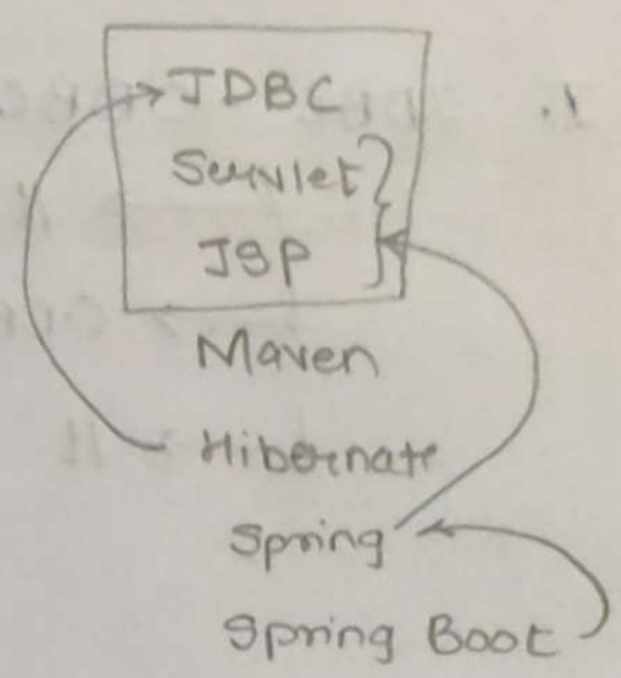
JAVA DATABASE Connectivity.

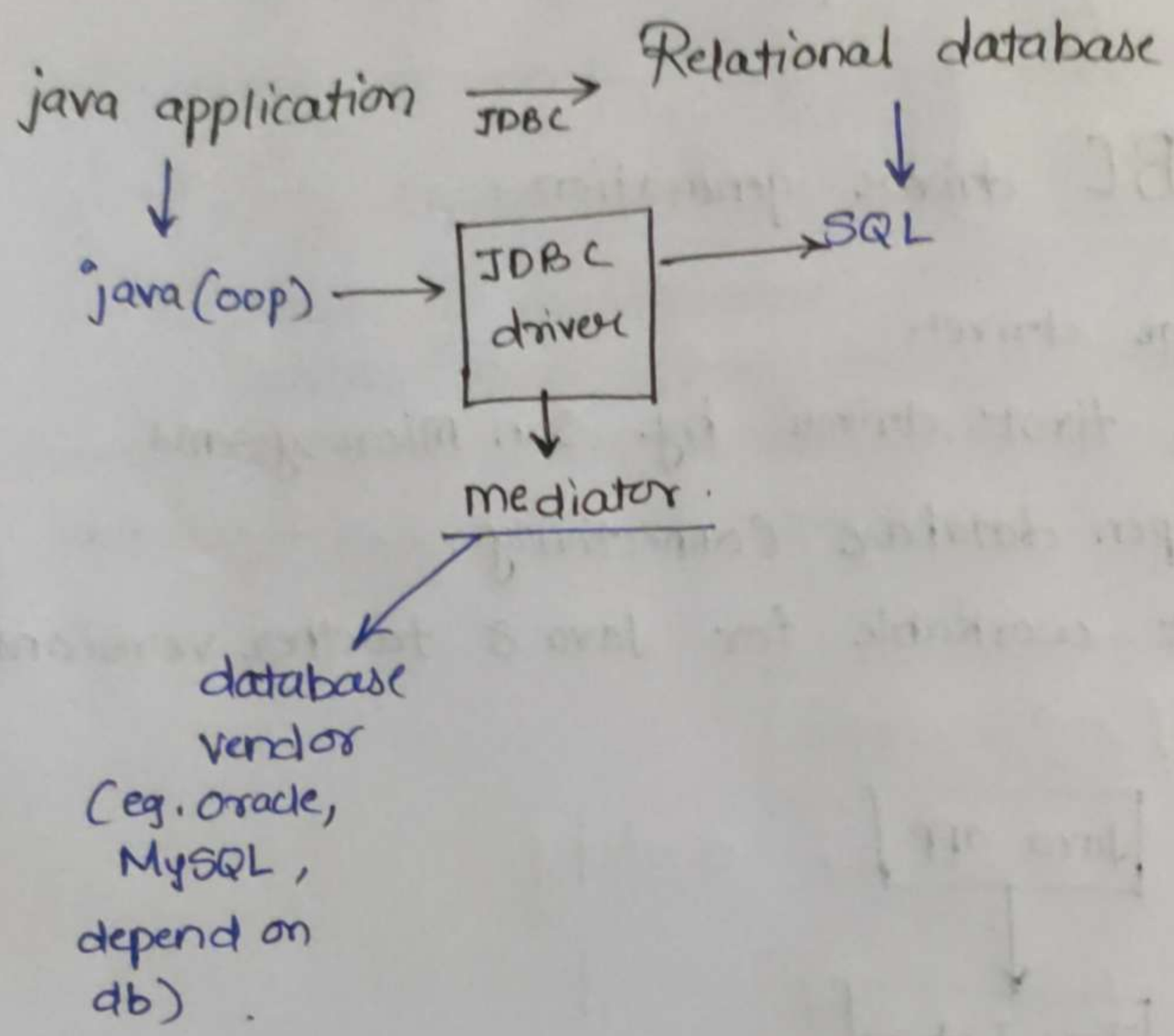
Agenda.

1. What is JDBC?
2. JDBC driver types.
3. JDBC specification.
4. JDBC steps.



JDBC API

↳ It is useful for connecting relational database.

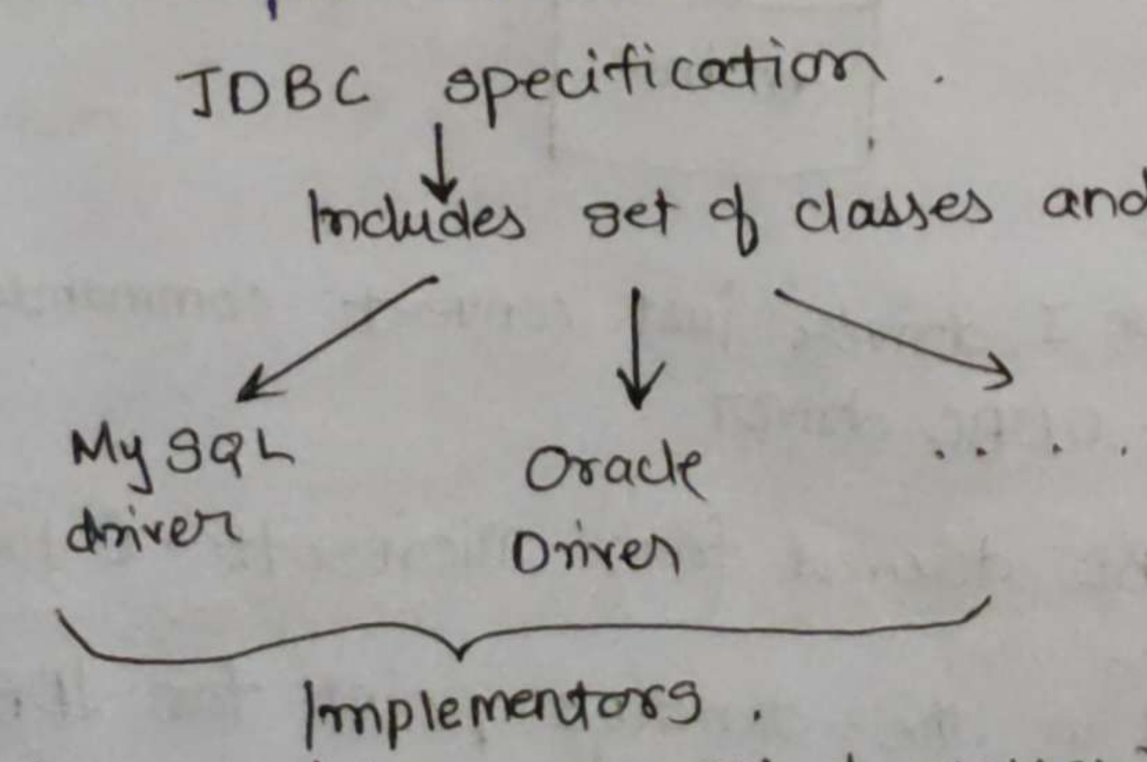


JDBC specification: It helps in converting from one db to another.

↳ It is given by Sun Microsystems.

JDBC specification Includes set of classes and Interfaces.

Interfaceses are Implemented by different vendors where this vendors works as implementors.



(Where drivers are set of classes)
They are (classes) are in jar files ie. java archieves

⇒ Database engines are different with db.

* JDBC driver is nothing but a JAR file.

Connection Interface.

↳ It is a Interface which is present in JAR file.

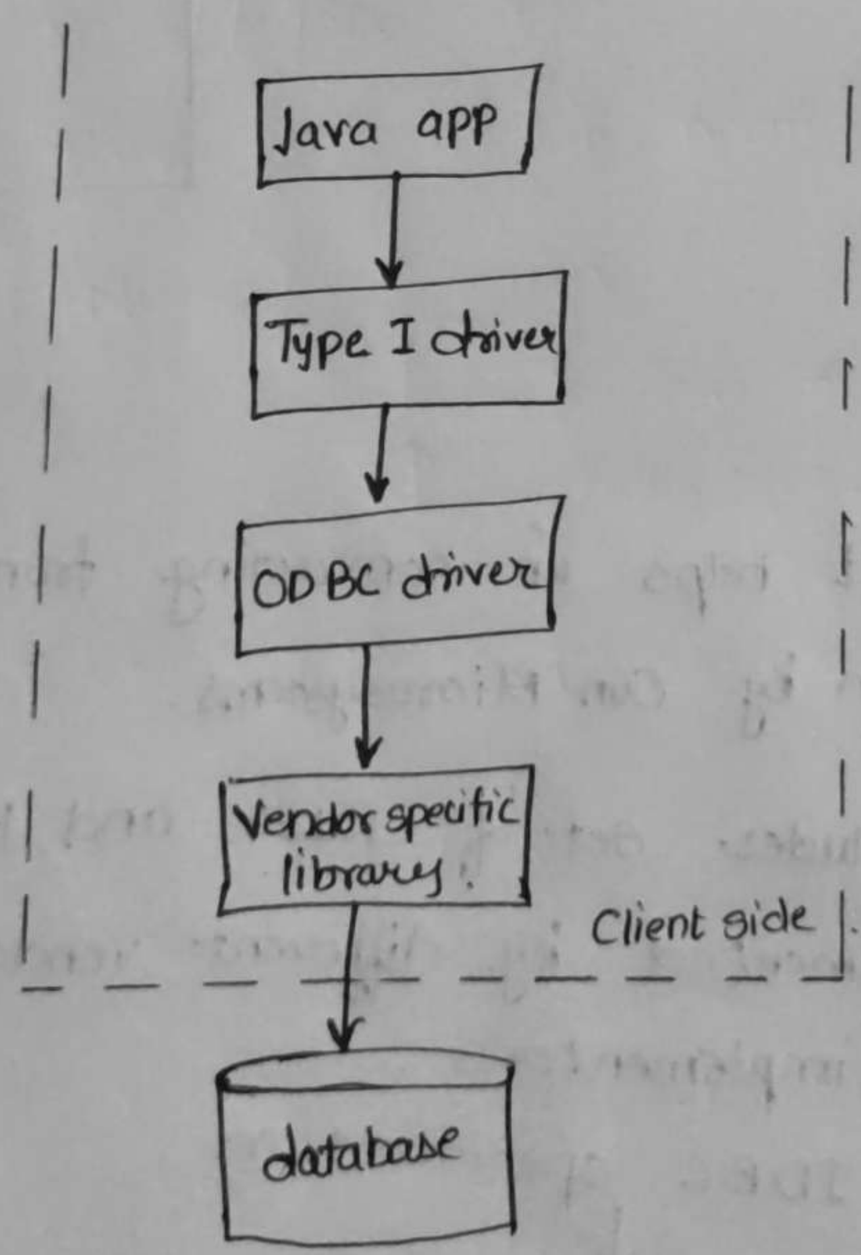
JDBC driver generation.

I. JDBC - ODBC Bridge driver.

↳ It is the first driver by Sun Microsystems.

↳ ODBC - Open database Connectivity.

↳ It is not workable for Java 8 further versions.



↳ Type I driver just convert commands from Java app to ODBC driver.

↳ ODBC derived from Microsoft - C Language.

↳ DSN is the starting point for JDBC technology.

II. Native API - partly java driver

↳ It has no dependency on ODBC anymore (Advantage)

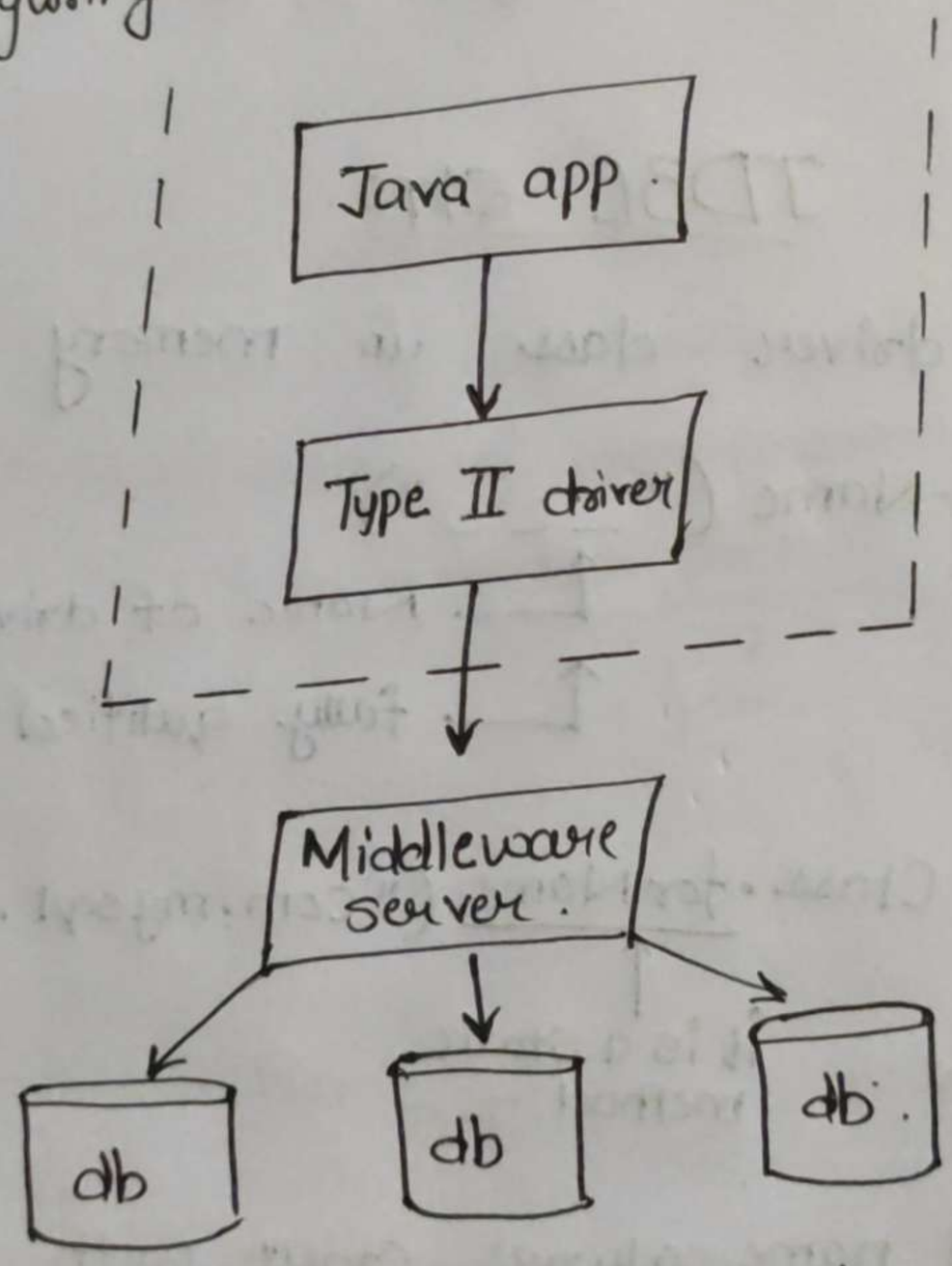
↳ It is not fully implemented in java which is one of its disadvantage.

↳ Vendor specific library is required on client side (disadvantage)

III. Network protocol - pure java driver.

↳ Fully Implemented In Java (Advantage)

↳ Need of middleware server (disadvantage).
means first vendor need to contact with middleware server. where connecting with middleware server is difficult task requires maintaing /configuring server.



↳ Most flexible - Many services - can be implemented (Advantage)

↳ No installation (vendor specific library) is required on client side. (Advantage).

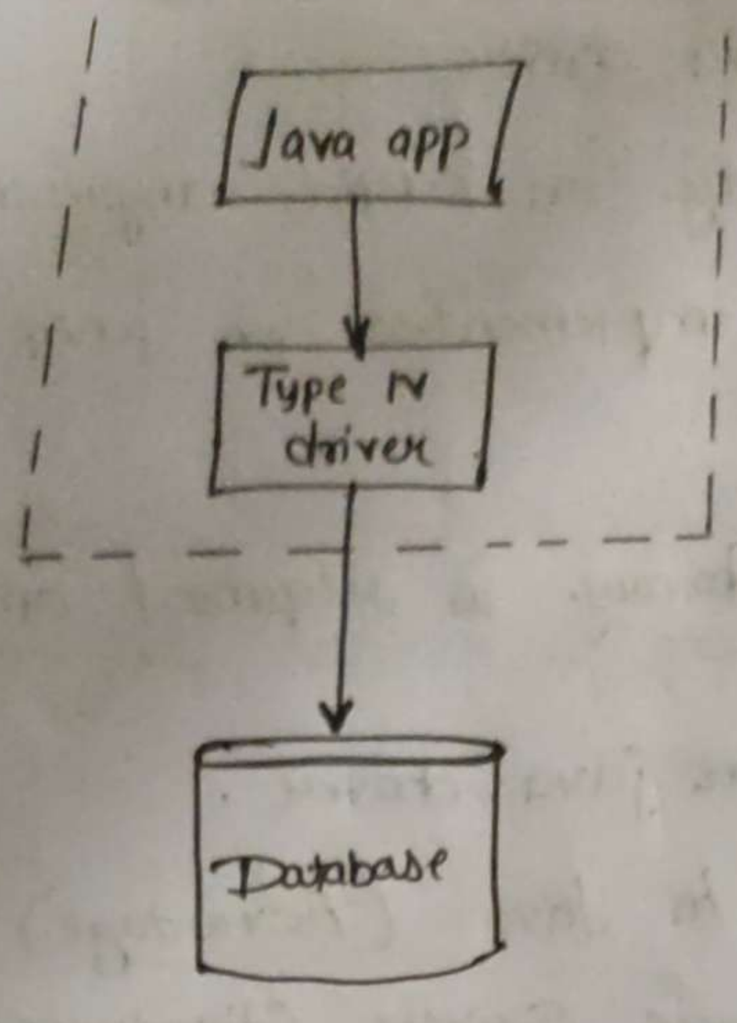
IV. Native protocol - pure java driver (thin driver)

↳ It directly connected with db (Advantage)

↳ fully implemented in java (Advantage)

↳ No client side installation is required (Advantage).

↳ No need of middleware server.



↳ Performance is better (Advantage)

↳ Driver needs to be change if db is changed (disadvantage)

JDBC Steps.

1. Loading the driver class in memory

Class.forName("___")

↑ Name of driver class.

↑ fully qulified name.

eg.

Class C = Class.forName("com.mysql.cj.jdbc.Driver");

↑ This part is not written because it is used to check class.

↑ it is a static method.

Note: fully qualified name always starts with root package followed by . for further packages.

If name of driver class is not written properly then it will give. "ClassNotfoundException".

→ Working of forName()

↳ ① Class get loaded in memory

* ② Static block get executed - registers the driver with Driver-Manager.

