PAGE NO:
DATE:

3/05/2024

J2EE

→ J2EE stands for java 2 Enterprise Edition.

→ By using java we can develop Stand Alone Application.

STANDALONE APPLICATION

An application which is ristricted to a particular system is known as standalone application.

Ex- Notepad, MS paint, Calculator, etc.

Need for J2EE -

J2EE is needed for simplification of web application development.

J2EE
(Java 2 Enterprice Edition) (Advance Java)

↓

Category

↓ ↓

JDBC — Servlets

(Java DataBase Connectivity) — (Server side java Program)

↓

[Java + SQL]

↓

DML
DQL
DDL

[Java + Web-Tech]

↓

HTML

PAGE NO:
DATE:

JAR File

→ Jar means Java archive (compress)
→ It is a file format based on Zip file format which is used to compress many files into one single file.

Contents of Jar File.

The contents of jar file are -

(i) • Java File -
It contains source code statements in it.

(ii) • class File -
It contains byte code statements in it.

(iii) Config Files -
→ It contains configuration data in it.
→ Their are two categories of config files present -

(a) • xml -
It is used to configure the resources with the help of tags.
Ex - web.xml

(b) • properties -
It is used to provide set of properties in the form of key and value pair.

PAGE NO
DATE

1/08/2024

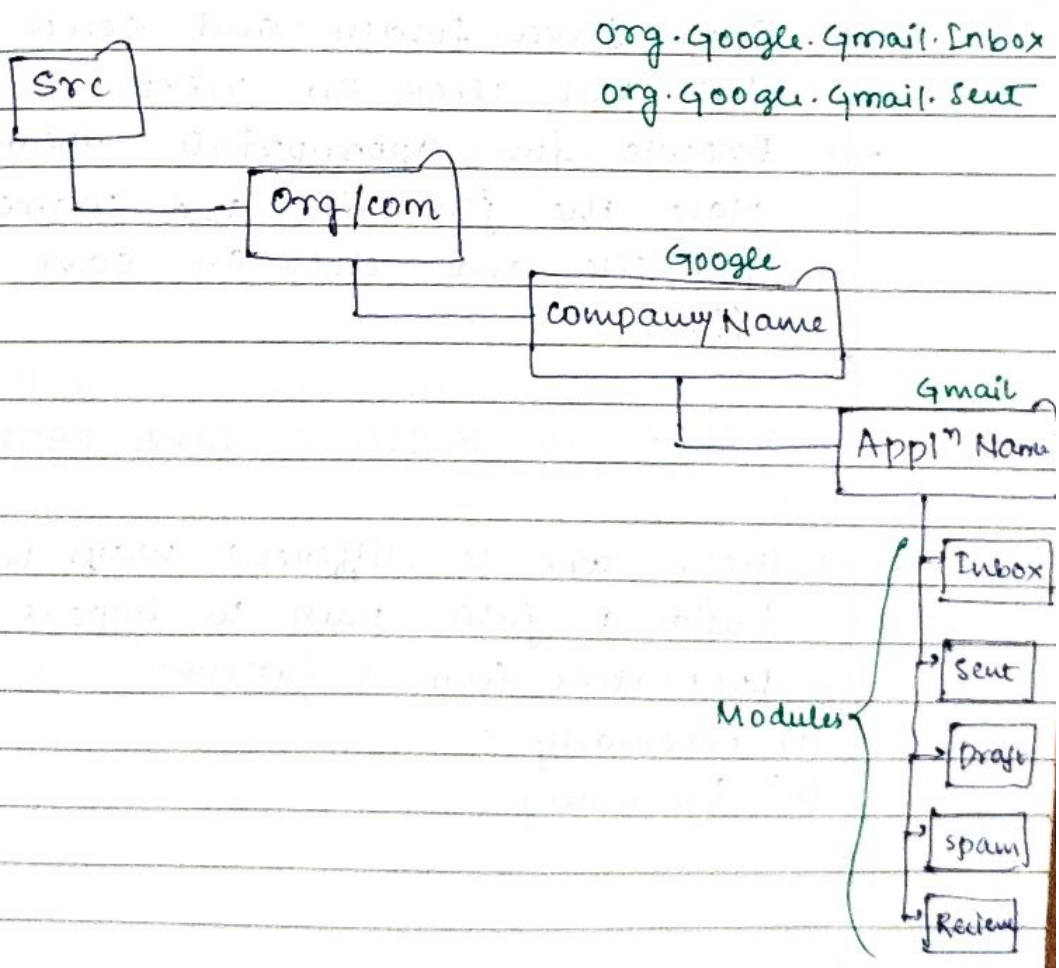
Need for JAR File

Jar file is needed to import the properties based on the requirement.

(if their is one specific task which we are not capable of to might if it is present in the different place we can use that through some formats of calling the particular function.)

STANDARD PACKAGE STRUCTURE

we need to create a standard package structure to differentiate between the modules of an application.



PAGE NO :
DATE :

STEPS TO CREATE A BASIC JDBC PROJECT

(1) Open eclipse in Java Perspective.
(2) Open .navigator mode.
(3) Right click within the navigator and create a new Java Project and name it.
(4) Select src folder and hit control +n to create a standard package structure.
(5) Select the application name and hit control +n to create a class.

STEPS TO CREATE A JAR FILE

→ Right click on Project and select export option.
→ Open Java folder and select jar file and click on next.
→ 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

→ There are 2 different ways we can build a java path to import the properties from a jarfile.
(1) Externally
(2) Internally

5/2024

PAGE NO :
DATE :

(1) EXTERNALLY

→ Right click on Project and select properties option.
→ Select Java Build Path and click on libraries tab.
→ Click on add external jars and select a jar file from the respective path and click on Apply, Apply and close.

(2) INTERNALLY

→ Right click on project and create a new folder by name lib.
→ Add the respective jar file into the lib.
→ Right click on project and select properties option.
→ Select Java Build Path and click on libraries tab.
→ Click on add jars and select the jar file from the respective project and click on Apply, Apply and close.

** ABSTRACTION

→ Hiding the implementation and providing the functionalities to the user with the help of interface is known as Abstraction.

→ We need to hide the implementation to reduce the complexity.

PAGE NO :
DATE :

** INTERFACE -

→ It is a media to communicate between user and ~~many~~ device.

OR

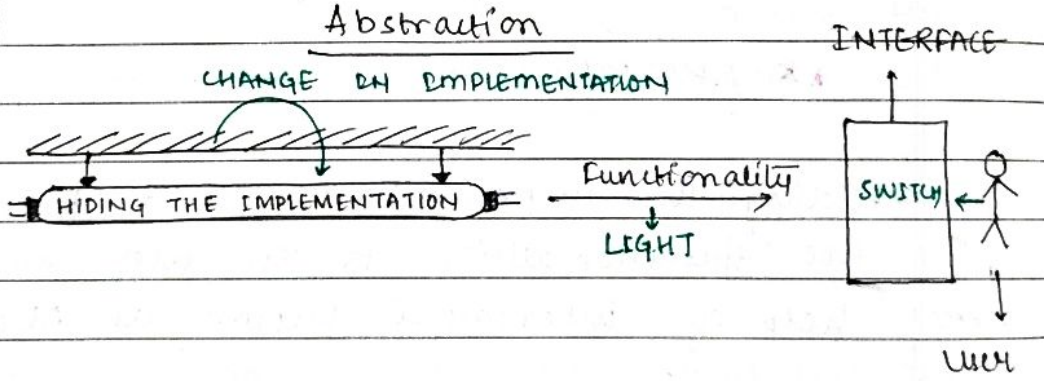
→ It is a coding contract between the service and the consumer.

TIGHT-COUPLING :-

Change in the implementation which affects the user is known as tight-coupling.

LOOSE-COUPLING :-

Change in the implementation which doesn't effect the user is known as loose coupling.



PORT NUMBER

Port number is the one which helps us to get connected to a particular server.

