explicitly commit the results by using

```
con.commit();
```

Program 12 :


```
package org.jspiders.dbapp;
import java.sql.*;
public class FetchAllStudent
{
  p.s.v.m(String[] args)
  {
    Connection con=null;
    Statement stmt=null;
    String selQry = "Select * from jjmll.student";
final String URL = "jdbc:mysql://localhost:3306/?user=root&password=dinga";
 try
{
 Class.forName("com.mysql.jdbc.Driver");
 con= DriverManager.getConnection(URL);
  stmt = con.createStatement();
 ResultSet rs = stmt.executeQuery(selQry);
 while(rs.next())
 {
  String name = rs.getString("sname");
  String id = rs.getString("sid");
  double perc = rs.getDouble("perc");
  String stream = rs.getString(4);
 S.o.p(name+"    "+id+"    "+perc+"    "+stream);
 }
catch(Exception e)
{
 e.printStackTrace();
}
finally
{
 //close all resources
}
}
}
```

Program 13:

```
package org.jspiders.dbapp;
import java.sql.*;
import java.util.Scanner;
public class LoginValidation
{
 p.s.v.m(String[] args)
 {
  Connection con=null;
  Statement stmt=null;
final String URL="jdbc:mysql://localhost:3306?user=root&password=dinga";
  Scanner scan=new Scanner(System.in);
  S.o.p("Enter your user name");
  String un=scan.next();
  S.o.p("Enter password");
  String pas=scan.next();
  scan.close();
 String qry="select name from jjmll.user "+
            "where usernm='"+unt"'"+
            "and pass='"+past"'";
 try
 {
  Class.forName("com.mysql.jdbc.Driver");
  con=DriverManager.getConnection(URL);
  stmt=con.createStatement();
  ResultSet rs=stmt.executeQuery(qry);
  if(rs.next())
  {
   /* valid user */
   S.o.p("welcome  "+rs.getString(1));
  }
  else
  {
  /* invalid user */
  S.o.p("Invalid user");
  }
  catch(Exception e)
  {
  e.printStackTrace();
 }
 finally
 {
  //close all resources
  }
 }
}
```

Program 14:

```
package org.shishira.dbapp;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Properties;
public class GetAllUsers
{
 public static void main(String[] args)
  {
   Connection con=null;
   Statement stmt=null;
   Properties props=new Properties();
final String URL="jdbc:mysql://localhost:3306";
   String qry="select * from jjmll.user";
   try
   {
    fr=new FileReader("config/dbconfig.properties");
    props.load(fr);
    Class.forName("com.mysql.jdbc.Driver");
    /* over-loaded */
   con=DriverManager.getConnection(URL,props);
   stmt=con.createStatement();
   ResultSet rs=stmt.executeQuery(qry);
   while(rs.next())
   {
    String name=rs.getString("name");
    S.o.p(name);
   }
  catch(Exception e)
   {
    e.printStackTrace();
   }
  finally {
try{
   if(fr!=null)
   {
    fr.close();
   }
  catch(IOException e)
  {
  e.printStackTrace();
  }
 }
```

```
try
{
 if (stmt != null)
  {
   stmt.close();
  }
 if (con != null)
  {
   con.close();
 }
}
catch (Exception e)
{
  e.printStackTrace();
}
}
}
```


Program 15 :

```
package org.shishira.dbapp;
import java.io.*;
import java.sql.*;
import java.util.properties;
public class ExecuteSelectDemo
{
  p.s.v.main(String[] args)
  {
   S.o.p(" App start ");
   String selQry = " Select * from jjmll.employee ";
    FileReader fr = null;
   Properties props = new Properties();
final String URL = "jdbc:mysql://localhost:3306";
   try
   {
   fr = new FileReader("config/dbconfig.properties");
    props.load(fr);
   Class.forName("com.mysql.jdbc.Driver");
   con = DriverManager.getConnection(URL, props);
   stmt = con.createStatment();
   boolean result = stmt.execute(selQry);
   if (result)
  {
   ResultSet rs = stmt.getResultSet();
   while (rs.next())
```

```
    String name = rs.getString("ename");
    double salary = rs.getDouble("sal");
    S.o.p(name + "      " + sal);
   }
  }
 }
  catch(FileNotFoundException e)
  {
   e.printStackTrace();
  }
  catch(IOException e)
  {
   e.printStackTrace();
  }
  catch(ClassNotFoundException e)
  {
   e.printStackTrace();
  }
 finally
 {
  // close all costly resources
 }
 }
}
```


