2. Date:-20-9-18

J2EE:- *It stands for java 2 Enterprise Edition where 2 refers to the version
*It contains larger set of library compare to j2SE using which some additional functionalities can be performed.

>>Need for J2EE:-*It is needed for the simplification of web application development.

>>JAR file:-JAR means Java Archive (compressed).
It is a file format based on ZIP file format which is used to compress 'many files into 1 single file.'

>>contains of jar file:-
.java (optional) --> (contains source code statement in it)
.class (contains byte code statements in it) -->Machine level language.
config Files--->{contains configuration data of file} e.g. any of s/w
[(a) .xml--> it is used to configure the resource with the help of user Define or custom tags.
b) .properties---> it is used to provide a set of properties in the form Of key (must be unique) and value pair]

>> Need for jar file:-*it is needed to 'import the properties based on the requirement'.
>> Total inbuilt class in java is 3000+ present in rt (run time).jar

>> Standard package or folder structure:-
[org/com] org:-Organisation
[Company name] :- Google
[Application name]:-mail
|----> Module 1 :-Inbox
---> Module 2 :-compose
---->Module n

----> In case of package structure, we use dot (.) as separator.
---->In case of folder structure, we use slash (/) as a separator

>> Steps to create a JAR file:-
1. **Right click on project and select Export option.
2. Open java folder and select jar file and click on next.
3. Browse for appropriate location to place the jar file and name the jar file and click on save and finish.

>>Steps to build a Java path:-
1. There are 2 different ways to build a java path to import the properties from a jar file namely
a) Externally b) Internally
..a> Externally:-
1. Right click on project and select properties option.
2. Select java build path and click on libraries tab.

3. Click on add externals jar and select a jar file from the respective path and click on open and OK.

---

*3. Date:-21-9-18

>> b) Internally:-

1. Right click on project and create a new folder with name lib (user choice).
2. Add the respective jar file into the lib folder (copy from source and paste).
3. Right click on project and select properties option.
4. Select java build path and click on libraries tab.
5. click on add jars and select jar file from the lib folder of respective project and click on open and ok.

Advantage: - if we will delete the existing jar file after setting the path internally the execution will work, but not in externally.

*Note: - It is always a good practise to import the properties from a jar file 'internally'.

>>Parameter: - *It is the one which holds an argument.

>> Argument: - *Data or value passed to any methods is an argument.

>>Note: - *1.Whenever the method parameter type is an interface, then the argument for that method must be any of its implementation class object.

*2. Whenever the method parameter type is super class, then the argument for that method must be any of its sub class object

*3. Whenever the method parameter type is an abstract class, then the argument for that method must be any of its sub class or implementation class object.

>>**Design Pattern**>>:-An optimised solution for commonly re-Occurring design problems or design needs is known as a design pattern.

e.g. ---->MVC (Model View Controller) architecture

>>There are 2 different categories of Design pattern namely i). Creational Design pattern
ii) Factory Design pattern

>>i) Creational Design pattern:-It involves only object creation logic.

>>ii)** Factory Design pattern:-Factory :- *It is the one which creates or produces 'different objects of same type'.

*Factory generally takes an i/p of 'String type'.

fig:- String as i/p object as o/p

("HMT")---------->[WATCH Factory]--------->HMT

{Fastrack}--->|<-----{HMT}

```
+ class WatchFactory  // factory class
{
+ static Watch getWatch(String type)  // factory /helper method
```

```
{
if(type.equalsIgnoreCase("Fastrack"))
      return new Fastrack();

else if(type.equalsIgnoreCase("HMT"))
      return new HMT();

else
      sop("Watch not found")
return null;
}
}
```

Date:-24-9-18

>>Abstraction:-*Hiding the implementation and providing functionalities to the user with the help of interface is known as Abstraction.
*the o/p of abstraction is loose coupling which can be achieved with the help of Interface.

>> Interface: -*It is media to communicate b/w user any device.

>>Loose Coupling:-Change in an implementation which does not affect the user is known as loose coupling.
>>Tight Coupling: - Change in the implementation which effect the user is known as tight coupling.

Assignment Question:-Q. Define jar file and explain the contents and Need of jar file.
Q. Define parameter and Argument.
Q. Define Design pattern along with example.
Q. Define Creational and Factory Design pattern with code.

Factory Design pattern is always Associated with 3 different types of logic Namely:-
i). Implementation Logic. ii) Object creation Logic iii).Consumer or Utilisation Logic

---

Date:-26-9-18

>>i). Implementation Logic:-*It is the basic fundamental logic which contains only Implementations according to which an implementation object has to be created.

>>ii) Object creation Logic:-*It is use to create implementation object according to the implementation logic by using a 'factory or helper' method within the factory class.

>> iii).Consumer or Utilisation Logic:-*it is the most important logic which is used to access the functionalities from the implementations.

>>Note:-without writing a consumer or utilisation logic, none of the implementations will work.

>>Costly resources (Scanner, stream, fileReader etc.):-Resources which makes use of system properties are known as costly resources.

**>>Factory or Helper method (job):-It is used to create implementation object.

>>NullPointerException:-Pointing towards an object which is not present throws an Exception called NullPointerException.

```
package org.btm.lightApp;
public interface ISwitch {
	void sOn();
	void sOff();
}

package org.btm.lightApp;
public class TubeLightImpl implements ISwitch {
	@Override
	public void sOn() {
		System.out.println("TubeLight is On!!");
	}
	@Override
	public void sOff() {
		System.out.println("TubeLight is Off!!");
	}
}

package org.btm.lightApp;
public class LedLightImpl implements ISwitch {
	@Override
	public void sOn() {
		System.out.println("LedLight is On!!");
	}
	@Override
	public void sOff() {
		System.out.println("ledLight is Off!!");
	}
}
package org.btm.lightApp;
public class LightFactory {
	public static ISwitch getLight(String type) {
		if (type.equalsIgnoreCase("ledLight"))
			return new LedLightImpl();
		else if (type.equalsIgnoreCase("TubeLight"))
			return new TubeLightImpl();
```

```
                else
                        System.err.println("No such light found");
                return null;
        }

}
package org.btm.lightApp;
import java.util.Scanner;
public class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("enter ur choice of light");
                String type = sc.next();
                sc.close();
                ISwitch is = LightFactory.getLight(type);
                if (is != null) {
                        is.sOff();
                        is.sOn();
                }
        }
}
```

Date:-27-9-18

>>Note:-1.Whenever the user provides implementation then the same user has to even create an object of implementation.

2. Whenever the vender provides implementation then the same vender has to even create an object of implementation.

>>Application programming Interface (API):-

```
                                              ------> [A2]
    [AI]-----------------> [API] ----------------->------> [A3]          Inter Application Communication
Java Application    Abstraction               ------> [A4]
```

>>API: -*It used for inter application communication i.e. one application can communicate with other application with the help of API to achieve loose coupling..

*Backbone of API is Abstraction.

*The output of abstraction is loose coupling which is achieved with the help of interface.

>>Loose Coupling: - Change in the implementation which does not affect the user is known as loose coupling.

>> Example of API's are: - Appache POI, Taxcel, JDBC, Server...etc.

API (.class, .properties, .xml)

O ----------->Interface

/|\ ----------> [A1] ----------→----------->Helper class ---------> [A2]

/\ java application ----------->Implementation class

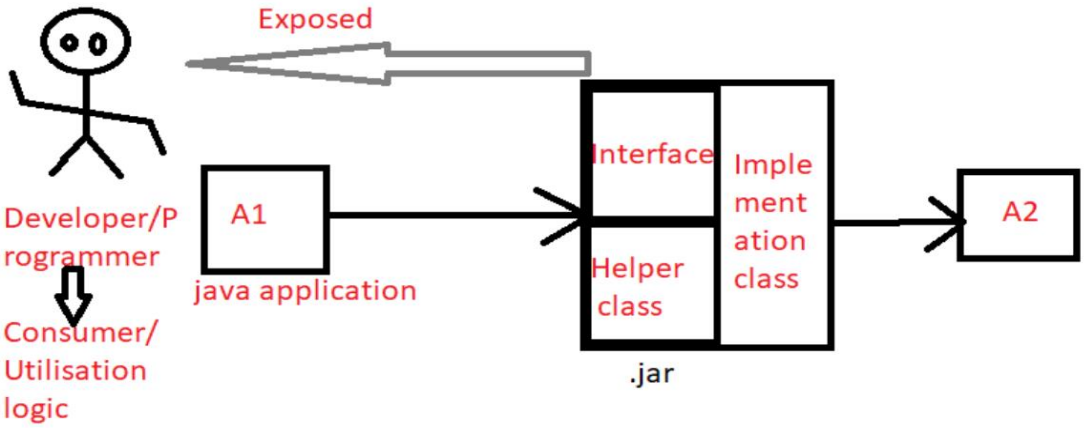
>> Software Application:-A set of codes which is used to perform some specific task is known as s/w application.