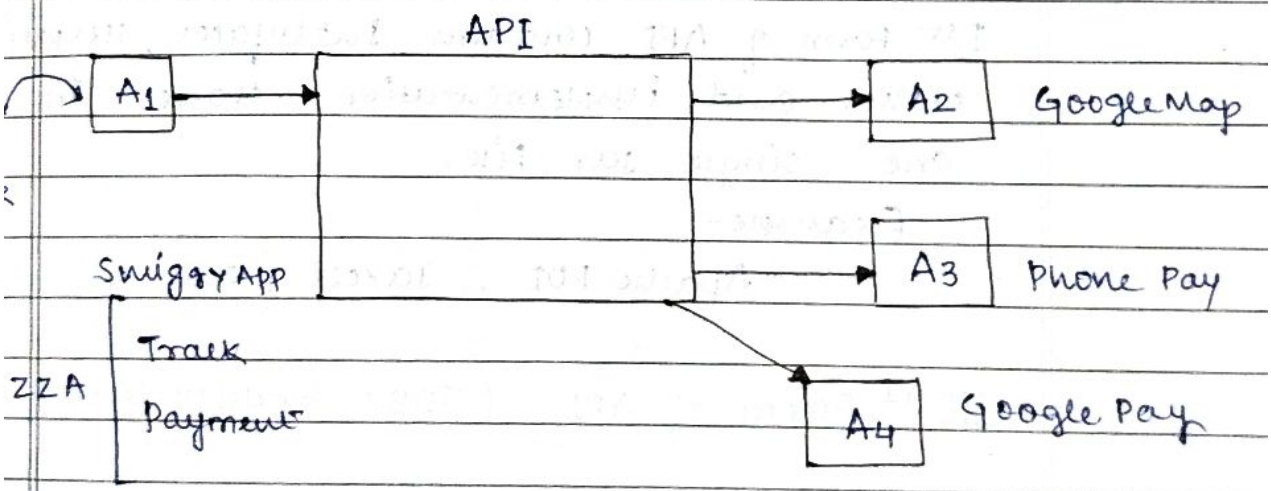
DataBase Server	Port Number
Oracle	1521
MySQL	3306
MS-SQL	1433
Derby	1527

05/2024

API

→ API stands for Application Programming Interface.

→ API is used for inter application communication i.e. one application can communicate with another application with the help of API. to achieve loose coupling.



→ Examples for API -

Apache - POI

Jaxcel

JDBC

Servlet, etc

→ APIs are given in the form of jar file.

PAGE NO :
DATE :

CONTENTS OF API -

The contents of API are -

(1) Interfaces
(2) Helper classes
(3) Implementation Classes

FORMS OF API -

Their are two different forms of APIs are present -

(i) I Form of API
(ii) II Form of API

→ I st Form of API -

I st Form of API contains Interfaces, Helper classes and implementation classes in one single jar file.

Example -

Apache POI, Jaxcel, etc.

→ II nd Form of API - (JDBC Architecture)

JDBC API

→ JDBC API was given by Sun Micro System to achieve loose coupling between Java Application & Data Base Server.

→ JDBC API contains Interfaces and helper class in the form of jar file.

→ JDBC API is distributed into two different packages

- java.sql
- javax.sql

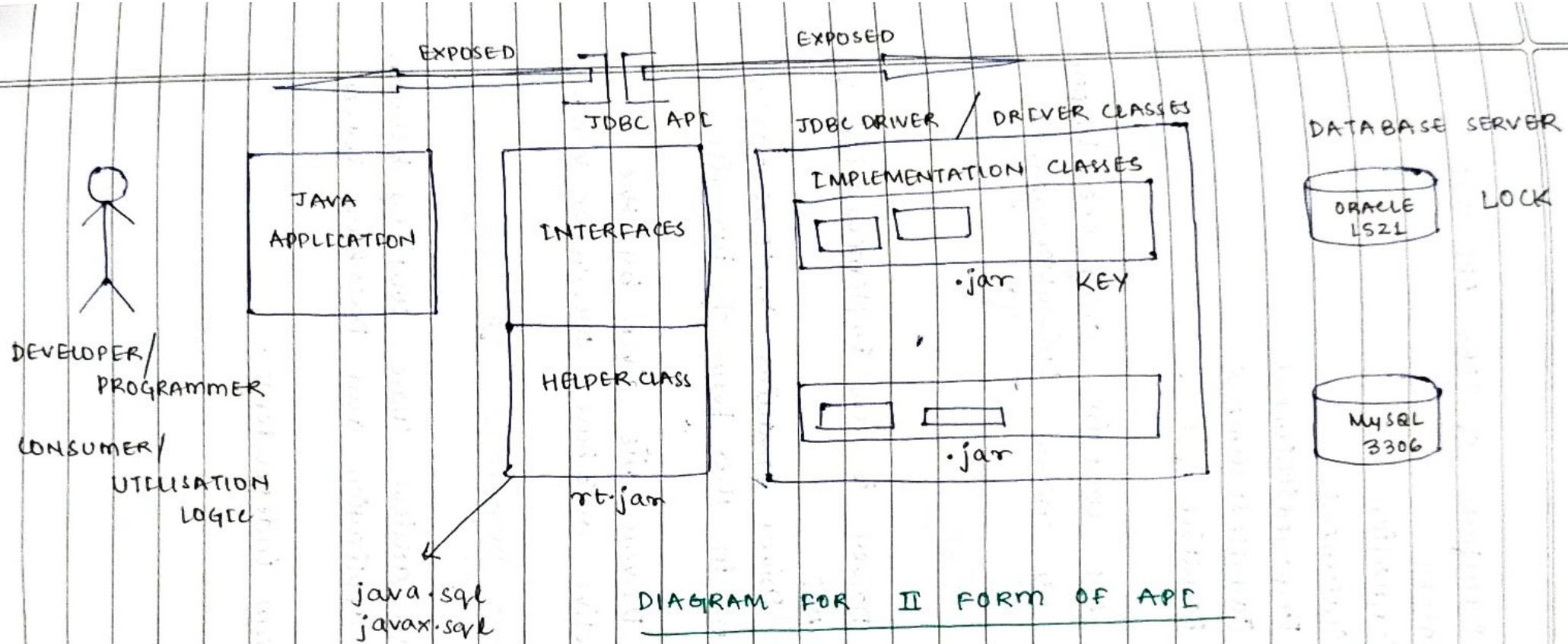


DIAGRAM FOR II FORM OF API

[NOTE :- FOR US TO COMMUNICATE WITH ANY DATA BASE SERVER AND TO PERFORM ANY DATABASE OPERATIONS. THEIR ARE TWO COMPONENTS REQUIRED MANDATORLY

(i) JDBC API
(ii) JDBC DRIVER]

→ The Interfaces of JDBC API are
(i) Driver
(ii) Connection
(iii) Statement
(iv) PreparedStatement
(v) CallableStatement
(vi) ResultSet, etc.

→ JDBC API contains only one helper class by name DriverManager.

JDBC DRIVER

→ JDBC Driver is an implementation of JDBC API.

→ JDBC Driver contains implementation classes in the form of jar file.

