Advance Java

Jar file

Jar → Java Archive

→ It is same as zip file format.

→ It contains .java file, .class file, .xml
.config file

.config files

→ It is used for configuration purpose

① .xml file

→ xml → extended mark of Language.

→ This xml file is used to configure the programm by using custom task or user define task.

② : property file

→ It is used to configure the resources by giving the data in key & value pair.

ArrayList → arr.length * 3/2;

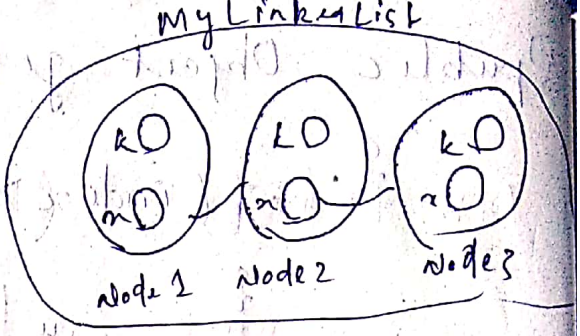
Vector → arr.length * 2;

```
class X
{
    psvm(---)
    {
        MyArrayList l = new MyArrayList();
        l.add("java");
        l.add('z');
        l.add(10);
        l.add(20);
    }

class MyArrayList
{
    Object [] arr=new Object [3];
    int n=0;
① public boolean add(Object ab)
    {
        if(n==arr.length)
        {
            increseSize();
        }
        arr[n]=ab;
        return true;
    }

② public void increseSize()
    {
        Object [] temp=new Object [arr.length*2];
        for(int i=0; i<arr.length; i++)
        {
            temp[i]=arr[i];
        }
        arr=temp;
```

③ public Object get(int index)
{
if(index >= n)
{
SOP("No such element");
return null;
}
return arr[index];
}

④
public int size()
{
return n;
}

⑤
public String toString()
{
if(n==0)
{
String temp = "[";
}
else
{
String temp = "[" + arr[0];
}
for(int i=1; i<n; i++)
{
temp = temp + " " + arr[i];
}
temp = temp + "]";
}




```
class MyLinkedList
{   int count=0;
private class Node
{
    Object k;
    Node x;
    Node (Object m)
    {
        k = m;  count++;
    }
}
Node start;   int count=0;
Node last;
```

① → `public boolean add (Object ab)`

```
{
    if (start == null)
    {
        start = new Node(ab);
        last = start; return true;
    }
    last.x = new Node(ab);
    last = last.x;
    return true;
}
```

② → `public int size()`

```
{
    return count;
}
```

```
public Object get(int index)
{
    if(index >= count)
    {
        SOP("No element is present");
        return null;
    }
    Node temp = start;
    for(int i=0; i<index; i++)
    {
        temp = temp.x;
    }
    return temp.k;
}
public String toString()
{
    String k = ""; Node tp = start;
    if(count == 0)
    {
        k = "[";
    }
    else
    {
        k = "[" + tp.k;
    }
    for(int i=1; i< count; i++)
    {
        tp = tp.x;
        k = k + " " + tp.k;
    }
    k = k + "]";
    return k;
}
```

jar file

Right click (package/project) → Export → jar format

→ give destination → file name → save

* Import internally
  Import externally
  ] 2 ways to import

* change Workspace → Reopen eclipse

→ create new project (java) → name → create

→ within src create package → then create new class

Externally

(Project (Right click) → properties → java BuildPath

→ add external jars → search jar file then click ok to import.)

~~Internally~~

→ But if we delete that jar file by default then that particular class will not available. So we if is recomended to import internally.

Internally

~~Project~~ Create a new folder in project → project folder (lib)

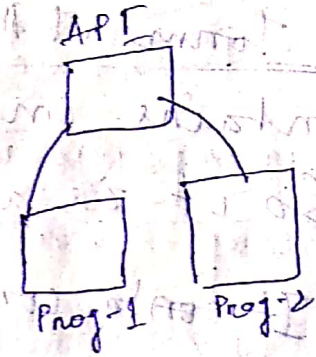
then copy jar file within lib folder.

→ Then go for,
Project → Propoties → java Build Path → remove default jar file ⇒ click on import jARs then it will available for your project.

API

Application Programming Interface

→ It is connecting ~~between~~ ^ ~~two~~ programs in a loosly coupled manner.

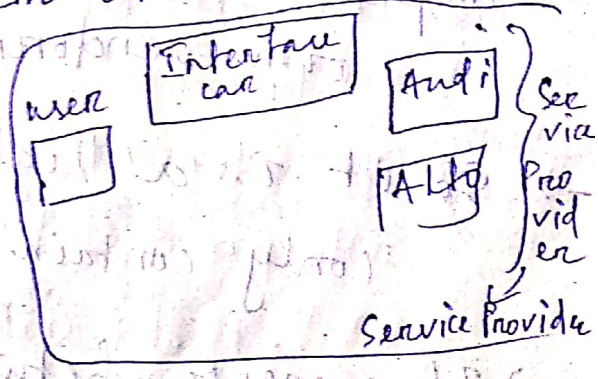


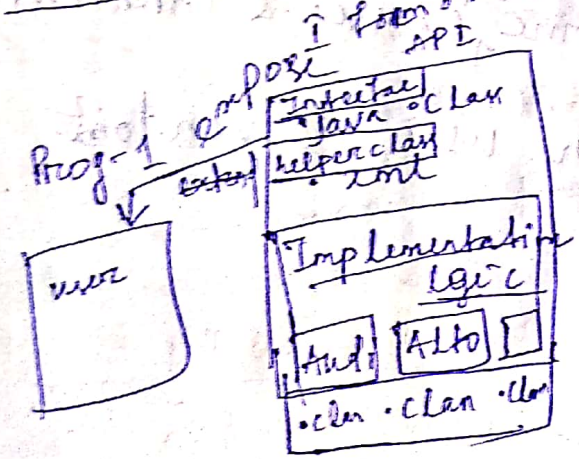
→ There is a 2 type of APIs.

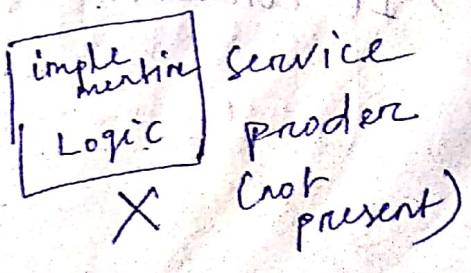
(i) I form of API
(ii) II form of API

→ If the API is developed by Java then it is given in the form of Java file.

I form of API







(Fig)

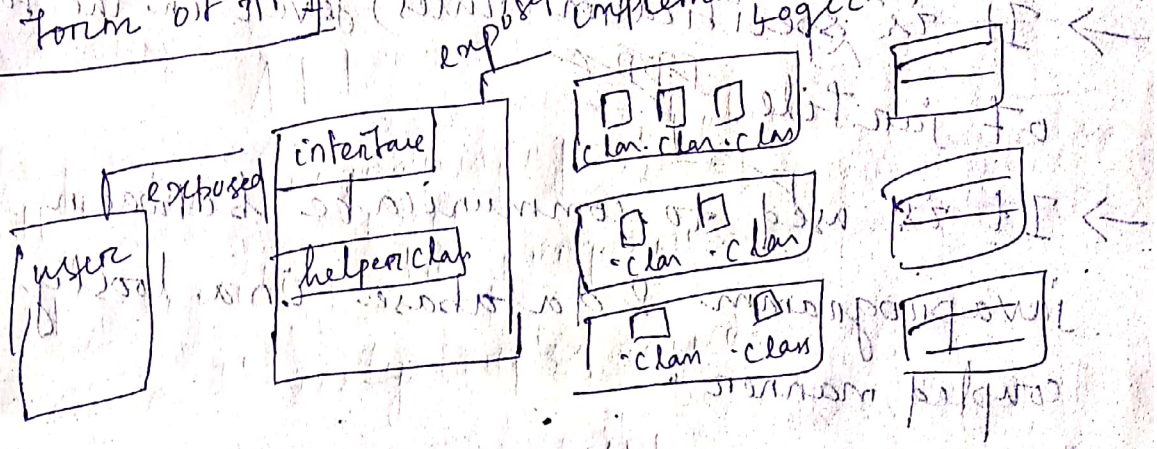
I form of API

→ In I form of API it cointains implementation logic

→ As it cointains implemention logic then I form of API are heavy.

→ I form of API is exposed towards the user programs, not towards venders.

I form of API



→ In I forms of API implemention logic are not present, The implementation logics are provided by respective service provider or vendors

→ As the API doesnot contain implemenbin logic it is light weight

→ It is exposed towards user side as well as service providers or vendor side

Implemention classes are also known as drievers

ODBC

→ It is outdated
→ present in every computer
→ Windows provided ODBC.

JDBC

→ Java Database Connectivity
→ It is specification (rules) given in the form of jar file
→ It is used to communicate between javaprogramm & database in a loosly coupled manner.