2. Establishing connection with database.

Connection con = DriverManager.getConnection(①, ②, ③)

Connection → interface present at java.sql.

DriverManager → it will return the instance (object) which is of the class which has implemented connection Interface

- The class is present at JDBC driver jar file. → (database vendor)
- Sun microsystems.

Where,

① → JDBC url
- ↳ url of any source (website)
- ↳ database identification.
- ↳ format → protocol : sub protocol : subname
  - protocol → jdbc
  - sub protocol → vendor specific protocol
  - subname → on what machine / → path of db server.

② → User name.

③ → password.

Example:

```
Class.forName("com.mysql.cj.jdbc.Driver");
String jdbcurl = "jdbc:mysql://localhost:3306/test";
Connection con = Drivermanager.getConnection(jdbcurl, "root", "root");
```

Drivermanager →
- only class - JDBC API
- Utility class → Static methods
- Maintains list of registered driver

Driver manager : It will check the all the drivers and registers valid drivers
↳ utility class

That's why step 1 is related with step 2.

And this will all happen in static block.

⇒ N number of driver can be registered with driver manager.

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

↳ ① loads driver class in memory
② register itself with DM.

Note: 1. If step 1 is not done then step 2 gets failed.
2. Step 1 will throw fileNotfound Exception.
3. Step 2 will throw SQLException.

Steps in Eclipse
↳ 1. new
↳ 2. project
↳ 3. Java project
↳ 4. Java SE -17
↳ 5. property
↳ Build path
↳ libraries.
↳ Add external JARs
Select lib
Select my-sql Connection
Create class

Note: We can't use try-catch or throw when class is overload

3. Creating the statement.
↳ It is nothing but encapsulating SQL query.

↳ Steps:
ⓐ Statement
↳ executing query without any parameters

It is used when requirement from user.

ⓑ Prepared Statement
↳ executing query with parameters (? as placeholder)

ⓒ Callable Statement
↳ useful for executing store

*CCEE
4. Executing Statement.

↳ ⓐ executeQuery () ← used for type of query.

ⓑ executeUpdate () ← used for DM / DDL type of queries.
↑ generally DDL is not done by Java use.

ⓒ execute () ← can be used ← especially.

Imp CCEE ↓

1. Return type of executeQuery () is ResultSet
2. Return type of executeUpdate () is
3. Return type of execute () is boole

Example :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class TestConnection {
    public static void main (String [] args) {
        try {
            // Load the driver class
            Class.forName ("com.mysql.cj.jdbc.Driver");
            // Establish the connection
            String jdbcUrl = "jdbc:mysql://localhost:3306/car-pool";
            String username = "root";
            String password = "root";
            Connect con = DriverManager.getConnection (jdbcUrl, username,
                                                       password);
            // Print a success message.
            System.out.println ("Connection successful");
```

red procedure / function.

Create a statement and execute a query.

```
Statement stmt = con.
                 createStatement();
```

interface java.sql

implement Statement

```
String sql = "select * from pass";

ResultSet rs = stmt.
               executeQuery(sql);
```

interface in java.sql

implemented ResultSet

ResultSet

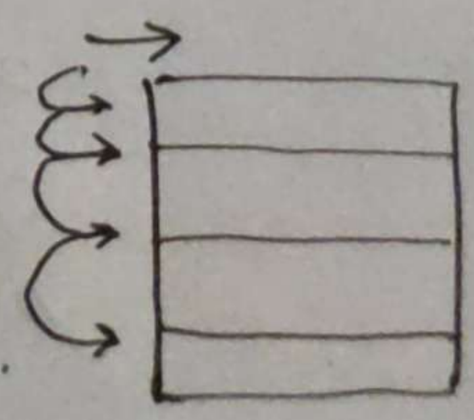
int.

olean.

Process the results.

```
while (rs.next()) {
int passId = rs.getInt("passId");
String name = rs.getString("name");
System.out.println("Pass ID" +
    passId + " Name :" + name);
```

```
java.sql.Statement stmt = con.createStatement();
String sql = "select * from passenger";
ResultSet rs = stmt.executeQuery(sql);
    // Retrieve data by column name.
while (rs.next()){
    int pass_id = rs.getInt("pass_id");
    String passenger_username = rs.getString("passenger_username");
        // Print the retrived data
    System.out.println("passenger ID:" + passenger_id + "userNam" +
    passenger_username);
}

catch (ClassNotFoundException e)
{
    SOUT("Driver class not found. Please ensure the MySQL
            JDBC driver is in our classpath");
}
catch (SQLException e)
{
    SOUT("connection error:" + e.getMessage());
}
finally {

    try
    {
        Connection con = null;
        if (con != null)
        {
            con.close();
        }
    }
    catch (SQLException e)
    {
        SOUT("Error closing connection:" + e.getMessage());
    }
    }
}
}
```



Note: record pointer is before of the first record

Note: The column which is fetch is always starts with 1.
ie, rs.getInt(1);

29/12/23.

Agenda.

1. executing simple queries.
2. executing DML queries.
3. prepared Statement
4. setting parameters
5. converting resultset to collection.

Projection of table:

filtering of table column is called projection of table.

Process resultset.

```
While (rs.next())
{
    sysout (rs.getInt(1));
    sysout (rs.getFloat(2));
    sysout (rs.getDate(3));
}
```

For Updation.

```
int n = stmt.executeUpdate("update emp set salary = 20000
        where empid = 10");
sysout (n + "records updated");
```

Example:

```
try {
    Class.forName ("com.mysql.cj.jdbc.Driver");
    String jdbcurl = "jdbc:mysql://localhost:3306/
                     test";
    Connection con = DriverManager.getConnection
                     (jdbcurl, "root", "root");
    System.out.println ("Connection esablished");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from
                   emp");
    While (rs.next())
    {
    Sout(rs.getInt(1));
    Sout(rs.getString (2));
    Sout(rs.getDate (5));
    Sout(rs.getFloat(6));
    }
```

```
int n = Stmt.executeUpdate("Update emp set comm = 500 where
    SOUT (n + " records update");
}
catch (Class Not Found Exception e)
  {
      SOUT ("driver class not loaded");
  }
catch (SQLException e)
{
    SOUT ("Connection rejected");
}
finally
 {
     try
     {
```

Prepared Statement — Step 3.

Executing queries with parameters where parameters are represented as ? ie.

~~Executing queries with parameters (?) = con.prepareStatement~~

Statement

PreparedStatement ps = con.prepareStatement ("select * from emp where deptno = ?");

interface in java.sql

Precomplied query.

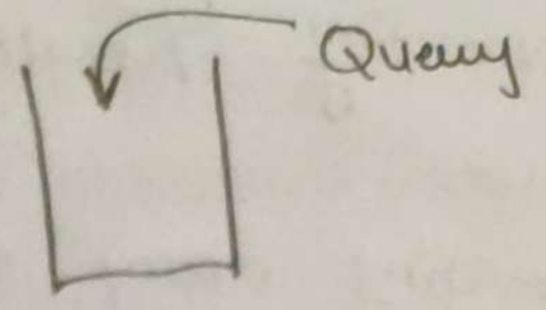
```
int dno = Integer.parseInt (br.readLine())
    ps.setInt (1, dno);
int eno = Integer.parseInt (br.readLine())
float n-sal = Float.parseFloat (br.readLine());
  ps.setFloat (1, n-sal);
  ps.setInt (2, eno);
```

error.

→ Prepared Statement

↳ It creates container with query

↳ If query is wrong gives error before hitting the db.

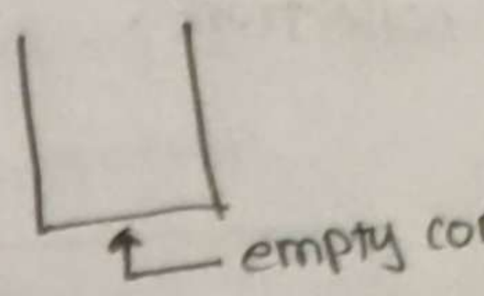


→ Statement.

↳ It creates container that is empty.

↳ If query is wrong gives error after hitting the db.

That's why very nearly statement is used over prepared statement.



Execute Statement — Step 4.

~~While~~
ResultSet rs = ps.executeQuery();

While (rs.next)
{

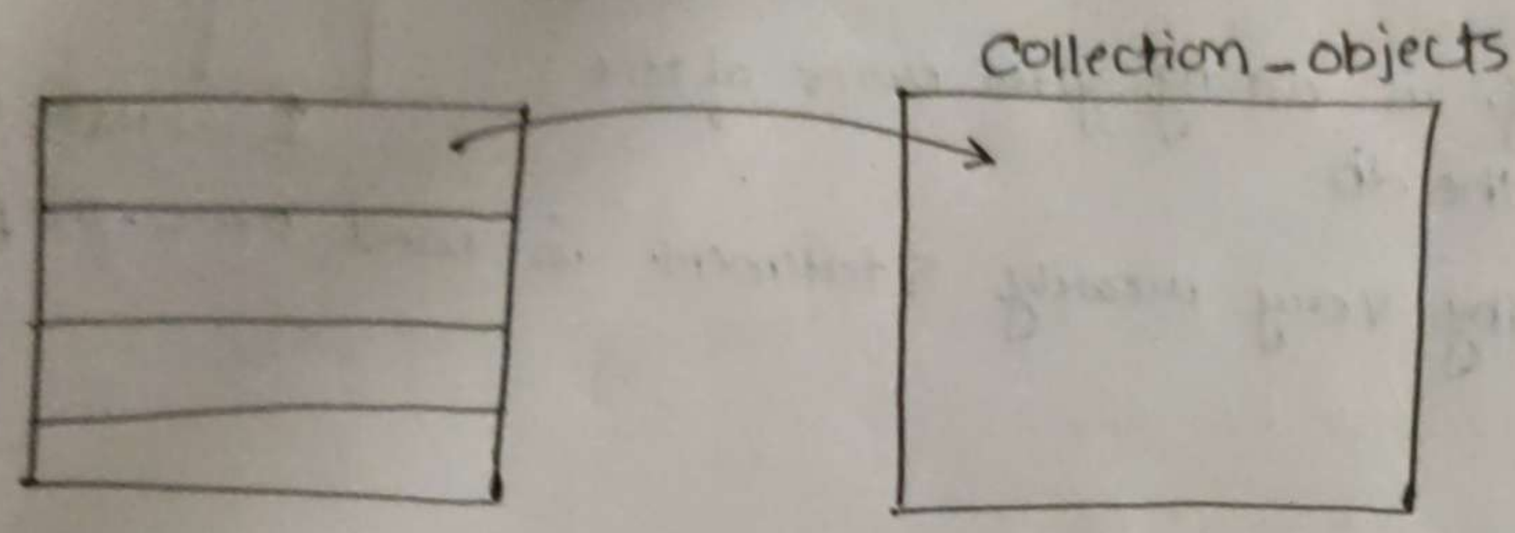
}

rs.next();
rs.get.

Converting Resultset into Collection

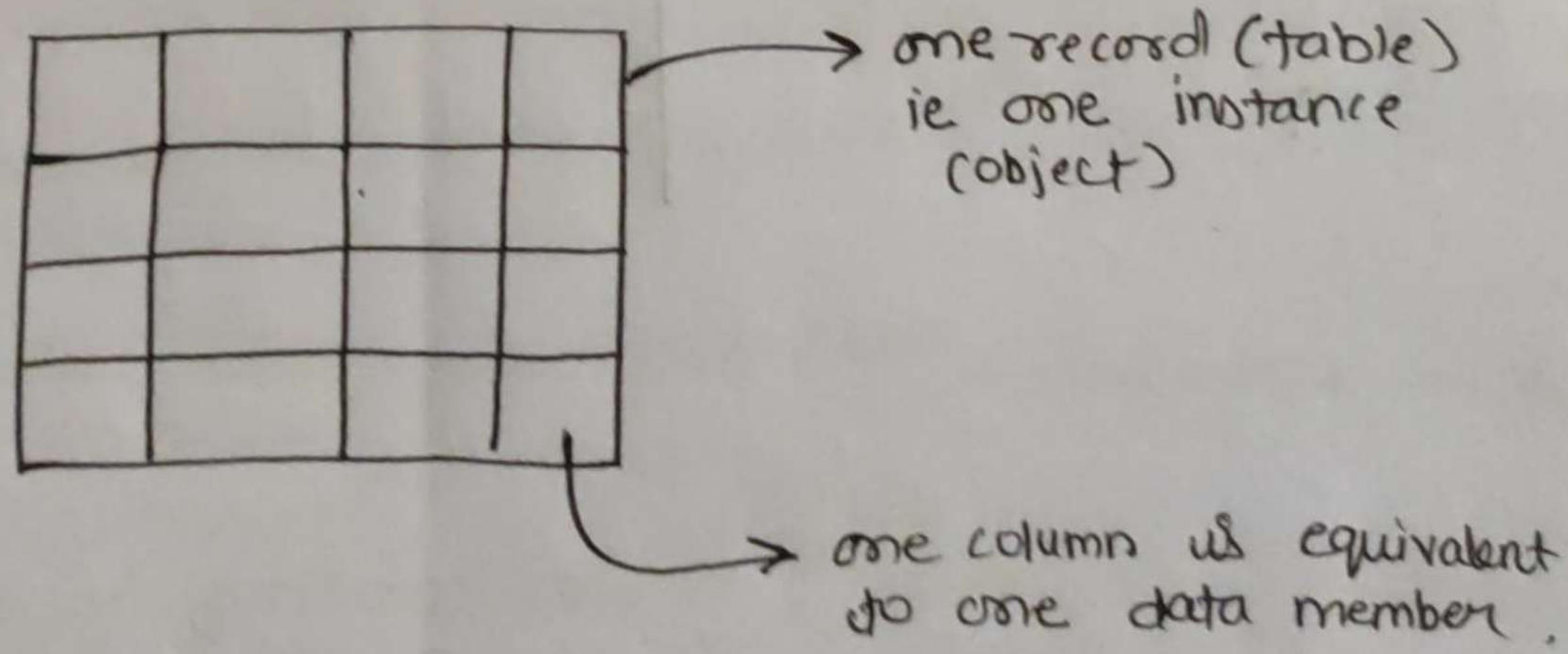
↳ it is used for OR mapping tool
Object Relational

Converting every set of table and then adding into collection which can be used in web development or processing of this collection.



Choosing the collection depends on the requirement in which format we want to convert the database data into collection.

Where one record of the database table is equivalent to one instance and one column is equivalent to one data member (variable)



Class = table.

instance = record.

property = column.

```
Set <Emp> emps = new HashSet();
While (rs.next())
{
    Emp e = new Emp();
    e.setEmpno(rs.getInt(1));
    e.setEname(rs.getString(2));
    =
    emps.add(e);
}
```

```
Class Emp implements Comparable
{
    int empno;
    String ename;
    float salary;

    //default constructor

    // parameterized constructor.

    // getter //setter.

    public
```

Callable Statement

It is used to execute stored procedures.

→ named PL/SQL block.
→ compiled and saved in the db.
→ frequently required tasks.

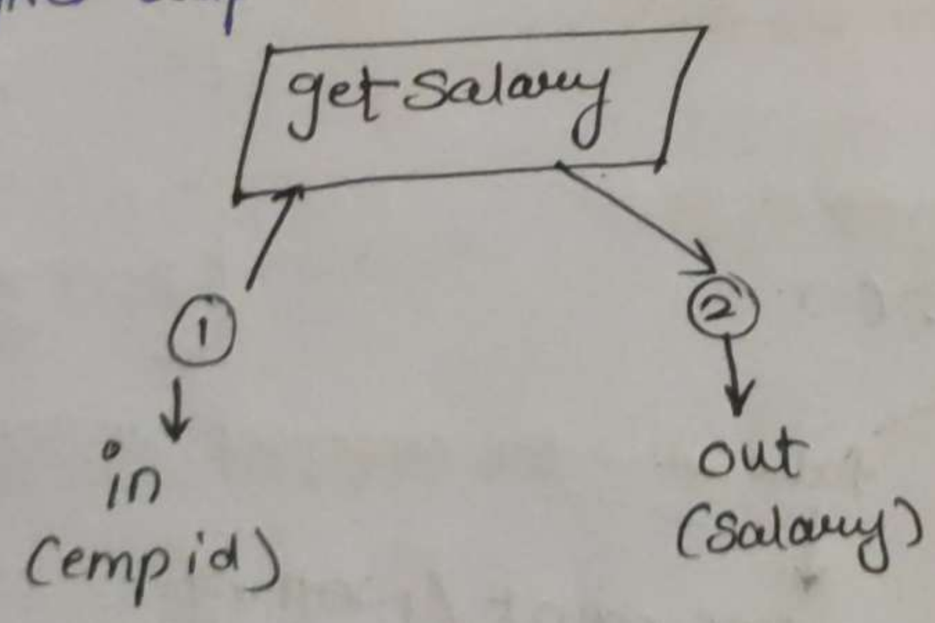
Agenda.
30/12/23.

1. Using callable Statement
2. Scrollable and updatable result sets.
3. transaction management.

→ Java application → Stored Procedures.
↓
SQL query

→ Statement + PS — Java + SQL
→ any SQL part
→ Stored Procedures.

Let take a stored procedure getSalary which takes input as empid and give output as salary.



To use stored procedures for Java connectivity with db we uses Callable Statement : ~~Statement~~
↑
P.S
↑
Callable Statement cs = con.prepareCall(" ")
↓
interface.
java.sql

in out

```
CallableStatement cs = con.prepareCall("{call getSalary(?,?)}");
```

curly brackets tells that this is not a SQL query.

⇒ for in parameter set method is required.

⇒ for out parameter registerOutParameter is required.

Taking Input using buffer.

```
int eno = Integer.parseInt(br.readLine());
cs.setInt(1, eno);
cs.registerOutParameter(2, Types.FLOAT);
```

mapping sql data types.

java.sql

```
                        , Types.VARCHAR);
                        , Types.INTEGER);
```

capital letters means it is final type.

When knowing the method to use

```
boolean res = cs.execute()
```

← Usefull for DDL statements

get resultset() ← true → resultset

false ← for empty set or for Insertion → update count / Nothing

empty set.

```
if (res)
{
    ResultSet rs = cs.getResultSet();
}
```

if true

OR

When don't know which method to use

```
cs.execute();
float sal = cs.getFloat(2);
```

exQ → Statement

exeU → PS.

exe → CS.

When datatype is compatible then we use inout.

getDept
↓
1
↓
inout
↗ setter ↖ registerOut parameter.

eg.

```
Callable Statement cs = con.prepareCall("{call getDept(?)}");
int eno = Integer.parseInt(br.readLine());
cs.setInt(1, eno);
cs.registerOutParameter(1, Types.INTEGER);
cs.execute();
int deptno = cs.getInt(1)
```

db(test)
↳ table( )
↳ right click
↳ click SP.

Java application.

```
public class CallableDemo {
  pusvm( )
{ Class.forName("      ");
  String jdbcurl = "      ";
  Connection con = Driver manager.get
      Conne "      "

  Callable Statement cs = con.prepareCall
  ("{call getsal(?,?)}");

cs.~~setInt(1,0);~~
  BufferedReader br = new Buffered
                Reader(new Input
  Stream Reader(
    SOUT("Enter emp no");
    int eno = Integer.parseInt(br.
                readLine());
    cs.setInt(1, eno);
    cs.registerOutParameter(2,
    Types.FLOAT);
```

db.

```
Create procedure 'getSalary'
(in eno integer, out salary float)
BEGIN
  Select sal into salary from
   emp where empno = 123;
END
```

```
cs.execute();

float sal = cs.getFloat(2);
SOUT ("Salary" +sal);
 cs.close();
 cs.close();
}
}
```

db.