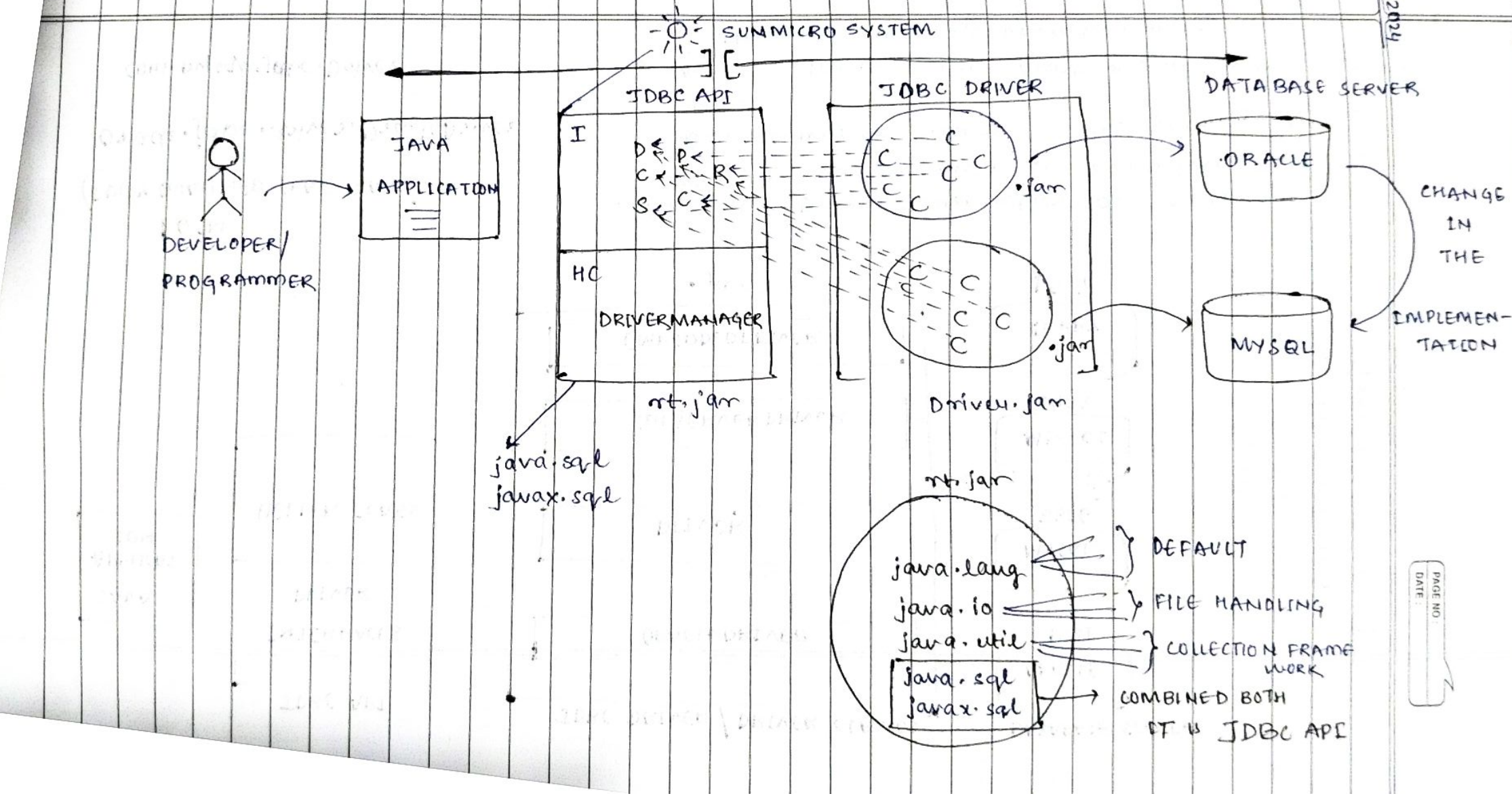
→ JDBC Drivers are given by the respective data base servers or vendors

~~APPLICATI~~ ADVANTAGES OF JDBC

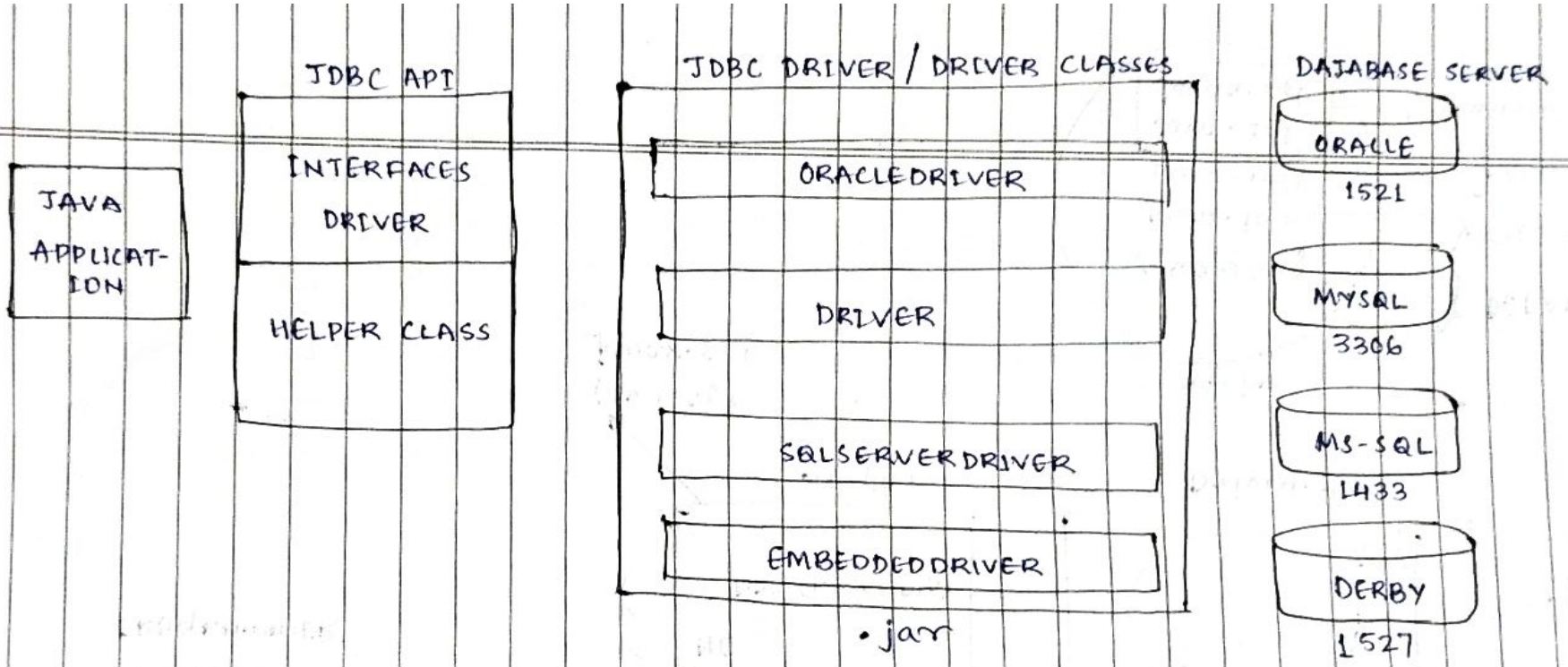
→ we can achieve loose coupling between java application and Data Base server.

→ Platform independent.

17/05/2024



PAGE NO.
DATE :



DIFFERENT DRIVER CLASSES RESPECTIVE DATA BASE SERVERS

FQCN
(FULLY QUALIFIED CLASS NAME)

Oracle.jdbc.driver.OracleDriver

com.mysql.jdbc.Driver

→ All the Driver classes which are part of JDBC Driver or given by respective Data Base server or vendor in the form of jar file.

→ All the Driver classes must mandatorily implements java.sql.Driver interface.

8/05/2024

CLASS LOADING

→ Loading a dot class file into the JVM memory is known as class loading.

→ public class Pen

{

==

==

}

saved as → Pen.java —Javac (compiled)→ Pen.class —stored in→ JVM MEMORY for execution.

→ A class is generally loaded in two different ways -

- By calling any of the members of a class which can either be a constructor, method, variable, blocks, etc.

→ By using a static method called forName() method which is declared inside a class by name class itself and that class is present in java.lang.package.

→ Whenever we use this method it throws a checked Exception called ClassNotFound-Exception.