JDBC API

→ It is already present in java.sql package
→ It contains some interfaces and helper classes

interfaces
(I) Connection
(II) Driver
(III) Statement
(IV) PreparedStatement
(V) CallableStatement
(VI) ResultSet
(VII) ResultSetMetaData
DatabaseMetaData

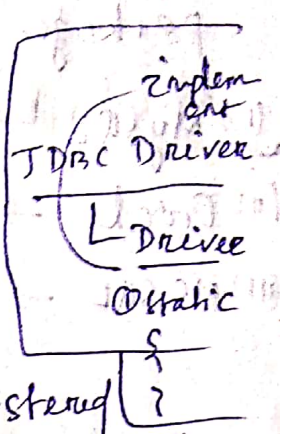
helperclass
(I) DriverManager

JDBC Driver

→ Their are implementation logic provided by service provider & vendors.

Rules for providing JDBC driver

(i) The driver class of JDBC driver must implement driver interface of java.sql package.



(ii) Every Driver class must get registered with driver manager (which is a helper class present in JDBC API)

(iii) Every Driver class must contain a static block in it & the code which is used get register with driver manager should be placed inside this static block.

Steps of JDBC

step-I → Load & register the driver class.

step-II → establish the connection between Java & Database.

step-III → create a platform to execute the Quary.

step-IV → Execute the Quary

step-V → fetch the result (optional)

step-VI → close all the costly resources.

Load and Register Driver class

Driver class for different database

→ Driver classname for with package name are,

(I) MySQL → com.mysql.jdbc.Driver (3306)

(II) Oracle → Oracle.jdbc.driver.OracleDriver.

(III) MS SQL → com.microsoft.sqlserver.jdbc.SQLServerDriver (1433)

(IV) Derby → org.apache.derby.jdbc.EmbeddedDriver

* There are two ways to load & register the driver class.

(1) In this way of registering we create the object of Driver & register the Driver through registerDriver().

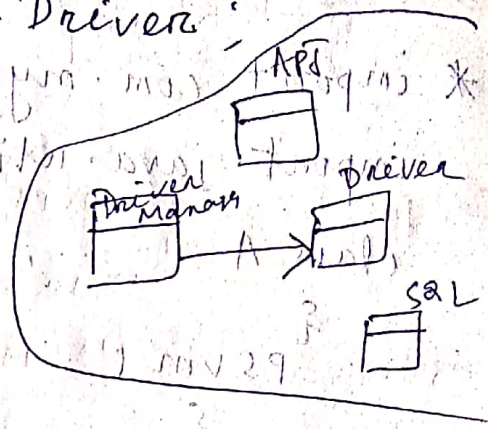
registerDriver()

→ public static void registerDriver (Driver d)
{
}

→ This method is a method of Driver manager class.

Programm

```
import java.sql.Exception;
import com.mysql.jdbc.Driver;
public class Test
    {
    psvm(.....)
      {
    try
    {
    Driver d=new Driver();
    DriverManager.registerDriver(d);
      }
    catch(.....)
      {
      }
    }
```



→ But in this way of registering the programm will become tight coupling.

2nd Way of Registering (preferable)

→ In this way of registering we use forName().

forName()

```
public static void forName(String fullyQualifiedName)
    {
    }
```

→ It is a method of Class called as "Class".
→ It is a static method & it shows checked Exception. The name of Exception of classNotFound Exception.

→ It is used to load the class.

```
* import com.mysql.jdbc.Driver;
  import java.util.Scanner;
  class A
    {
      psvm(-----)
        {
          Scanner t = new Scanner(System.in)
    SOP("Enter driver Name");
    try
      {
      Class.forName(t.next());
(or) Class.forName("com.mysql.jdbc.Driver");
      SOP("Loading & Registering");
      }
    catch(-----)
      {
      }
    }
```

Host

→ It is a system which runs the application and maintain the resources.

→ There is a two type of Host

(i) Local Host

(ii) Remort Host

Local Host

→ The Host which is running the same system application and maintain the same system resources is called Local Host.

Remort Host

→ The Host which is running the ~~same~~ applicatin and maintain the resources in another system is called Remort Host.

portno

It is same as pin no. There are unique portnos for diff. URLs.

URL

format → protocol : subprotocol :: // host-Name : portNumber ? data

(optional)

(separator)

URL for mysql →

jdbc : mysql : // local host : 3306 ? —

Step-2 → Establish the connection between java program and Database.

~~public static getConn Connection getConnection~~

We can establish the connection between the java programm and database through getConnection()

① public static Connection getConnection(String url)
{
}

② public static Connection getConnection
(String url, String user, String pwd)
{
}

③ public static Connection getConnection
(String url, Propoties propoty)
{
}

Programm (1st way)

```
import com.mysql.jdbc.Driver;
public class A
{
    PSVM(String[] args)
    {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            SOP("Loading & registering");
            Connection cn =
            DriverManager.getConnection
                ("jdbc:mysql://localhost:3306?
                user=root&password=abcd");
            SOP("Connection is established");
        } catch(ClassNotFoundException | SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```

getConnection() present in DriverManager class

2nd way

DriverManager.getConnection("jdbc:mysql://localhost:3306?","root","abcd");

3rd way

Return type of getConnection() is Connection which is an interface present in java.sql package.

Open notepad, then type, user=root (save as Hi.properties)

password=abcd

There is a class called as Connection present in Driver class

We are using Connection interface as returntype by doing upcasting.

Step-III

Create a platform to execute the Quary

We can establish 3 type of platform

(i) Statement

(ii) Prepared Statement

(iii) Callable Statement

interface Connection
{
createStatement();
}

class Connection implements Connection
{
}

upcast

Connection cn = new Connection();

Statement

Prepared Statement

Callable Statement

Statement

→ It is an interface present in java.sql package

→ We can create the object of Statement by using createStatement(), which is present in Connection Interface.

public Statement createStatement()
{
}

→ It is a method present in Connection interface.

Connection cn = DriverManager.getConnection("...");

Statement st = cn.createStatement();

Step-iv

executing the SQL Quary

We can execute the Quries in the following methods.

→ All the methods are present inside the Statement interface.

(i) execute()
(ii) executeUpdate()
(iii) executeQuery()

execute()

→ It is generic method used to execute all type of quary.

Syntax: public boolean execute(String quary);

→ It returns true, if it is DQL
false, if it is DML & DDL

executeUpdate()

→ It is a special type of method used to execute only DML Quaries

Syntax → public int executeUpdate(String query);

→ how many rows are going to affect, then it will return in int value.

→ it gives integer value which is equivalent to no of rows is got affected to DML quaries.

executeQuery()

→ It is also a special type of method used to execute DQL Quary only.

Syntax → public ResultSet executeQuery(String quary)

→ This methods returns the object of interface ResultSet.

Programm

```
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.DriverManager;
public class Test
{
    PSVM(...)
    {
    Connection cn=null;
    Statement st=null;
    String qu="insert into hi.student
              values(2,'Sampad',34.99);
```

```
try
{
Class.forName("com.mysql.jdbc.Driver");
SOP("Loading & registering");
cn = DriverManager.getConnection
        ("jdbc:mysql://localhost:3306 ?
                user=root&password=abcd");
SOP(" Connection is established");

st = cn.createStatement();
SOP("platform established");

SOP(st.execute(qu));
SOP("data inserted");
}
catch (ClassNotFound Exception || SQLException e)
    {
     e.printStackTrace();
    }
}
finally
 {
  if (st != null)
     {
        or try {
              st
              cn.close();
        }
        catch(SQLException e)
        {
        }
```

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

Fetching the result

```
public ResultSet executeQuary(String quary);
```

→ To fetch the result we have to make use of interface ResultSet

⇒ ~~Fetch~~ ResultSet

→ It is an interface present in java.sql package. It comes under the JDBC API.

Methods of ResultSet

```
public boolean next()
```

→ It is used to check there is next record present or not & move the pointer to next record.

true - next record present

false - if next record is not present

* public ****** get****** (int columnNumber)
* public ****** get****** (String columnName)

→ get****** method is used to fetch the column data.

→ Here ***** means any datatype, it can be int or String or Double etc.

→ In get******(-) we can specify column number or column name. Both are respected to buffer memory result.

Programm to fetch the Record

```
import java.sql.Connection;
public class Test
{
    psvm(-!-)
    {
    String s = "select * from hi.student";
    Connection cn = null;
    Statement st = null;
    ResultSet rs = null;
try
{
    Class.forName("com.mysql.jdbc.Driver");
    cn = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/
                user=root&password=abcd");
```

```
st = cn.createStatement();
rs = st.executeQuary(s);
while(rs.next())
  {
    SOP(rs.getInt(1)+"\t");  // rs.getInt("id");
    SOP(rs.getString(2)+"\t");  // rs.getString("name");
    SOP(rs.getDouble(3));  // rs.getDouble("marks");
  }
catch(ClassNotFoundException | SQLException e)
  {
    e.printStackTrace();
  }
finally
  {
  if(rs!=null)
    {
    try
      {
      rs.close();
      }
    catch(SQLException e)
      {
      }
    }
  if(st!=null)
    {
    try{
      st.close();
      }
    catch(SQLException e)
      {
      }
    }
```

(go for columnname) Preferable ↓

Step-6

close all the connection by the resource

```
if(cn != null)
{
  try{
    cn.close()
  }
  catch(SQLException e)
  {
  }
}
```

Write a programm to insert 3 Quarys at a time