>>Note:-Whenever the API's are given in the form of jar files, then such API's developed using java as programming language.

>> There are two different form of API present namely:-

:-i) 1st form of API ii) 2nd form of API

>>i) 1st form of API: -*



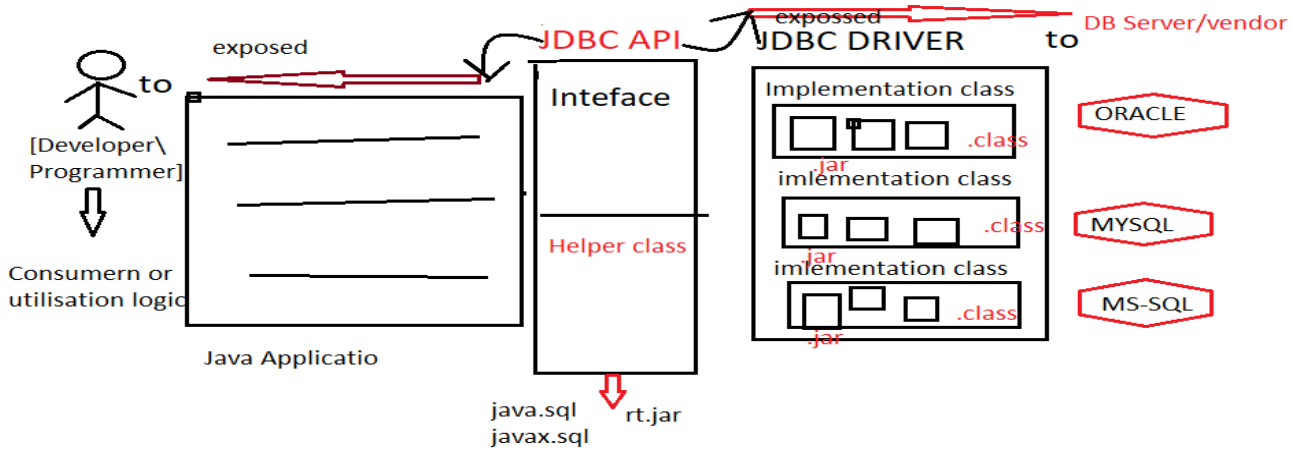
*This form of API contains interfaces, helper classes and implementation classes in the form of jar file.

*This form of API is exposed to the developer or the programmer to write the consumer or utilisation logic.

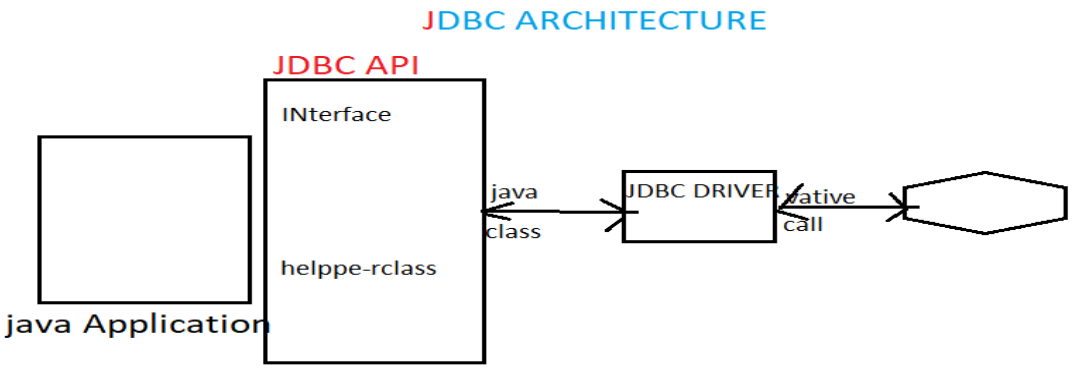
>>NOTE:-A consumer or utilization logic can always be written only after the implementation are known.

e.g.- Appache[0], taxcel.

Date:-28-9-18



>>JDBC Architecture:-



>>JDBC API:-* It was given by 'Sun micro System' to achieve loose coupling b/w java application and data base server.

*It Contains interfaces and class in the form of jar file.

*It is distributed into 2 different packages

namely: - i) java.sql(mostly uses) ii) javax.sql

*The interfaces of jdbc Api are: - i) Driver ii) Connection iii) Statement iv) PreparedStatement v)CallableStatement vi) ResultSet vii) MetaDAta etc.

*** It contains only one helper class in it by name DriverManager.

>>JDBC DRIVER:-** It is an implementation of jdbc API.

*It contains implementation classes in the form of jar file.

*It is always specific to the particular data base server or vendor.

*These are provided by respective database server or vendors.

>>NOTE:-** For us to communicate with any data base server and perform any data base operation, we need to have 2 different components mandatorily namely JDBC API and JDBC DRIVER

JDBC---> [[JDBC API] [JDBC Driver]]

>> Advantages of JDBC:-*We can achieve loose coupling b/w java application and data base Server.

*platform independent

>>NOTE:-In java JDBC program we always write only the consumer or utilisation logic.

Date:-1-10-18

>>Port No :-*It is the one which helps us to get connected to a particular server.

>>Different port no:-Oracle--1521

**MYSQL--3306

MS-SQL--1433

Derby--1527

>>Thin-client application:-*It is a light-weight s/w which is used to communicate with a particular data base server or data base vendor.

*there are 2 different categories of thin client application: -

i) Mobile client application ii) Database client application.

>> i) Mobile client application: -Banking s/w, fb, Messenger.

>> ii) Database client application:-Tood, Squiral ,SQLworkbench. SQLdeveloper ,SQLyog, SQLultima etc.

>>HOST:-*It is a platform on which all the applications can be executed.

>>there are 2 different types of host present namely i) Local ii) Remote

>> i) Local host: -In this case, the applications are restricted or limited only to that particular System.

e.g. - standalone application. ----> (like Share-it)

ii) Remote host: - In case of remote host, the application are not limited or restricted to a particular system.

e.g.-Any real time application----> web application. (you-tube)

>>Date:-2-10-18

>>URL (Uniform Resource Locator):-it is the path using which we can access the resources uniquely over the.

>> The contents of url are:-

Protocol, Host +Port/Domain name, Resource Name, Data (optional).

Compression tool

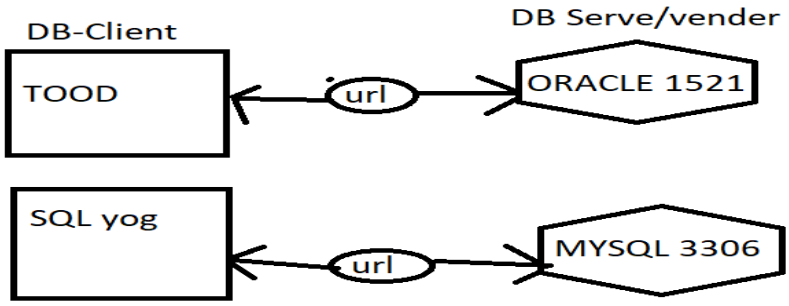
Application name

>>the protocol for web application are: -http/https

>>the protocol for JDBC is: - jdbc:sub-protocol.

>>In case of url, data refers to Key and Value pair which is provided by the user which is Optional

>>E.g for JDBC url:-



>Standard way of writing any url: -mainprotocol:subprotocol://Host+Port/dbname(optional)

1. for MY SQL:-[user name must be root password anything]

(? is) separator

(Host -information) <--|--> (user credentials/user data)

Local Host:-jdbc:mysql://localhost:3306/Studentdb?user=root&password=admin

jdbc:mysql://localhost:3306?user=root&password=admin

Remote Host: - jdbc:mysql://192.168.10.16:3306/Studentdb?user=root&password=admin

jdbc:mysql://192.168.10.16:3306?user=root&password=admin

2. for ORACLE:-[user name must be Scott password anything (like tiger)]

Local host: - jdbc:oracle://localhost:1521/Studentdb?user=scott&password=tiger

jdbc:oracle://localhost:1521user=scott&password=tiger

Remote Host :-jdbc:oracle://192.168.10.16:1521/Studentdb?user=scott&password=tiger

jdbc:oracle://192.168.10.16:1521?user=scott&password=tiger

>>NOTE: - no space b/w url and everything is in small letter (Generally)

>>DATE:-3-10-18

>>CLASS LOADING: -*Loading a dot class file into the jvm memory is known as class loading.