Program 16 :

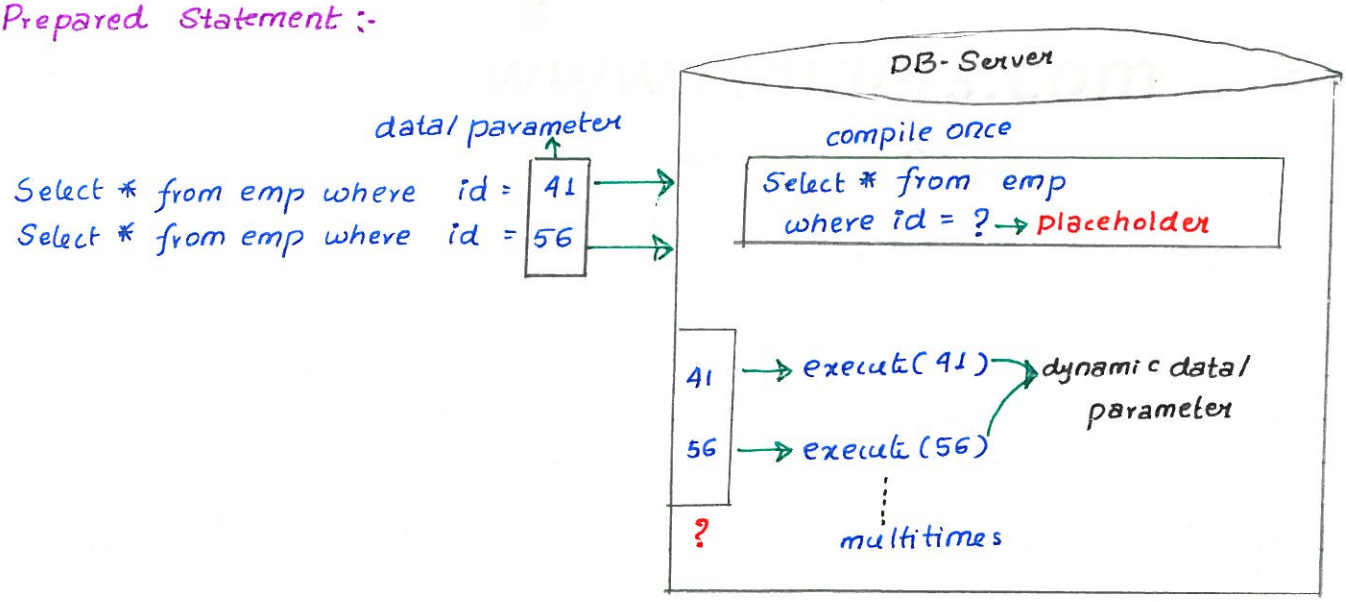
```
package org.shishira.dbapp;
import java.io.*;
import java.util.properties;
public class PropertyReadyUtility
{
 public static string getProperty(String key)
 {
  String value = null;
  FileReader fr = null;
  Properties props = new Properties();
  try
  {
   fr = new FileReader("config/dbconfig.properties");
   props.load(fr);
   value = props.getProperty(key, "No key found"); }
  catch(Exception e){
   e.printStackTrace();
  }
```

```
    catch ( IOException e)
    {
      e.printStackTrace();
    }
    finally
    {
    if (fr! =null)
    {
     try
     {
      fr.close();
     }
    catch (IOException e)
    {
     e.printStackTrace();
    }
    }
    }
    return value;
    }
}

public class ExecuteDMLDemo
{
 p.s.v.main(String[] args)
 {
  Connection con=null;
  Statement stmt = null;
  String driver = PropertyReaderUtility.getProperty ("driver");
 String url = PropertyReaderUtility.getProperty ("url");
 String pass = PropertyReaderUtility.getProperty ("password"); [get username also]
 String qry = "delete from jjmll.employee where dept = 'pool'";
 try
 {
  Class.forName (driver);
  con = DriverManager.getConnection (url, user, pass);
  stmt = con.createStatement();
  boolean res = stmt.execute (qry);
  if (!res)
  {
   int nord = stmt.getUpdateCount();
    S.o.p (nord+ " records deleted");
  }
  catch (Exception e) {
```

```
    e.printStackTrace();
   }
   finally
   {
   try
   {
    if (stmt!=null)
     {
      stmt.close();
     }
    if(con!=null)
     {
      con.close();
     }
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }
}
```