```
class A
{
  psvm(....) throws Exception
  {
    Connection cn = null;
    Statement st = null;
    ResultSet rs = null;
    try{
    Class.forName("com.mysql.jdbc.Driver");
cn = DriverManager.getConnection
        ("jdbc:mysql://localhost:3306/
            user=root&password=abcd");
st = cn.createStatement();
sop("take the no of insert you want
        to give at a time");
```

```
String s1 = "insert into hi.student values
                (1, "Sonali", 99.99);
String s2 = "insert into hi.student values
                (2, "Gudi", 1069);
String s3 = "insert into hi.student values
                (3, "hi", 101);
String s4 = "select * from hi.student";
    st.execute(s1);
    st.execute(s2);
    st.execute(s3);
    rs = st.executeQuary(s4);
    while(rs.next())
        {
        sop(rs.getInt("id"));
        sop(rs.getString("name"));
        sop(rs.getDouble("mark"));
        }
    catch(
        {
        }
    finally
    {
        while(rs!=null)
            {
            rs.close();
        while(st!=null)
            {
            rs.close();
        while(cn!=null)
            {
            cn.close();
            }
```

In the above example the Quary is passed through database & (compiled + executed)

→ As we have are having 3 Quary, 3 time compilation & 3 time execution will happen that

→ This will take process time more.

→ To overcome this we use PreparedStatement platform instead of Statement platform.

Prepared Statement

It is an interface present in Java.sql package

→ It is an interface of Java API.

→ It extends Statement interface.

→ Prepared Statement support place holder concept which helps to make programm dynamic

→ In preparedStatement Compilation of the Quary 1 time & execution will be many time

→ We can get the object of Prepared Statement by calling a method prepareStatement()

```
public PreparedStatement prepareStatement
                          (String quary)
{
}
```

delete - 10
update - 10
insert - 10

] preparedStatement better if multiple time run the Quary

→ prepareStatement() ——) present in Connection interface.

Place Holder

→ It is integrated by the symbol (?)

→ Whenever the data is unavailable during the Quary compilation we can use Place Holder

→ It is necessary to set all the placeholder before executing the Quary.

→ We can set the place Holder by using a method set****( )

public void set****(int placeholder number, **** value);

→ set****() method present in the PreparedStatement interface.

Programm

```
public class Test
{
psvm(...)
{
Connection cn = null;
PreparedStatement psn = null;
String qu = " insert into hi.student
            values(?,?,?) ";
Scanner sc = new Scanner(System.in);
```

```
try {
Class.forName("com.mysql.jdbc.Driver");
cn = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/");
psmt = cn.prepareStatement(qu);
SOP("how many reiords you want to insert")
int n = sc.nextInt();
for(int i=0; i<=n; i++)
  {
  SOP("Enter id, name & marks");
   pst.setInt(1, sc.nextInt());
   pst.setString(2, sc.next());
   pst.setDouble(3, sc.nextDouble());
   pst.executeUpdate();
  }
catch(...)
   {
   }
finally
   {
   ss.pst.close();
     cn.close();
   }
```

Updated using PreparedStatement

```
class A
{ psvm(.....)
  {
  ResultSet rs = null;
  Statement st2 = null;
  Connection cn = null;
PreparedStatement pst = null;

try {
  Class.forName("- - - - - - - - - - ");
  cn = DriverManager.getConnection("- - - - - - - - - - ");
String st = "update hi.student set mark=?
                          where id=?";

pst = cn.prepareStatement(st);
sop("put the new mark");
pst.setDouble(1, sc.nextDouble());
sop("put the id where you want to change");
pst.setInt(2, sc.nextInt());
 pst.executel);

}
catch(- - - -)
finally(- - - - )
```

Batch file

It is a file which containing multiple (ie) Quarys.

→ By using Batch file I can execute multiple Quaries at one sort.

→ This reduces the costly operation & process time.

→ We can add the Quary by calling a method addBatch();

```
public void addBatch();
```

→ addBatch() is present in prepared statement

→ we can execute the Batch file by using a method executeBatch()

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

→ executeBatch() method gives integer array.

→ individual elements of the array represent the no of rows got affected by the individual Quaries of the batchfile.

executeBatch()

returns how many rows affect		
1	0	insert
1	1	insert
3	2	update
2	3	update
5	4	delete

→ BatchFile Concept is only applicable for **DML** Quaries

(DML → insert, update, delete)

```
class A
{
 PSVM
 {
   Connection cn = null;
   PreparedStatement pst = null;
 String qu = "insert into histudent values(?,?,?)";
 Scanner sc = new Scanner(System.in);
 try{
  Class.forName("----");
  cn = DriverManager.getConnection("----");
  pst = cn.prepareStatement(qu);
  SOP("How many record you want to insert");
  int n = sc.nextInt();
  for(int i=1; i<=n; i++)
   {
    SOP("Enter id, name and marks");
    pst.setInt(1, sc.nextInt());
    pst.setString(2, sc.next());
    pst.setDouble(3, sc.nextDouble());
    pst.addBatch();
   }
 pst.executeBatch();      //int arr[] = pst.executeBatch();
catch(-----)
```

Scanned by CamScanner

```
for(int y : arr)
    {sop(y);
    }
catch(....)
   {
   }
finally
  {
  }
```

database (.hi)
↓
create Stored Procs
↓
give name

Callable Statement

⟺ Store procedure

→ It is a set of pre compiled Quaries pres in the database.

→ They decreses the process time & increses the reusability factor

→ They are the frequently used Quaries.

Callable Statement

→ It is an interface present in java.sql package & it is a part of Java-API.

→ We can get the Object of Callable Statem by calling a method prepareCall()

```
public CallableStatement prepareCall
                        (String st)
```

⟹ prepareCall() method present in Connection Interface.

→ Callable Statement is used to call the store procedure present in database.

→ Callable statements supports place holder concept as it is a sub interface of Prepared statement too

Quary for Store Procedure

```
DELIMITER &&;
DROP PROCEDURE IF EXISTS 'hi'.'qsp' &&
CREATE PROCEDURE 'qsp'()
BEGIN
select * from hi.student;
END&&
DELIMITER ;&&
```

Programm

```
class A
{ psvm(---)
  { Connection cn = null;
    Callable Statement pst = null;
    ResultSet rs = null;
    String qu = "call hi.qsp()";
Scanner sc = new Scanner(System.in);
try {
  Class.forName("com.mysql.jdbc.Driver");
  cn = DriverManager.getConnection
       ("-------");
```

```
pst = cn.prepareCall(qu);
rs = pst.executeQuary();
while(rs.next())
{
sysO( rs.getInt(1) + " " + rs.getString(2) +
                    rs.getDouble(3));
}
catch(-----)
{
}
finally(-----)
{
}
```

Update using Store Procedure Concept

In SQL Code

```
DELIMITER $$;
DROP PROCEDURE IF EXISTS 'hi'.'hello' $$
CREATE PROCEDURE 'hello' (in id int, in na varchar(3)
                          in ma mark);
BEGIN
insert into hi.student values (id, na, ma);
END$$
DELIMITER ; $$
```

Program (Java)

```
public class A
{
PSVM(-----)
  {
  Connection cn = null;
  CallableStatement pst = null;
  ResultSet rs = null;
  String qu = "call hi.hello(?,?,?)";
  Scanner sc = new Scanner(System.in);
try {
  Class.forName("-----");
  cn = DriverManager.getConnection("-----");
  pst = cn.prepareCall(qu);
  SOP("Enter data");
  pst.setInt(1, sc.nextInt());
  pst.setString(2, sc.next());
  pst.setDouble(3, sc.nextDouble());
  pst.executeUpdate();
}
catch(-----)
  {
  }
finally
  {
  }
```

Login Page

```
class A
  {
    PSVM(.....)
      {
      Scanner sc = new Scanner (System.in);
      SOP("Enter userName");
      Connection cn = null;
      PreparedStatement st = null;
      ResultSet rs = null;
  String q = "select * from hi.login where name = ?";
      Class.forName(".....");
      cn = DriverManager.getConnection(.....);
      st = cn.prepareStatement(q);
      st.setString(1, sc.next());
      rs = st.executeQuary();
if (rs.next())
      {
        SOP("Enter Password");
        String s = sc.next();
          if(rs.getString("password").equals(s))
            {
            SOP("Login successfully");
            SOP(rs.getString("Name"));
            SOP(rs.getInt("id"));
            SOP(rs.getDouble("Mark");
            }
          else
            {
            SOP("Password is not correct");
            }
else
  {
  SOP("Login id is not present");
```

Metadata

ResultSetMetaData

```
class A
{
    PSVM
    {
        Connection cn=null;
        PreparedStatement pst=null;
        ResultSet rs=null;
        ResultSetMetaData rsmt=null;

try {
Class.forName("com.mysql.jdbc.Driver");
cn=DriverManager.getConnection("u-p-.....");
~~rs = ps~~ ~~ps~~ rs=
~~ps~~ = pst=cn.prepareStatement("select * from hi.student");
rs=pst.executeQuery();
rsmd=rs.getMetaData();
int i=rsmd.getColoumCount();
for(int i=1; i<=i; i++)
{
SOP(rsmd.getColoumnName(i));
}
SOP(rsmd.getTableName(2));
catch(
{
}
```

→ ResultSetMetaData is used to give some information about the table like no of column, table name, column datatype etc.

→ It is present in java.sql package

→ We can ~~make~~ get the object of ResultSetMetaData by using getMetaData() method which is present in ResultSet Interface.

* → ResultSet interface object we can get in 2 ways

(i) ExecuteQuery()

(ii) getResultSet(), present in Statement or PreparedStatement or CallableStatement

DatabaseMetaData

→ By using databaseMetaData we can get the information about the database.

like @) database version, database name, drivername

→ We can get the object of DatabaseMetadata by calling a method getMetaData(), which is present in ~~Re~~ Connection Interface.