Syntax :-

Class.forName("FQCN");

↓ java.lang ↓ static method

Example –

```
package org.jsp.loadApp;

public class Trainer{
  S.O.P(" J2EE Trainer is scolding students
        without any reason");
  }
}
```

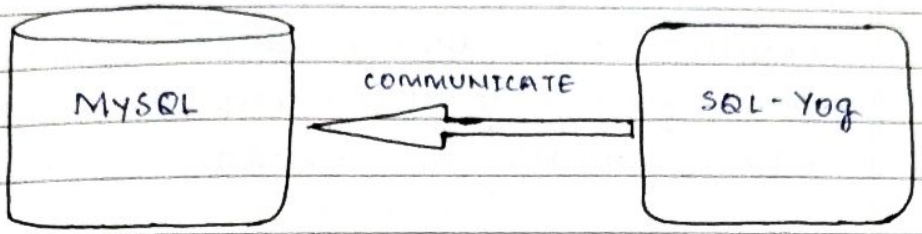
```
package org.jsp.loadApp;

public class Test {
    public static void main (String[] args)
    {
      try {
        Class.forName ("org.jsp.loadApp.Trainer");
      }
      catch ( ClassNotFoundException e) {
          e.printStackTrace();
      }
    }
}
```

present in java.lang.package

20/05/2024

PAGE NO :
DATE :

SERVER INSTALLATION



JDBC

Java Database Connectivity is a specification which is given in the form of abstraction API to achieve loose coupling between java application and the database server.

Standard Steps of JDBC (JDB)

There are 6 standard steps present w.r.t. JDBC -

(i) Load and Register the driver
(ii) Establish a connection between java application and the data server.
(iii) Create a statement or a platform.
(iv) Execute the sql queries or SQL statements.
(v) Process the resultant data (optional)
(vi) close all the costly resources.

21/05/2024

1st STEP:

LOAD AND REGISTER THE DRIVER

In this step we need to load and register the driver classes which are given by the respective data base servers or vendors.

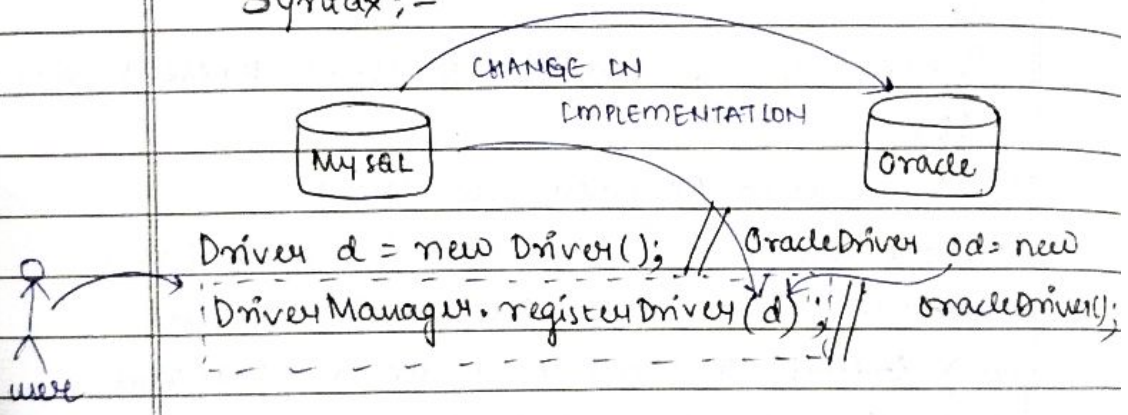
→ There are two different ways in which we can load and register the driver class -

(i) 1st WAY:

→ By creating an object of driver class then, register that driver class with driverManager by using a static method called

registerDriver()

Syntax;-



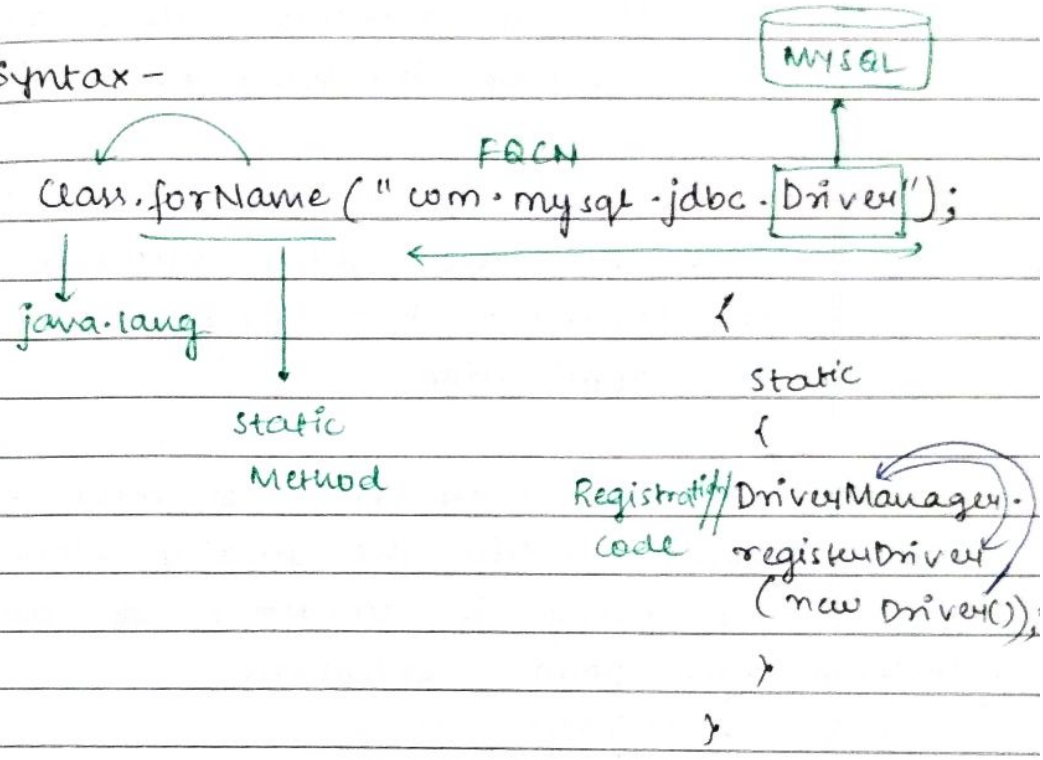
NOTE-

The First way of loading & Registering a driver class is not a good practice since it causes tight coupling b/w Java appln and the database server

(ii) 2nd WAY:

→ By using a static method called forName() which is declared inside a class by name class itself.
Whenever we try to use this method it ~~throug~~ throws a checked exception called classNotFoundException.

Syntax -



★
```
package org.jsp.demoApp;
public class Test
{
    public static void main(String[] args)
    {
     try{
        Class.forName("com.mysql.jdbc.Driver");
        S.O.P("Driver class is loaded and
                Registered!");
     } catch (ClassNotFoundException e) {
            e.printStackTrace();
     }
}
```

PAGE NO:
DATE:

COSTLY RESOURCES

Resources which makes use of System properties in the form of Stream are known as costly resources.

Example -

Scanner, PrintWriter etc.

All the ~~methods~~ classes which are using File handler.

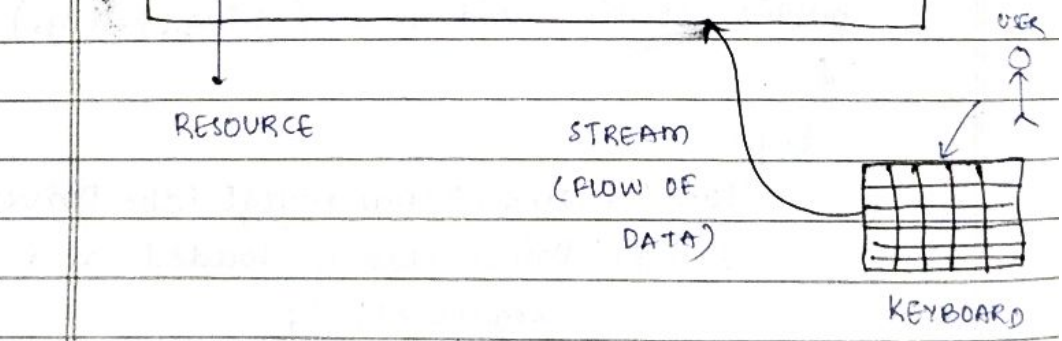
→ It is always a good pratice to close all the costly resources since it decreases the performance of an application.

→ ** All the costly resources must be closed within the finally block by using if condition to avoid null pointer exception.

05/2024

SYSTEM PROPERTIES { MEMORY, CPU TIME

```
Scanner sc = new Scanner(System.in);
sc.next();
sc.close();
```



→ All the interfaces of JDBC API are considered to be costly resources w.r.t JDBC.