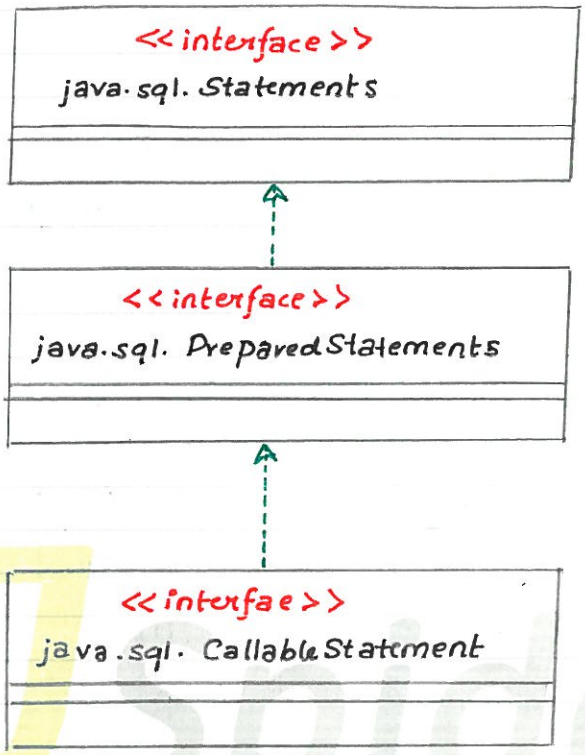

Prepared Statement :-



* PreparedStatements are designed to support execution plan
* The advantages of execution plan is the similar SQL statement or query is compiled only once & can be executed multiple times.
* The execution plan provides placeholders (?) to support dynamic data.
* Any no of placeholders can be present which starts with index of number 1
* Prepared statement is an interface which is present in java.sql. package
* PreparedStatement is a part of JDBC API for which implementation is given by

Author:-Shishira Bhat
Technical Architect

different DB-Vendors as the part of JDBC Driver.
* Prepared statement extends statement.



Steps for preparedStatement :-
Step1 : create an object for PreparedStatement
Step2 : Set the values for placeholders
Step3 : Execute.

→ Ex: PreparedStatement pstmt = con.prepareStatement("select * from emp where id=?");
pstmt.setInt(1, 25); // set dynamic data to placeholder.
ResultSet rs = pstmt.executeQuery(); // execute.

Points to be considered while writing PreparedStatement program :
→ The query of SQL statement along with placeholders must be passed while creating prepared statement object.
→ Dynamic data must be set to the placeholder before execution.
→ The no.of dynamic data & value must exactly match with the no.of placeholders otherwise we get SQLException.
→ While executing the query we dont pass the SQL statement for executeQuery() method.