```
class A
  {psvm(- - - - - -)
    {
    Connection cn=null;
    Statement st=null;
    ResultSet rs=null;
   DatabaseMetaData dbmt=null;
try {
  Class.forName("- - - - - - - - -");
  cn=DriverManager.getConnection("- - - - - - - -");
  dbmt=cn.getMetaData();
  sop(dbmt.getDriverName());
  sop(dbmt.getURL());
  sop(dbmt.getUserName());
  sop(dbmt.getDatabaseMajorVersion());
  String [] st={"TABLE"}
  rs=dbmt.getTables("hi",null,null,st);
  while(rs.next())
    {
    sop(rs.getString(3);
    }
}
catch(- - - - - -)
    {
    }
```

Servlet (server Component)

Server

→ It is a machine which accepts the user invest & process it & get back some response & also the server manages the resources.

→ There are 2 type server
(i) Web server
(ii) Application server

Web Server

→ Web server only accept web (HTTP) request & process it & get back some result

Exp → ~~glassfish~~ Apache Tomcat

Application Server

→ It acts as both webservers & manages the current system applications also

Exp → ~~Apache Tomcat~~ glassfish

So
→ Every server must contain container which are responsable for executing of application.

Servlet

→ It is a set of rules to build the web application

→ It is an API given in the form of jar file
→ It is an interface present in servlet API.
→ It is a serverside java program.
→ It is given in the two packages

(javax.servlet, javax.servlet.http) } Not present in java. Provided by separate file like apache

Interfaces of Javax.servlet

(I) Servlet
(II) ServletRequest
(III) ServletResponse
(IV) filter
(V) filterChain
(VI) filterConfig
(VII) ServletConfig
(VIII) ServletContext
(IX) SingleThreadModel
(X) ServletContextListner
(XI) ServletContextAttributeListner

Classes of Javax.servlet

(I) GenericServlet
(II) ServletContextEvent
(III) ServletContextAttributeEvent

Right Click (Project)
↓
JavaEE tool
↓
Generate deployment description stub.

Servelet Lifecycle

→ There are 4 stages in lifecycle

(I) Instatiation
(II) Intialization
(III) service ⟶ service(_, _)
{
}
(IV) Destroy

Instatiation

→ In this phase when the user make a Ist request the web container going to load the class 1st by the help of class loader.
→ then it creates the object of servelet.
→ This happens only once at the begining.

Intialization

→ In intialization phase it is going to call the init method after the instatiation page

→ In init() we can intialize the component for the servlet.

```
public void init(ServletConfig ab);
```

(It is an abstract method because it is present in ServletInterface)

→ This stage also happens only once at the begining of the servlet.

Service →

→ In service phase page the container creates request object & response object for every user request. & It is going to call the service method for each request.

```
protected void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException;
```

→ init() execute one time, then after for next user directly service() will come. init() method will not execute.

→ Inside service(), actual Bussiness logic is present.

→ In this method we are going to process & getback some result.

Destroy phase

→ This phase is going to execute when the server is stop or the application is undeploy

→ In this phase the object of servlet is going to destroyed but before deleting the servlet object it is going to call destroy()

```
public void destroy()
```

Servlet Interface

→ It is an interface present in javax.Servlet package

⇒

→ Method of Servlet Interface

① public void init(ServletConfig ab)

② public void destroy();

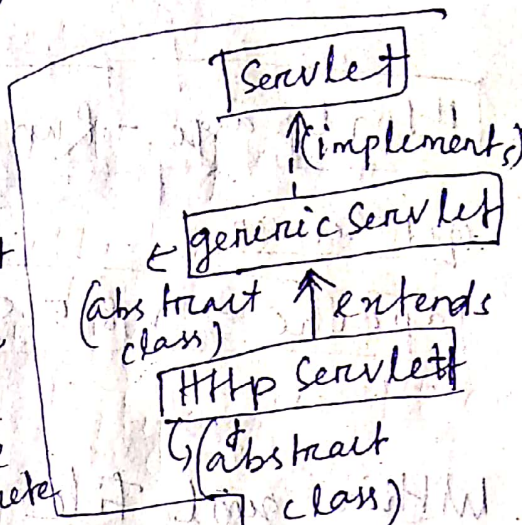
③ ~~public~~ protected void service(ServiceRequest sr, ServiceResponse sr) throws ServletException, IOException

④ Public ServletConfig getServletConfig();

⑤ public String getServletInfo();

→ init(), service(), destroy() are called as a life cycle methods.

→ generic Servlet override init(), service(), getServletConfig(), getServletInfo() but not service() so they are concrete & service() is an abstract method in generic Servlet.



→ There are 2 subclasses of Servlet

→ HttpServlet overrides all the 5 method & all are concrete method.

→ HttpServlet present in javax.servlet.http.

* ① Create new HTML file in web Content

② Java Resources → src → create package → new class → don't give main method

③ apache Tomcat → lib → servlet-api copy → web-INF → lib → paste

<body>

web.xml → source → delete everything except <web-app>

class → copy qualified name

→ <webapp> is root tag for tomcat.

Html page - Run As → Run Server

Get Server

Apache - v8.5 server → Next → browse → folder upto apache tomcat → finish

Within xml File

```
<servlet>
  <servlet-name>prabhas</servlet-name>
  <servlet-class>com.demo</servlet-class>
</servlet>
```

(com.demo — fully package name)

```
<servlet-mapping>
<servlet-name>prabhas</servlet-name>
<url-pattern>/mindtree</url-pattern>
</servlet-mapping>
```

* html File

```
<body>
<form action="mindtree">
<input type="~~text~~ submit">
</body>
```

Java file

```
class Demo extends GenericServlet
{
public void service(.....)
{
sop("Hi Good Morning"); sop("Rupesh")
}
}
```

o/p → Hi Good Morning RUPESH

Html file

```
<body>
<form action = "mindtree">
username: <input name = "un"><br><br>
Password: <input type = "password" name = "ps">
<input type = "submit">
</body>
```

Java