→ The Interfaces of JDBC API are -

- Driver
- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet, etc....

} Costly Resources

SPECIFICATIONS OF JDBC

Totally there are three different specifications present w.r.t. jdbc -

(i) All the Driver classes must contain one static block in it.

(ii) All the Driver classes must mandatorily implements java.sql.Driver Interface which is a part of JDBC API.

(iii) All the Driver classes must mandatorily be registered with driver manager by using a static method called registerDriver().

II

public class Driver implements java.sql.Driver

↓ Interface of JDBC API

{

static

{ I

DriverManager.registerDriver(new Driver()); III

}

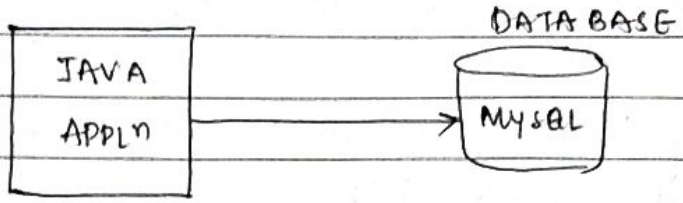
}

IInd STEP:

ESTABLISH A CONNECTION BETWEEN JAVA APPLICATION AND DATABASE SERVER.

OR

ESTABLISH A CONNECTION WITH THE DATABASE SERVER.



→ In this we need to establish a connection between java applⁿ and the data base server.

→ In order to establish a connection between java applⁿ and the data base server JDBC API provides one interface by name connection.

```
java.sql.Connection con = DriverManager.getConnection("URL");
```

- `java.sql.Connection` → Interface of JDBC API
- `DriverManager` → Helper class of JDBC API
- `getConnection("URL")` → static Factory Helper Method

return new ConnectionImpl();

return new OracleConnectionImpl();

JDBC URL – "jdbc:mysql://localhost:3306?user=root&password=admin"
FOR MYSQL

2nd STEP :- java.sql.Connection con = DriverManager.getConnection("URL")

Connection → Interface → JDBC API → Implm class Object of CONNECTION Interface

DriverManager → Helper class → JDBC API

getConnection → { static Factory / Helper Method

JDBC Driver → MYSQL → Driver.Jar

Implementation class
ConnectionImpl
MYSQL
@override
m1()
{
}

Implm class Object of CONNECTION Interface

Create & return

return: new ConnectionImpl();

return: new OracleConnectionImpl();
}

Connection con = new ConnectionImpl();
= new OracleConnectionImpl();

Connection (I)

MYSQL → ConnectionImpl

Oracle → OracleConnectionImpl

public class Projector implements Remote
{
freeze()
{
}
}

23/05/2024

getConnection() -

→ getConnection() method is used to create and return the implementation class Object of connection Interface based on the URL.

→ Hence the return type for getConnection() method is Connection Interface.

→ There are three overloaded variants of getConnection() method present -

Overloading
- (i) getConnection (String URL)
- (ii) getConnection (String URL, Properties info)
- (iii) getConnection (String URL, String user, String password)

This is the best real time example for method overloading.
Because getConnection() method is overloaded in DriverManager class.

→ Whenever we try to use either of these overloaded variants of getConnection method it throws a checked exception called SQLException.

JDBC URL for MYSQL

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

PAGE NO:
DATE:

java.sql.Connection

Interface of

→ It is an ^JDBC of API and the implementations are provided by the respective database servers as a part of JDBC Drivers.

→ Since Connection is an Interface which is considered to be the costly resources w.r.t. jdbc it is always a good practice to close connection interface.

java.sql.DriverManager

↓

can be called as Factory class/Helper class.

→ It is a helper class of JDBC API which contains two static methods in it.

- ↳ getConnection()
- ↳ registerDriver()

FACTORY / HELPER METHOD

→ It is used to create and return the implementation class object of interface.

```
package org.jsp.demoApp;
import java.sql.*;

public class Test
{
    p.s.v.m(S[] a){

        Connection con = null;
```

```
try {
    Class.forName ("com.mysql.jdbc.Driver");
    S.O.P("Driver class is loaded and
          Registered!!");
    con = DriverManager.getConnection
      ("jdbc:mysql://localhost:3306?
       user=root & password=admin");
     S.O.P ("connection established
           with the database server");
   }
catch (ClassNotFoundException | SQLExcep-
        tion e) {
      e.printStackTrace();
 }
finally
 {
    if(con!=null)
    {
      try{
        con.close();
      }
      catch (SQLException e) {
         e.printStackTrace();
      }
      S.O.P. ("closed all the coming
           resources");
    }
  }
 }
}
```

24/09/2024

IIIrd STEP:

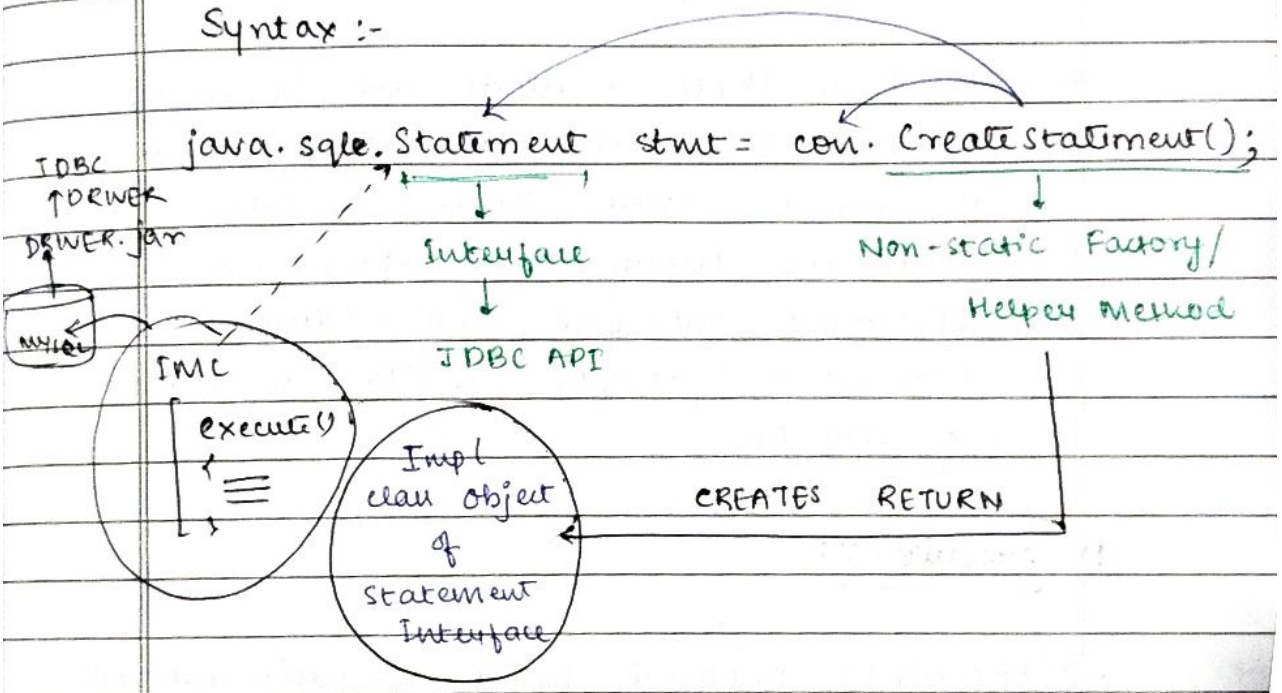
CREATE A STATEMENT OR A PLATFORM

→ We need to create statement or a platform to execute the SQL queries or sql statements.

→ A platform can be created either by using statement or prepared statement or callable statement Interface which is a part of JDBC API.

Syntax :-

```
java.sql.Statement stmt = con.CreateStatement();
```



→ Create statement method is a non-static factory/helper method which is used to create and return the implementation class object of statement Interface.

→ Hence the return type of create statement method is statement Interface.