```
Callable Statement cs = con.prepare
Call (" {call getEmpInfo(?,?,?)
Buffered "      "
SOUT ("Enter emp no");
int eno = Integer.parseInt (br.readline());
 cs.setInt (1, eno);
cs.registerOutParameter (2, Types.FLOAT);
cs.registerOutparameter (3, Types.VARCHAR);
es.execute();
float tsal = cs.getFloat (2);
string email = cs.getString (3);

SOUT (tsal);
SOUT ( email);
 cs.close();
 con.close();
```

```
Create procedure 'getEmpInfo'
(in eno integer, out tsal float,
out mailid
Begin
Select sal + ifnull (comm,0),
email into tsal, mailid from
emp where
```

Scrollable and Updatable Result Sets.

Result set :. Forward only ← method next() is used.

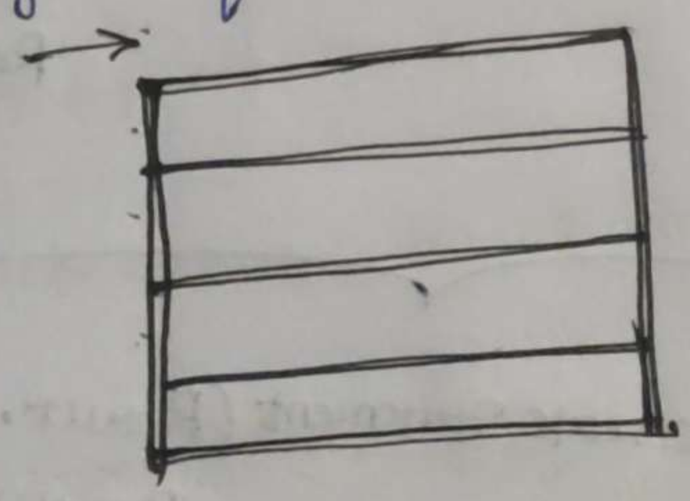
Read only ← can not modify with the use of get() ← column index.

Result set : scrollable

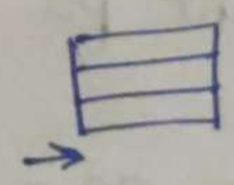
if Resultset is made scrollable then we have following functions.

- next()
- → previous()
- → first()
- → last()
- → absolute()
- → relative()
- → before first()
- → after Last().

Note: Pointer always before first set in the db.



1. rs.absolute(4) ← point from zeroth record.
2. rs.relative(-2) ← point from current record.
3. rs.afterLast() ← goes out of last.



4. rs.previous() ← goes to previous in opposite direction.

eg. while (rs.previous())
{
}

Resultset : Updatable

↳ update XXX ( )

↳ update the ~~table~~ value in result set.

table.

updateRow ( ).

Making resultset scrollable and updable.

↳ specified at creating statement.

★ Statement stmt = con . createStatement (① , ②);

① → resultset type.

② → resultset concurrency.

✓ ResultSet . TYPE_FORWARD_ONLY

Resultset . TYPE_SCROLL_INSENSITIVE

Resultset . TYPE_SCROLL_SENSITIVE

✓ ResultSet . CONCUR_READ_ONLY

Resultset . CONCUR_UPDATABLE

Statement stmt = con . createStatement (Result . TYPE_SCROLL_INSENSITNE, Resultset . CONCUR_UPDATABLE);

PreparedStatement ps = con . prepareStatement (query , type , concurrency);

CallableStatement cs = con . prepareCall (String , resultset type , resultset concurrency);

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

rs . absolute (5); ← 5th row

rs . updateFloat (6, 5000.00f); ← column index

rs . updateRow ();

Transaction Management

Transaction betn two db is done with help of commit.

Transaction is done as all or none.

In transaction the requirement is that if first query fails then second manage it and if next fails then it get rolls back to previous queries.

Where autocommit mode is alway true for automatically commit.

```
con.setAutocommit(false);   ← taking autocommit off and taking it our hands.
try
{
    // insert into t1
    // insert into t2.
    // insert into t3.
    con.commit();   ← when pointer comes to this means all above commands are successful.
}
catch (Exception e).
{   con.rollback();   ← it got failed
}.
Con.setAutoCommit(true);
```

executed as transaction.

Q CCEE Autocommit is false or true at the time of transaction?

Java I/O Streams

1. In Java streams are the sequence of data that are read from the source and written to the destination.

2. Stream
   - Input Stream
   - Output Stream

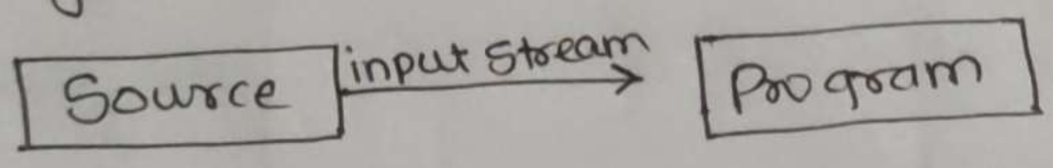
3. Input Stream: An input stream is used to read data from the source.

4. Output Stream: An output Stream is used to write data to the destination.

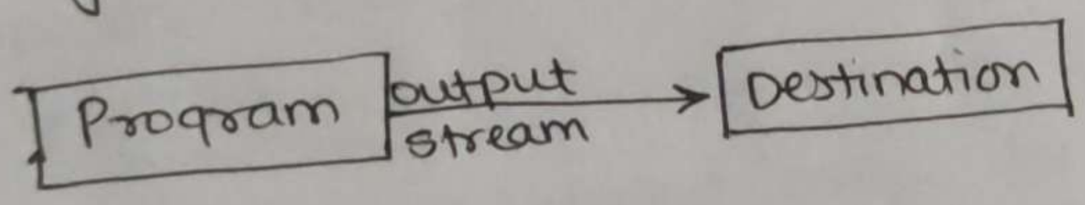
5. Where, System.in is Input Stream.

6. System.out is Output Stream.

7. Reading data from source.



8. Writing data to destination.



Types of Streams

Depending upon the data a stream holds, it can be classified into:

1. Byte Stream
2. Character Stream.

Agenda.
02/01/24

1. executing stored function.
2. batch updates.
3. meta data.
4. transaction isolation.

CCEE

I. executing Stored function.

Stored procedures never returns value
Stored functions returns value

3. Callable Statement cs = con.prepare("{ ? = call getGrossSal(?) }");

```
cs.registerOutParameter(1, Types.FLOAT);
cs.setFloat(2, sal);
```

4. cs.execute();
   float gross = cs.getFloat(1);

5. sysout("Gross sal:" + gross);

II. Batch Updates.

→ .batch file

→ .batch - few commands.

→ batch update - multiple SQL queries one after another in single execution.

→ batch updates are restricted only for DML statements.

→ ~~be~~ Select queries are not allowed in batch updates.

3. Statement struct = con.createStatement();
   struct.addBatch("insert into ---");
   struct.addBatch("insert into ---");
   struct.addBatch("insert into ---");

   } batch of queries

→ Return type of this is integer.

4. int[] updatecount = struct.executeBatch()

Batch Updation Exception.

↳ If any exception happen in between then it will give batch updation exception. Which should be handled by us.

↳ It executed within a transaction we take the control of commit or rollback.

↳ If no exception is raised commit all updates.

5. Prepared Statement ps = con. prepare Statement ("insert into t1 (?,?,?)");

```
ps. setInt (1, n);
ps. setString (2, str);
ps. setFloat (3, f);
ps. addBatch ();
ps. setInt (1, n1);
ps. setString (2, str1);
ps. setFloat (3, f1);
ps. addBatch ();
```

The changes here are parameter changes.

III. Meta Data.

- additional information.
- about resultset, connection.

resultset:
- getColumnCount
- getColumnType.
- getColumnName.

connection:
- Database major version.
- Database minor version
- Driver major version.
- Driver minor version.
- Isolation level.
- supports Stored Procedure.

Result Set Meta Data.

1. Object of this gives result sets like:
   i. get column count
   ii. get column Type.
   iii. get Column Name.

```
Statement struct = con. create Statement();
ResultSet rs = struct. execute Query
                ("select * from emp);

Result-set Meta Data rsmd = rs. get Meta
                            Data();

int col-count = rsmd. get Column Count
                ();

while (rs. next ())
{
 // one record = col-count columns
   for (i=1; i<=col-count ; i++)
   {  sout (rs. get string (i));
   }
}
```

OR.

```
for (i=1; i<= col-count; i++)
{ SOUT (rsmd. get Column Name (i) +
        " : " + rsmd. get Column Type
        (i));
}
```

Database Meta Data.

1. Object of this gives Database connection ~~result~~ sets like:
   i. Database " "
   ii. " "
   iii " " "
   "
   "

```
Database Meta Data dbmd =
        con.get Meta Data();
```

IV. Transaction Isolation Levels. most imp ccec

Type of transaction for any transaction is done ~~by~~ or decided by transaction Isolation levels.

1. Dirty Reads:

When one transaction is not completely commited but still the second tranansaction can read the inserted record of the first transaction is called dirty reads.

Transaction A	Transaction B.
→ inserted one record	→ read inserted record
→ rolled back.	When uncommitted records are allowed.

2. Repeatable Reads:

When one transaction is not completed but at the same time another transaction proceded which will resulted into addition or reduction of records from first transaction.

Repeateable reads are for ~~set~~ usefull set of records.

Transaction A	Transaction B.
→ reading a set of records.	→ inserting new record matching the condition
→ reading same records for a particular condition	

Wheather or not transaction should happend or not is decided by TIL.

3. Phantom Reads.

Transaction A

→ Read a particular records.

Transaction B.

→ will update same records.

→ Diff values for same records.

Isolation Levels.

↳ 1> TRANSACTION - NONE

↳ 2> TRANSACTION - READ - UNCOMMITED.

↳ 3> TRANSACTION - READ - COMMITTED.

↳ 4> TRANSACTION - REPEATABLE - READ.

↳ 5> TRANSACTION - SERIALIZABLE.

Sr. No.	Dirty reads	Repeatable reads	Phantom reads.
1> Transaction - None			✓
2> Tran^n - read - uncomm	✓	✓	✓
3> Transa^n read - comm	X	✓	
4> Tran^n repetable - read	X	X	✓
5> Tran^n serializable.	X	X	X

Note: Consistency increases as we go down but concurrency and performance decreases because of more locks on records.

↓ consistency increases.
concurrency decreases.] more locks on records.
performance decreases

For Connection Set.

- getTransactionIsolation()

For check whether supported.

- setTransactionIsolation()

CCEE: JDBC drivers.
↳ statement, execution.
↳ transaction isolation.

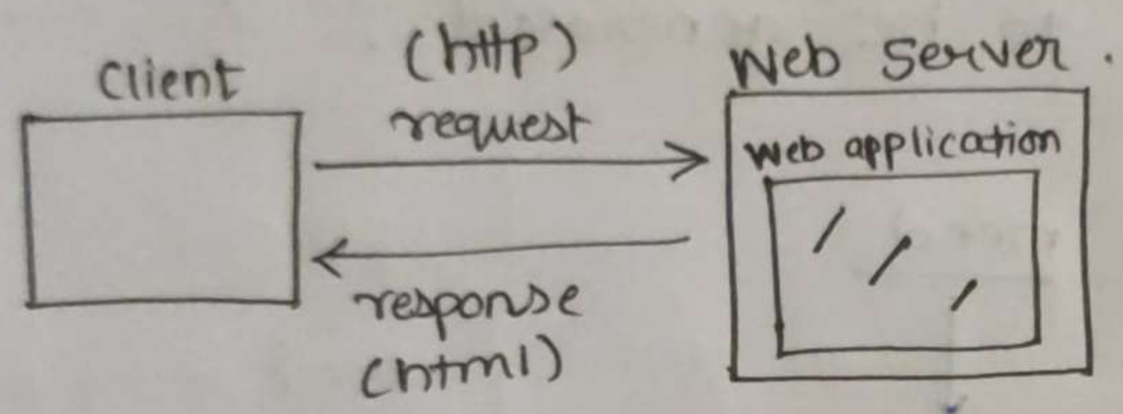
⇒ Maximum isolation level is 8.

Servlet

Agenda.

1. Web server
2. Static and dynamic request
3. What is servlet
4. creating simple servlet
5. deployment and execution
6. servlet life cycle.

1. This is apart from console based application
2. It is all about web application.
3. It is a type of client-server architecture.
4. In this we use Java to developing web application.



5. Web applications are residing web server.
6. Web Server: It is some software which control machine to behave as web server.

It is a machine which is in possition to communicate with the clients.

Client communication — Accept request, Send response.

Apache: It is one of the web server which is used for servlet.

7. Web server can't generate dynamic response.

Suppose we have request like

http://www.google.com. ← This response is ready available

It is not dynamically generate, it is similar to all such type of request is called Static request response cycle.

In this response is ready available.

8. ~~Dynamic~~ Dynamic Request.

http://www.gmail.com/inbox.

This is not ready available this need to be generated dynamically at run time.
In which server gets failed alone for which it needs help.

i. Response is going to be different for diff. user.
ii. Response needs to be generated.

For which web server need

↓

programs java (java class)

Servlet = web server + java class (program java) → can generate dynamic response.

Servlet
↑
Generic Servlet
↑
HTTP Servlet.

Q. What is Servlet?

1. Servlet is a Java class which is always under Server.
2. Java class always residing in the environment of web server.
3. We used it generate dynamic response (html).
4. Life cycle of Java class is under the control of web server.
5. It gets executed only after receiving request from client.

6. The main difference betn Java class and servlet is that servlet don't have main method it is totally under server.

Simple Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
url → @WebServlet("/hello")
      public class HelloServlet extends HttpServlet
      {
        @Override
        protected void doGet(HttpServletRequest req,
                             HttpServletResponse resp) throws
                             ServletException, IOException
        {
          PrintWrite out = resp.getWriter();
          out.print("<h1>Hello from servlets</h1>");
        }
      }
```

annotation

for getting response from server.

⇒ Request, Response objects will be created by server.

⇒ Our work only is to give a proper structure and deploy in the server.

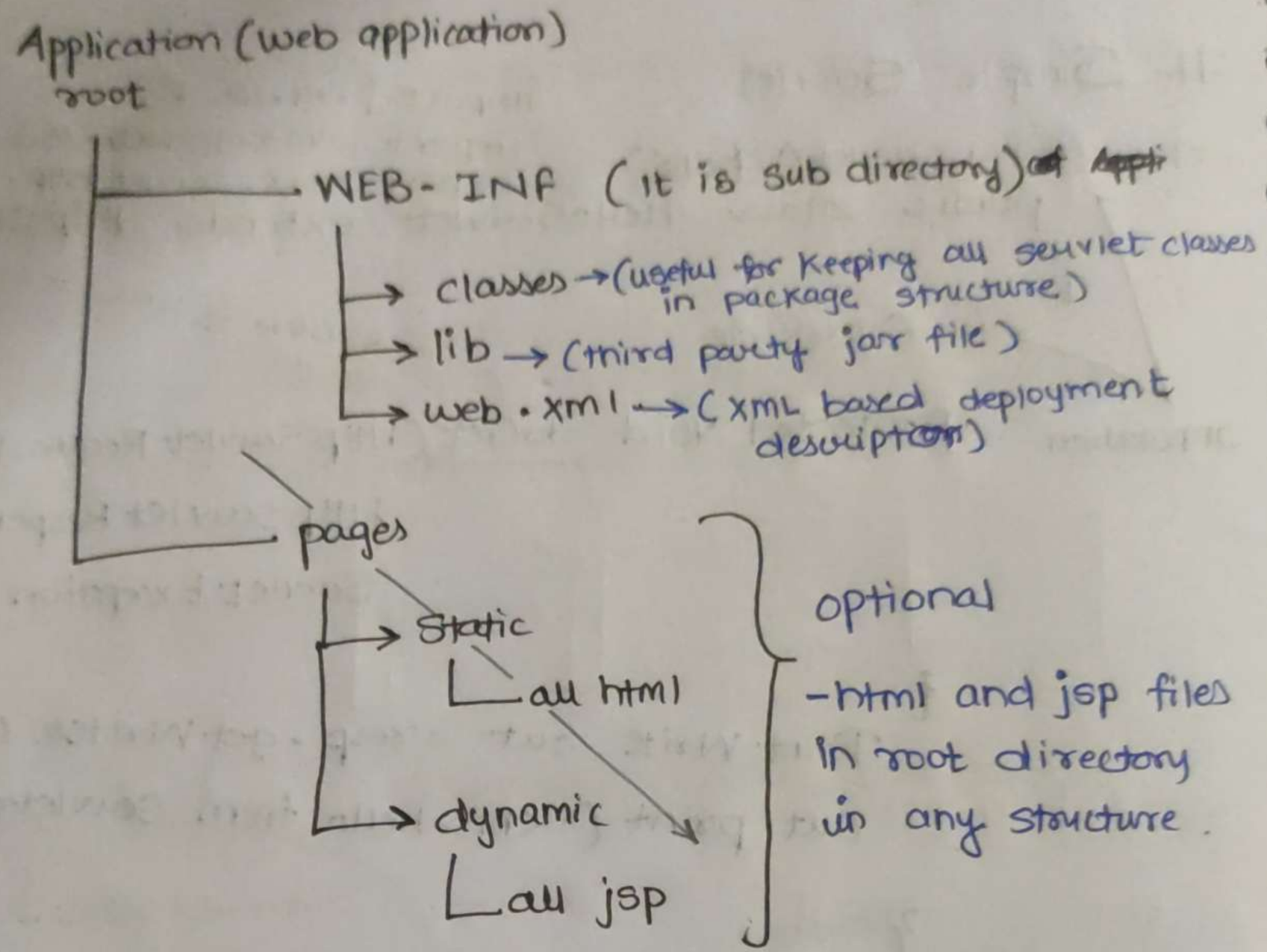
⇒ @web servlet(" ") ← it is a request.

protected void doGet(" ") ← it is request method.

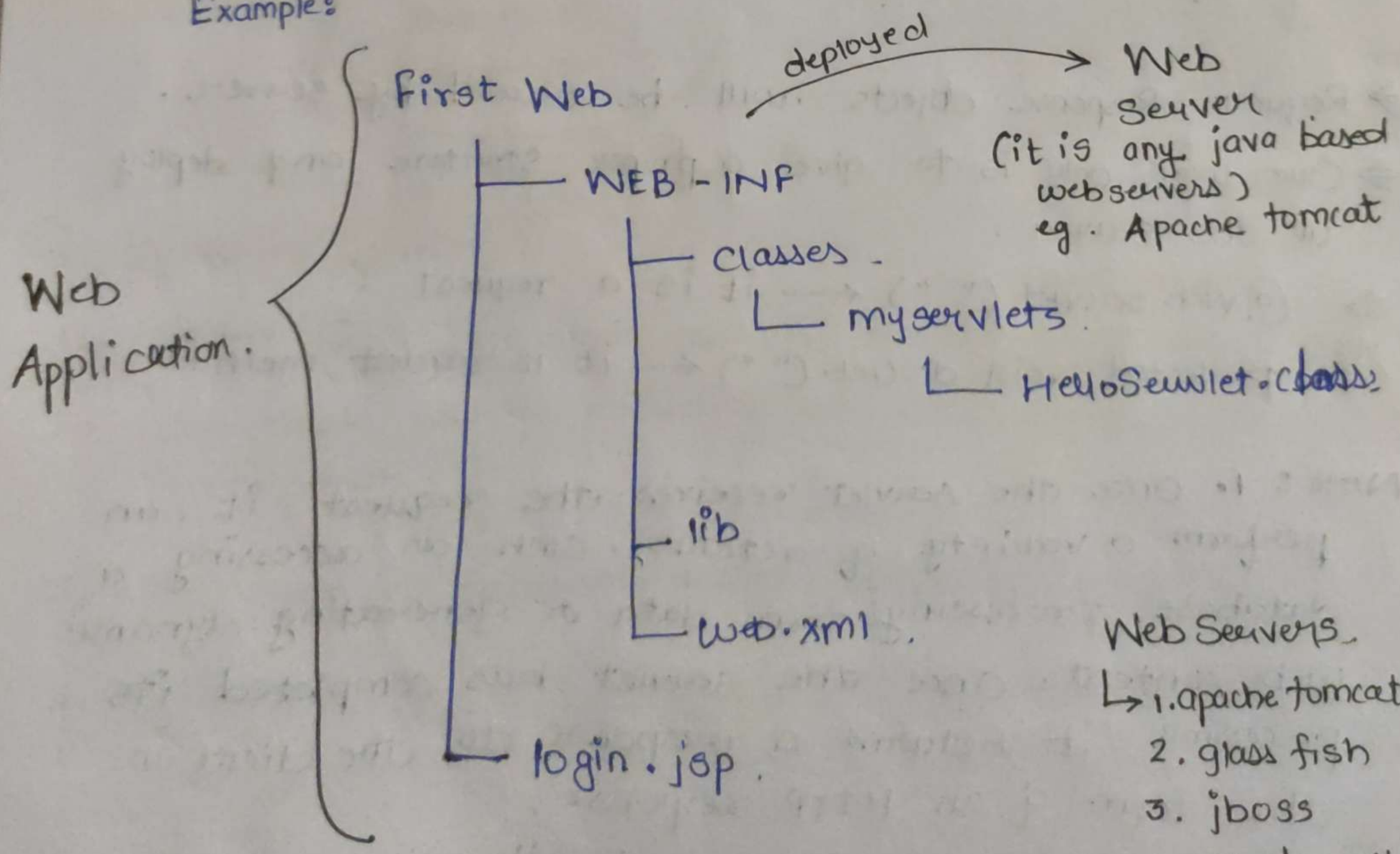
Note: 1. Once the servlet receives the request, it can perform a variety of actions, such as accessing a database, processing form data or generating dynamic web content. Once the servlet has completed its processing, it returns a response to the client in the form of an HTTP response.