```
class Demo extends GenericServlet
{
Public void service(SReq, SRes)
{
String usn = req.getParameter("un");
String pas = req.getParameter("ps");
sop(usn); sop(pas);
}
}
```

→ To take input from user there is a method getParameter() present in ServletRequest object.

Public ~~void~~ String getParameter(String st)
{
}

(This should be same as the particular name attribute)

* <input type="text" name="pan">

String name = req.getParameter("pass");
SOP(name);

✓ System.out.println("");

System
{
PrintStream out = new PrintStream();

PrintStream
↳ println() method is present

→ To take response from our programm there is a method setContentType() present in ServletResponse Object.

public void setContentType("text/html");

{

}

from this it will access only html code at output screen

→ Java Programm

public class A

{

public void service(

{

String user = req.getParameter("un");

String pass = req.getParameter("ps");

res.setContentType("text/html");

PrintWriter p = res.getWriter();

p.println("<html><body>Welcome
");

p.println("user" + "& password" + pass + "
");

p.println("</body></html>");

}

HTML Code

<body>

UserName: <input type="text" name="un">

Password: <input type="password" name="ps">

<input type="submit" name="sub" value="Login">

</body>

→ PrintWriter class present in java.lang package & inside printWriter println() method is present.

→ We can get object of printWriter by calling getWriter() method which is present in ServletResponse Interface.

PrintWriter p = res.getWriter();

---

getParameter()

→ It is a method present in ServletRequest.

→ It is used to get the value from the key.

public String getParameter(String a)
{
}

setContainType()

→ It is a method present in the ServletResponse

→ It is used to set the type of response given by the servlet

→ By default it is taking "text/html"

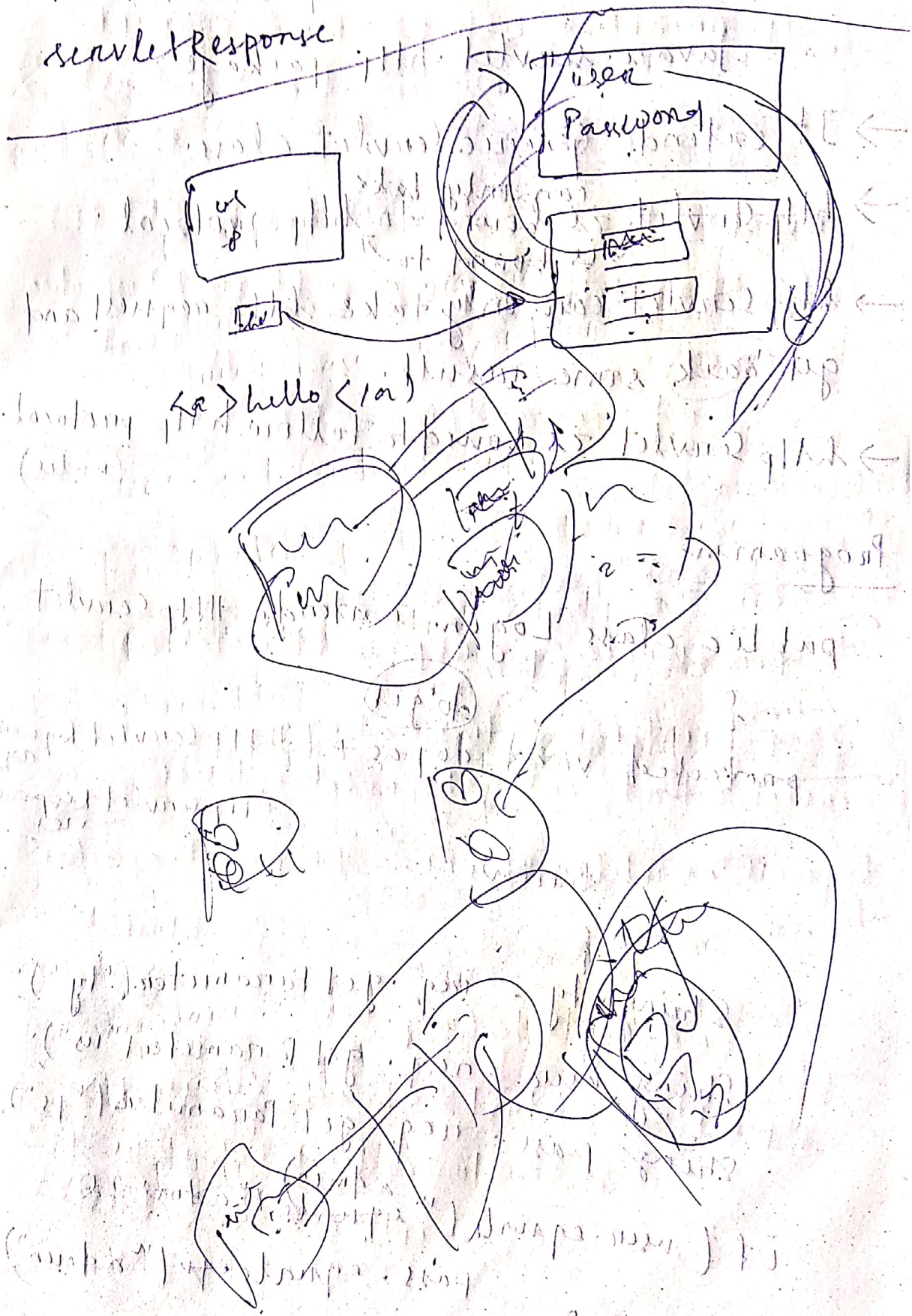
public void setContainType(String type)

⇒ PrintWriter

→ It is a class of io package

→ for using println() method of PrintWriter we can give the response through HTML file

→ By getWriter() method to create PrintWriter object & It is present in the servletResponse



HttpServlet

→ It is an abstract class present in javax.servlet.http package.

→ It extends GenericServlet class.

→ ~~HttpServlet is bound to http protocol.~~ (can only take) (is bound to)

→ HttpServlet can only take http request and get back some result.

→ HttpServlet is bound to follow http protocol (rules)

Programm

```
public class Login extends HttpServlet
{
protected void doPost (HttpServletRequest req,
                       HttpServletResponse res)
        throws . . .
{
String ty = req.getParameter("ty");
String user = req.getParameter("us");
String pass = req.getParameter("ps");
if (user.equals("admin") && pass.equals("admin"))
{
continue;
```

(doGet)

Html:

```
<body>
<h1> Welcome </h1>
<h3> <a href="Login.html"> Login </h3>
<h3> <a href="Customer Register.html"> Register </h3>
<h3> <a href="manager Register.html"> Register M </h3>
</body>
```

* `<form action="—" method="post">` (post / get)

`</form>`

→ `ServletContext cx = getServletContext();`

`RequestDispatcher rd = cx.getRequestDispatcher("Home.html");`

`rd.forward(req, res);` (two argument input) → redirect to html page

?

else

{

`ServletContext cx = getServletContext();`

`RequestDispatcher rd = cx.getRequestDispatcher("Login.html");`

`rd.forward(req, res);`

}

OR

else

{

PrintWriter p = resp.getWriter();

p.println("<html><head><script>

window.alert('check us &Ps')

</script></head></html>");

ServletContext cx = getServletContext();

RequestDispatcher rd = cx.getRequestDispatcher

("/login.html");

rd.forward(req, res);

}

OR

else

{

PrintWrite (p = resp.getWriter();

~~p.println("<html><hea~~

ServletContext cs = getServletContext();

RequestDispatcher rd = cs.getRequestDispacher

("/login.html");

rd.include(req, res);

p.println("<html><head><script>

Window.alert('check us &Ps')

</script></head></html>");

}

HttpServlet

→ It is an abstract class but it is not having any abstract method.

→ We are going to place our bussiness logic by doget() or dopost method

* protected void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException & IOException
{
}

* protected void doGet(HttpServletRequest Req, HttpServletResponse res)
throws

→ RequestDispature

* It is used to forward the request aswell as response object to another servlet or webpage & include the result of another servlet or html page in the current servlet

→ It is also an Interface present in javax.servlet packag

→ By calling getRequestDispature(), method we can get RequestDispature object.

→ Public RequestDispature getRequestDispature
(String url of html/servl
{
}

* It is present inside the ServletContext Interface.

→ RequestDispature contains mainly 2 methods. (i) forward(req, res)
(ii) include(req, res)

forward

It is used to forward the request & response obje to the servlet of Html page.

include

It is used to include the result of servlet or Html page.

→ Connect two Servlets

class Test extends HttpServlet
{ protected void doGet(- - - -)
{ SOP("Test started");

RequestDispature rd = getServletContext().
getRequestDispature(

rd.forward(req, resp)

SOP("Test ended");

}

}

→ class Test2 extends HttpServlet

{

protected void doGet(---

{

SOP("Test2 started");

SOP("Test2 ended");

}

}

→ web.xml

① <servlet>

<servlet-name> dog </servlet-name>

<servlet-mapping> Test </servlet-mapping>

</servlet>

<servlet-mapping>

<servlet-name> dog </servlet-name>

<url-pattern> /tr </url-pattern>

</servlet-mapping>

② Same as upper

<servlet mapping>

<servlet-name> cat </servlet-name>

<url-pattern> /jcb </url-pattern>

</servlet-mapping>

HTML

<body> <form action = "hi" >
<input type = "submit" >
</form>
</body>

Servlet Config

→ It is an interface present in javax.servlet
→ It is used to create a config object.
→ For every servlet there is one config
servlet is present & it is used to configure the servlet through xml file.
→ Config object is created by web container
→ We can get the object of ServletConfig by using getServletConfig()
→ This method present in javax.servlet.GenericServlet

```
class Test extends HttpServlet
{
    Protected void doGet(-,-)
    {
        ServletConfig p = getServletConfig();
        String s = p.getInitParameter("k1");
```

```
sop(p.getInitParameter("jdbc"));
sop(p.getInitParameter("driver"));
}
}
```

web.xml

```
<servlet>
<servlet-name>one</servlet-name>
<servlet-class>hi.Test</servlet-class>
<init-param>
<param-name>jdbc</param-name>
<param-value>pravas</param-value>
</init-param>
<init-param>
<param-name>driver</param-name>
<param-value>com.mysql.jdbc.Driver</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>one</servlet-name>
<url-pattern>/tr</url-pattern>
</servlet-mapping>
```

Html

```
<body><form action="tr">
<input type="submit">
</form></body>
```

→ Creating init parameter in xml file by <init-param>

→ name can be given in <param-name>
value can be given in <param-value>

@Override

→ It gives conformation for imidiate overriden method.

→ If the method is not overriden then it gives error

→ If Before overriden method, we have to write @Override.

~~public class A extends~~

@WebServlet(value = "/tr")

• url → html action = "u"

public class A extends HttpServlet

{
protected void doGet(

{
SOP("Hi");
}
}

→ We don't need to write anything within another web.xml

It is automatically connect to Html file

ServletConfig (continue)

Methods of ServletConfig;

① * public String getInitParameter(String key);

* → It is used to get the parameter or value. (<param-name> key </param-name>)

② public Enumeration getInitParameterNames();

→ It gives object of Enumeration, which can iterate the initial parameters names collection.

③ public ServletContext getServletContext();

→ By using this we can create the object of servletContext

④ public String getServletName();

→ It is used for get the servlet name (present class name)

Programm

```
class Test extends HttpServlet
{
  doPost ---- .
  {
Enumeration e = getServletConfig().getInitParameter
Servlet                                  Names();
```