IV TH STEP :

EXECUTING THE SQL QUERIES OR SQL STATEMENTS

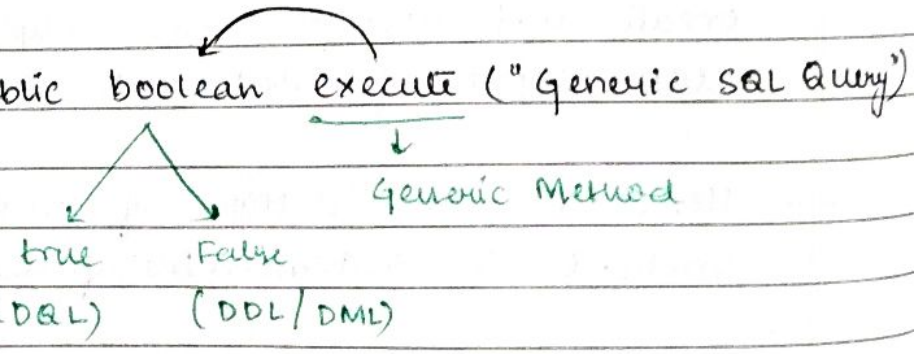
→ In order to execute the different set sql queries or sql statements totally there are three methods available.

(i) executeMethod()
(ii) executeUpdate()
(iii) executeQuery()

→ All these three methods are basically declared in statement interface but it can be used either w.r.t. statement interface or prepared statement interface or callable statement interface which is a part of JDBC API.

(1) execute() :-

→ execute() method is a generic method which is used to execute any type of SQL queries or SQL statements.



PAGE NO.
DATE

→ the return type of execute method is boolean.

→ execute() returns boolean True value in case of DQL and it returns boolean False value in case of (DDL/DML).

• What is the outcome of any executed DML Query?

→ The outcome of DML is 0 - n Integer value which gives the total number of records affected in the table.

28/05/2004

id	name	perc
101	PRERNA	98.34
102	CHINNI	70.09 ← 1 Record deleted
103	ADITYA	99.00 ← 1 Record updated
104	TEJAS	89.43 ← 1 record inserted.

3

DML (Data Manipulation Language) query

insert | into btm.student values (104, 'TEJAS', 89.4
delete | from btm.student where id = 102
update | btm.student set name = 'PINKY'
where id = 102
delete from btm.student where id = 105 // 0

(ii) executeUpdate() :-

executeUpdate() is a specialized method which is used to execute only DML queries and or DML statements.

METHOD SIGNATURE -

- public int executeUpdate("only DML query")

specialized method

→ The outcome of DML is 0 - n Integer value which gives the total number of records affected in the table. Hence the return type of execute update method is Integer.

→ whenever we try to execute DDL or DDL query by using execute update method it throws SQLException.

CODE TO INSERT ONE SINGLE RECORD INTO DATABASE SERVER BY USING STATEMENT INTERFACE -

```
★ package org.jsp.insertApp;
import java.sql.*;
public class InsertDemo
{
    public static void main (String[] args)
    {
        Connection con = null;
        Statement stmt = null;

        String qry = "insert into btm.
        student values(101, 'Prerna', 98.45);
                    // DML

        try {
            Class.forName("com.mysql.jdbc.
                Driver");
```

```
S.O.P("Driver class is loaded and
        registered");
con = DriverManager.getConnection("jdbc:
    mysql://localhost:3306?user=root&
    password=admin");
S.O.P("Connection Established with the
        database Server!!");
Stmt = con.createStatement();
S.O.P("Platform Created!!");
stmt.executeUpdate(qry);
S.O.P("Record Inserted!!");
}
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();
        }
    }
S.O.P("Closed all the costly resources");
}
```

29/05/2024

CODE TO UPDATE A RECORD IN THE DATABASE SERVER BY USING STATEMENT INTERFACE

```
package org.jsp.UpdateApp;
Import java.sql.*;
public class UpdateDemo {

    public static void main(String[] args){

        Connection con = null;
        Statement stmt = null;

        String qry = ("update btm.student
            set name = 'Prerna' where
                id = 102"); //DML query

        try{
            Class.forName("com.mysql.jdbc.
                Driver");
            S.O.P("Driver class is loaded and
                registered");
            con = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306
                ?user=root&password=admin");

            Stmt = con.createStatement();
            stmt.executeUpdate(qry);
            S.O.P("Record ~~Deleted~~ 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;
    }
  }
}
```

CODE TO DELETE A RECORD IN THE DATABASE SERVER BY USING STATEMENT INTERFACE

```
package org.jsp.DeleteApp;
import java.sql.*;
public class DeleteDemo {

  public static void main(String[] args){

    Connection con = null;
    Statement stmt = null;

    String qry = ("Delete from btm.student
                  where id = 101"); // DML query
```

```
try {
 Class.forName ("com.mysql.jdbc.Driver");

  S.O.P("Driver class is loaded and
          registered");
  con = DriverManager.getConnection
  ("jdbc:mysql://localhost:3306?
    user=root&password=admin");

   stmt = con.createStatement();
   con.executeUpdate(qry);
   S.O.P("Record Deleted");
}
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();
    }
  }
```

30/05/2024

CODE TO INSERT MULTIPLE RECORDS INTO THE DATABASE SERVER BY USING STATEMENT INTERFACE

```
Stmt.executeUpdate(qry1); // compilation + Execution of the query.
```

In case of statement interface compilation takes place each time along with the execution.

Statement

300 Records → 300 Queries → 300C + 300E → 600 tasks

Records to be inserted

Compilation

Execution

Prepared Statement

300 Records → 1 Query → 1 com + 300E → 301 tasks

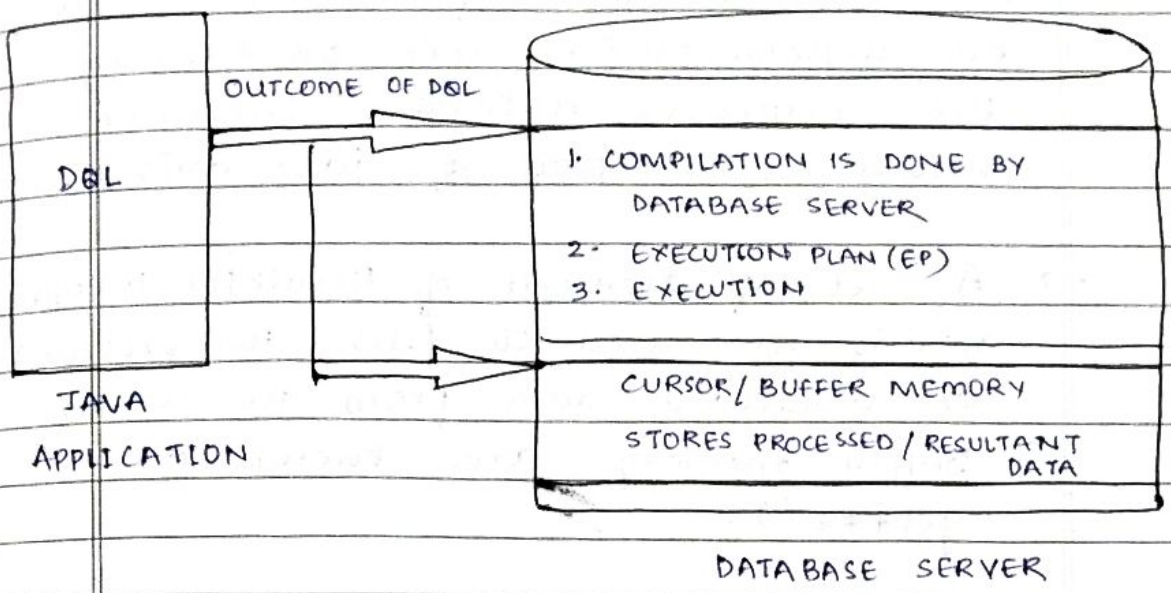
Records to be inserted

compilation

Execution

PAGE NO :
DATE :

FETCHING THE DATA FROM THE DATABASE SERVER



→ Whenever we execute DQL queries or DQL statement we may get a result which is reffered as processed or resultant data.