*A class is generally loaded in 2 different ways namely:-

i) By calling any of the members of a class which can either be a constructor, methods, variables, blocks etc.

ii) By using a static method called forName() method ,which is present in java.lang package.

Whenever we use this method, it throws an Exception calledClassNotFoundException (checked Exception) .Surround with try and catch.

>>Syntax:- Class.forName("FQCN");

>>e.g. of 1st way:-

```
public class Test {
```

```
	public static void main(String[] args) {
		Snake s =new Snake();
		s.eat();
	}

}
------------------
package org.btm.loadApp;
public class Snake {
	void eat()
	{
		System.out.println("eating frog");
	}
}

>> e.g. of 2nd type:-
package org.btm.loadApp;
public class Snake {
	static
	{
		System.out.println("eating frog");
	}

}
-----------------------------------------------
package org.btm.loadApp;
public class Test {
	public static void main(String[] args) {
		try{
			Class.forName("org.btm.loadApp.Snake");
		}
		catch(ClassNotFoundException e)
		{
			e.printStackTrace();
		}
	}

}
-----------------------------------------
```

**>>Definition of JDBC: - (Java Data Base Connectivity) is a specification given in the form of abstraction API to achieve loose coupling b/w java application and database server.

**>>Steps of JDBC:-There are 6 diff steps present in jdbc namely: -

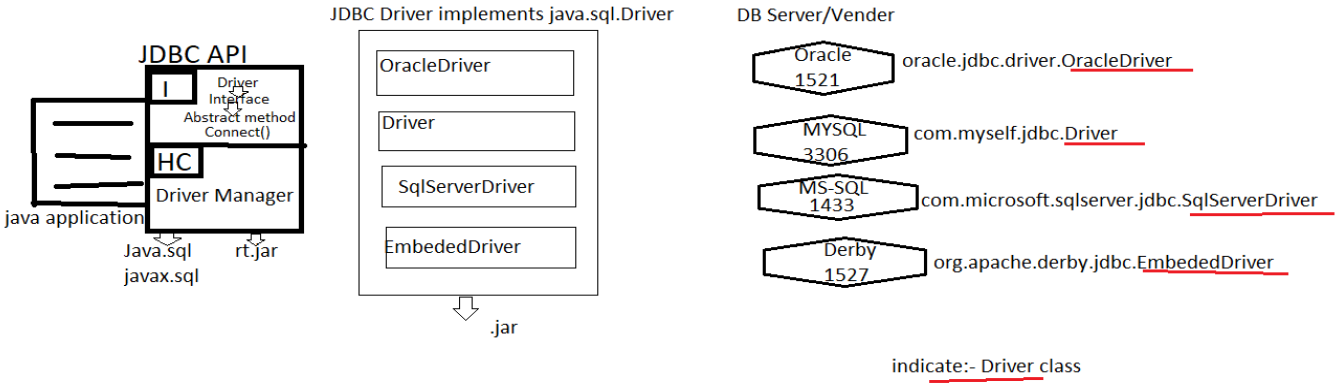
i) Load and register the Driver (It refers to Driver class provided by respective data base server or vender).

ii) Establish a connection with the data base server.

iii) Create statement or platform.

iv) Execute the sql queries or sql statement.
v) Process the resultant data (optional).
vi) Close all the costly resources.

>>Different Driver classes provided by respective data base servers or venders:-



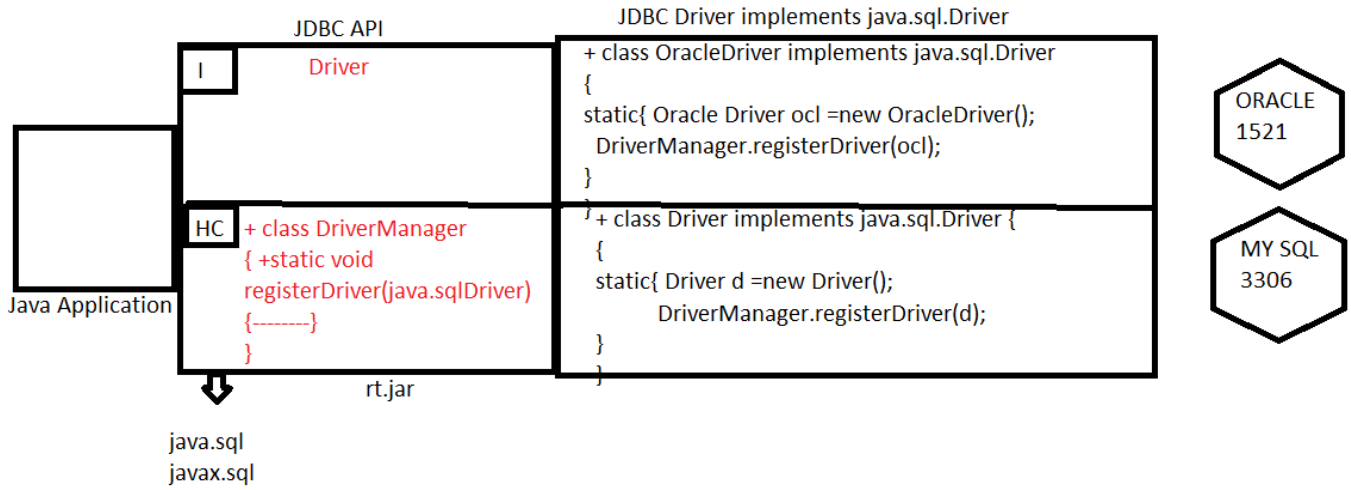
>>ALL the Driver classes which is a part of jdbc Driver are provided by respective data base servers or venders in the form of jar file.
>>All the Driver classes must mandatory implements: -java.sql.Driver interface which is a part of jdbc API.

>>For MY SQL
Driver class (JDBC Driver) Interface (JDBC API)
+class Driver implements java.sql.Driver
{
@override
Connect()
{

}
}

Date:-4-10-18.

>>Internals of Driver (Driver class):-



***>>Specifications of JDBC:-1 All the Driver classes must contains one static block in it.
2. All the Driver classes must mandatorily implements java.sql.Driver interface which is a part of jdbc Api.
3 All the Driver classes must be mandatory registered with DriverManager using a static method called 'registerDriver()' method.

>> 1 steps of JDBC Load and register the Driver:-

i) In this step, we have to load and register the Driver classes which is a part of JDBC Driver which are provided by respective data base server or vendors

ii) Driver classes can be loaded and registered in 2 diff ways namely: -

A)Manually: In this case an object of the Driver class is created by user and then registered with DriverManager using static method called registerDriver() method.

Syntax: - Driver d =new Driver(); DriverManager.registerDriver(d);

Note: - this is not a good practise since it causes tight coupling b/w java application and data base server.

B) By using a static method called forName() method we can load and register the Driver classes provided by respective data base servers or vendors.

syntax: - Class.forName("com.mysql.jdbc.Driver");

>>Date:-5-10-18

>>Steps for installation of MY-SQL server:-

1. Select the set up type to be complete and click next and install.
2. Skip the sign up page and click on next page and finish.
3. Select the config type to be standard configuration and click on next.
4. Check for two different properties mandatory.
   i) Launch the mysql server automatically.
   ii) Include the bin directory in windows path and click on next.