```
while(e.hasMoreElements())
  {
  SOP(e.nextElement());
  }
```

To get param-name

Create <init-param> & int (param-name) & <param-value>

To get param Value

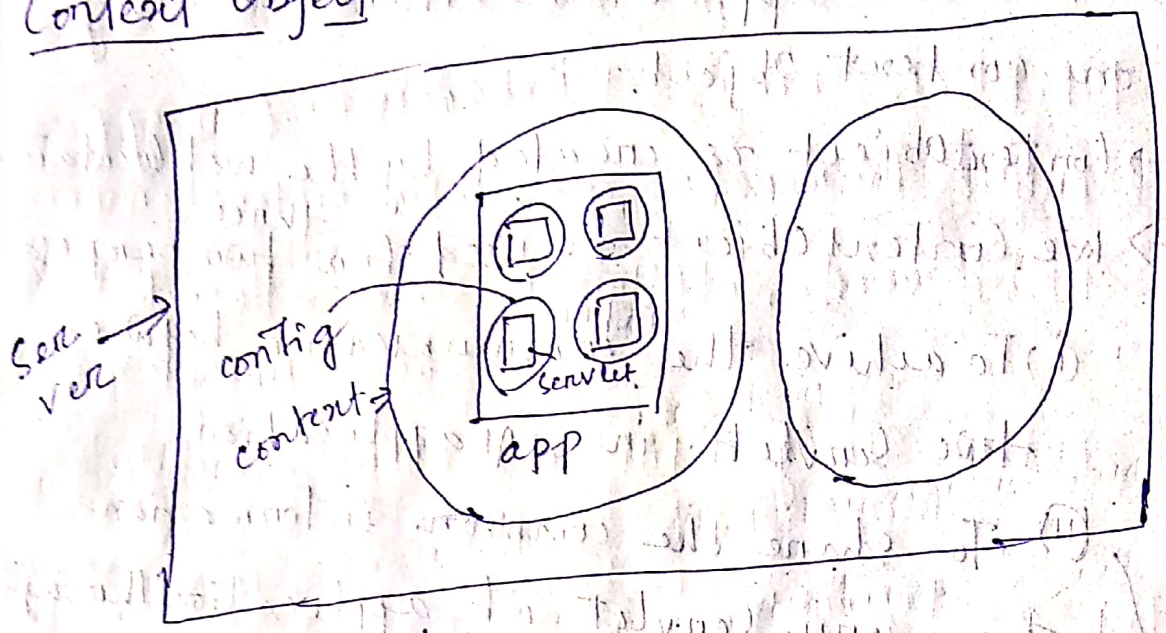
```
class A extends HttpServlet
  {
    doGet
      {
  ServletConfig cn = getServletConfig();
  Enumeration p = cn.getInitParameterNames();
  while(p.hasMoreElements())
    {
    String y = (String) p.nextElement();
    SOP(cn.getInitParameter(y));
    }
  }
```

→ hasMoreElements() & nextElement() are present inside the Enumeration.

→ Enumeration is like an Iterator

→ For Iteration perpose we use Enumeration.

Context Object



ServletContext

→ It is an interface present in javax.servlet package.

→ We can get the object of ServletContext in two ways.

* (I) By ~~directly~~ calling getServletContext() of ServletConfig

(III) ServletConfig sc = getServletConfig();
ServletContext sco = sc.getServletContext();

* (II) By calling directly getServletContext() of GenericServlet

ServletContext sc = getServletContext();

→ Context object is get created when the application is get deployed in the server.

→ For one application there is only one context object.

→ ContextObject is created by the web container

→ Context object is used for three purpose

(i) To achive the inter communication between two servlets in a application.

(ii) To share the common information for every servlet of application throughout web.xml

(iii) To achive the inter communication between two web application present in same server.

web.xml

→ Use of statement (ii)

```
<context-param>
<param-name> pravas </param-name>
<param-value> dracula </param-value>
</context-param>
<servlet>
</servlet>
<servlet-mapping>
</servlet-mapping>
```

```
class Test extends HttpServlet
{
  doGet
  {
  ServletContext sc = getServletContext();
    sc.getInitParameter("pravas");
    SOP("Rupesh");
  }
}

class Test2 extends HttpServlet
{
  doGet
  {
  ServletContext sc = getServletContext();
    sc.getInitParameter("pravas");
    SOP("Pravas");
  }
}
```

Html

```
<body>
<form action="ll">
<input type="submit">
</form>
<form action="gg">
<input type="submit">
</form>
```

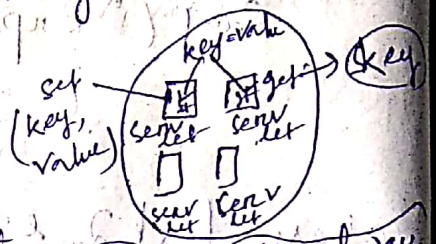
Methods of ServletContext

→ public String getInitParameter(String para...

→ public Enumaration getInitParameterNames()

→ public void setAttribute(String attributeName, String attributeValue)

→ public String getAttribute(String attributeName)



Attribute always given in the key & value pair

Exp → (Intercommunication between servlets)

```
public class A extends HttpServlet
{
  doPost(.....)
  {
ServletContext ct = getServletContext();
  ct.setAttribute("hi", "abcd");
  SOP("ct.getInitParameter("pravas");
  }
}
public class B extends HttpServlet
{
  doPost(.....)
  {
ServletContext ct = getServletContext();
  ct.getAttribute("hi");
  SOP("ct.getInitParameter("pravas");
```

→ To give common information

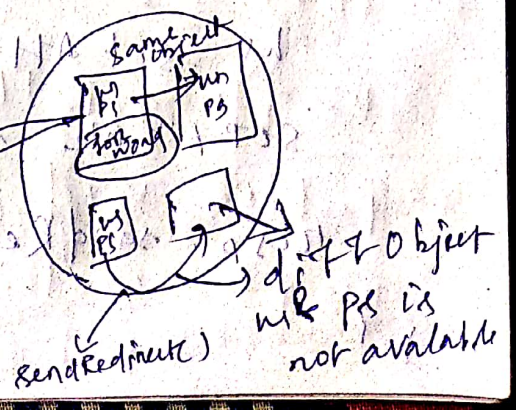
→ Inter Communication between two Applications

sendRedirect("/____");

→ By using sendRedirect(), we can redirect the request from one servlet to another servlet or Html or a website.

→ sendRedirect() is present inside the HttpServletResponse.

Forward()	SendRedirect()
→ It is a method of RequestDispature	→ It is a method of HttpServletResponse
→ Forward method pass the same request & response object to next.	→ sendRedirect method redirect to pass a new request to next
→ This is can only possible for servlet & HTML page	→ This is possible for servlet, HTML Page & other website.



Http Session

Html File

```
<body>
<form action="lo">
User Name: <input name="us">
Password: <input type="password" name="p">
<input type="submit">
</form>
</body>
```

Java

```
@WebServlet(value="lo")
class A extends HttpServlet
{
    doGet - - - - -
String user = req.getParameter("us");
String pass = req.getParameter("pass");
HttpSession hs = req.getSession();
hs.setAttribute("uid", user);
hs.setAttribute("p", pass);
sop("We are redirecting.....");
resp.sendRedirect("abc");
}
```

```
@WebServlet(value="/abc")
class B extends HttpServlet
{
  doGet(.....)
  {
    HttpSession t = req.getSession();
    SOP(t.getAttribute("uid"));
    SOP(t.getAttribute("p"));
    PrintWriter out = resp.getWriter();
    Date k = new Date(t.getLastAccessedTime());
    SOP(k);
    out.println("<html><Body><script>window.alert
                ('Logout successfully')</script></Body>
                </html>");
  }
}
```

```
@WebServlet(value="/log")
public class Logout extends HttpServlet
{
  doGet
  {
    HttpSession y = req.getSession();
    y.invalidate();
    PrintWriter out = resp.getWriter();
    RequestDispatcher rd = req.getRequestDispatcher
                           ("newFile.html");
    rd.include(req,resp);
    out.println("<html><head>
    <script>window.alert('Logout
    successfully');</script>
    </head></head>");
```

getRequestDispatcher also present in HttpServletRequest

Http Session

→ It is an interface present in javax.servlet.http package

→ It helps to keep the user activities like last login information, some of its details, value, etc.

→ We can also provide security for the user account. (max time)

→ It is a time session is a time between the user login to logout.

→ To get the object of Http session, by using we are by getSession() method present in Http servlet Request interface.

→ We can not use it for Generic Servlet

→ Session Object is created by the container & It maintains the session Id for every user

Methods of HttpSessions

1. public int getId();

→ used to get the session Id for any user

2. public void setAttribute(String attr name, Object String attr value);

→ used to create an attribute & put some value within it.

③ public String getAttribute(String Attribute name)
→ used to get the attribute value.

④ public long getLastAccessedTime();
→ used to get the last accessed time in milisecond starting from 1970 June.

⑤ public void invalidate();
→ used to kill the current session

⑥

Advantge of HTTPSession

→ It keeps the record of the user like last login time, login duration etc.
→ It can use for intercommunication between servlets.
→ If we can create a key & value but value can be anything, key must be String.

Disadvantges of HTTPSession

→ Session object is stored in the server side.
→ It will load the server & performance of the server will be decrese

Cookies

→ It is a small information stored in the clientside.

→ We can create object of Cookie by 2 ways

① Cookie c1 = new Cookie();
c1.setName("");
c1.setValue("");

② Cookie c2 = new Cookie(name, value);

HTML page