→ The processed/ Resultant data which is stored in the cursor or buffer memory. can be fetched with the help of ResultSet interface.

→ The structure of cursor/ buffer memory may differ from the structure of Table.

id	name	perc
101	Tej	89.54
102	Peru	98.45

DQL = Select * from btm.student where id = 102;

O/C

PROCESSED/ RESULTANT DATA

	1	2	3 → Column Number
Data Type w.r.t java	(int) id	(string) name	(double) perc
	102	Peru	98.45

cursor/ buffer memory

ResultSet (I)

Fetch → get***()
check → next()
absolute() → check particular record is present

Fetched

Java.sql.ResultSet

→ It is an Interface of JDBC API and the implementations are provided by the respective database servers or vendor as a part of jdbc drivers.

→ A set of Methods of ResultSet Interface which are used to fetch the processed or resultant data from the cursor/buffer memory are known as get***().

31/05/2024 (I) getXXX() :- (To fetch)

METHOD SIGNATURE -

(i) public *** get***(int column number)

OR

(ii) public *** get***(String column name)

refers to return type

refers to data type

(i) To Fetch id(int)

- public int getInt(1); // 102

OR

- public int getInt("id"); // 102

(ii) To Fetch name(String)

- public String getString(2); // Peru

OR

- public String getString("name"); // Peru

PAGE NO :
DATE :

(iii) To Fetch perc (double)

- public double getDouble (3); // 98.45
  OR
- public double getDouble ("perc"); // 98.45

NOTE -
By default ResultSet Interface cursor will not be pointing to any record in the cursor/ buffer memory.

(2) next() :- (To check)

→ next() method is used to check weather the next record is present in the cursor/ buffer memory. or not. and it returns a boolean value called true or false but not the record.

METHOD SIGNATURE-

```
public Boolean next();
```

true or false — check

→ we can use next method when even ~~they~~ their are minimum number of records present in the cursor/ buffer Memory.

(3) absolute() :- (To check the particular record)

→ absolute() is used to check weather a particular record is present in the cursor or buffer memory or not based on a parameter called integer row number, and

it returns a boolean value called true or false but not the record.

→ absolute() method can be used whenever their are n. no. of records present in the cursor / buffer memory.

Method SIGNATURE –

public boolean absolute (int row number);

boolean → true, false
absolute → To check a particular record

(iii) executeQuery() :–

→ executeQuery() method is a specialized method which is used to execute only DQL Queries or DQL statements.

- public ResultSet executeQuery ("only DQL query");

executeQuery → specialized Method

Return Type of executeQuery() is ResultSet (I) because executeQuery() method specialized method which only used in DQL query or statement which gives processed / resultand Data which is stored in cursor/buffer memory which can be fetched by methods present in which are - ResultSet Interface.

→ The outcome of DQL is processed or resultant data which is stored in the cursor or buffer memory which can be fetched with the help of result set interface. Hence the return type for executeQuery() method is ResultSet interface.

→ whenever we try to execute DML or DDL query by using executeQuery() method then it throws SQLException.

13/06/2024 PLACEHOLDER

Placeholder is a parameter which holds dynamic value at the run time by the user.

In case of JDBC placeholder is represented by "?" symbol.

Declaration of placeholder in the query.

```
String qry = " insert into btm.Student values
             (101, 'sonu', 60.14)";
String qry1 = " insert into btm.student
              value (?,?,?)";
String qry2 = " update btm.Student set
              name = ? where id = ?";
String qry3 = " delete from btm.student
              where id = ?";
String qry4 = " select * from btm.student
              where id = ?";
```

RULES TO SET THE DATA FOR PLACEHOLDER

Totally their are three different rules to set the data for place holder.

(i) we need to set the data for place holder before the execution.

(ii) The number of data must exactly match the no. of placeholders.

(iii) we need to set the data for place holder by using setXXX();

```
String qry = "insert into btm.student
             values (?, ?, ?)";
```

(1, 2, 3 → placeholder Number/index)

id (int) name (string) price (double) → datatype

METHOD SIGNATURE -

```
public void set***(int placeholder no/index,
                   ***);
```

(set*** → Datatype; *** → Type of Data)

Set before execution:

```
pstmt.setInt(1, 102);
pstmt.setString(2, "Bandbaja");
pstmt.setDouble(3, 96.01);
```

```
pstmt.executeUpdate(); // execution
```

PAGE NO:
DATE:

java.sql.PreparedStatment

↓

Interface

↓

JDBC API

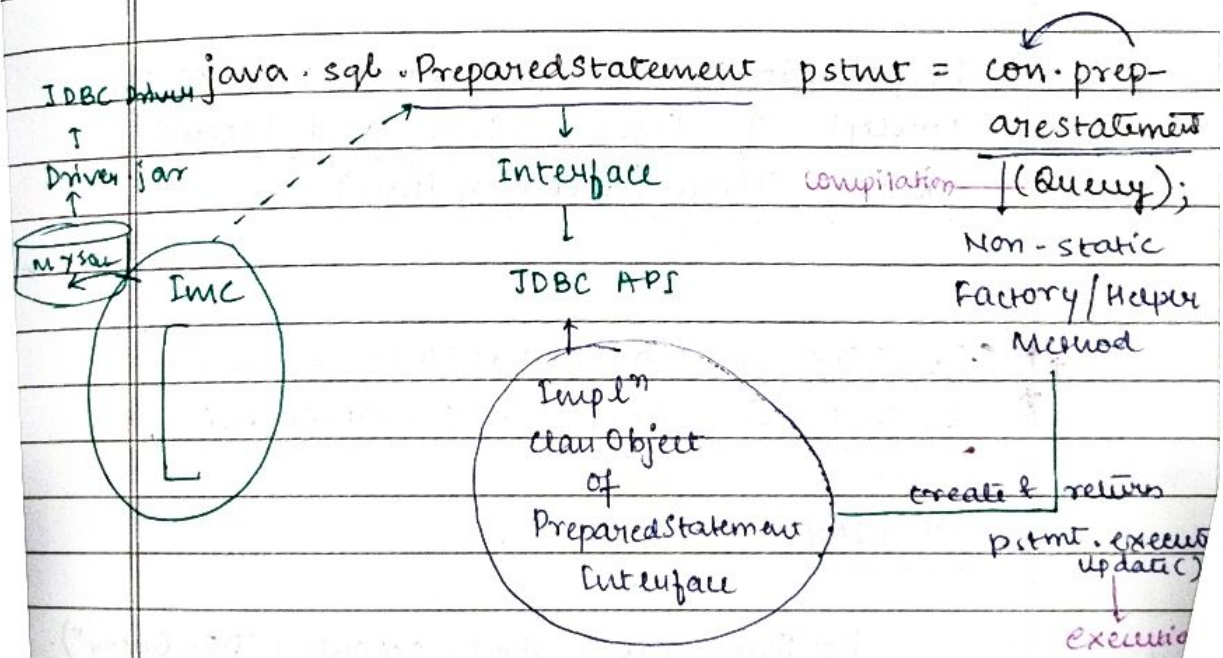
→ PreparedStatement is an Interface which is a part of JDBC API & the implementation are provided by respective database server or vender as a part of JDBC driver.

→ PreparedStatment Interface is a sub Interface of Statement Interface.

→ PreparedStatement Interface supports the concept of placeholder to take dynamic values at the runtime by the users.

05/06/2024

SYNTAX for PreparedStatement –



prepareStatement()

→ prepareStatement() method is a non-static Factory / Helper Method which is used to create and return the implementation class object of prepared statement Interface.
Hence the return type of prepared statement method is PreparedStatement Interface.

→ PreparedStatement Interface is also known as pre-compiled statement.

→ In case of PreparedStatement Interface the query compilation takes place at the time of implementation class object creation of PreparedStatement Interface.
Hence we called PreparedStatement Interface as pre-compiled statement.