Ex: String qry = "select count(*) from jjm11.employee";
//load & register the driver
// establish connection
// creat statement / preparedStatement

```
ResultSet rs = stmt.executeQuery(qry);
if(rs.next())
{
int ~~no of~~ numofRec = rs.getInt(1);
}
// close stmt & con
```

Prepared Statement

```
String q1 = "insert into jjm11.student
             values(?,?,?,?)";
PreparedStatement pstmt = con.prepareStatement(q1);
```

query

setData

1 2 4 3

?	?	?	?

```
pstmt.executeUpdate();
```

no query;

compile Execute

setData

4 1 3 2

?	?	?	?

dynamic data
[in parameter]

```
pstmt.executeUpdate();
```

Statement

```
String q1 = "insert into jjm11.student
             values(0,0,0,0)";
String q2 = "insert into jjm11.student
             values(0,0,0,0)";
String q3 = "insert into jjm11.student
             values(0,0,0,0)";
Statement stmt = con.createStatement();
```

no argument

```
stmt.executeUpdate(q1);
stmt.executeUpdate(q2);
stmt.executeUpdate(q3);
```

compile+Execute	compile+Execute	compile+Execute
q1	q2	q3

```
insert into jjm11.employee values (?,?,?,?)
                                   1 2 3 4

insert into jjm11.employee (eid, ename) values (?,?)
                            1    2.          1.  2.

Select * from jjm11.employee where dept = ?
                                      1.

updat jjm11.employee set sal = ? where eid = ?
                           1.              2.
```

Program 17:-


```
package org.shishira.dbapp;
import java.sql.*;
public class PstmtInsert-Demo
{
 p.s.v.main(String[] args)
 {
  Connection con=null;
  PreparedStatement pstmt=null;
  String driver = PropertyReaderUtility.getProperty("driver");
  String url = PropertyReaderUtility.getProperty("url");
  String user = PropertyReaderUtility.getProperty("user");
  String pass = PropertyReaderUtility.getProperty("password");

  String qry = "insert into jjm11.student values(?,?,?,?);
  try
  {
   Class.forName(driver);
   con=DriverManager.getConnection(url, user, pass);
   pstmt = con.prepareStatement(qry);

   /* set the data for placeholders */
   pstmt.setString(2, "Kishor");
   pstmt.setString(1, "S1");
   pstmt.setString(4, "CS");
   pstmt.setString(3
   pstmt.setDouble(3, 86.12);

  /* execute the query */
   int nori = pstmt.executeUpdate();
  S.o.p(nori + " records inserted");
  }
 catch(Exception e)
 {
  e.printStackTrace();
```

```
    }
    finally
    {
     //close all resources
    }
  }
}
```

Program 18 :

```
package org.shishira.dbapp;
public interface DBConstants
{
  String MYDRIVER = "com.mysql.jdbc.Driver";
  String MYURL = "jdbc:mysql://localhost:3306";
  String MYUSER = "root";
  String MYPASS = "dinga";
}
```



```
package org.shishira.dbapp;
import java.sql.*;
import java.util.Scanner;
import static final org.shishira.dbapp.DBConstants.*;

public class PstmtSelectDemo
{
 private static final String SQL = "Select * from jjmll.employee "+
                                   " where eid = ? ";
 public static void main(String[] args)
 {
  Connection con = null;
  Scanner scan = new Scanner(System.in);
  S.o.p("Enter employee ID");
  int id = scan.nextInt();
  try
  {
   Class.forName(MYDRIVER);
   con= DriverManager.getConnection(MYURL, MYUSER, MYPASS);
   PreparedStatement pstmt = con.prepareStatement(SQL);
    pstmt.setInt(1, id);
   ResultSet rs = pstmt.executeQuery();
   if(rs.next())
  {
   String name = rs.getString(2);
   double sal = rs.getDouble("sal");
```

```
String dept = rs.getString("dept");
S.o.p(name+"        "+sal+"        "+dept);
}
else
{
S.o.p("No data found");
}
catch(ClassNotFoundException e)
{
e.printStackTrace();
}
catch(SQLException e)
{
e.printStackTrace();
}
finally
{
// close all the resources
}
}
}
```


Batch Updates :

→ A batch update is a batch of updates grouped together, and sent to the database in one "batch", rather than sending the updates one by one.