2. They can be configured to handle different types of HTTP requests such as GET, POST and PUT requests. They can also be configured to handle diff. type of data HTML, XML & JSON

Web Application Structure.



Example:



Web Servers.
↳ 1. apache tomcat
2. glass fish
3. jboss
4. Sun's application server.

Web server.

→ deployment directory.

→ webapps → first web

apache tomcat

different server has diff. names.

Request Syntax :

default port no. of tomcat

http : // localhost : 8080 / first Web / hello

- http → protocol
- localhost → host
- first Web → application root
- hello → request url (@ webServlet)

project
↳ Preference
↳ Server
↳ server Runtime Environment
↳ Add Apache (v9.0)

☆ perspective
↳ java.
↳ java EE

http : // localhost : 8080 / ← home page of tomcat

Dynamic web project
↳ first web
Target runtime : Apache Tomcat v9.0

04/01/24

Agenda.

1. Servlet life cycle
2. Servlet API
3. Handling request parameters.
4.

Servlet Life Cycle.

Properties of Servlet.

→ 1. java class
2. dynamic HTML (response)
3. Hosted in web server.

As a client servlet can be visible in the format of url pattern.

→ Servlet holds the business logic.

Deployment :- It is a special copy.
- It is workspace project copy ~~and~~ ~~from~~ to deployment directory of webserver (web apps)
- This operation is done by eclipse itself.

Workspace project → deployment directory of web server. (web apps)

Servlet life Cycle. * CCE

→ Client hits the URL

→ Web server will get the resource
http://localhost:8080/firstWeb/hello
http://localhost:8080/firstWeb/logic.jsp.

→ Web server maps URL pattern to servlet class.
/hello - ?

Annotation.
@WebServlet.

1. @Web Servlet → meta information about the servlet - to web servlet.

1. Annotation - metadata - begin with @ symbol.
2. Above the class / method / variable.
3. Method parameters.
4. provide compile time / deploy time / run time information
5. Give information at time of deployment
6. We servlet will identify that /hello - myservlets. Hello servlet map to appropriate servlet class.

2. @ Override.

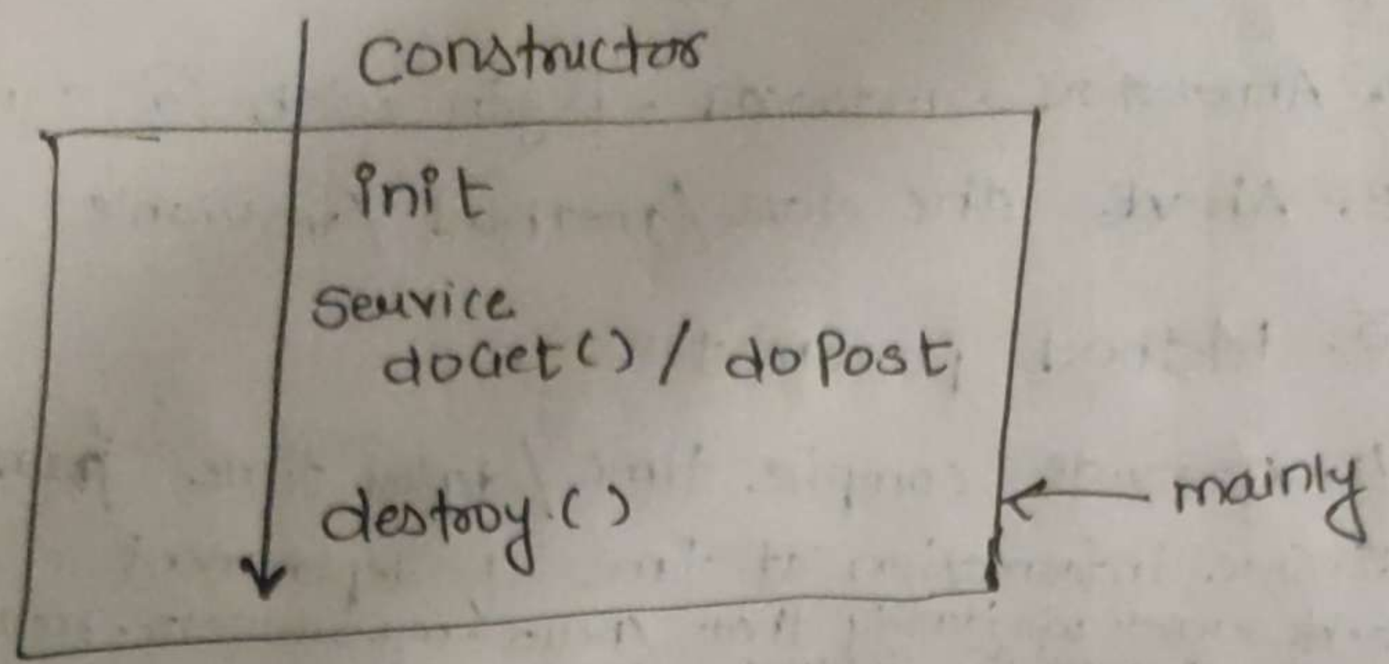
1. come from Runnable Interface.
2. Tells to compiler that this is come another

Web Server:

- Web server loads servlet class into the memory.
- Servlet class gets instantiated (servlet instance is created) (constructor).
- servlet instance will be initialized - if any resource are need - jdbc connection -

init ()
↑ it is inherit from super class.

- web server creates separate request, response objects and gives ~~all~~ a call to appropriate service method.
  ↓
  based on http method (doGet / doPost)
- Generated response will be send back to the client
- Servlet will get destroyed - using destroy()



If we send the same ~~request~~ as first request ~~is~~ so ~~first~~ step are same they are :

every request

- Web server creates separate request, response objects and give a call to appropriate service method
  ie. doget / dopost
- Generated response will be sent back to the client

But step which runs only once for same request are

first request

- Web server loads servlet class into the memory
- servlet class gets instantiated
- servlet instance will be initialize - if any resources are needed - jdbc connection
  ie. init()
- servlet will get destroyed.
  destroy()

Javax Package . ☆ cccc

Servlet — servlet life cycle init, service, destroy
↑ (Servlet is interface)

Generic Servlet — service.x — protocol independent
(All methods are implemented by Generic servlet except service.x)
↑

Http Servlet — service
— do Get(), doPost()
— http protocol communication
↑

HelloServlet — do Get()

Generic Servlet implements all servlet methods ie init, service, destroy except ~~destro~~ service x

Generic Servlet is useful to communicate while

Servlet Request (interface) → - get Parameter()
- get Port
- get Protocol

↑ extends

Http Servlet Request (interface) → - getHeader
- getCookies
- get session

Servlet Request : 1. get Parameter()
2. get Port
3. get Protocol

Http Servlet Request : 1. get Header.
2. get Cookies
3. get Session.

Servlet Response (interface) → - get writer()
- set content Type()

↑ extends

HttpServlet Response (interface) → - Set cookies()
- Set Status()

Servlet Response : 1. get writer()
2. set content Type()

HttpServlet Response : 1. set cookies()
2. set Status()

Web application
└ JRE system.
└ target runtime ×

if value ~~not available~~ is none than it will give compilation error because we need the imports the javax.servlet and javax.servlet.http.

javax.servlet
javax.servlet.http } Important.

Two things are important at Server side they are:
1. url
2. method.

We are mapping url by.
→ @web servlet

Handling Request Parameters.

* Request object → Provides method
  → Reading request parameter

request.getParameter ("uid")
→ name of form field.

Where uid is:

http://localhost:8080/firstWeb/logincheck ? uid=object & pwd=object

if not write it will give null value exception

request body.

request.getParameter ("uid")
→ name of form field.

```
@webServlet ("/logic check")
public class LogicCheckServlet extends HttpServlet
{
  protected void doPost (HttpServletRequest req,
                         HttpServletResponse resp)
  {
    String uid = request.getParameter ("uid")
    String pwd = request.getParameter ("pwd")
    ~~String~~
    if(uid.equals ("object") && pwd.equals ("object 123")
      resp.getWriter().print ("Login success");
  }
}
<form action = "http:8080/firstWeb/logincheck"
      method = "post" >
  Userid : <input type = "text" name = "uid" />
  pwd : <input type = "password" name = "pwd" />
  <input type = "submit" value = "LOGIN" />
</form>
```

In Tomcat v10
↳ Has instead of Javax to Jakarta

x.equal(y) ← reference type

x==y ← value type

src main java →

```
@ WebServlet ("/logincheck")
public class LogicCheckServlet extends HttpServlet
{
    protected void doPost (HttpServletRequest req, HttpServletResponse resp)
    {
        String uid = request.getParameter ("uid");
        String pwd = request.getParameter ("pwd")
        if (uid.equals ("object") && pwd.equals ("object123")
        {
            resp.getWriter ().print ("Login success");
        }
    }
}
```

~~Web~~ Web app →

```
<form action ="http 8080/FirstWeb/logincheck" method ="post">
Uid id : <input type ="text " name ="uid" />
Pwd : <input type = "password " name ="pwd" />

<input type ="submit " value ="LOGIN" />
</form>
```

→ Servlet request processing.

- init — first 2 steps
- service — last 2 steps.
  - doPost

→ Adding JDBC driver jar file.

- JRE system.
- server runtime
- third party jar - mysql connector

} for this two steps this jar file is required.

→ Adding - build path.

* - jar files → WEB-INF / lib - deployment assembly
  - instruction given to eclipse
  - ie. project → deployed application.

java class.

→ src/ main /java
deployment
/WEB-INF /classes.

Note: Default value of method is get method.

webapp
/
→ /WEB-INF/lib

Exception
→ try-catch block

```
try {
    class . forName ("    ");
    String jdbcurl = "jdbc:mysql://"
    con = DriverManager .getConnection "
    SOUT ("con established)
}
catch (Exception e)
{ e.printStackTrace();
```

Agenda.
5/01/25
↓

1. Using JDBC in Servlet
2. init and content parameters
3. request dispatching
4. request redirection.
5. Listeners.
6. attributes

required
↳ init
↳ doGet
↳ doPost
↳ destroy
optional

JAR
project
↓
properties
→ Build path
→ class path
→ Deployment Ass

jar
↑
JavaBuildpathentry

lib → jar file

Config (init) and context parameters.

value of request parameter is given by client via browser.

- Request parameter → String
  - name
  - value
  - initialized by client.

- init (config)
- context

} supplied at the time of deployment.

- initialized by web developer.
- name (String) - value (String)

Common betn Request parameters and config is both are String types.

★ Config → specific for one servlet.
↳ @WebInitParam.

★ context → applicable for all servlets in one web application.
↳ Configured in web.xml.
↳ Deployment Description.

WebInit provides

There are two seperat objects created by web server.

↳ config

↳ context.

⇒ Web server creates object (instance) of servlet.

⇒ config & content objects.

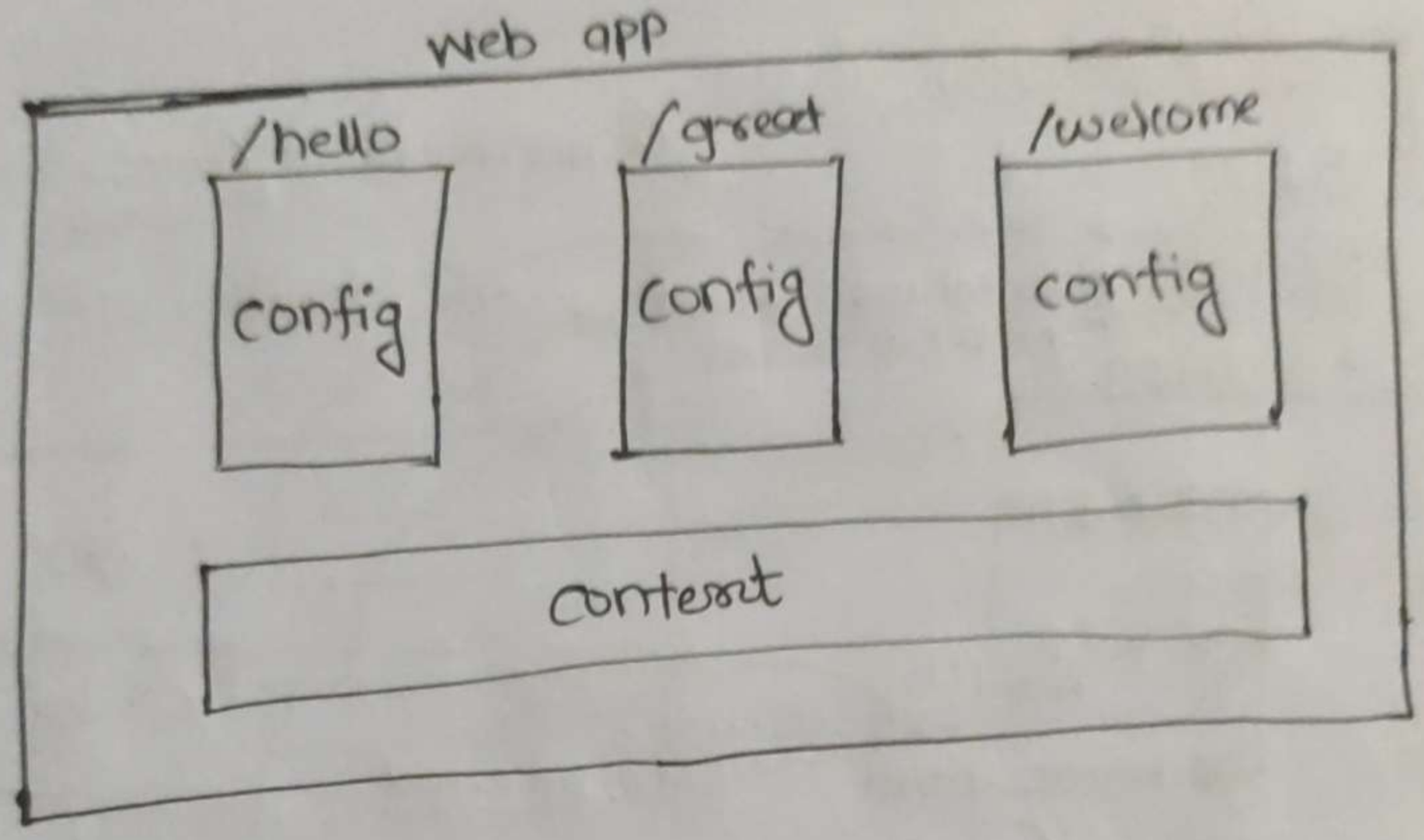
↳ created by web server.

↳ made available to every servlet.

⇒ config - specific for one servlet.

⇒ content - specific for one web application

As many web app as many web objects created by webserver.



For Generating the response.

/home.html

PG - DAC modules.

Core Java - /Java.

Database - /dbt

Web Programming - /wpt

/java.

@webInitParam.

subject = - - -

subject = - - -

subject = - - -

Web.xml

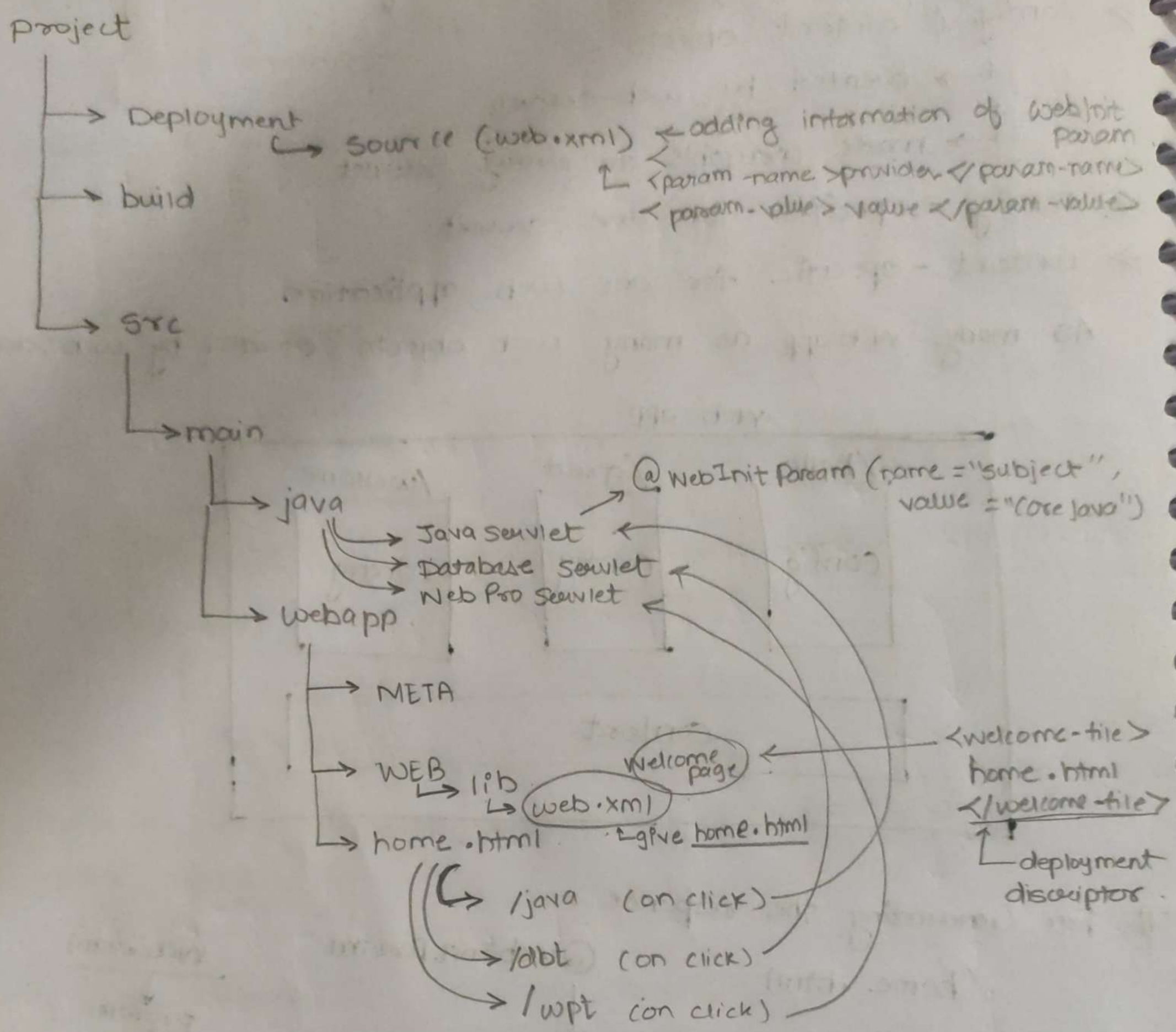
↓

provider

↓

KnowIT, CDAC ACTS.

```
Protected void doGet(- - - -)
{
    PrintWriter out - -
    ServletConfig config = getServletConfig();
    String sub = config.getInitParameter("subject")
    ServletContext content = getServletContext()
    String provider = content.getInitParameter("provider");
    out.print("Training on" + sub + "by" + provider);
}
```



Q What is root tag of web app in web.xml ?