→ PreparedStatement Interface supports the concept of Compile Once and Execute Many Times (Execution Plan)

** CREATION OF IMPLEMENTATION CLASS OBJECT OF RESULT SET INTERFACE

1st WAY:-

```
boolean val = stmt.execute("DQL Query");
                (True)
if (val)
{
```

java.sql.ResultSet rs = stmt.getResultSet();

rs.next();

IMC [ResultSetImpl]

[next()]

2nd WAY :-

java.sql.ResultSet rs = stmt.executeQuery ("DQL");

ResultSet — C/B memory P/R data — Outcome

CODE TO INSERT MULTIPLE RECORDS INTO THE DATABASE SERVER BY USING PREPARED STATEMENT INTERFACE ALONG WITH PLACE HOLDER.

```
package org.jsp.insertpreApp;
import java.sql.*;
public class InsertPreDemo {
    p.s.v.m (S[] a) {

        Connection con = null;
        PreparedStatement pstmt = null;
        String qry = "insert into btm.student
                value (?,?,?)";
        try {
            Class.forName("com.mysql.jdbc.
                        Driver");
            con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306?user=root&
                        password=admin");
```

```
pstmt = con.prepareStatement(qry); //compile
// Set the DATA for Place holder.
pstmt.setInt(1, 101);
pstmt.setString(2, "Peru");
pstmt.setDouble(3, 94.44");
// Execute the query
pstmt.executeUpdate(); // Execution - 1
pstmt.setInt(1, 102);
pstmt.setString(2," Tej ");
pstmt.setDouble(3, 90.19);
pstmt.executeUpdate(); // Execution - 2
pstmt.setInt(1, 103);
pstmt.setString(2, "Palat");
pstmt.setDouble(3, 91.11);
pstmt.executeUpdate(); // Execution - 3
S.O.P("Records inserted");
}
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();
```

PAGE NO:
DATE:

06/06/2024

CODE TO FETCH ALL THE RECORDS FROM CURSOR/BUFFER MEMORY BY USING RESULTSET INTERFACE

```
package org.jsp.fetchApp;
import java.sql.*;
public class FetchDemo{

    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");
            con = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306?user=
              root&password=admin");
            pstmt = con.prepareStatement(qry);
            rs = pstmt.executeQuery();

            while(rs.next()){
                int sid = rs.getInt(1);
                String sname = rs.getString("name");
                double sperc = rs.getDouble(3);
                S.O.P("Student id is "+sid);
                S.O.P("Student name is "+sname);
                S.O.P("Student perce is"+sperc);
                S.O.P("-----------");
            }
        }
        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();
            }
        }
    }
}
```

PAGE NO :
DATE :

06/2024

CODE TO FETCH A PARTICULAR RECORD FROM A CURSOR OR BUFFER MEMORY WHERE ID = ? (PLACEHOLDER)

```
package org.jsp.fetchidApp;
import java.sql.*;
import java.util.Scanner;

public class fetchdemo{

    public static void main(String[] args){

        Scanner sc = new scanner(System.in);
          S.O.P("Enter id");
          int sid = sc.nextInt();

        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;

        String qry = " select * from btm.
                      student where id=?";

        try{
         Class.forName("com.mysql.jdbc.
                        Driver");
         con = DriverManager.getConnection
             ("jdbc:mysql://localhost:3306?
              user=root & password=admin");
         Pstmt = con.prepareStatement(qry);
         pstmt=setInt(1,sid);
         rs = pstmt.executeQuery(qry);
        }
```

```
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();
        }
    }
}
}
}
```

PAGE NO :
DATE :

CODE TO FETCH A PARTICULAR RECORD FROM THE CURSOR/BUFFER MEMORY WHERE NAME IS EQUAL TO ? (PLACEHOLDER)

```
package org.jsp.loginApp;
import java.sql.~~conn~~ *;
import java.util.Scanner;

Public clas login_Demo {

    public static void main (string[] args) {

        Scanner sc = new Scanner (System.in)
        S.O.P ("Enter name");
        String nm = sc.next();
        ~~S.O.P~~ (

        Connection con = null;
        PreparedStatement pstmt = null;
        Resultset rs = null;

        String qry = " select * from btm.
                     student where name = ?";

        try {
        Clas.forName ("com.mysql.jdbc.
                       Driver");
        con = DriverManager.getconnection
        ("jdbc:mysql://localhost:3306?
         user = root & password = admin");

        Pstmt = con.preparestatement (qry);
        pstmt = setstring (1, nm);
        rs = pstmt.executeQuery();
        }
```

```
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();
        }
    }
}
}
}
}
```

10/06/2024

java.sql.CallableStatement -

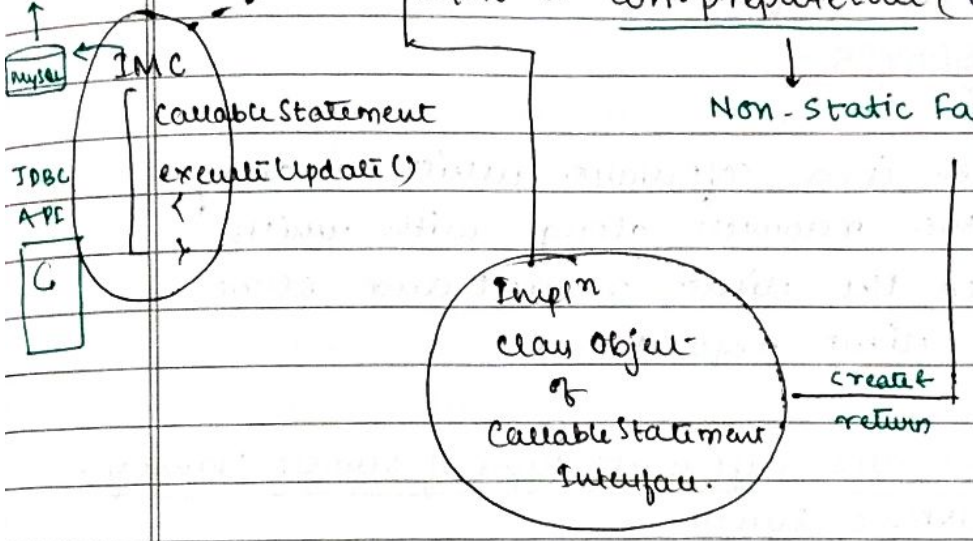
→ It is an interface of JDBC API and the implementation are provided by the respective database server or vendor as a part of JDBC Driver.

SYNTAX -

java.sql.CallableStatement

↳ Interface → JDBC API

CStmt = con.prepareCall(Query);



→ prepareCall() is a non-static / factory / Helper method which is used to create and return the implementation class object of CallableStatement Interface.
Hence, the return type of prepareCall() is callableStatement interface.

→ CallableStatement Interface is used to call the procedures which are saved in the database (stored procedure).

PAGE NO:
DATE:

SERVLETS

→ Servelet is a server side Java Program which peforms all the three different type of logics such as -

(i) Presentation logic
(ii) persistance logic
(iii) Buimess logic

Along with which we it proces the client request and server the client request.

11/06/2024 SERVER -

Server is a software which manages all the resources along with which proces the client request and serve the client request.

There are different type of server Namely -

→ DataBase server
→ Application Server
→ Web Server

1. DATABASE SERVER - Database ~~server~~ server is used to deal with the data.
Ex - Oracle, My-SQL, Derby, MongoDB Sybase etc.

2. APPLICATION SERVER - Application server is used to execute a dynamic application or a realtime application.

Dynamic Application / Real time ~~Ap~~ Application -

An application which performs all the 3 different types of logic such as presentation logic, ~~Persistance~~, Persistance logic & Business logic is known as Dynamic / Real Time Application.

3. WEB SERVER —

Web server is used to execute only web application.

Ex- Apache - Tomcat server & Oracle Glassfish.

DEPLOYMENT

Making all the resources available to the server is known as deployment.

There are 2 different types of deployment present namely -

Apache - Tomcat Servers comes with 2 different variant -

(1) .zip

(2) .exe

12/06/2024

PORT NUMBER

→ Port Number is the one which helps us to get connected to a particular server.

→ Default Port Number for Apache - Tomcat server is - 8080