```
<body>
<form action="hi">
Username: <input name="us">
Password: <input type="password" name="ps">
<input type="submit">
</form>
</body>
```

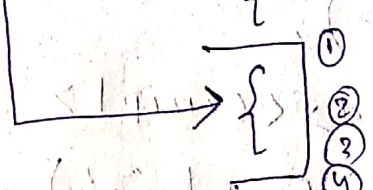
```
<form action="pp">
<a href="pp">
<button> your page
```

Java Code

```
@WebServlet(value="/hi")
public class one extends HttpServlet
{
    doGet(.....)
    {
PrintWriter out = resp.getWriter();
```

String user = req.getParameter("us");
String pass = req.getParameter("ps");

{
① Cookie c1 = new Cookie("name", user);
② Cookie c2 = new Cookie("pas", pass);
③ resp.addCookie(c1);
④ resp.addCookie(c2);

if (user.equals("hello"))
{ if (pass.equals("Hi"))
{


~~out.println("<html><body>Hi you are in</body></html>");~~
"</body></html>");

RequestDispature rd = ~~getServlet~~
req.getRequestDispature("protile.html");
rd.forward(req, resp)

}
}

@WebServlet(value="PP")
class two extends HttpServlet
{ doGet - - - - -
{

← PrintWriter out = resp.getWriter();

Cookie

arr[]=req.getCookies();

if (arr!=null)
{
for(int i=0; i<arr.length; i++)
{
SOP(arr[i].getValue());
}
}

else
{
req.getRequestDispature("NewFile.html")
include(req, resp);
}

out.println("<html><head><script>
window.alert('First Login')
</script></head></html>");

Profile.html

<body>
<h1> Welcome to Profile </h1>
 Log Out
</body>

```
@WebServlet(value = "/Log")
class LogOut extends HttpServlet{
    { doGet( ..... )
      {
HttpSession hs = req.getSession();
hs.invalidate();
req.getRequestDispatcher("NewFile.html").
      forward(req, resp);
      }
}
```

* ~~We ca~~ Cookie present in javax.servlet.http servlet

* We can add the cookie to the response by calling addCookie() i.e.

Exp → res.addCookie(c1);

* We can take out the cookies by using getCookies();

```
public Cookie[] getCookies()
  {
  }
```

→ getCookies() present in HttpServletRequest.

→ Cookie is also used to communicate between two servlets.

Html code ①

```
<body>
<form action = "personal">
<h1> personal info </h1>
First Name: <input type = "text" name = "fn"
Last Name: <input type = "text" name = "ln"
</form>
</body>
```

② Form — Form2.html

```
<body>
<form action = "profile">
Employee id: <input name = "id"><br><br>
Employee email: <input type = "email" name = "e
<br><br>
<input type = "submit">
</form>
</body>
```

Java Code

```
@WebServlet(value = "/personal")
class A extends HttpServlet
{ doGet - - - - - }
```

String fname = req.getParameter("fn");
String lname = req.getParameter("ln");

Cookie c1 = new Cookie("fna", fname);
Cookie c2 = new Cookie("lna", lname);

resp.addCookie(c1);
resp.addCookie(c2);

req.getRequestDispatcher("form2.html").include(req, resp);

}

@WebServlet(value="/pro7")
class B extends HttpServlet {
doGet(...)
{
String id = req.getParameter("id");
String em = req.getParameter("em");
Cookie[] arr = req.getCookies();
sop("your info");
for(int i=0; i<arr.length; i++)
{ sop(arr[i].getValue()); }
sop(id); sop(em);
arr[0].setMaxAge(0);
~~arr[0].setValue()~~

arr[i].setMaxAge(0);

Cookie c1 = new Cookie(arr[0].getName(), "");

new Cookie(arr[i].getName(), "");

}

}

delete the cookie

→ There are two type of cookies

(I) persistance → save data for longer time

(II) non persistance → save data for a shorter time.

~~Url-Rewriting~~

There are 4 type of temporary storage

(i) HttpSession

(ii) Cookies

(iii) Url-Rewriting

(iv) Hidden form

There is 1 type of permanate storage

(i) database

~~Url Rewriting~~

Hidden FormField

```
@WebServlet("value = "/personal")
public class Test extends HttpServlet
{
    doGet - - - - - - - -
    {
String fname = req.getParameter("fn");
String lname = req.getParameter("ln");
PrintWriter out = resp.getWriter();
out.println("<html><body><form action='profile'"
    + "Employee Id :<input name='id'><br><br>"
    + "Email:<input type='email' name='em'><br>"
    + "<input type='submit'></body>"
    + "<input type='hidden' name='fna'
                value=" + fname + ">"
+"<input type='hidden' name='lna' value="+lname
                + ">"
+"<input type='submit'></body></html>");
    }
}
```

```
@WebServlet(value = "/profile")
class B extends HttpServlet
{
    doGet()
    {
String id = req.getParameter("id");
String em = req.getParameter("em");
String f = req.getParameter("fna");
String l = req.getParameter("lna");
SOP(id); SOP(em); SOP(f); SOP(l)
    }
```

HTML Code

```
<body>
<form action = "personal">
<h1> personal info </h1>
firstName: <input type = "text" name = "fn">
LastName: <input type = "text" name = "ln">
</form>
</body>
```

→ ~~Hidden for~~ We are hiding the form before final submiting. ~~We hide for~~

→ Redirecting ~~the~~ another Html page by writing Html code within println. We have to take all the information by giving key & value (name, value) & type as hidden.

Url Rewriting

→ We have to write URL in anchor tag

```
@WebServlet(value = "/personal")
class A extends HttpServlet
    {
      doGet
      {
String fname = req.getParameter("fn");
String lname = req.getParameter("ln");
PrintWriter out = resp.getWriter();
out.println("<html><body><h1> You want to
                subscribe for this also </h1>
+"<br><br><a href='proff?na=" +fname
+"&ln=" +lname+ "'></body></html>")
    }
}
```

```
@WebServlet(value = "/proff")
{
class B extends HttpServlet
  {
    doGet
    {
String first = req.getParameter("na");
String last = req.getParameter("ln");
```

```
        SOP(first);
        SOP(last);
    }
}
```

HTML code

```
<body>
<form action = "personal">
<h1> personal info </h1>
firstname : <input type = "text" name = "fn">
Lastname : <input type = "text" name = "ln">

</form>
</body>
```

→ When two servlet are connected two database & we want to send the data to another table then we use url rewriting

→ Here in 2nd servlet the data is automatically save, no need to write another time.

Filter

→ Filter is an API present in javax.servlet package

→ filter is also an interface

→ It is a plugable component used for decription, incription, on keeping log information etc.

→ In Filter API mainly 3 interfaces are avalable (i) Filter
(ii) FilterConfig
(iii) FilterChain

Methods of filter interface,

(i) public void init(FilterConfig arg0) throws ServletException {

(ii) public void doFilter(ServletRequest req, ServletResponse resp, FilterChain ch) throws IOException, ServletException {

(iii) public void destroy() {

```
<servlet>
<servlet-name> one </servlet-name>
<servlet-class> Test1 </servlet-class>
</servlet>
<servlet-Mapping>
<servlet-name> one </servlet-name>
<url-pattern> /aa </url-pattern>
</servlet-Mapping>
```

<Filter>
<Filter-name>qwe</Filter-name>
<Filter-class>hi.kk.jj.FilterEx</Filter-class>
</Filter>
<Filter-Mapping>
<Filter-name>qwe</Filter-name>
<url-~~mapping~~ pattern>/aa</url-~~mapping~~ pattern>
</Filter-mapping>

---

* class Test1 extends HttpServlet
{ doGet(---)
{
SOP("servlet started");
}
}

HTML code

<body>
<form action="aa">
</form>
</body>

* class FilterEx implements Filter {
public void destroy()
{
}
public void doFilter(---)
{
SOP("filter is executed");
ch.doFilter(req, resp);
}
public void init(FilterConfig args)
{
}

Html Code

```
<body>
<Html> <form action="aa">
Email:<input type=text name="em" />
Mob: <input type="tel" name="mob" />
</form>
</body>
```

→

```
class FilterEx implements filter
{
    public void destroy(){
    }
    public void doFilter(req, res, FilterChain fc)
    {
        String email = req.getParameter("em");
        String mob = req.getParameter("mob");
        req.setAttribute("use", email+"@gmail.com");
        req.setAttribute("mob", "+91-"+mobile);
        fc.doFilter(req, res);
    }
    public void init(FilterConfig arg0)
    {
    }
}
```

→

```
class A extends HttpServlet
{
    doget
    {
        sop(req.getAttribute("use"));
        sop(req.getAttribute("mob"));
```

xml

```
<servlet>
<servlet-name> one </servlet-name>
<servlet-class> A </servlet-class>
</servlet>
<servlet-mapping>
<servlet-name> one </servlet-name>
<url-pattern> /aa </url-pattern>
</servlet-mapping>