↳ <web app>

Q No spaces in xml file ?

↳ true

Q. one content param is equalivalent with one content parameter ?

↳ true

web app (parent of content-param)

↳ <content-param> (parent of param name and value)

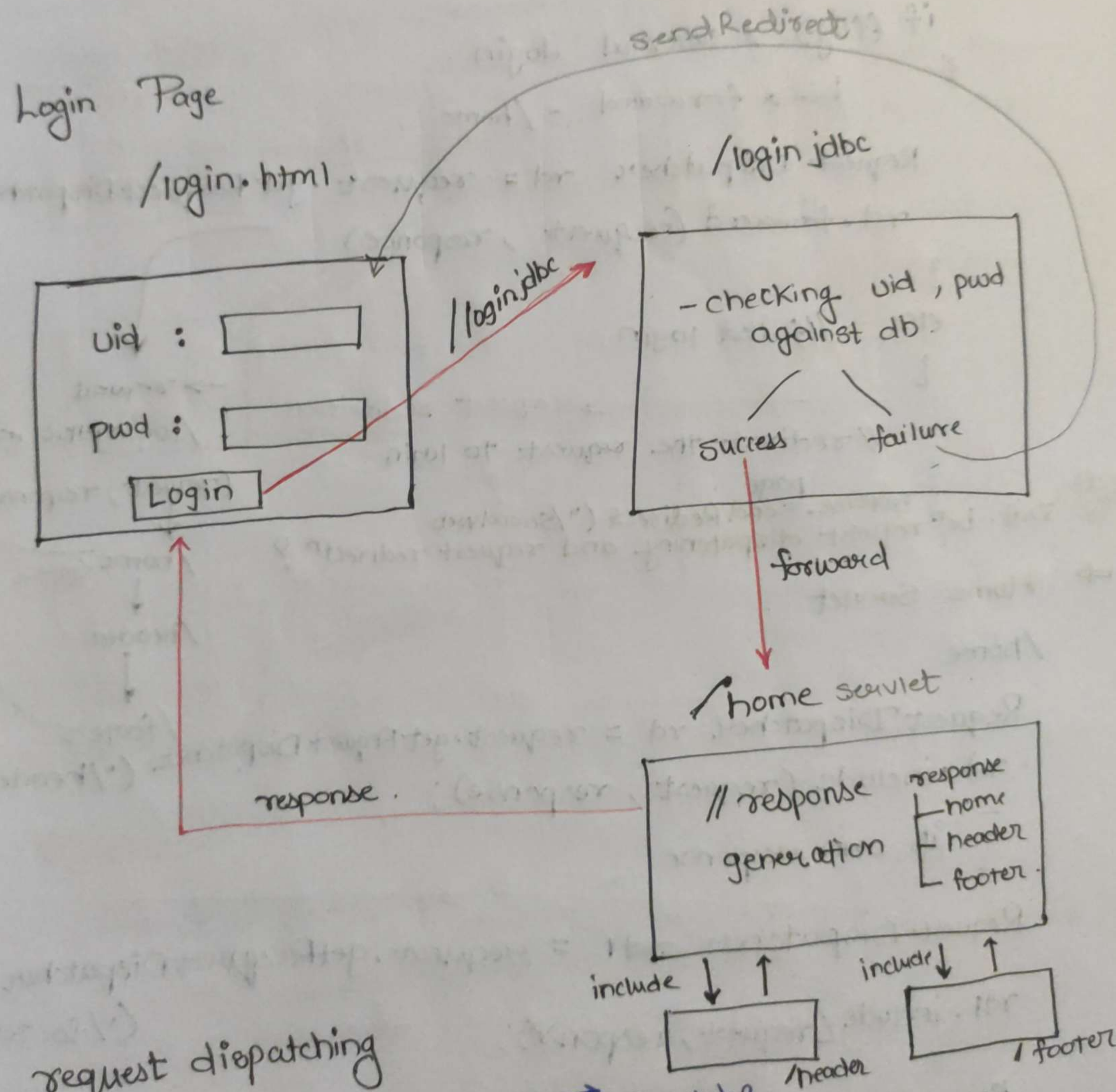
Q.

→ Request Dispatching

→ It is a process within web application for getting help of other servlet and JSP for generating response.

→ Internal management - divides the task of request processing in multiple servlets (JSP)

→ It is never known to the client - URL in the address bar is not getting changed.



request dispatching

→ forward - servlet 1 ⇄ servlet 2.

→ include - servlet 1 ⇄ servlet 2.

We only have to make object of cookie else other's object are created internally.

Request Dispatcher
↓
interface. ← creating instance
/log in jdbc.

if (flag) // successful login.
{
→ forward - /home.

Request Dispatcher rd = request.getRequestDispatcher("/home");
rd.forward (request, response)
}
else //failed login.
{
→ request redirection.
// redirect the request to login page.
response.sendRedirect ("/secondweb

→ request
/loginjdbc
(request, response)
↓
/home
↓
/header
↓
/footer

CCEE
Q. Diff. betn request dispatching and request redirectn?

⇒ Home Servlet.

/home

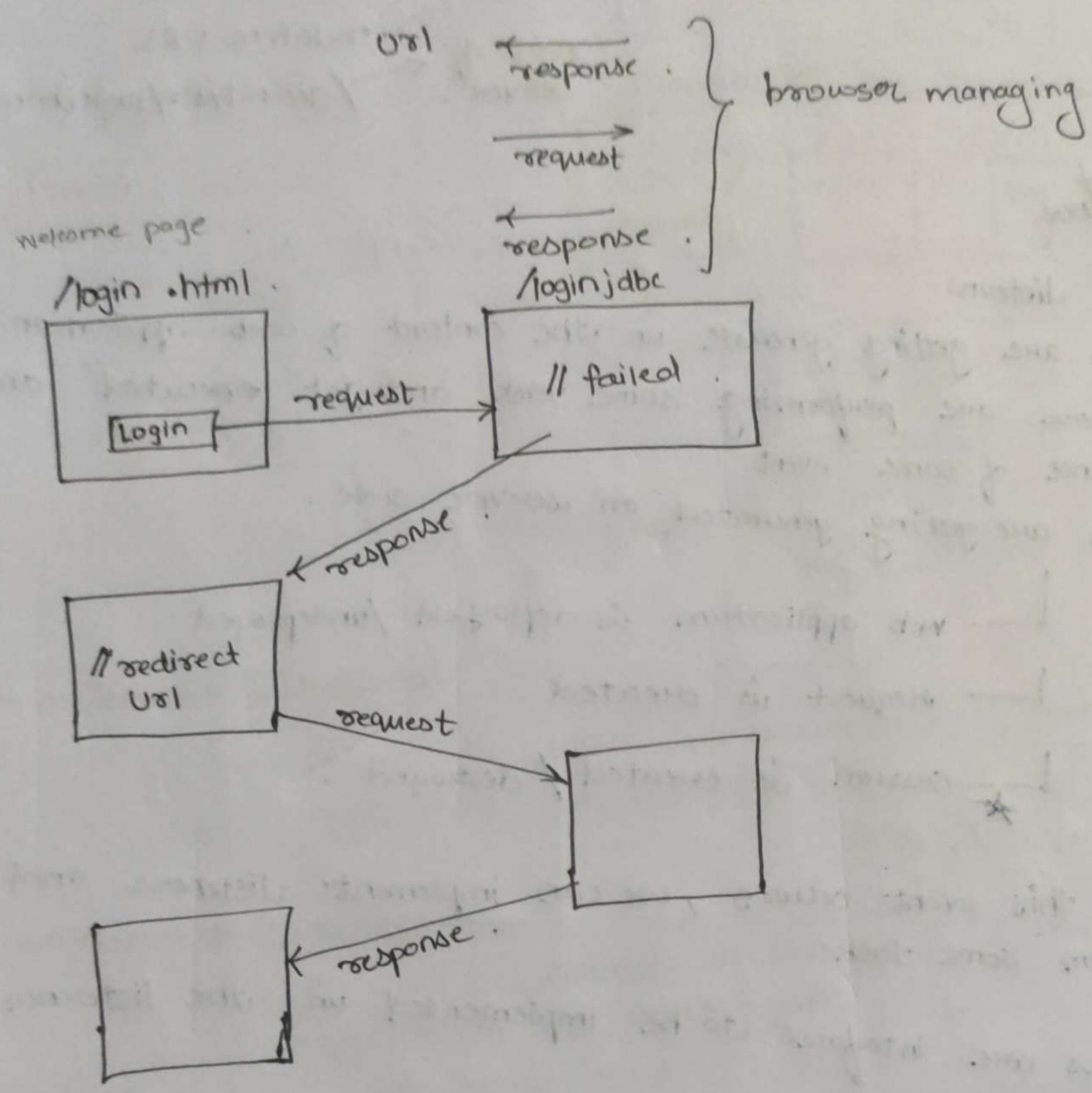
Request Dispatcher rd = request.getRequestDispatcher ("/header");
rd.include (request, response);
= its own response.
=
Request Dispatcher rd1 = request.getRequestDispatcher ("/footer");
rd1.include (request, response);

Q. What is diff. creating header, footer in one servlet and multiple servlet?

Q. When clicking on hyper link always get method is used?

→ Response only has url.

→ When clicking on search at that time there are two different ~~request~~ and response and ~~one~~ request which is called request redirection.



Redirection.

↳ It is known to the client

↳ can observe change in the URL in the address bar of browser.

↳ response received from server.

↳ contains reference of URL

↳ browser is expected to make the separate request to specified URL

response.sendRedirect("---")

URL ← absolute URL.
eg. http://www.abc.com

relative URL

root of server. ← /secondWeb/login.html

Listers

+ event listerns.
+ Events are getting generate in the context of web application.
+ Listerners are performing some task and get executed on occurance of some event.
+ Events are getting generated on server side.
  - web application is deployed /undeployed.
  - request is created.
  - session is created / destroyed.

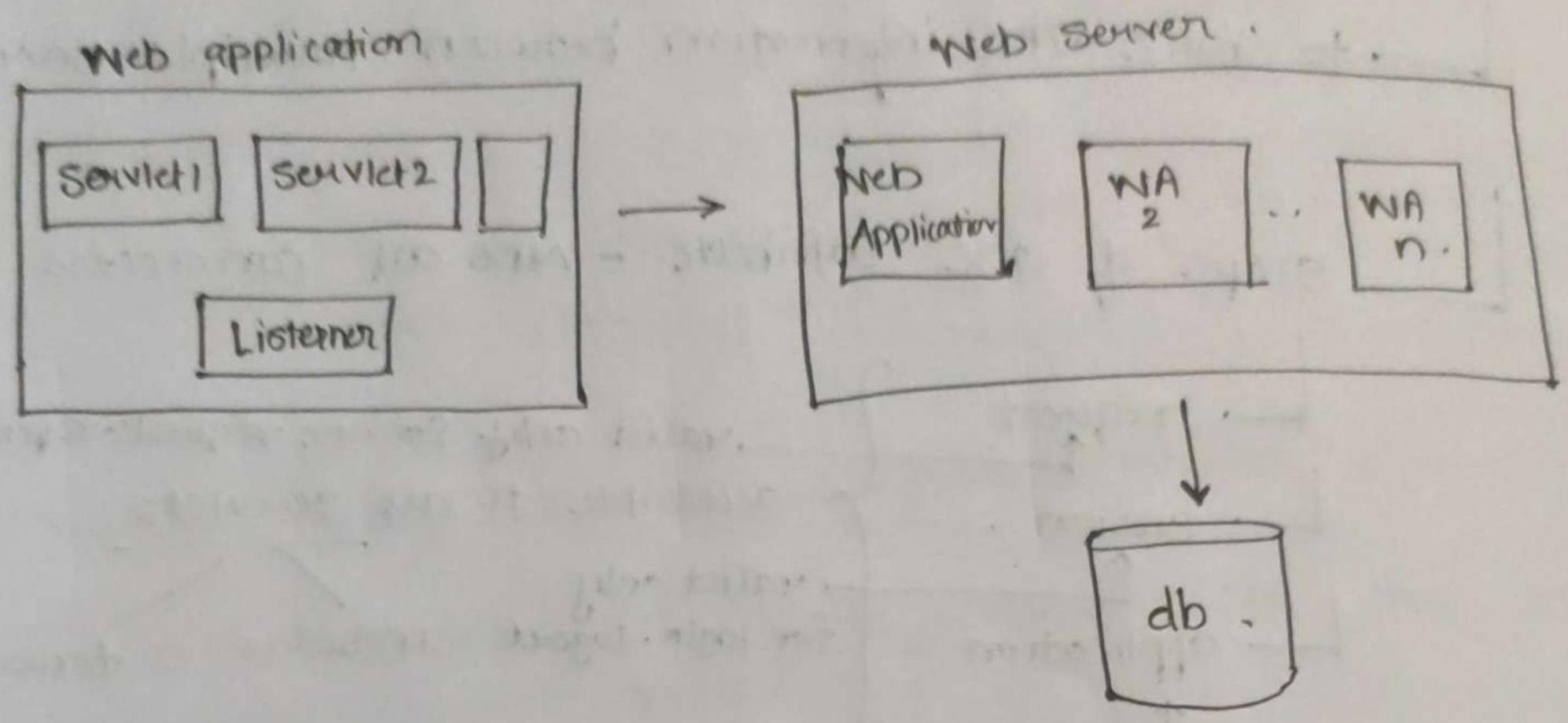
+ When this event occurs, we can implement listeners and perform some task.
+ Listers are interfaces to be implemented in the listerner class.
+ Listerns need to be registered with web server
  - ~~web application is deployed/undeployed~~.
    web.xml
  - annotation - @webListerner.
    ✓
    Created at the time of Servlet created.

+ Different types of listerns interfaces.
  1. ServletContextLister ← interface.
  2. HttpSessionLister.
  3. ServletRequestListener
  ⋮

→ Servlet Content Listener.

└ content Initialized → establishing connection.

└ content Destroyed → closing connection.



→ Content Initialized.

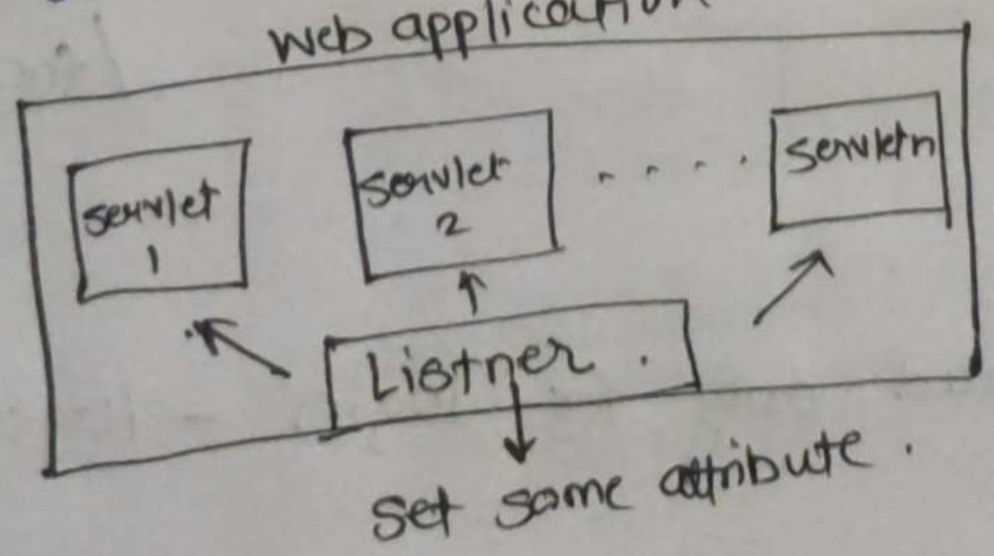
↑ It will be called on occurance of event - application is deployed.

→ Content Destroyed.

↑ It will be called on ocurance of event - application is undeployed.

Attribute.

⇒ All servelets will use connection created by listner, for that listner will create set of some attribute.



Note: Parameters are always string.

Parameter - name - value

string.

Attribute

- name - value
  - name → String
  - value → object (In attribute)
- to share the information between multiple servlets.

- scope of the attribute - who all can access
  - request
  - session
  - application

  request: valid only for one request-response cycle

  session: valid only for login-logout

  application: accessible to all servlets
    - included
    - forwarded

  Accessible to all servlets & JSPs.

1. Request
   - valid only for one request - response cycle
     - accessible to home.
       - /login jdbc
         - /home
         - /header
         - /footer.

2. Session
   - valid only for login - logout.

3. Application.
   - ~~valid only for one request - re~~
     Accessible to all servlets & JSPs.

Reading request parameters.

uid : [ ] //uid.

pwd : [ ] //pwd.

[login] //submit.

if. method = get.

action = "http://localhost:8080/firstweb/logincheck?
user = jhon & pass = java..

if method = post
action = "http://localhost:8080/firstweb/logincheck.

Q. If servlet wants to make dialog with the database then what servlet make use of?

⇒ JDBC → ① Establish or load driver class.
② Establish connection.
③ Create statement.
④ Execute statement.

Q. What type of statement we use for JDBC and servlet connection?

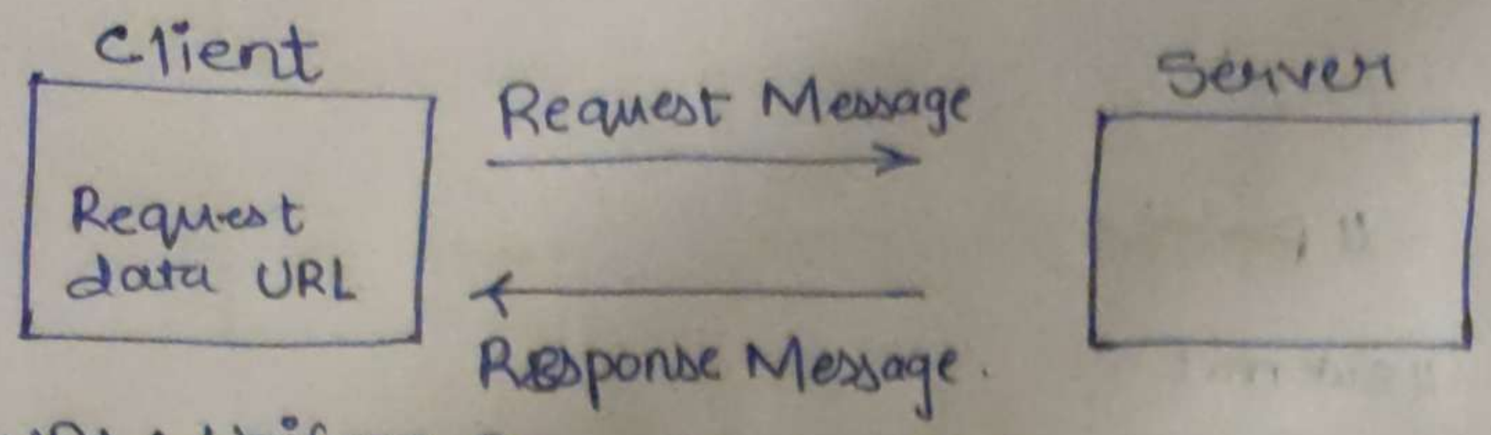
⇒ Prepared statement because userid and password we don't know where data is fetch from db and taken from client and dynamically decided.

So, we use queries like:
select * from user where user-id = ? and pass = ?;

⇒ Data is send to server in key-value pair.

```
void doPost(--req, --res)
{
    String uid = req.getParameter("uid");
    String pwd = req.getParameter("pwd");

}
```

HTTP Methods



URL : Uniform Resource Locator.