5. Enter a password of your choice for the user called root and reconfirm the password and click on next.
6. Click on execute to make sure that all the four properties are executed without which server fails to work.

```
Package org.btm.jdbcApp;
public class JdbcDemo
{
psvm(------)
{
try
{
Class.forName("com.mysql.jdbc.Driver");
sop("Driver class loaded and registered");
}
catch(ClassNotFoundException e)
{
e.printStackTrace();
}
}
}
```

Date:-8-10-18
Q. 1 Define jar file and explain contents and need for jar file.
Q. 2 Define design pattern and explain factory design pattern, integrating all the 3 diff types of logic.
Q. 3 Define the following things:- abstraction Loose coupling ,interface ,factory or helper method, costly resources, NullPointerException.
Q. 4 Define API and explain the 2nd form of API in details
Q. 5 Define JDBC and explain specifications of JDBC along with its advantages.
Q. 6 Explain the step of JDBC.
Date:-10-10-18

>>costly resources or heavy weight object:-
[data.txt]
>>FileReader fr =new FileReader("Data.txt"); here fr will get back to Data.txt by(using System properties in the form of Stream)

>>Costly Resources :-
>>Resources which make use of System properties in the form of Stream are known as costly resources.
>>It is always a good practice to close all the costly resources since it decreases the performance of an application.
>>All the costly resources must be close within the finally block using an if condition to avoid NullPointerException.

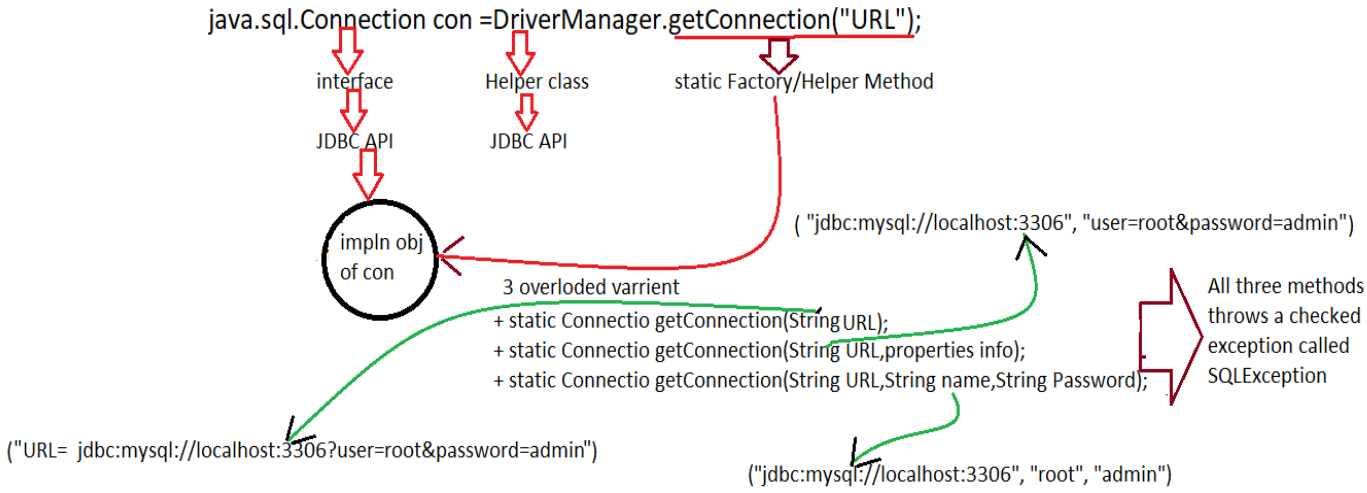
>>NOTE:-All the interfaces of JDBC api are considered to be costly resources w.r.t to JDBC.

>>2nd step of JDBC Establish the connection b/w Data Base Server and java Application:-

To make connection we use these things

[JAVA APPLICATION]---------- (URL/USER/PASSWORD) ----------> [DATABASE SERVER]

>>



1. In this step, we have to establish a connection b/w the java application and the data base Server by using getConnection("URL") method.
2. getConnection() method is static factory/helper method which is used to create and return an implementation object of Connection Interface 'based on URL'.
3. Hence, the return type for the getConnection() method is Connection interface.
4. There are 3 diff overloaded variants of getConnetion() method present namely refer the above fig.
5 Whenever we use either of the overloaded variants of getConnection() it throw a checked Exception called SQLException.
Syntax:-java.sql.connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=admin");

6. *** java.sql.Connection:- It is an interface which is a part of JDBC Api and the implementations are provided by respective database servers or vendors as a part of JDBC Driver.
7. It is always a good practice to close Connection Interface since it is considered to be a costly resources which decreases the performance of the application.

8 java.sql.DriverManger:-It is a Helper class which is a part of JDBC Api which contains 2 important static method in it's namely :- i). registerDriver() ii) . getConnection()

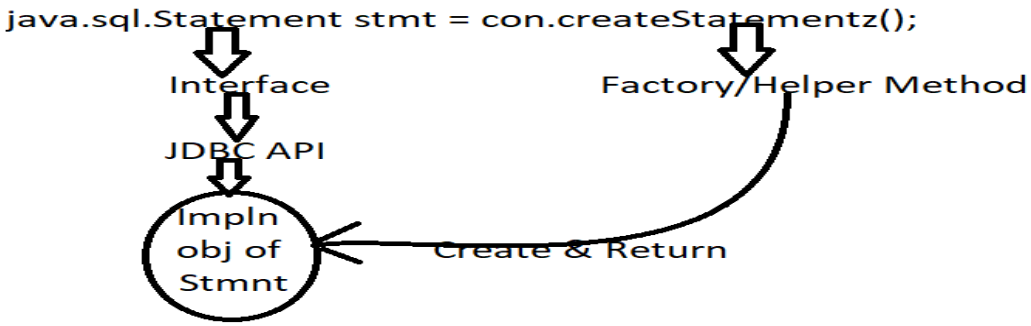
Date:-11-10-18

>> 3rd step of JDBC Create a statement or platform:-

>We need to crate statement or platform to execute the 'sql queries or sql statements'.
>A platform can either be created by using Statement or PreparedStatement or CallableStatement interface which are the part of JDBC Api.

Syntax:-
Statement<------ (extends)--PreparedStatement<------(extends)-------CallableStatement

>>java.sql.Statement:-*this is the interface which is the part of JDBC Api and the implementations are provided by respective Data base Servers or venders as a part of JDBC Driver.
>>



createStatement() is a factory method which is use to create a return implementation object of Statement Interface.
>Hence, the return type for createStatement() is Statement Interface.

```
package org.btm.jdbcApp;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JdbcDemo {
	public static void main(String[] args) {
		Connection con = null;
		Statement stmt = null;
		try {
		Class.forName("com.mysql.jdbc.Driver");
		System.out.println("Driver class loaded and register ");
		con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
		System.out.println("connection Established");
		stmt = con.createStatement();
		System.out.println("Plateform created");
		}
		catch (ClassNotFoundException | SQLException e) {
			e.printStackTrace();
		} finally {
			if (stmt != null) {
				try {
					stmt.close();
				} catch (SQLException e) {
					e.printStackTrace();
```

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

Date:-12-10-18

>> 4th step of JDBC execute the sql queries or sql statements:-

:-Diff type of Sql statement or queries:-DML:-Insert,Update,Delete
:-DDL:-Create,Drop,Alter,Rename,Truncate
:-DQL:-Select

1. To execute the diff type sql statement or queries, 3 diff methods are present namely execute() ,executeUpdate(), executeQuery() .
2. All these methods are declared in Statement interface but it can be used either wrt to Statement or PreparedStatement or CallableStatement interface which is a part of JDBC Api.

>>execute():- +boolean execute("Generic sql Query ");

- *It is a generic method since it is used to execute any type of Sql queries or statements.
- *Hence, return type of this method is boolean.
- *execute() returns a boolean true value in case of DQL and Boolean false in case of DML or DDL.
- *e.g:-syntax:- stmt.execute("Generic sql queries");

>**[the outcome of DML is 0 to n integer value which gives the total no of records affected in the data base server.]

>getUpdateCount():- + int getUpdateCount("only DML");

- * It is used to get the count of total no of records affected in the data base server 'at once'.
- *It is used to execute only DML queries or DML statements.
- *The outcome of DML is 0 to n integer value which gives the total no of records affected in the data base server.
- *hence, the return type for this method is int.
- *Whenever we try to execute a DDL or DQL queris using this method it throws an exception called SQLException.(which can not be handled by try catch or throws so to avoid this Exception we wont use any other queries apart from DML).