→ Batch processing allows you to group related SQL statements into a batch and submit them with one call to the database.

→ When you send several SQL statements to the database at once, you reduce the amount of communication overhead, thereby improving performance.

→ Sending a batch of updates to the database in one go, is faster than sending them one by one.

→ There is less network traffic involved in sending one batch of updates (only 1 round trip)

→ You can batch both SQL inserts, updates and deletes. It does not make sence to the batch select statements.

→ There are 2 ways to execute batch updates.

- Using Statement
- Using PreparedStatment

→ The addBatch() method of Statement or PreparedStatement is used to add individual statements to the batch.

→ Syntax is

```
public void addBatch()
```

→ The executeBatch() is used to start the execution of all the statement grouped

together

→ Syntax is :

```
public int[] executeBatch( )
```

→ The executeBatch() returns an array of integers, and each element of the array represents the update count for the respective update statement.

→ Just as you can add statements to a batch for processing, you can remove them with the clearBatch() method.

→ This method removes all the statements you added, However you cannot selectively choose which statement to remove.

Program 19 :


```
package org.shishira.dbapp;
import static org.shishira.dbapp.DBConstants.*;
import java.sql.*;
public class BatchUpdateDemo
{
 p.s.v.main(String[] args)
 {
  S.o.p("App started");
  Connection con=null;
  PreparedStatement pstmt=null;
  String insertQry="insert into jjmll.student values(?,?,?)";
  try
  {
  Class.forName(DRIVER);
  con= DriverManager.getConnection(URL, USER, PWD);
   pstmt=con.prepareStatement(insertQry);
  /* Set 1st row */
   pstmt.setString(1, "S56");
   pstmt.setString(2, "Pavan");
   pstmt.setDouble(3, 51.9);
    pstmt.addBatch(); //add 1st record to batch

  /* Set 2nd row */
  pstmt.setString(1, "S107");
  pstmt.setString(2, "Anish");
  pstmt.setDouble(3, 66.86);
  pstmt.addBatch(); // add 2nd record to batch

 /* Set 3rd row */
 pstmt.setString(1, "S103");
 pstmt.setString(2, "Niki");
 pstmt.setDouble(3, 90.86);
```

```
    pstmt.addBatch(); //add 3rd row to batch

    pstmt.executeBatch();
   }
   catch(Exception e)
   {
    e.printStackTrace();
   }
   finally
   {
    //close all resources
   }
  }
}
```


→ By using prepared statement we can use single query but multiple records can be updated or inserted or deleted at a time.
→ Combination of those are not possible.
Ex:- 1 insert, 1 delete, 2 update is not possible
→ PreparedStatement with BatchUpdates gives high performance.

Singleton Design Pattern:-

Program 20:

```
package org.shishira.dbapp;
 //singleton
public class Vishweshwariah
 {
  /* 1. */
 private Vishweshwariah()
 {

 }
/* 2. */
private static final Vishweshwariah() ONLY_ONE = new Vishweshwariah();

/* 3. */
public static Vishweshwariah getInstance()
{
 return ONLY_ONE;
}
}
```

Author:-Shishira Bhat
Technical Architect

Applying Singleton Design Pattern for JDBC Connection :-

Program 21 :

```
package org.shishira.dbapp;
import java.sql.*;
public class DBSingleton
{
private static final DBSingleton ONLY-ONE = new DBSingleton();
private Connection con; //non-static
private DBSingleton()
{
 try
 {
  Class.forName("com.mysql.jdbc.Driver");
  con = DriverManager.getConnection("jdbc:mysql://localhost:3306?
                                     user=root&password=dinga");
 }
 catch(Exception e)
 {
  e.printStackTrace();
 }
}
public static DBSingleton getInstance()
{
 return ONLY-ONE;
}
//non-static
public Connection getCon()
{
 return con;
}
}
```