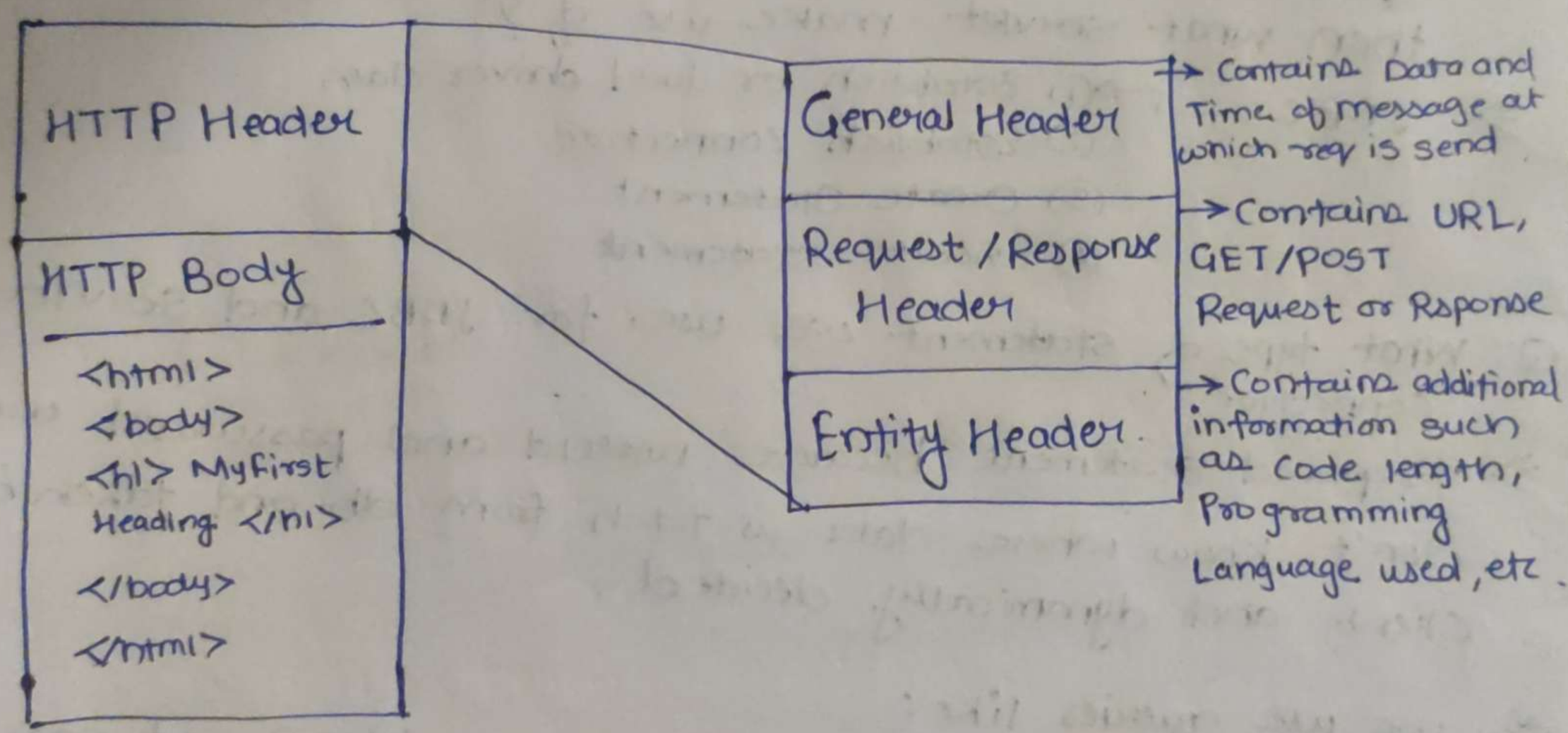
Request & Response Message format.

→ Both message has same format.

→ The format is divided into parts.

↳ HTTP Header

↳ HTTP Body.



→ HTTP body is empty at the time of HTTP request, only HTTP header is used at the time of HTTP request.

→ In Response Message HTTP Header and also has HTTP body which has HTML ~~body~~ program. which get open on the client device

GET Request Method.

- URL (Link)

Example:

https://example.com/test/demo_form.php?name1=value1&name2=value2.

It contains an Url which works as request and for that server sends response as web page or data get updated on web page.

Post Request Method.

- It has a URL (Link)
- And it also contain request body.

Example:

URL ⇒ https://example.com/test/demo_form.php HTTP/1.1

Request body ⇒ name1 = value1 & name2 = value2

↳ it contains the data which want to send from client to server.

When we click on submit button then all the filled data ~~on form~~ get saved into request body and using url send to server.

Get Request Method.

1. Contains only URL
2. Data is visible in URL anyone can change it or copy it.
3. Unsafe
4. Maximum length of data which is send is 2048 maximum data length.
5. Only ASCII characters can be sent over Get method.

Post Request Method.

1. URL + Request Body (Hidden)
2. It is URL containing Request body which is hidden.
3. Safe.
4. Maximum length of data which is send is Unlimited data length.
6. ASCII, Decimal, Binary all types of data can be sent.

→ Connection with database in servlet is done at init method.

That is Establing and load driver } done at init
Establing connection.

Creating statement } done at service
Executing statement } method.

→ MySQL connector jar file is the must file for JDBC servlet connection.

→ MySQL connector jar file should be added at
└ build path.
└ deployment assembly.

→ MySQL connector should be at location in deployment assembly is lib

→ One servlet can have multiple URL patterns.

eg. Servlet name : Test Servlet
URL mapping : /TestServlet
/ test

→ But one url pattern can't be used for multiple servlet.

→ ~~Testing~~ ~~[illegible]~~ ~~[illegible]~~.

→ ~~Web Server~~ V/S Application ~~Server~~.

Web Server	Application Server
1. Provides static content to clients.	1. It provides dynamic conent.
2. It handles request and response for static content, such as HTML pages, files, images and videos.	2. An app

Web Server.

1. A web Server provides Static content to clients.

2. A web server handles a requests and responses for static content such as HTML pages, files, images and videos.

3. A web server only handles HTTP requests and Responses.

4. ~~Example~~ Web Server only has one container whose name is web container. which provides servlet API, JSP, API, Security, Connection pooling, JNDI (naming service)

5. Less Powerful.

6. Example: Apache

Application Server.

1. An application server provides dynamic content.

2. An application server exposes business logic to clients, which generates dynamic content. Dynamic content includes real-time updates, personalized information and customer support.

3. The application server generates responses based on business logic that can change depending on the request parameters.

4. Application Server has web container as well as EJB (enterprise java bean) container and all J2EE services implementation.

5. More Powerful.

6. Example ~~Glo~~ Tomee. Glassfish

HTTP BASIC

1. HTTP : Hyper Text Transfer Protocol.
2. It is an application level protocol for transfering textual data between client and server.
3. HTTP is stateless protocol.
4. HTTP request methods and Headers.

eg URL : http://www.server.com:8080/bank/login.html

port Header

Whenever the client sends the request to server for any resource such as a HTML document, it specifies.

1. HTTP method (get/post/put/delete...)
2. Request URI (eg. /bank/login.html)
3. Protocol version.
4. Optional Header Information (eg: accept, cookies)

Statelessness : It means once the response is returned, server doesn't remember anything about the client (even the very next request)

5. HTTP server will listen on TCP port 80 by default which is the well-known port number for an HTTP server.
6. This way any HTTP client that connects to HTTP servers can use the default of TCP port 80.

7. ~~HTTP~~ Response ~~contains~~ :

i. Response ~~headers~~ ~~(eg: cookie~~ 404/200)

ii.

7. HTTP Response contains

1. Response status information (eg: 404 / 200)
2. Response headers (eg: cookies / resp cont type eg: text / html, application / json image / gif ... / cont length).
3. Response data.

8. $\underbrace{\text{GET}}_{\text{method}}$ / $\underbrace{\text{tutorial.html}}_{\text{path}}$ $\underbrace{\text{HTTP / 1.1}}_{\text{protocol}}$. ← Following is the example of HTTP request header, which provides additional information about the request. uses GET request method to ask for HTML document named tutorial.html using HTTP protocol ver. 1.1

9. User-Agent : mozilla / 4.0 (compatible ; MSIE 4.0 ; Windows 95)
Accept : image / gif , image / jpeg , text / * , */*.
eg : text / html , text / xml , application / json.

↑ This header specifies the information about the client software and what MIME (Multi-purpose Internet Mail Extension) type the client accepts in response, using User-Agent and Accept headers.

⇒ HTTP request Methods.

The request method indicates to the server what action to perform on the resource identified by the request-URI.

HTTP 1.1 specifies these request methods:

GET, POST, HEAD, OPTIONS, PUT, TRACE, DELETE, CONNECT

↑ Servlets can process any of the above request.

But GET and POST methods are most commonly used.

Attributes and Listeners.

Attributes.

name - value pair

- name → String
- value → Object
  - String
  - connection
  - session factory
  - Array List
  - Emp.

→ Attributes - scope.

↳ request → limited one request response cycle.

↳ session → login - logout.

↳ application → common for all servlet (JSP)

→ Make use of methods.

```
setAttribute ("name", object);
setAttribute ("name");
```

* one servlet or JSP - set the attribute other servlets or JSP - get the attribute.

- setAttribute ↓

```
request.setAttribute("name", obj)
request.getsession().setAttribute("name2", obj)
getservletcontext().setAttribute("name3", obj)
```

→ same scope should be used while retriving.

String msg = (String)
request . getAttribute ("name1");
↓
object

eg. connection conn = (connection) getservlet Content().
getAttribute ("name 3");

Listeners:

Classes - listerner Interface → Servlet Contentlistener.

Code ↓

```
(2)
@ weblistener
public class Shopping Applistener Implements Servlet content (1)
                                                listener
{
    connection conn; (3)
//deploy p.v content Initialized (Servlet ContentEvent e)
    {
        // establish conn;
        e. getservlet Content ().set Attribute ("jdbc.com", conn);
    }
    (4)
    p.v. content Destroyed (servlet ContentEvent e)
    {
        // close the connection
    }
}
```

State Management.

Why?

↳ http - Stateless protocol

- every request considered as new fresh reque

- so it unables to generates the dialog betw

client - server.

ongoing conversatio

- maintain some relation between multipl request.

What?

↳ Saving some data received in the request.

↳ either on the client side or server side.

↳ Data always get saved in the form of ke value pairs.

how?

↳ State management techniques.

Client

- url rewriting *

- hidden form fields *

↳ request.getparameter(" ")

✓ cookies.

Server.

✓ session.

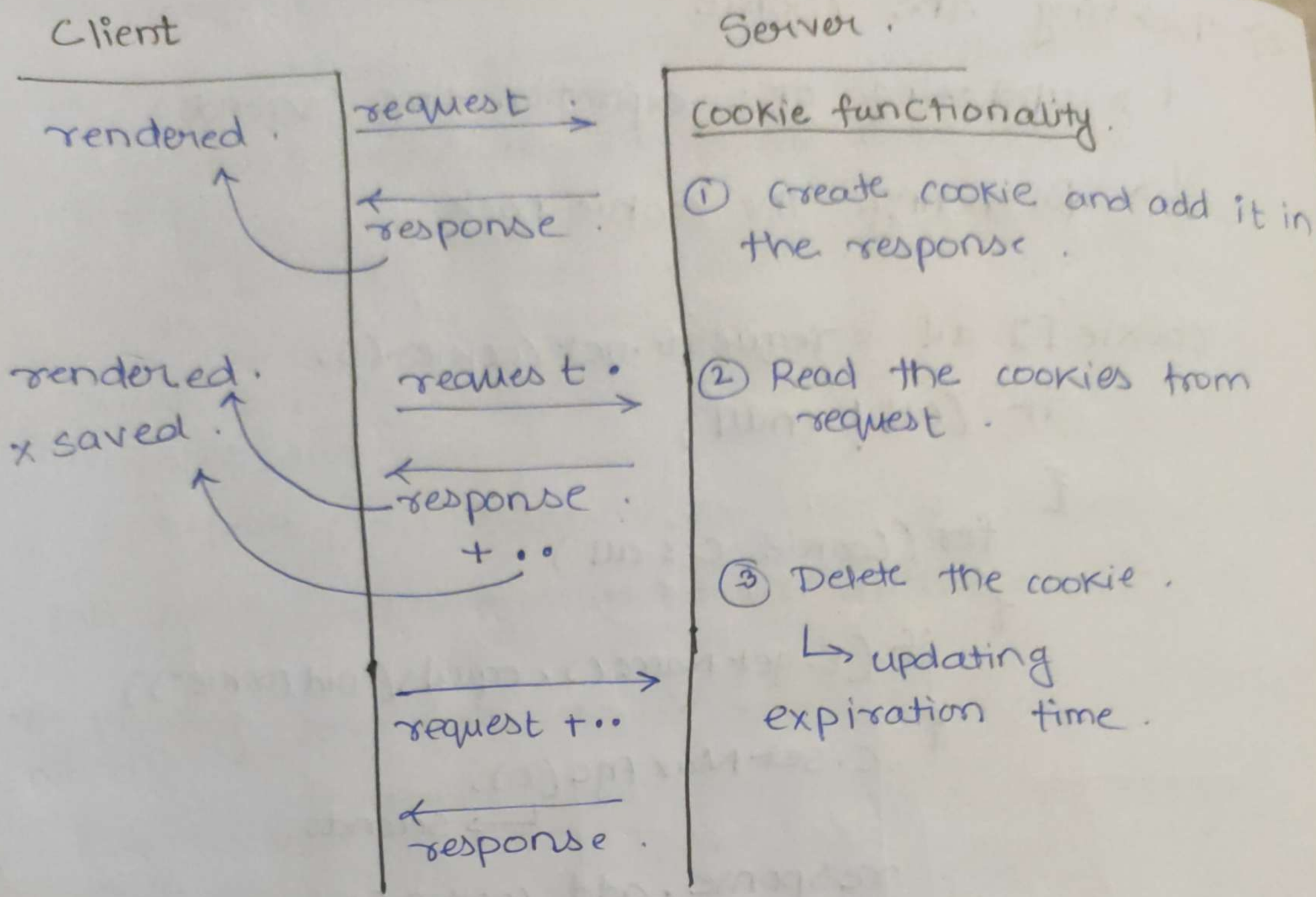
Client	Server
Cookies	**Session**
⊕ Not utilizing server side's memory.	⊖ Utilizes servers resource.
⊖ readability - anyone can read info from the cookies (not secure)	⊕ Info in the session can not be read. (secure)
⊖ cookies can be disabled on client side (not reliable)	⊕ session can not be disabled (reliable)
⊖ It increased network traffic.	⊕ No increase in network traffic.

Cookie:

- name - value pair on the client side.
- cookies will be generated on server side and sending cookies along with response to client.

① Saving the cookie received in response.

② Resending the cookie to the server in subsequent request.



1) creating cookie and add it in the response :-

```
cookie cookie = new Cookie ("uid cookie", "request.getparameter ("uid"));

response.addCookie (cookie);
```

2) Retrieving Cookies from request:

```
cookie [] all cookies = request.getCookies ();
 if (all cookies! = null)
 {  for (cookie c : all cookies).
    {  if (c.getName().equals ("uid cookie")).
       {
         out.print ("<hi> welcome" +c.getvalue() +"</hi>");
       }
    }
 }
```

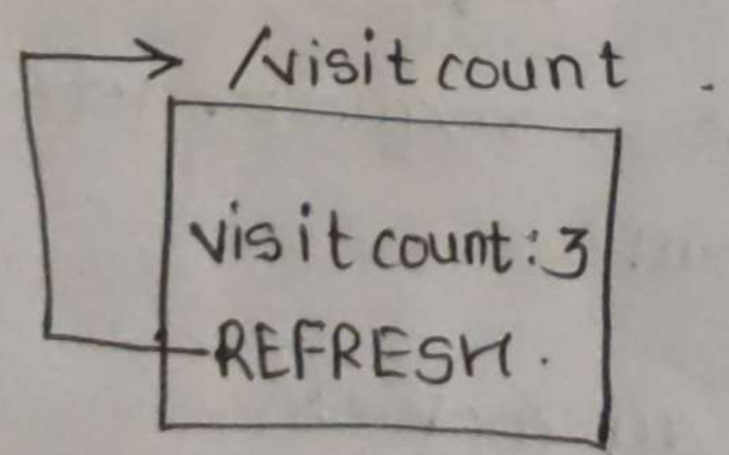
3) Deleting the cookie:

↳ updating the expiration time (seconds)

↳ Resending the cookie back.

```
cookie [] all = request.getCookies();
if (all != null)
{
    for (Cookie c : all):
    {
        if (c.getName().equals("uid cookie"))
        {
            c.setMaxAge(0);
                ↳ seconds.
            response.addCookie(c);
        }
    }
}
```

→ cookie should set one servlet / JSP and should be accessed by another servlet / JSP.



/visit count:

```
print writer out = response.getWriter();
int n=0;
cookie [] all = request.getCookie();
if (all != null)
{
```

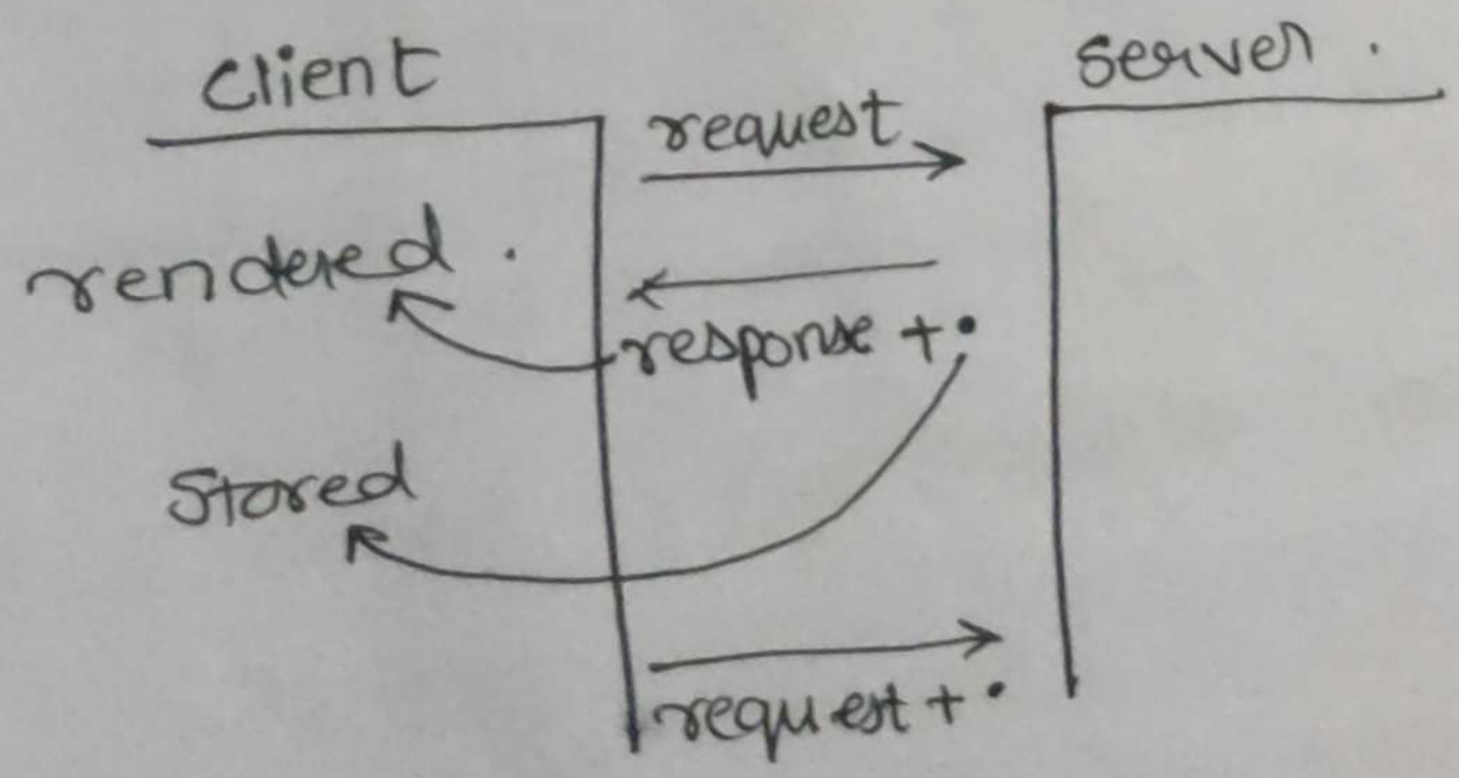
```
for (cookie c : all)
{ if (c.getName().equals("visitcount cookie"))
  {  n= Integer.parseInt(c.getvalue());
  }
}
  n++;

Cookie cookie = new Cookie("visitcount cookie","" +n);
 response.addCookie(cookie);
  out.print("visit count:" +n);
 out.print("<a href ="visitcount" > REFRESH </a>");
```

Writing Java Code in JSP:

<u>shortcut:</u>

```
<%
    // java code
        <p> ${ cookie.log1nerror.value } </p>
%>
```

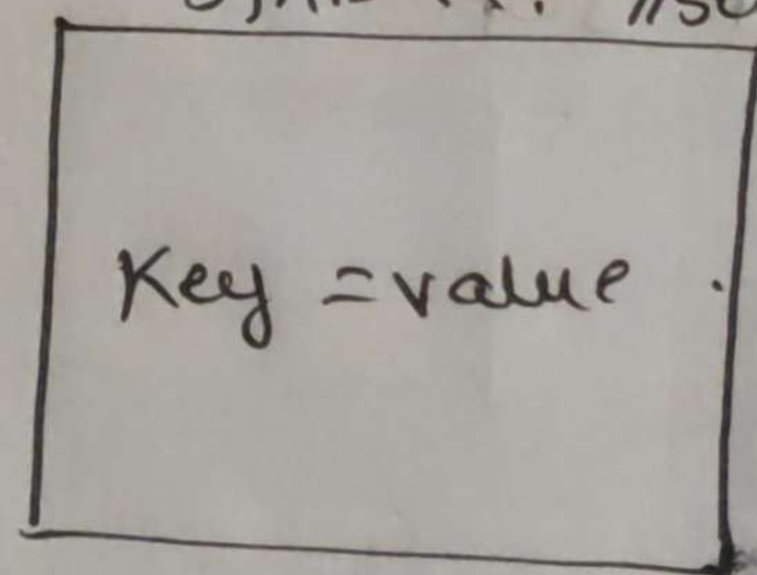
* session - server side state mgmt technique.
  - more secure, reliable than cookies.
* - sessions are internally dependant on cookies.



first request :

if (session is cookie is not available).
{
// allocating some block of memory.
// Identifying unique session id.
// jsessionid =
// cookie is sent in response.
}.