>>executeUpdate():- signatue:- + int executeUpdate("only DML");

*This method is specialised method which is used to execute only DML queries or stamnt .
*hence, the return type for executeUpdate() is int.
*whenever we try to execute a DDL or DQL query using this method, it throws an exception called SQLException(cant handle)

..CODE TO INSERT 1 single record into the data base server using Statement interface.

```
package org.btm.jdbcApp;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JdbcDemo {
	public static void main(String[] args) {
		Connection con = null;
		Statement stmt = null;
		string qry=  "delete from BTM.Student where id=4";
		try {
		Class.forName("com.mysql.jdbc.Driver");
		System.out.println("Driver class loaded and register ");
		con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
		System.out.println("connection Established");
		stmt = con.createStatement();
		System.out.println("Plateform created");
		stmt.executeUpdate(qry);
		System.out.println("data updated");
		}
		catch (ClassNotFoundException | SQLException e) {
			e.printStackTrace();
		} finally {
			if (stmt != null) {
				try {
					stmt.close();
				} catch (SQLException e) {
					e.printStackTrace();
				}
			}
			if(con!=null)
			{
				try {
					con.close();
				} catch (SQLException e) {
					e.printStackTrace();
				}
			}
```

```
		}

	}

}
```

Date:-15-10-18

>>Code to insert multiple data into a table using Statement Interface:-

```
package org.btm.jdbcApp;
import java.sql.*;
public class MultipleIsert {
	public static void main(String[] args) {
		Connection con = null;
		Statement stmt = null;
		String insqry1="insert into BTM.Student values(1,'Pappu',94.80)";
		String insqry2="insert into BTM.Student values(2,'Dappu',95.80)";
		String insqry3="insert into BTM.Student values(3,'Gappu',96.80)";
		try {
			Class.forName("com.mysql.jdbc.Driver");
			System.out.println("Driver class loaded and register ");
			con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
			System.out.println("connection Established");
			stmt = con.createStatement();
			System.out.println("Plateform created");
			stmt.executeUpdate(insqry1);
			stmt.executeUpdate(insqry2);
			stmt.executeUpdate(insqry3);
			System.out.println("inserted successfuly");
		} catch (ClassNotFoundException | SQLException e) {
			e.printStackTrace();
		} finally {
			if (stmt != null) {
				try {
					stmt.close();
				} catch (SQLException e) {
					e.printStackTrace();
				}
			}
			if(con!=null)
			{
				try {
					con.close();
				} catch (SQLException e) {
					e.printStackTrace();
```

```
                    }
                }
                System.out.println("all costly resources closed");
            }
        }
}
```

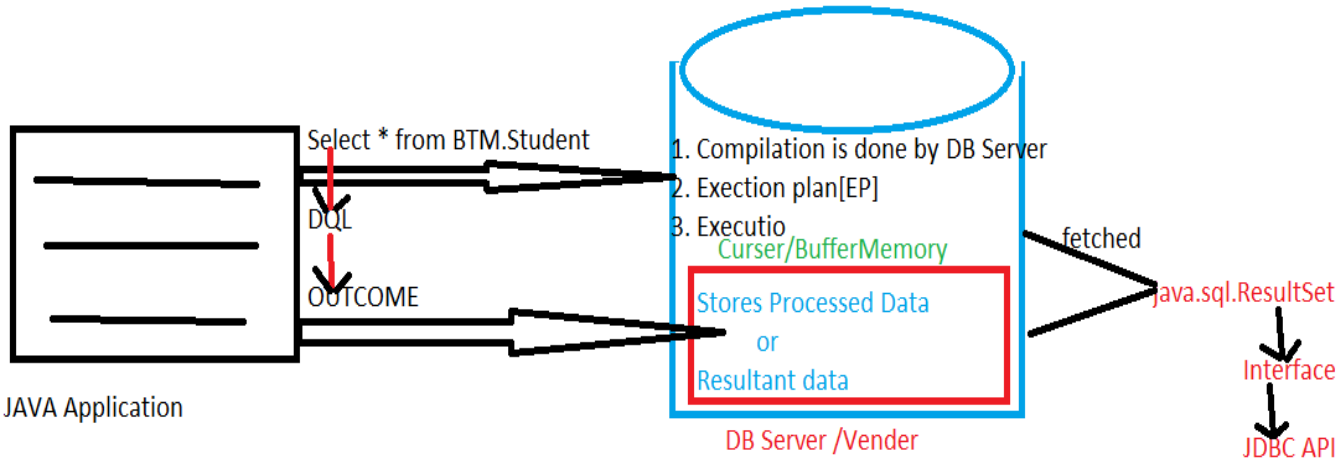
>>Statement stmt =con.createStatement(qry);

stmt.execute("Query"); // where query get compiled first then execute(2 step).

>>Since compilation takes place each time along with execution whenever we execute query using Statement interface, performance of an application decreases.
>>Hence,it is a good practice to make use of 'PreparedStatement interface' to deal with multiple records.

Date:-16-10-18

>> Fetching the data from data base server: -



*Whenever we execute ant DQL query or DQL Statement we may get a result which is referred as processed or resultant data
*The processed or resultant data stored in the Curser or Buffered memory, which can be fetched by using ResultSet interface which is a part of JDBC API.
*The Structure of Curser or Buffered memory may differ from the Structure of data base.
*NOTE:-Column name is also known as column label
Column number is also known as column index

>>java.sql.ResultSet:-It is an interface which is a part of jdbc Api and the implementations are provided by respective data base servers or venders as a part of jdbc Driver.
*>> A set of methods of ResultSet interface are used to fetch the processed or resultant data from the curser or buffer memory which are known as 'getXXX() methods'.
>> Any method which deals with data from buffer or curser memory must be declare in ResultSet.
>> There are 2 different overloaded variants or method of getX present namely:-
+ XXX getXXX(int columnnumber) + XXXgetXXX(String Columnname)
>> return type of this method is respective data type.

+ int getInt(int columnNumber) + String getString(int columnNumber)
+ double getDouble(int columnNumber)

+ int getInt(String columnname) + String getString(String columnname)
+ double getDouble(String columnname)

BTM----> Student

no.	type		
1.	int	2. String	3. double
id		name	per
1		ABC	22.22
2		DEF	33.33
3		GHI	44.44
4		JKL	55.55

>>NOTE:-**By default the ResultSet interface does not point to any record in the curser or buffer memory.

>> next():- signature:- + boolean next()
* It is used to check whether the next record is present in the curser or buffer memory or not and return a Boolean value called true or false but not the record.
* It can be used whenever there are minimum no of records present in the curser or buffer memory.
* When there are more 'n' no of data in curser memory then use of next() each 'n' time, decrease the performance of app.

>> absolute() :- signature:- + boolean absolute(int rownumber)
*It is used to check whether a particular record present is present in the curser or buffer memory' based on the parameter called int rowNo' and returns a boolean values but not the records.
* It can be used whenever there are n no of records present in the curser or buffer memory.

* both next() and absolute() are declared in ResultSet interface.

>> executeQuery():- Signature:- + ResultSet executeQuery("Only Dql");
*It is a 'specialised method' which is used to execute only dql queries or dql statement .
*the outcome of dql is processed or resultant data which is stored in the curser or buffer memory which can be fetched throgh the help of ResultSet interface which is a part of jdbc api.
*[hence, the return type for executeQuery() is ResultSet interface].
*whenever we try to execute a DML or DDL query using this method, it thrwos an Exception called SQLException(can't handle).

Date:-17-10-18

>> Creating an implementation object of ResultSet interface:-

*there are 2 diff ways to create an implementation object of resultSet interface which are as follows:-

1st way:-

```
boolean val=stmt.execute("DQL query");
if(val)
{
java.sql.ResultSet rs =stmt.getResultSet();
//ResultSet is interface of JDBC Api
//getResultSet() is helper method which create and return,
//implementation object of ResultSet interface.
 rs.next();
//use getXXX() .
}
```

2nd way:-

```
ResultSet rs =stmt.executeQuery("DQL")
```

>>place holder:-* It is a parameters which holds dynamic values at the run time by the user.
* In case of JDBC, place holder is represented as '?'.

```
String inqry="insert into BTM.Student values(?,?,?)";
String upqry ="update BTM.Student set name=? where id=?";
String dlqrry="delete from BTM.Student where id=?";
String selqry="select * from BTM.Student where id =?";
```

>>Rules to set the data for a place holder:-

There are 3 different rules to set the data for a place holder namely :-

**1. We have to set the data for a place holder 'before the exection'.
2. The no of data must exactlly match the no place holder.
3. We have to set the data for a place holder by using 'setXXX()' method.