Code to access the Connection :

```
public static void main(String[] args)
{
DBSingleton dbs = DBSingleton.getInstance();
Connection con = dbs.getCon();
 /* OR */

Connection con1 = DBSingleton.getInstance().getCon();
}
```

Author:-Shishira Bhat
Technical Architect

JDBC Transactions :-

```
Ex:- con.setAutoCommit(false);
     Begin ---------->
     op1 ✓
     op2 ✓
     op3 x

     End ------------->
     con.commit();

   catch(SQLException e)
   {
    con.rollback();
   }
```


→ If all the operations are successfull, it will be committed.
→ JDBC transaction management and Savepoint.
→ A transaction is a set of actions to be carried out as a single, atomic action.
→ Either all of the actions are carried out, or one of them are.
→ The classic example of when transactions are necessory is the example of bank accounts.
→ You need to transfer Rs 1000 from one account to other account. You do so by subtracting Rs 1000 from 1st account, and adding Rs 1000 to other account.
→ If the process fails after you have subtracted the Rs 1000 from the 1st bank account, the Rs 1000 are never added to the second bank account. The money is lost in cyber space
→ To solve this problem the subtraction and addition of the Rs 1000 are grouped into a transation.
→ If the transaction succeeds, but the addition fails, you can "rollback" the first subtraction.
→ That way the database is left in the same state as before the subtraction was executed.
→ JDBC Transaction let you control how & when a transaction should commit into database

```
// Transaction block start

// SQL insert statement
// SQL updat statement
// sql delete statement

// Transaction block end.
```

→ In simple, JDBC transaction make sure SQL statements within a transaction

block are all executed successful, if either one of the SQL statement within transaction block is failed, abort and rollback everything within the transaction block

→ By default when we create a DB Connection, it runs in auto Commit mode.
→ It means that whenever we execute a query and it's completed, the transaction commit is fired automatically.
→ So every SQL query we fire is a transaction and if we are running some DML or DDL queries, the changes are getting saved into Database after every SQL statement finishes
→ Sometimes we want to a group of SQL queries ~~runs~~ to be part of transaction so that we can commit them when all queries runs fine and if we get any exception, we have a choice of rollback all the queries executed as part of the transaction.

JDBC Transaction Management :-

→ To enable manual-transaction support instead of auto-commit mode that the JDBC driver uses by default, use the connection object's setAutoCommit() method.
→ If you pass a boolean false to setAutoCommit(), you turn off auto-commit.
→ You can pass a boolean true to turn it back on again
→ Syntax is

```
public void setAutoCommit(boolean value);
Ex: con.setAutoCommit(false);
```

Commit() & RollBack();

→ Once you are done with your changes and you want to commit the changes then call commit() method on connection object as follows:

```
con.commit();
```

→ Otherwise to rollback updates to the database made, use the following code:

```
con.rollback();
```

Program 23 :

```
package org.jspiders.trnxapp;
import java.sql.*;
import java.util.Scanner;
public class JDBCTransactionDemo
{
 p.s.v.main(String[] args)
 {
  Scanner scan = new Scanner(System.in);
  S.o.p("Enter name");
  String name = scan.next();
  S.o.p("Enter Id");
  Int id = scan.nextInt();
```