• active for multiple request response cycle.
user specific
SyAb ... //session id.



Understanding sessions.

- Server side
- Secure
- reliable.

Agenda.

+ Concept of session
+ Session functionality
+ implementing session in stopping
+ intro to filters.

first request →
1) allocates block of memory
2) unique ~~id~~ session id generated
3) jsession id - ~~~~
4) send in the response.

subsequent → jsessionid
1) read the cookie - jsessionid - request.
2) refer session for that client.

1. Sessions are dependent on cookies.

To deal with this problem:

(a) Straight way reject the request till cookie is enabled.
(1) JS coding.
(2) send the cookie and check whether cookies are received in subsequent request.

(b) Use different mode of transport for sessionid.
└ in the form of url.

Here instead of cookie, sessid will be used for travelling between client and server.

http://localhost..../getProducts?jsessionid=---
↓
encoded URL.

2. Session - actually using memory on server side.
└ session should be destroyed.
└ automatically.
└ (1) Server shuts down *
(2) once inactive time is over.

For example:

If we do not do anything on website at that time after some time the website give us message that session get expired. that means that memory gives to another client.

└ explicitly when we intentially want to shout down session
└ at the time of logout
(no further dialog required)

1> Retrieve the session.

HttpSession session = request.getSession()
└ session exists, it will be returned.
└ session does not exist, it will created and returned.

boolean
↓
HttpSession session = request.getSession(false)
└ if session exists it will be returned.
└ if session is not existing, it will not be created.

This can be null.

2> Save the info in the session.

```
HttpSession session = request.getSession();
session.getAttribute("usciinfo", user);
```
└ read from database.

3> Retrieve info from the session.

```
HttpSession session = request.getSession
User user = (User)session.getAttribute("userinfo");
```

It's return type is object.

```
userinfo = { }
msg = " "
... = [ ]
! =
```

Note: * CCEE

Life of an session is in minitus and life of cookies is in seconds.

4) Remove the attributes.

session.removeAttribute("--");

5) Setting inactive time period.

Session.setMaxInactiveInterval (20);
↳ Web.xml
↑ in min.

6) Destroying the session.

```
Http session session = request.get session(false);
if (session != null)
{ session.invalidate();
  // memory block will be free
```

If we refresh the page the session id remains same.

→ /session

sessionid : _____

REFRESH

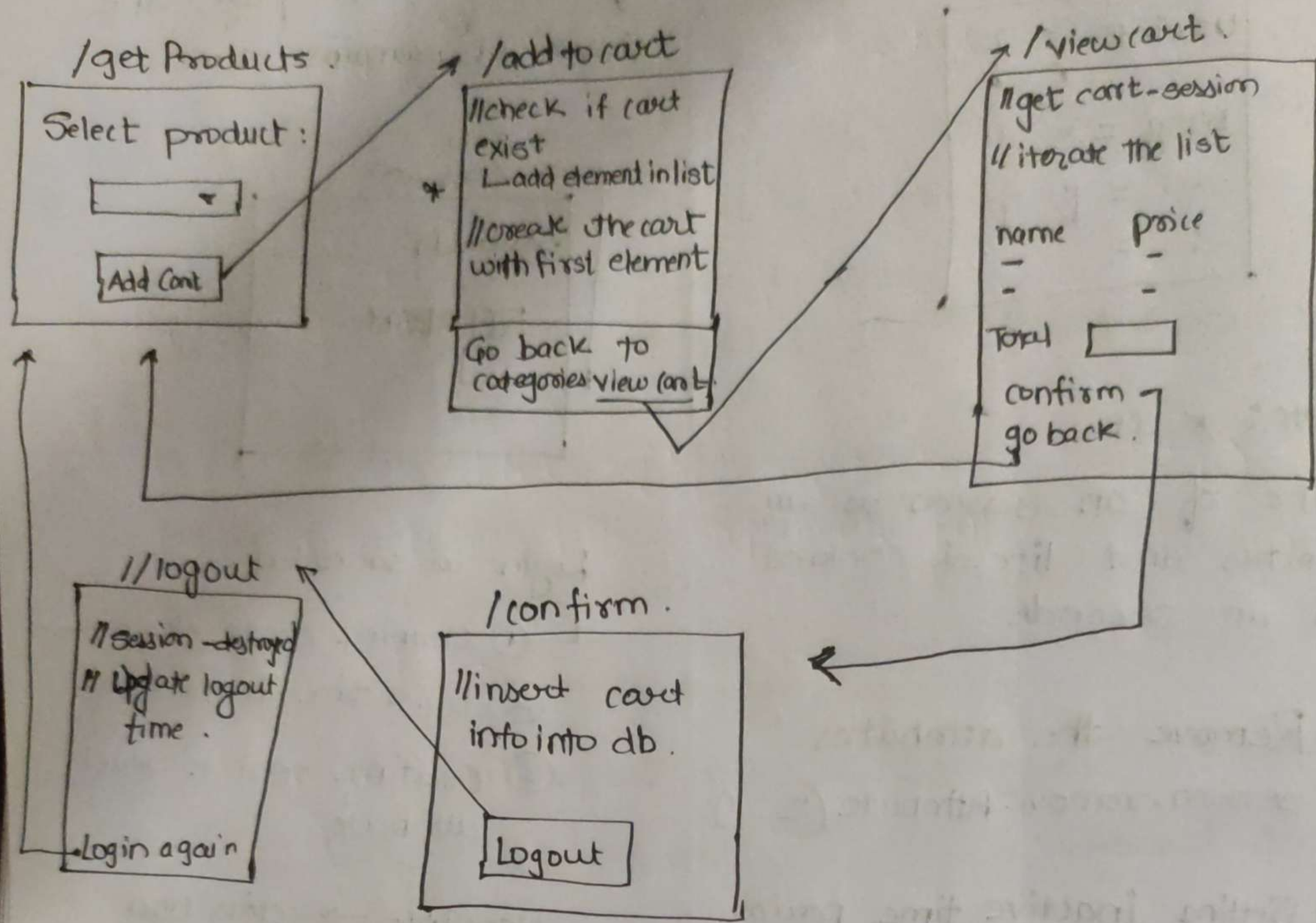
Login is successful.

1. Cleaning cookie - loginerror
2. login time is inserted
3. Creating session level attribute

User info → class User
{ =
}

* Note: In the first request there is no session.

If we delete cookie id and then refresh the session at that time it will generate new session id.



autobaxing.

Servlet filters.

1. Filters are java classes which allows to execute some additional task along with core business logic.

   - logging request
   - Encryption and decryption of pwd.
   - finding out response time.
   - Modifying request header or request data.
   - compressing the response.

2. Filters are pluggable (either web.xml or annotation)
   @ WebFilter.

3. Filter - implement Filter

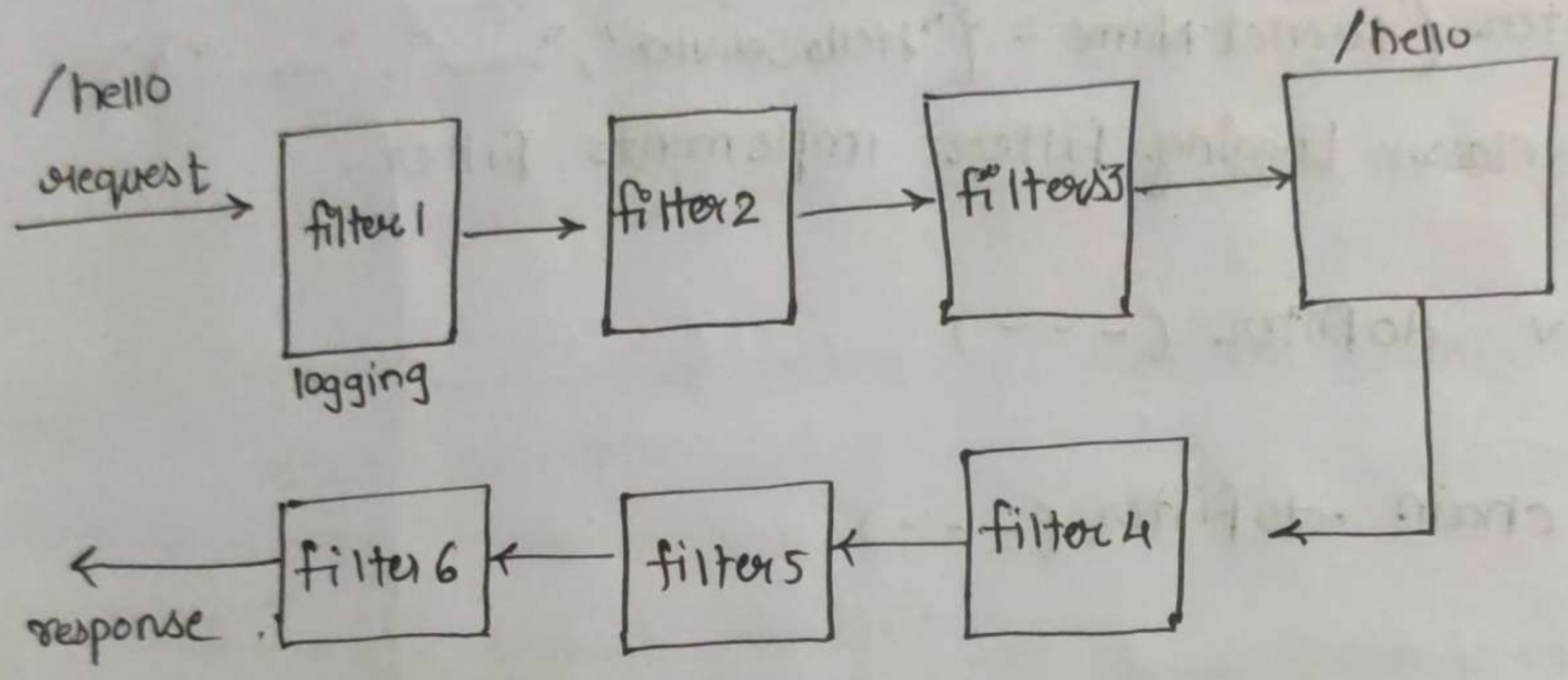
life cycle of filter.

init() - 1.

destroy()-1

defilter() - n no. of times - request, response, chain (filter chain)

Execution of filters.



Note: Just like servlet is a class, listner is a class similarly filter is a class but the only difference is that no one will call filter just like servlet class, it get automatically get.

How to Write filters.

@WebFilter (urlPatterns = {"/hello", "/Welcome", "/login"})

This means when client called for servlets /hello or /Welcome or /login then filter will also called for this classes.

```
public class LoggingFilter implements Filter.
{
    public void doFilter (...)
    {
        Chain.doFilter (...)
    }
}
```

Use for moving of filters.

Filter mapping can be done by.

↳ 1. Url Pattern ← seen in the last code.

2. Servlet Names.

Using Servlet Names.

```
@WebFilter (servletName = {"HelloServlet", "___", "___"})
public class LoggingFilter implements Filter.
{
    p.v doFilter (---)
    {
        chain.doFilter (---)
    }
}.
```

Note: if we use url as

http://localhost:8080/shoppingApp/addtocart

In this case filters will instead of taking directly to addtocart page to redirect the user to login page if login is not done in this way filters will be useful.

JSP → JSP is not replacement for servlet
→ complementary technology for servlet

Java Server Pages.

1. JSP is supportive technology that has been developed after servlets
2. Servlet:
   i. Generating HTML in servlet is very difficult.
      └ HTML written as string and given as a argument to out.print()

   ii. Modification of maintenance of this HTML view is very difficult.

   iii. No support is given in editor for writing HTML.

   iv. JSP - supportive technology that has been developed after servlets.

request → 1> request processing. — When java part then Servlets (java & HTML) ✗
└ db operation. (java).
No out.print statement in servlets.

2> Generate the response (html + dynamic data). — JSP (HTML & java) ✗
No scriplet in JSP
ie. <%
%>

v. JSP developed on top of servlet.

vi. JSP internally gets converted to servlet
also applicable for JSP.
→ config,
→ content
→ dispatching
→ redirection.
→ attributes
→ cookies, sessions.

vii. Extension of JSP is .jsp

viii. JSP looks like HTML.

ix. Translation from JSP to servlet is done by web server.

JSP Life Cycle.

1. Translation of JSP to servlet

   first.jsp → first-jsp.java.

2. Compilation of generated servlet class

   first-jsp.java → first-jsp.class.

3. Loading servlet class.

4. Instantiating servlet class - constructor.
   Creation of constructor.

5. Initialization.
   init()

6. Service
   get() post()

7. destroy.
   destroy()

} Servlet life cycle

init() can be call only once whereas service() can call multiple times.

JSP elements.

1. HTML tags. - <h1> Welcome to JSP </h1>

   ↓ translation.

   out.print("<h1>Welcome to JSP </h1>");

   ↑ service method.

2. HTML comment -

   <!-- HTML comment -->

   ↓ translation.

   out.print("<!-- HTML comment -->");

   ↑ service method.

   HTML comment will not visible in client's browser.

3. JSP comment.

<%-- JSP comment --%>

↓ translation.

not added in generated servlet.

Note: The difference betn HTML and JSP comment are HTML comment will move from server to client browser but not get displayed whereas JSP comment scope is only upto servlet.

4. Scriptlet

i. Inserting java code in JSP.

ii. valid servlet code - scriplet of JSP.

iii. scriplet processing is easy for webserver.

```
<%
String val = request.getParameter("key");
out.print("<p>" + val +"</p>");
%>
```

} java statements ended with ;

↓ translation.

only symbols are removed, code gets added as it is, Service method.

5. Expression → shortcut of out.print() statement

```
<% = Calendar.getInstance().getTime() %>
```

→ argument for out.print()

↓ translation.

```
out.print(calendar.getInstance().getTime());
```

→ service method.

6. Declaration → helps in instance variable and method declaration.

```
<%! int n=0; %>
```

↑ instance variable of servlet

↑ Statement gets added in generated servlet outside service method

<% int n = 0; %>

↳ added in service method.

7. Directives - providing addition into to JSP engine.

<% @ ---- %>

3 types of directives

↳ a. page → This page directive used to provide instructions to the container regarding the structure and behavior of JSP

b. include → The include directive is used to include the content of another resource (like HTML or JSP)

c. taglib → It is used to declare and define custom taglib that can be used in the JSP page.

<% @ page ____ = " " ____ = " " %>

directive. attribute. value of attribute.

import statement.

<% @ page import = "java.util.*, java.io.File" %>

directive. attribute. value of attribute.

Note: page directives are written on top of servlet JSP.

8. Std (JSP) actions.
9. EL
10. JSTL
11. Custom tags.

} Scriptless JSP elements → do not insert java code directly.

↑ it is made by us.

└ it is made by us.

page :

```
<%@ page language ="java"
contentType ="text/html;
charset =UTF-8"
import ="java.util.*" %>
```

include :

```
<%@ include file ="included
content.jsp"
%>
```

taglib :

```
<%@ taglib uri="http://java..."
prefix ="c" %>
```

In Java local value should be initialize with zero

Agenda:
1. JSP elements.
2. implicit objects.
3. JSP directives.
4. Use of scriplets, expression, declaration
5. MVC architecture.
6. MVC implementation.

⇒ Implicit objects.

1) out - JSP Writer. → response.getWriter()

2) request - HttpServletRequest. } available in service method as argument.

3) response - HttpServletResponse }

4) session - HttpSession. — request.getSession()

5) config - Servlet Config. - get Servlet Config()

6) application - Servlet Context — get Servlet Context()

7) page Context - page Context — single point of contact.
└ page context.getOut() → out

8) page - this — rarely used.

9) exception - Exception. — error page in JSP.
└ Not for all JSP pages.

Note: Always use curly braces for if condition.
Use scriptless JSP

Servlet - JSP
└ CCPE

⇒ JSP directives. ← Written on top of scriplet

3 types of directives -

1. Page directive
- language. — java.
- content Type. — text/html
- page Encoding. — ISO-88591
- isELIgnored. — false.
- session. — true.
- autoFlush. — true.

- buffer — 8 kb.
- isThreadSafe - true.
- isErrorPage
- errorPage
- import.

Use of error page.

/viewcart.jsp.

```
<% @page errorPage = "/error.jsp" %>
        isErrorPage = "false"
<%
    int n = 30/0;
%>
```

Work as try block

/error.jsp.

```
<% @page isErrorPage = "true" %>
<% = exception.getMessage() %>.
```

This work as catch block.

exception — additional implicit object available only for error pages.

/header.jsp

```
<% @page errorPage = "/error.jsp"
         isErrorPage = "false" %>
<% use user = (User) session.getAttribute ("Userinto")
%>
<% = user.getfname() %>
```

exception occur

So, any exception happen in any page then only one page will handle all the exception.

When jsp page act as an error page when isErrorPage = "true"

2. include directive. → include std action.

content.jsp

<% @include file="/header.jsp" %>

↳ helps in adding the code from other html or JSP files.

<%

⋮

Inclusion of header.jsp takes place before translation.

⊖ jsp pages become very bulky.

⊕ content of header are accessible in included file.

Scriptless JSP elements.

1. scriptlet, expression, declaration (Always avoid).

↳ insert java code in JSP.

↳ it becomes difficult to read, modify, maintain.

↳ Java + html (out.print).
↑ difficult

html + java code (scriplet)
↑ difficult

2. Scriptless JSP elements.

↳ do not insert java code directly in JSP.

↳ java code gets inserted at the time of translation of scriptless JSP elements.

↳ script less JSP (std. (JSP)) ← actions - tag.

Expression Language - ${ }

JSP std Tag Library - tag.

Custom tags - Tag.

3. std actions (JSP action)

↳ written in the form of tags which start with jsp prefix.

<jsp : include

prefix → name space → set of tags.

name of tag.