Signature of getXXX() :- + setXXX(int placeholdernumber/placeholderindex ,XXX data);

+ void setInt(int placeholdernumber/placeholderindex , int data)
+ void setString(int placeholdernumber/placeholderindex ,String data)
+ void setDouble(int placeholdernumber/placeholderindex ,double data)

e.g:-
```
setInt(1,34);
setString(2,"Rakesh");
setDouble(3,55.55);
```

>>NOTE:- By default, the return type for getXXX() method is respective data types, where as return type of setXXX() method is void.

>>java.sql.PreparedStatement:-*It is an interface which is the part of JDBC Api ,and its implementation are provided by respective data base servers or venders as a part of JDBC Driver.

*PreparedStatement interface is sub interface of Statement interface.
*It is supports the concept of place holder to take dynamic value at the run time by the user
*It supports the concept of compile once and execute many times (execution plan)

*In case of PreparedStatement, the query is passed at the time of implimentation object creation of PreparedStatement interface but not in the execute() method.
Hence,PreparedStatement is also known as 'precompiled statement'.

>> PreparedStatement is faster than Statement cause of its 2nd and 3rd rules.

>>-> Statement stmt =con.createStatement();
stmt.execute(qury);----> compile and execute each time.

>> java.sql.PreparedStatement pstmt=con.prepareStatement(Query); ----->pre compiled
//where ,PreparedStatement is interface of JDBC Api
pstmt.execute();----> Execution

* prepareStatement() is helper method which create and return,implementation object of reparedStatement interface. Hence, the return type for prepareStatement() is preparedStatement interface.

>> Code to insert multiple data into the data base server using PreparedStatement interface along with place holder:-

```
package org.btm.jdbcApp;
import java.sql.*;
public class MultiplWithPS {
	public static void main(String[] args) {
		Connection con = null;
		PreparedStatement pstmt = null;
		String insqry="insert into BTM.Student values(?,?,?)";

		try {
			Class.forName("com.mysql.jdbc.Driver");
			System.out.println("Driver class loaded and register ");
			con = DriverManager.getConnection("jdbc:mysql://localhost:3306?
                                                    user=root&password=tiger");
			System.out.println("connection Established");
			pstmt = con.prepareStatement(insqry);
			System.out.println("Plateform created");
			//SET THE VALUE PLACE HOLDER BEFORE EXECUTION//
			pstmt.setInt(1, 1);
			pstmt.setString(2, "RAKESH");
			pstmt.setDouble(3, 75.00);
			pstmt.executeUpdate();

			pstmt.setInt(1, 2);
			pstmt.setString(2, "RISHABH");
			pstmt.setDouble(3, 70.00);
			pstmt.executeUpdate();

			pstmt.setInt(1, 3);
```

```
                pstmt.setString(2, "PAPPU");
                pstmt.setDouble(3, 80.00);
                pstmt.executeUpdate();
                System.out.println("inserted successfuly");
            } catch (ClassNotFoundException | SQLException e) {
                e.printStackTrace();
            } finally {
                if (pstmt != null) {
                    try {
                        pstmt.close();
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
                if(con!=null)
                {
                    try {
                        con.close();
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("all costly resources closed");
            }
        }
}
```

Date:-18-10-18

>> Code to fetch multiple record from the curser or buffer memory by using ResulSet interface along with getXXX() method.

```
package org.btm.jdbcApp;
import java.sql.*;
public class FetchData {
    public static void main(String[] args) {
        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        String qry = "select * from BTM.Student ";

        try {
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("Driver class loaded and register ");
            con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
            System.out.println("connection Established");
            pstmt = con.prepareStatement(qry);
```

```
                System.out.println("Plateform created");
                rs = pstmt.executeQuery();
                while (rs.next()) {
                        int id = rs.getInt("id");
                        String name = rs.getString(2);
                        double per = rs.getDouble(3);
                        System.out.println(id + " " + name + " " + per);
                }
        } catch (ClassNotFoundException | SQLException e) {
                e.printStackTrace();
        } finally {
                if (rs != null) {
                        try {
                                rs.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                if (pstmt != null) {
                        try {
                                pstmt.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                if (con != null) {
                        try {
                                con.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                System.out.println("all costly resources closed");
        }

    }

}
```

>> code to fetch a single record from curser memory using ResultSet interface along with getXXX() method:-

```
String qry = "select * from BTM.Student where id =3";
        while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString(2);
                double per = rs.getDouble(3);
                System.out.println(id + " " + name + " " + per);
```

```
                }
-------------------------------------------------------------------------------------------------------------------------
-------------------------
>> code to fetch a particular record from the curser or buffer memory where id is=?:-
package org.btm.jdbcApp;
import java.sql.*;
import java.util.Scanner;
public class FetchDataOnId {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                Connection con = null;
                PreparedStatement pstmt = null;
                ResultSet rs = null;
                String qry = "select * from BTM.Student where id=? ";

                try {
                        Class.forName("com.mysql.jdbc.Driver");
                        System.out.println("Driver class loaded and register ");
                        con = DriverManager.getConnection("jdbc:mysql://localhost:3306?

        user=root&password=tiger");
                        System.out.println("connection Established");
                        pstmt = con.prepareStatement(qry);
                        System.out.println("Plateform created");
                        System.out.println("please enter the id for u want to fetch the  data");
                        int n = sc.nextInt();
                        // SET THE VALUE PLACE HOLDER BEFORE EXECUTION//
                        pstmt.setInt(1, n);
                        rs = pstmt.executeQuery();
                        if (rs.next()) {
                                int id = rs.getInt("id");
                                String name = rs.getString(2);
                                double per = rs.getDouble(3);
                                System.out.println(id + " " + name + " " + per);
                        } else
                                System.err.println("no such data is present on the id=" + n);
                } catch (ClassNotFoundException | SQLException e) {
                        e.printStackTrace();
                } finally {
                        if (rs != null) {
                                try {
                                        rs.close();
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                }
                        }
                        if (pstmt != null) {
```

```
                        try {
                                pstmt.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                if (con != null) {
                        try {
                                con.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                sc.close();
                System.out.println("all costly resources closed");
        }

    }
}
```

---

code to fetch a particular record from the curser or buffer memory where name is=?:-

```
package org.btm.jdbcApp;

import java.sql.*;
import java.util.Scanner;

public class FetchDataOnName {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                Connection con = null;
                PreparedStatement pstmt = null;
                ResultSet rs = null;
                String qry = "select * from BTM.Student where name=? ";

                try {
                        Class.forName("com.mysql.jdbc.Driver");
                        System.out.println("Driver class loaded and register ");
                        con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
                        System.out.println("connection Established");
                        pstmt = con.prepareStatement(qry);
                        System.out.println("Plateform created");
                        System.out.println("enter name  u want to fetch the data");
                        String nam = sc.nextLine();
                        // SET THE VALUE PLACE HOLDER BEFORE EXECUTION//
                        pstmt.setString(1, nam);// here 1 is not the column number it
                                        //is no of position of place holder in the query
```

```
                        rs = pstmt.executeQuery();
                        if (rs.next()) {
                                int id = rs.getInt("id");
                                String name = rs.getString(2);
                                double per = rs.getDouble(3);
                                System.out.println(id + " " + name + " " + per);
                        } else
                System.err.println("no such data is present on the name=" + nam);
                } catch (ClassNotFoundException | SQLException e) {
                        e.printStackTrace();
                } finally {
                        if (rs != null) {
                                try {
                                        rs.close();
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                }
                        }
                        if (pstmt != null) {
                                try {
                                        pstmt.close();
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                }
                        }
                        if (con != null) {
                                try {
                                        con.close();
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                }
                        }
                        sc.close();
                        System.out.println("all costly resources closed");
                }

        }

}
```