SHISHIRA BHAT
Technical Architect
www.jspiders.com

Author:-Shishira Bhat
Technical Architect

```
S.o.p ("Enter percentage");
double perc = scan.nextDouble();
S.o.p("Enter stream");
String stream = scan.next();
S.o.p("Enter city");
String city = scan.next();
S.o.p("Enter state");
String state = scan.next();
S.o.p("Enter country");
String country = scan.next();
scan.close();
Connection con = null;
PreparedStatement pstmt1 = null;
PreparedStatement pstmt2 = null;
String insertQry1 = "insert into jjmll.student.education_details" +
                    "values (?, ?, ?, ?)";
String insertQry2 = "insert into jjmll.student.student_address" +
                    "values (?, ?, ?, ?)";
final String URL = "jdbc:mysql://localhost:3306?user=root&password=dinga";
try
{
Class.forName("com.mysql.jdbc.Driver");
con = DriverManager.getConnection(URL);
/* Transaction begins */
con.setAutoCommit(false);

//operation1
pstmt1 = con.prepareStatement(insertQry1);
pstmt1.setString(2, name);
pstmt1.setInt(1, id);
pstmt1.setDouble(3, perc);
pstmt1.setString(4, stream);
pstmt1.executeUpdate();

//operation2
pstmt2 = con.prepareStatement(insertQry2);
pstmt2.setInt(1, id);
pstmt2.setString(2, name);
pstmt2.setString(3, city);
pstmt2.setString(4, state);
pstmt2.setString(5, country);
pstmt2.executeUpdate();

/* transaction ends. */
```

```
con.commit();
}
catch(Exception e )
{
 e.printStackTrace();
}
catch(SQLException e)
 {
  try
  {
   con.rollback ();
    S.o.p (" Transaction failed ");
  }
  catch (SQLException e1)
  {
   e1.printStackTrace();
  }
finally
 {
  //close all JDBC resources
 }
}
}
```


SavePoint :-

```
SavePoint sp=null;
Transaction begin ----->
OP1 ✓
OP2 ✓
OP3 ✓
      ------> sp= new SavePoint ("one");
OP4 X
OP5
con.commit();
Transaction ends.------>
catch (SQLException e)
 {
  if (sp! = null)
 {
  con.rollback (sp);
   con.commit();
  }
 else
 {
  con.rollback();
 } }
```

Metadata :-

- Data about a data is called as metadata.
- A database contains many tables & table contains lots of data.
- Apart from data, the DB and table has its own data like no. of columns, datatype of column, length or size, is the column primary key? etc.
- These data are called as metadata.
- We can get the metadata info by using below 2 interfaces
  - DataBaseMetaData
  - ResultSetMetaData
- Both are present in java.sql. package
- DatabaseMetaData deals with the metadata info of DB like DBName etc.
- The ResultSetMetadata deals with the metadata info of the table like the no. of columns, column length etc.

Program 24:-

```
package org.shishira.dbapp;
import java.sql.*;
public class MetaDataDemo
{
private static final String URL = "jdbc:mysql://localhost:3306?user=root&
                                   password=dinga";
private static final String DRIVER = "com.mysql.jdbc.Driver";
public static void main(String[] args)
{
 Connection con = null;
 try
 {
  Class.forName(DRIVER);
  con = DriverManager.getConnection(URL);
  DatabaseMetaData dbmeta = con.getMetaData();
  String dbname = dbmeta.getDatabaseProductName();
  String driverName = dbmeta.getDriverName();
  S.o.p(dbName + "      " + driverName);
 }
 catch(Exception e)
 {
  e.printStackTrace();
 }
finally
{
 //close all resources
 }
}
}
```

Program 25 :-

```
package org.shishira.dbapp;
import java.sql.*;
public class RSMetaDataDemo
{
private static final String URL = "jdbc:mysql://localhost:3306/jjm11?
                                   user=root & password=dinga";
private static final String DRIVER = "com.mysql.jdbc.Driver";
p.s.v.main(String[] args)
{
  Connection con=null;
  Statement stmt = null;
  String qry = "Select * from jjm11.employee";
  try
  {
  Class.forName(DRIVER);
  con = DriverManager.getConnection(URL);
   stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery(qry);
  ResultSetMetaData rsmeta = rs.getMetaData();
  int noColumns = rsmeta.getColumnCount();
 String colName = rsmeta.getColumnName(3);
 S.o.p(noColumns + "          "+ colName);
 }
catch(Exception e)
{
 e.printStackTrace();
}
finally
 {
 // close all resources
 }
}
}
```