<filter>
<filter-name> two </filter-name>
<filter-class> filterEx </filter-class>
</filter>
<filter-mapping>
<filter-name> two </filter-name>
<url-pattern> /aa </url-pattern>
</filter-mapping>
```

FilterConfig

.xml

```
<servlet>
<serv
:
</servlet-mapping>
<filter>
:
<filter-class> filterEx </filter-class>
```

```
<init-param>
<param-name>Asd</param-name>
<param-value>hello</param-value>
</init-param>
</filter>
<filter-mapping>
```

---

filterex class

```
class filterEx implements Filter
    {
    FilterConfig h;
public void destroy() { };
public void doFilter(. . . . . )
    {
  SOP(h.getInitParameter("asd");
                ch.doFilter(req, res);
    }
public void init(ServletConfig arg0)
    {
      h = arg0;
    }
}
```

JSP

→ Java Server Page
→ JSP is a combination of HTML & Java Programming
→ JSP is also an API
→ It is a extended version of servlet

→ **Advantages**

① It reduces the code.
② Easier to do the programm than servlet
③ Execution is faster

Lifecycle of JSP

(I) translation
(II) Instatiation
(III) Intialization
(IV) service (_jspService)
(V) destroy

Translation → In this phase the JSP page is converted into servlet & Everything is done by container

Instatiation → In this phase servlet compilation will happen & instance is created by the container.

Intialization → In this phase it is going to call jspInit() method for the intialization perpose.

service → In this phase the actual bussiness logic is present in the _jspService() method.

→ In service phase it is going to call _jspService()

→ All the request are going to present process & get back some response.

destroy → In this phase jspDestroy() method is called. It is executed when the application is removed on server is shutdown.

JSP API

→ It is present in ① javax.servlet.jsp

② javax.servlet.http.jsp.tagext

JSP Tags

→ There are 3 types of tags are available

(i) script-let tag (<% %>) (Should be inside jsp service)

(ii) Expression tag (<%= %>) (Printing)

(iii) Declaration tag (<!% %>) (Should be outside jsp service)

Scriptlet tag

→ It cointains on going to be placed inside the jspService() method.

Expression tag

→ It is used to print the cointain.

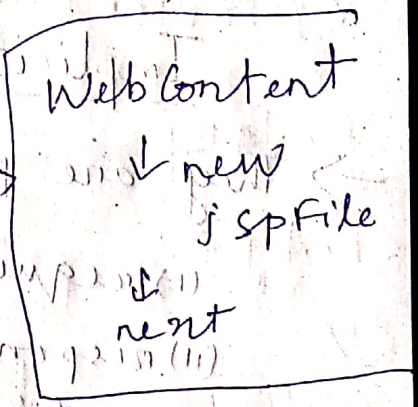
Declaration tag

→ ~~It is used~~ to The cointain of declaration tag placed outside of JspService() method.

→ We use declaration tag for creating method or static & nonstatic variables.

Exp

```
<body>
<h1> Thereis will no mock...</h1>
<%
PrintWriter t = response.
                getWriter();
for(int i=0; i<=10; i++)
  {
    t.println(i);
  }
t.println("<body></html>");
t.println("<html><body>");
%>
</body>
```



```
<body>
<%
 for (int i=1 ; i<=10; i++)
  {
%>
The value of i = <%= i %> <br>
```

```
<%
  }
%>
</body>
```

We can write html code by ending scriptlet tag & restart scriptlet tag

Implicit Objects

→ They are predefine objects in JSP

(i) request → servletRequest
(ii) response → servletResponse
(iii) config → servletConfig
(iv) application → servletContext
(v) out → printWriter
(vi) session → HttpSession
(vii) page → Object
(viii) exception → Throwable
(ix) pageContext → PageContext

Html Code

```
<form action = "hi.jsp">
Firstname: <input type name = "fname">
Lastname: <input name = "lname">
<input type = "submit">
```

hi.jsp

```
<body>
<% String fname = request.getParameter("fname");
request.getParameter("lname");
%>
Welcome: <%= fname %>
</body>
```

Connect through web.xml

Html Code

```
<form action = "gg">
Firstname: <input name = "fname">
Lastname: <input name = "lname">
<input type = "submit">
```

web.xml

```
<servlet>
<servlet-name>one</servlet-name>
<jsp-file>/hi.jsp</jsp-file>
</servlet>
```

<servlet-mapping>

<init-param>
<param-name>77</param-name>
<param-value>Hello</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>one</servlet-name>
<url-pattern>gg</url-pattern>
</servlet-mapping>

hi.gjsp

<body>
<%
String name = request.getParameter("fname");
String lname = request.getParameter("lname");
String h = config.getInitParameter("gg");
%>
Welcome <%=name%>
Config object is <%=h%>
</body>

JSP

Directives

→ These are the information about the container

→ There are 3 types of directives are available

(i) page (<@ page >
(ii) include (<@ include >
(iii) taglib (<@ taglib >

Directives are always specified by <@ >

page directive

→ It gives the information about the current JSP file.

→ It has some attributes, like

(i) language = "java" → It is always Java
It gives the information of language which is included in JSP file.

(ii) contentType = "text/html;"
It gives the return type form which information of html file.

(iii) pageEncoding = "ISO-8859-1"
It gives what type of charset is include the jsp file.

(iv) import = "java.util.Scanner"
When we want to import some class.

(iv) extend = "java.util."

It is used for extending the classes

(v) isErrorPage = "true"
"false"]

It is true when if the current JSP file is going to handle the exception; otherwise it always false (by default)

(vi) error Page = "file"

This attribute gives the information about the JSP file to which the control will be passed if an exception arose

(vii) info = "abild"

It is used for getting / set the information about the current servlet.

→ To get that info value (getServletInfo())

Example of JSP exception

Html colder

<body>

<input type = "number" name = "n1" />

<input type = "number" name = "n2" />

<input type = "test" name = "" />
submit

handExc.jsp

<%@ ~~errorPage~~ isErrorPage = "true" %>
<body>
<h1> Exception name is ..... <%= exception %> </h1>
</body>

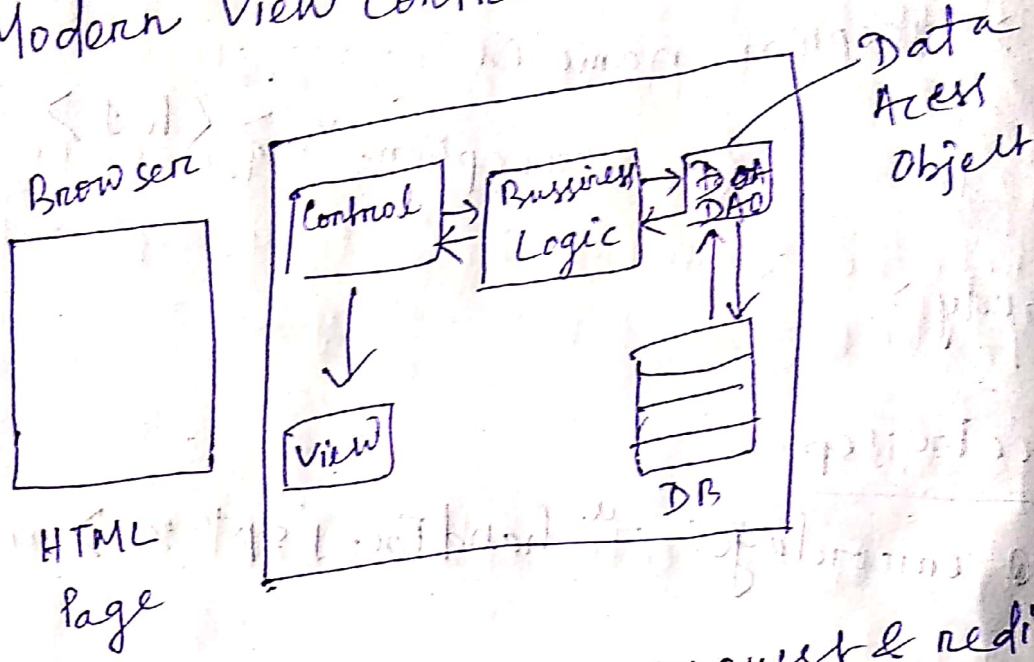
exceExc.jsp

<%@ errorPage = "handExc.jsp" %>

<body>
<%
int c = Integer.parseInt(request.getParameter("n1"));
int h = Integer.parseInt(request.getParameter("n2"));
int c = k/h;
%>
Result = <%= c %>
</body>

MVC Architecture

Modern View Control



Controller ⇒ It receives the request & redirect the request to respective destination / servlet / class / JSP

→ Usally controller are developed by servlet

Bussiness Logic → It is the actual processing of our request.

→ It is the exact classes or servlet which process the request.

DAO → These are all the classes which interacts with the bussiness logic and database

View → View is the servlet or Jsp file which furnish the response and give it to the response.

Listener

event → Events are the action which occure in the server

→ Event are also classes present in javax.servlet package

Exp → ServletContextEvent
ServletContextAttributeEvent
ServletRequestEvent
ServletRequestAttributeEvent
HttpSessionEvent

Listener → Listener are the classes which is executed when the event occurs.

→ Listeners are interfaces

Exp → ServletContextAttributeListener
ServletContextListener
ServletRequestListener
HttpSessionListener