>> code for login vallidation:-

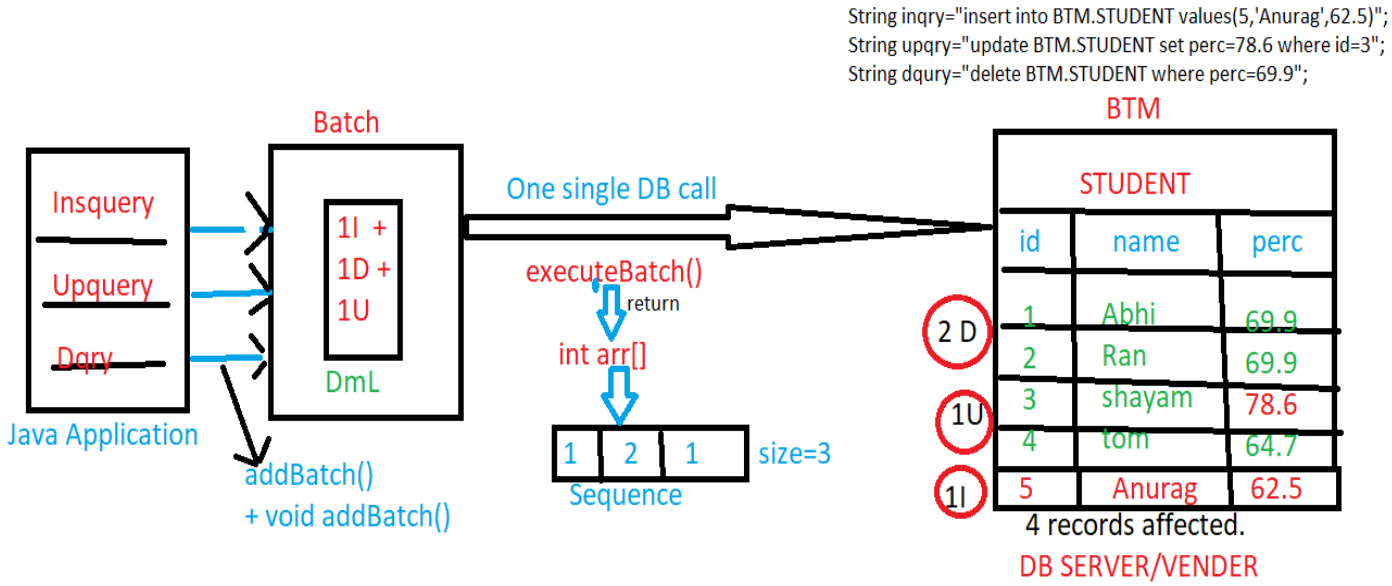
```
package org.btm.jdbcApp;
import java.sql.*;
import java.util.Scanner;
public class LoginVallidation {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
```

```
		Connection con = null;
		PreparedStatement pstmt = null;
		ResultSet rs = null;
		String qry = "select username from BTM.User where name=? and password=?
";

		try {
			Class.forName("com.mysql.jdbc.Driver");
			System.out.println("Driver class loaded and register ");
			con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
			System.out.println("connection Established");
			pstmt = con.prepareStatement(qry);
			System.out.println("Plateform created");
			System.out.println("please enter the name ");
			String nam = sc.nextLine();
			System.out.println("please enter the password");
			String pwd = sc.nextLine();
			// SET THE VALUE PLACE HOLDER BEFORE EXECUTION//
			pstmt.setString(1, nam);// here 1 is not the column number it is no of
			pstmt.setString(2, pwd);	//position of place holder in the query
			rs = pstmt.executeQuery();
			if (rs.next()) {

				System.out.println("welcom   "+rs.getString(1));
			} else
		System.err.println("no data is present for name=" + nam+", or password ="+pwd);
		} catch (ClassNotFoundException | SQLException e) {
			e.printStackTrace();
		} finally {
			if (rs != null) {
				try {
					rs.close();
				} catch (SQLException e) {
					e.printStackTrace();
				}
			}
			if (pstmt != null) {
				try {
					pstmt.close();
				} catch (SQLException e) {
					e.printStackTrace();
				}
			}
			if (con != null) {
				try {
					con.close();
```

```
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                sc.close();
                System.out.println("all costly resources closed");
        }
    }
}
```

Date:-19-10-18

>> Batch Update:-



* Each and every individual db call which is made from java application to the data base server is considered to be a costly operation.
*The more no of DB call reduces the performance of an application. Hence, it is better to use Batch update.

>> Definition for Batch:-Batch is collection of only DML Query or DML Statements.

>> Need for Batch Update:-Batch update is needed to make one single DB call and affect multiple records of multiple tables 'at once'.

>> Advantage for Batch Update:-It greatly improves the performance of an application.

*It is applicable only for DML query or DML statement.
*In case of Batch Update, after adding all the DML queries into the batch, the entire batch will be executed only once by making one single DB call.
** It can be used both wrt Statement as well as PreparedStatement interface
*There are 2 different methods assosiated wrt to Batch namely:-
i) + void addBatch ii) executeBatch()

>>i) + void addBatch :- It is used to add all the DML query into the Batch.

>>ii) executeBatch():-It is used to execute all DML queries present in the Batch only once by making one single DB call.

** executeBatch() returns an 'int array'.
the size of this array represent the total no DML queries added into the Batch.
sequence of the int array is same as the sequence in which the DML queries are 'added into the batch'.
And the values in the array are no of records affected in that table for respective sequential Queries.

>> Code for Batch w.r.t to Statement interface:-

```
package org.btm.jdbcApp;
import java.sql.*;
public class BatchUpdateDemo {
	public static void main(String[] args) {
		Connection con = null;
		Statement stmt = null;
		String inqry = "insert into BTM.User values('Rajesh','455','id')";
		String delqry = "delete from BTM.Student where perc=55.55";
		String upqry = "update BTM.Student set perc=89.99 where id=1";
		try {
			Class.forName("com.mysql.jdbc.Driver");
			System.out.println("Driver class loaded and register ");
			con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
			System.out.println("connection Established");
			stmt = con.createStatement();
			System.out.println("Plateform created");
			stmt.addBatch(inqry);
			stmt.addBatch(delqry);
			stmt.addBatch(upqry);
			int arr[] = stmt.executeBatch();
			for (int i : arr) {
				System.out.print(i);
			}

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

```
                        try {
                                con.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                System.out.println("aal costly resources closed");
        }
    }
}
```

Code for Batch wrt to PreparedStatement interface:-

```
package org.btm.jdbcApp;
import java.sql.*;
public class BatchUpdateDemo {
    public static void main(String[] args) {
        Connection con = null;
        PreparedStatement pstmt=null;
        PreparedStatement pstmt1=null;
        String inqry = "insert into BTM.User values('Rajesh','455','id')";
        String delqry = "delete from BTM.Student where perc=55.55";
        try {
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("Driver class loaded and register ");
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306?

    user=root&password=tiger");
            System.out.println("connection Established");
            pstmt = con.prepareStatement(inqry);
            System.out.println("Plateform created");
            pstmt.addBatch();
            int arr[] = pstmt.executeBatch();
            for (int i : arr) {
                System.out.print(i);
            }
            pstmt1 = con.prepareStatement(delqry);
            System.out.println("Plateform created");
            pstmt1.addBatch();
            int arr[] = pstmt1.executeBatch();
            for (int i : arr) {
                System.out.print(i);
            }

        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        } finally {
```

```
                        if (pstmt1 != null) {
                                try {
                                        pstmt1.close();
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                }
                        }
                        if (pstmt != null) {
                                try {
                                        pstmt.close();
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                }
                        }
                        if (con != null) {
                                try {
                                        con.close();
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                }
                        }
                        System.out.println("aal costly resources closed");
                }
        }
}
```

>> JDBC Transaction:-

i) Exchange of data or information b/w 2 media is known as Transaction.

ii) By default, the AutoCommit mode is set to Boolean true value because of which all the data are automatically saved into the database server wherever we perform any data base operation.

iii) We can explicitly disable AutoCommit method {setAutoCommit(false)} and passing a Boolean false argument.

Syntax:- + void setAutoCommit()

eg:- con.setAutoCommit(false);

>> Note*:- i) We have to disable the AutoCommit mod before we begain the transaction but after the establishing a connection with database server.

ii) Once the AutoCommit mode is disable we have to explicitly save the data into the data base server by using commit() method at the end of the Transaction.

Syntax: - + void commit()

eg :- con.commit();

---> If any one of the database operation fails, then the rollback operation is called. Which is used to Rollback 'entire executed database operation' and Transaction starts from beginning.

Syntax: - + void rollBack()

e.g:- con.rollBack();

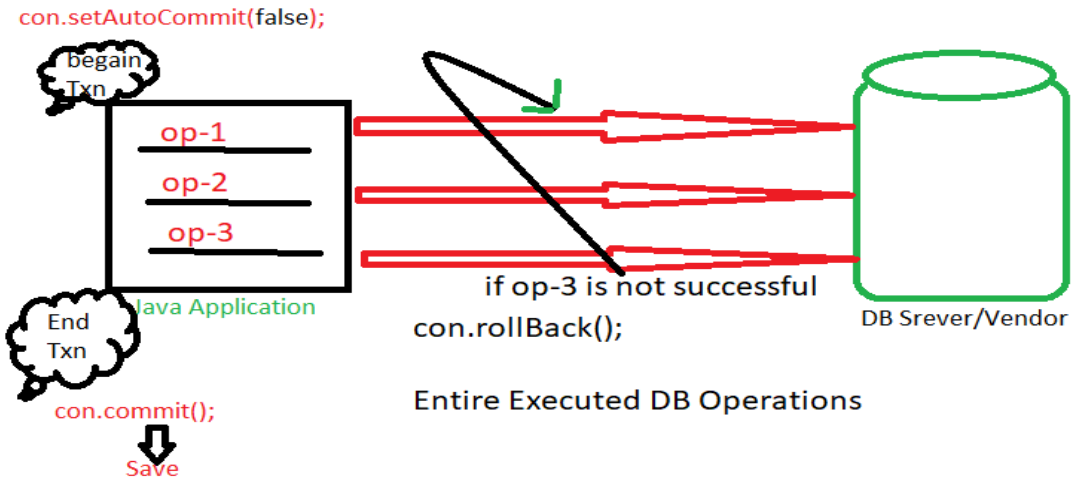
>> Defination for JDBC Transaction:-
JDBC transaction is considered to be a 'single Business unit'. Which may have multiple SQL Statements which must be executed.

>>Need of JDBC Transaction:-
JDBC Transaction is needed to maintain data consistency in the DB Server.

>>Advantage of JDBC Transaction:-
It is used to achive ACID properties or rules where A refers to Automicity, C for Consistency, I for Isolation and D for Durability.

--> Automicity: - It means do everything or do nothing, do everything refers to complete or successful transaction where if all the database operation are successfully executed, then the data's are saved into the database server leading to data Consistency.
do nothing refers to incomplete or unsuccessful transaction where if any one of database operation fails ,then the rollBack operation is called which is used to roll back the entire executed DB operation and transaction starts from beginning without saving any data's into the DB server due to Data Inconsistency.

Date:-22-10-18



AccNo	AccName	AccBal			
421	Bipasha	1000000	-10000	Debit	__Maintain Data ->JDBC Transaction
420	Abc	0	+10000	Credit	Consistancy in DB Server

>> Code for JDBC Transaction:-

```
package org.btm.jdbcApp;
```

```
import java.sql.*;
import java.util.Scanner;
public class Transcation {
	public static void main(String[] args) {
		Connection con = null;
		PreparedStatement pstmt = null;
		PreparedStatement pstmt1 = null;
		String inqry1 = "insert into BTM.Student1 values(?,?,?,?)";
		String inqry2 = "insert into BTM.Student2 values(?,?,?)";
		Scanner sc =new Scanner(System.in);
		System.out.println("enter ID??");
		int id =sc.nextInt();
		System.out.println("Enter name");
		String name=sc.next();
		System.out.println("Enter dept");
		String dept=sc.next();
		System.out.println("enter perc");
		double perc=sc.nextDouble();
		System.out.println("Enter place");
		String place=sc.next();
		sc.close();
		try {
			Class.forName("com.mysql.jdbc.Driver");
			con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
			//disable AutoCommit()//
			con.setAutoCommit(false);

			pstmt=con.prepareStatement(inqry1);
			pstmt.setInt(1, id);
			pstmt.setString(2, name);
			pstmt.setString(3, dept);
			pstmt.setDouble(4,perc);
			pstmt.executeUpdate();
			System.out.println("Student1 complete");

			pstmt1=con.prepareStatement(inqry2);
			pstmt1.setInt(1, id);
			pstmt1.setString(2, name);
			pstmt1.setString(3, place);
			pstmt1.executeUpdate();
			System.out.println("Student2 complete");

			con.commit();

		} catch (ClassNotFoundException | SQLException e) {
			try {
```

```
                        con.rollback();
                        System.out.println("rolled out");
                    } catch (SQLException e1) {
                        e1.printStackTrace();
                    }
                    e.printStackTrace();
                } finally {
                    if (pstmt1 != null) {
                        try {
                            pstmt.close();
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                    }
                    if (pstmt != null) {
                        try {
                            pstmt.close();
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                    }
                    if (con != null) {
                        try {
                            con.close();
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                    }
                    System.out.println("aal costly resources closed");
                }
            }
}
```

Date:-23-10-18

>> java.sql.Savepoint sp=con.setSavepoint();

:-*setSavepoint() is a factory or helper method which is used to create and return implementation object of Savepoint interface.

*Hence, the return type of this method is Savepoint interface.

*Note:-In general the return type for setSavepoint() is void.

syntax:- + void setSavepoint();

>> Code for JDBC transaction using Savepoint Interface:-

```
package org.btm.jdbcApp;
import java.sql.*;
import java.util.Scanner;
public class TransactionWithSavepoint {
```

```
	public static void main(String[] args) {
		Connection con = null;
		PreparedStatement pstmt = null;
		PreparedStatement pstmt1 = null;
		Savepoint sp=null;
		String inqry1 = "insert into BTM.Student1 values(?,?,?,?)";
		String inqry2 = "insert into BTM.Student2 values(?,?,?)";
		Scanner sc =new Scanner(System.in);
		System.out.println("enter ID??");
		int id =sc.nextInt();
		System.out.println("Enter name");
		String name=sc.next();
		System.out.println("Enter dept");
		String dept=sc.next();
		System.out.println("enter perc");
		double perc=sc.nextDouble();
		System.out.println("Enter place");
		String place=sc.next();
		sc.close();
		try {
			Class.forName("com.mysql.jdbc.Driver");
			con =
DriverManager.getConnection("jdbc:mysql://localhost:3306?user=root&password=tiger");
			//disable AutoCommit()//
			con.setAutoCommit(false);

			pstmt=con.prepareStatement(inqry1);
			pstmt.setInt(1, id);
			pstmt.setString(2, name);
			pstmt.setString(3, dept);
			pstmt.setDouble(4,perc);
			pstmt.executeUpdate();
			System.out.println("Student1 complete");

			sp=con.setSavepoint();
			pstmt1=con.prepareStatement(inqry2);
			pstmt1.setInt(1, id);
			pstmt1.setString(2, name);
			pstmt1.setString(3, place);
			pstmt1.executeUpdate();
			System.out.println("Student2 complete");

			con.commit();

		} catch (ClassNotFoundException | SQLException e) {
			try {
				con.rollback(sp);
```

```
                        con.commit();
                        System.out.println("rolled out");
                    } catch (SQLException e1) {
                        e1.printStackTrace();
                    }
                    e.printStackTrace();
                } finally {
                    if (pstmt1 != null) {
                        try {
                            pstmt.close();
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                    }
                    if (pstmt != null) {
                        try {
                            pstmt.close();
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                    }
                    if (con != null) {
                        try {
                            con.close();
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                    }
                    System.out.println("aal costly resources closed");
                }
            }
}
```

---

gautham.r92@gmail.com

>>Interview Question for JDBC:-

1. Define Design Pattern and explain the 3 diff types og logic associated w.r.t to factory Design pattern.
2. Define the following components:- Abstraction, Loose Coupling ,Interface, Factory /helper Method, NullPointerException.
3. Define JDBC and and Explain the steps of JDBC along with its advantages.

*4. What are the Specification for JDBC.

5 Can we deal with multiple records using Statement interface.

6 Is Statement interface is faster in performance compare to PreparedStatement in case of BatchUpdate.

*7 Name the different Driver classes provided by respective data base server or vendors along with their port no.

8 Define placeholder and explain 3 diff rules w.r.t placeholders

9 What is the outcome of DML and DQL queries.
10 What is the return type for getXXX() and setXXX().
11 Explain execute() ,executeUpdate() and executeQuery().
12 Explain addBatch() and executeBatch() in details.
13 Explain JDBC Transaction in brief.

>> Coding Question:-
1. Write a code to insert update and delete a record from the data base server.
2. Write a code for login validation using standard steps of JDBC.
*3. Write a code to insert multiple records into the database server using PreparedSataement and place holder.
*4. Write a code to fetch a particular record from buffer memory where id is place holder.
5 Write a code for JDBC Transaction.

GUDIED BY: -

Gowtham Sir

WRITTEN BY: -

Rakesh Kumar Rai