<jsp : forward page="--" />

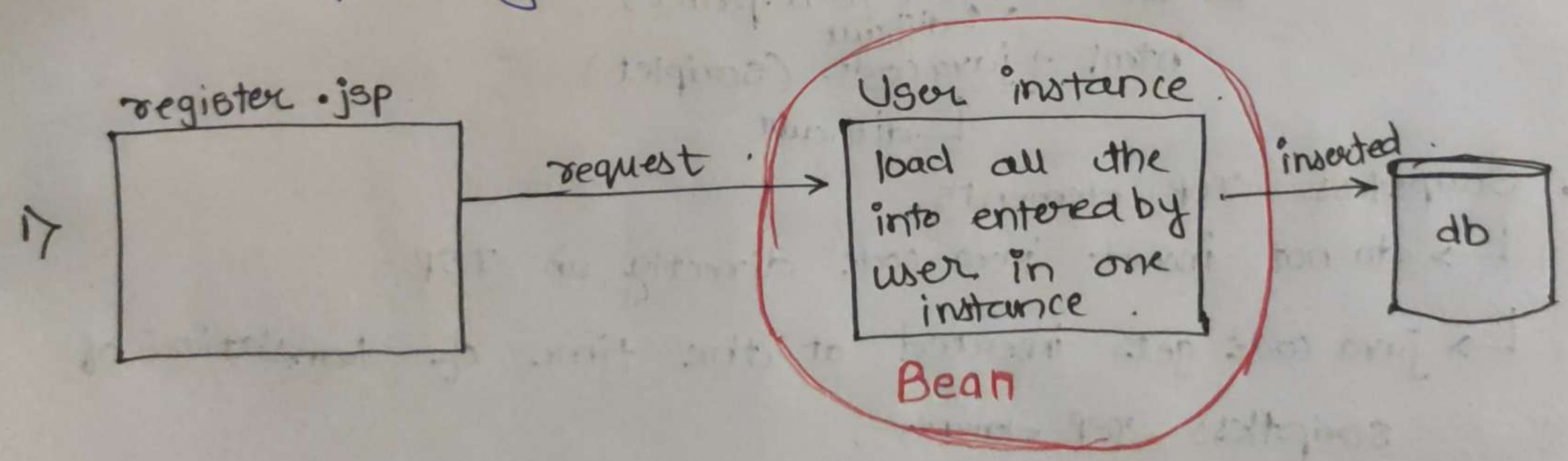
<jsp : useBean --- />
<jsp : setProperty -- />
<jsp : getProperty -- />

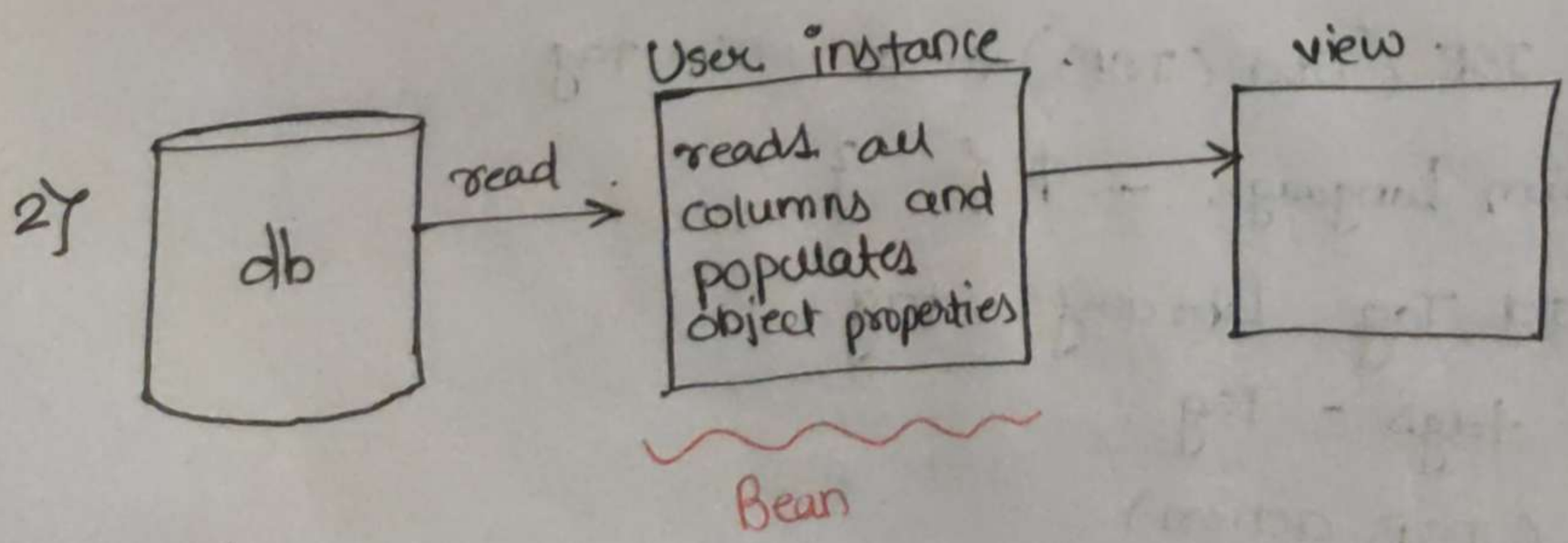
} bean related std actions.

This are defaultly available to JSP.

Bean :- Normal java class - POJO → plane old Java object.

- properties, getter, setter methods. ← automatically called.
- reflect the information from permanent storage / into that is getting inserted into permanent storage
- representing business data - in memory.





⇒ bean class - properties needed.

Compulsory

* ↳ set of properties.
* ↳ getter, setter method.
* ↳ default constructor.

↳ parameterized constructors.

↳ extends from super class.

↳ interface to be implemented.

↳ other business logic methods.

Note:

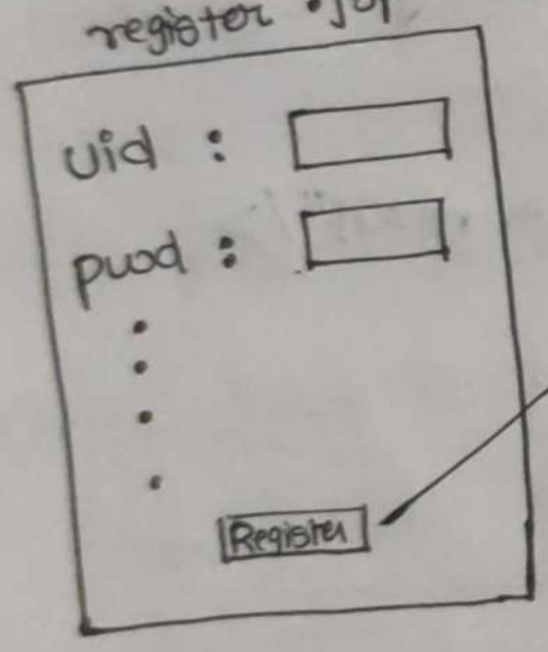
1. Class name always singular

Users ✗ User ✓

2. getter setter naming convension.

fname — String getfname()
— void setfname(String)

⇒ Bean related std actions.



/ create Bean.jsp

// bean related std actions.

```
<jsp:useBean id="userinfo" class="beans.User"
scope="session"/>
```

if bean is available, use it

if bean is not available, create and use it

```
beans.User user = (beans.User)session.getAttribute
("userinfo").

if (user == null)
{
    user = new beans.User();
}
session.setAttribute("userinfo", user);
```

It will happen at the time of translation.

Note:

Bean is an object which has certain properties.

Example :

```
<jsp : use Bean id ="userinfo" class ="beans.User" scope ="session"/>
<jsp : setProperty name ="userinfo" property = "fname" param ="fname"/>
```

└ populating (assiging) bean property.

```
userinfo.setFname ("Balak");
Userinfo.setFname (request.getParameter ("fname")).
```

```
<jsp : use Bean id ="user info" class ="beans.User" scope"session"/>
<jsp : setProperty name =user info" property ="fname" param ="fnam"/>
```

property name in the class

```
<jsp : set Property name ="user info" property = "email"
param = "email" />
```

↓

```
< input type = "text" name ="email"/>
```

```
Class User.
{
    String fname, email;
    Userinfo.setFname (request.getParameter ("fname"))
}
```

```
< jsp : Use Bean id ="user info" class ="beans.User" Scope "session"/>
< jsp : setProperty name ="user info" property ="fnam" param ="fnam"/>
< jsp : set Property name = "User info" property ="email" param="email"/>
< jsp : set Property name ="Userinfo" property =" *"/>
```

↑ applied on all

/ display Bean . jsp

```
<jsp : Use Bean  id = "user info"  class = "beans . User"  scope = "session" />
<h1> Welcome  <jsp : get Property  name = "user info"  property = "fname" />
                ↑ Never use alone
              <jsp : get Property  name = "user info"  property = "lname" />

</h1>
```

MVC Architecture ?

→ It is a design guideline for web applications.

→ It is loosely coupled - total separation business data, logic, UI requirements.

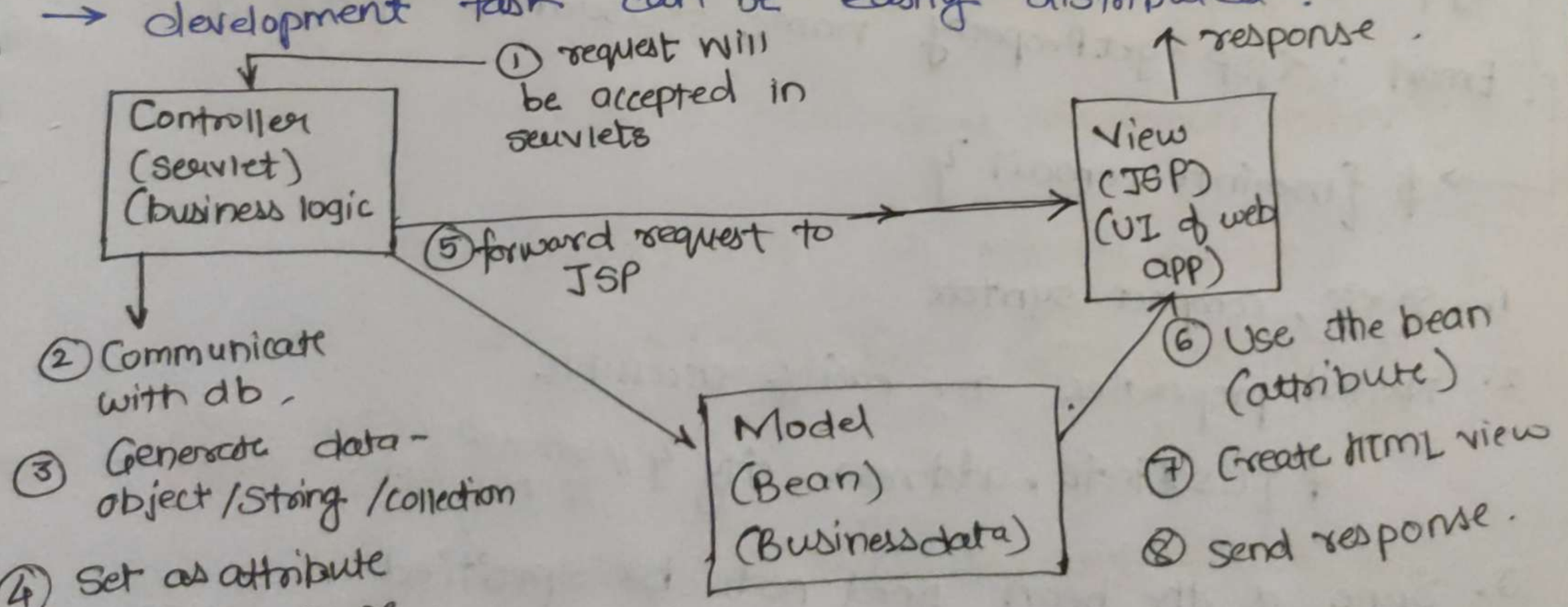
Model - business data - beans.

View - UI - HTML - JSP → HTML / scriptlet, express declaration / Std action, EL, JSTL

Controller - business logic - java - Servlet.

→ Easy modification & maintenance.

→ development task can be easily distributed.



1. scriplet, expression ×
2. Std actions (JSP) ×
3. EL ✓
4. JSTL ✓

} Usage.

Expression Language. (EL)

10/01/24.

Agenda.

1. MVC steps.
2. limitations of std action.
3. Use of EL
4. MVC implementation
5. Need of JSTL
6. diff tags in JSTL
7. using EL & JSTL in blogging app.

1. It is used for displaying data in JSP.
2. Syntax :

$\$\{first.second\}$

- first → - attribute name
  - EL implicit object
- second → - property.

$\$\{first["second"]\}$

This syntax is used when first is array or collection and second is index key.

- first → array/collection.
- "second" → index key

```
<jsp: useBean id="userinfo" class="--" scope="--"/>
Email : <jsp:getProperty name="userinfo" property="email"/>
```

→ $\$\{userinfo.email\}$

1. Short, compact syntax
2. Nested properties are easily accessible

   $\$\{userinfo.address.city\}$ // internally called getter method if not written getter in class

3. Scope of the bean need not be specified.
   ↳ Searching least ~~accep~~ accessible scope to most accessible scope.
   ↳ page request - session application.

Current Time Greeting in JSP.

```
<%@ page import="java.text.SimpleDateFormat, java.util.
    Calendar"%>
<%@ page language="java" contentType="text/html;
    charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<html>
<head>
  <title> Insert title here </title>
  </head>
<body>
    <%
      SimpleDateFormat sdf = new SimpleDateFormat("HH");
      String hh = sdf.format(Calendar.getInstance().
                                    getTime());
      int hours = Integer.parseInt(hh);
      if (hours >6 && hours <= 12)
      {
          out.print("<h1> Good Morning </h1>");
      }
      else if (hours >12 && hours <=16)
      {
          out.print("<h1> Good Afternoon </h1>");
      }
      else if (hours > 16 && hours <=20)
      {
          out.print("<h1> Good Evening </h1>");
      }
      else
      {
          out.print(" <h1> Good Night </h1>");
      }
    %>
</body>
</html>
```

4. It treats null values gracefully.

↳ If available, displayed.

↳ If not available, remain silent.

5. Not allowing reading data from only beans - provides implicit objects - cookies, content, paramets, ...

EL implicit objects.

1. Readily available objects to EL.

↳ 1. param.
↳ 2. paramValues. } request parameters.

$ {param.email}

out.print (request.getParameter ("email"))

paramValues : Used for reading value - parameter having more than one value.

$ {paramValues.city ["0"]}

Multiple selected value goes as:

city : Pune , city : Mumbai

↳ 3. header, headerValues } reading request headers.

$ {header.host}

$ header ["user-agent"]}
↳ info about client's browser.

↳ 4. initParam ← reading content parameters (web.xml)

$ {initParam.adminemail}
↳ name of content parameter

5. Cookie - allows to read into from cookie.

$ {cookie.loginerror.value}

6. Request_Scope.
Session_Scope.
application_Scope

$ {sessionScope.userinfo.email}
↳ Even if it will not written it get executed internally.

MVC implementation.

project.
↳ src — java → controllerservlet
└ beans.
↳ webapp.
└ register.jsp.

valid EL implecet object :→ cookie.

valid get implecet object :→ request, session, out.

JSTL.
↳ JSP Standard tag library.

1. It is a library of tags
2. Not defaultly available to JSP.
3. JSTL jar files needs to be added in the application.
① ↳ location → build path.
→ deployment assembly.

4. ② Adding taglib directive on JSP.

uri always unique

<%@ taglib uri = " " prefix = " " %>

1. identifier for tag library.

2. Which appears in the form of URL.

used as libraries.

eg.

<%@ taglib uri = "https://java.sun.com/jsp/jstl/core" prefix = "c" %>

unique, it is just not like URL but unique created by sun microsystems so that no body else will use it

/jstl/sql
/jstl/format
/jstl/function
} libraries.

③ Use the tag on JSP.

<c : forEach var : "hobby" items = "${hobby list}">

<li> ${hobby} </li>

same thing has to continue

</c: forEach>

</UI>

Steps:

1. Adding library. (JAR file).

2. Build path.
   → classpath.

3. Use taglib directive

Core Tags.

1. for Each → Iterating

2. <c: if test ="- - ">

3. </c : if>

5. <c: set var = " " value = " "/>

4. <c : choose>
   ↑ works as switch case.

   <c: when

   <c: otherwise

   </c : choose>.

SQL Tags.

1. <%. @ taglib uri ="- - - -/sql"
   prefix ="sql"%>

2. <sql : set Date Source driver ="-
   - - " url="- - -", user ="- - -",
   password = "- - -" var ="- -"/>

3. <sql : query

4. <sql : update

Formatting tags.

1. library which helps in formatting number, date, time, time zone.

<% @ taglib uri ="- - - /format"
prefix = "format"%>

< format : formatDate date =". -"
pattern =" - -" />

< format : formatNumber

< format : setTimeZone.

Number of tags are more in all libraries except SQL.

<foreach tag is most important

——— X ———

Blogging_App.

$\underbrace{\text{user, topic, comments}}_{\text{db.}}$

index.jsp.
↓
view topics → /getTopics.
view comments.

needs support of Topic class (bean)

↓

1. db connection.
2. topics table.
3. iterating resultset
4. every record - instance of Topic
   ↓
5. adding collection

for collection part we use set collection because we don't to repeat topics.

6. forwarding request to JSP
↳ done with the help of Servlet.

/showTopics.jsp

```
<%@taglib uri="---"/core"
 prefix="c" %>

<ol>
 <c:forEach var="topic"
  items="${applicationScope.tlist}"
 <li><a href="---">
  ${topic.name}.
 </a></li>
 </c:forEach>
</ol>

href="getComments?topicid=$
  {topic.topicid}"
```

Custom Tags.

Tags developed according to application's requirement.

Steps:

1. TSTL (adding jar files)
2. implementing tag in the form of class.
3. Describe tag in XML file.
4. Adding taglib directive in JSP.
5. actually using tag - prefix and attributes.
6.

```
<mine:greet user="abc" />
```

prefix → mine; tag name → greet; attribute → user

↓

Welcome abc.

Server has the responsibility to translate the tag at the time of translation. and suppling translation is our responsibility.

1. Develop a class -

Business logic to be used at the time of translation.

↳ web server uses this class while translating custom tag to servlet code.

— implement Tag interface directly or indirectly.

```
public class Greet Tag extends
    Simple Tag Support
{
                    → attribute name in tag
    String user;   <mine:great user="Abc"/>

    // getter & setter.

    public void doTag () --
    {
hidden → [this].get JspContent ().get Out().
                    out
    print ("<hi style="---" >
        Welcome + user + "</hi>");
```

```
this.get Jsp(ontext().getout().print (" <hi style=" ">
    welcome + user + "</hi>");
```

2. Describing the tag in the form of XML file (.tld)

```
<taglib>  → There can be multiple <tag> in <taglib>
    <tlib-version> --
    <jsp-version> --
    <short-name>mytags</short-name>

    <tag>
     <name>greet </name>
     <tag-class>tag classes.GreetTag</>
    <body content> empty </>
    <attribute>
      <name>user </name>
      <required> true </required>
      <rtexprvalue> false </>
    </attribute>
    </tag>
    </tag lib>
```

one <taglib> can have multiple <tag> and one <tag> can have multiple attributes.

<rtexprvalue>

↳ run time express value.

mytags.tld.

↳ put the file at location /WEB-INF/mytags.tld.

3. Adding taglib directive.

path of tld file.

```
<%@ taglib uri = "/WEB-INF/
        mytags.tld" prefix="mine"
%>
```

4. Using the tags.

```
<mine:greet user="  " />
```

Class : GreetTag.

Super class : javax.servlet.jsp.tagext.
SimpleTagSupport.

MAVEN

→ Maven is build tool.

→ Maven is about building of enterprise application

↳ generating source code (eg bean)

↳ for example in the bean class we provide db to the tool and it will generate us the source code. classes.

↳ generating documentation.

In Servlet we always get

```
/**
 *
 */
```

This is called javadoc comments.

This contents the documentation for code.

↳ compilation of source code.

project
↓
build automatically

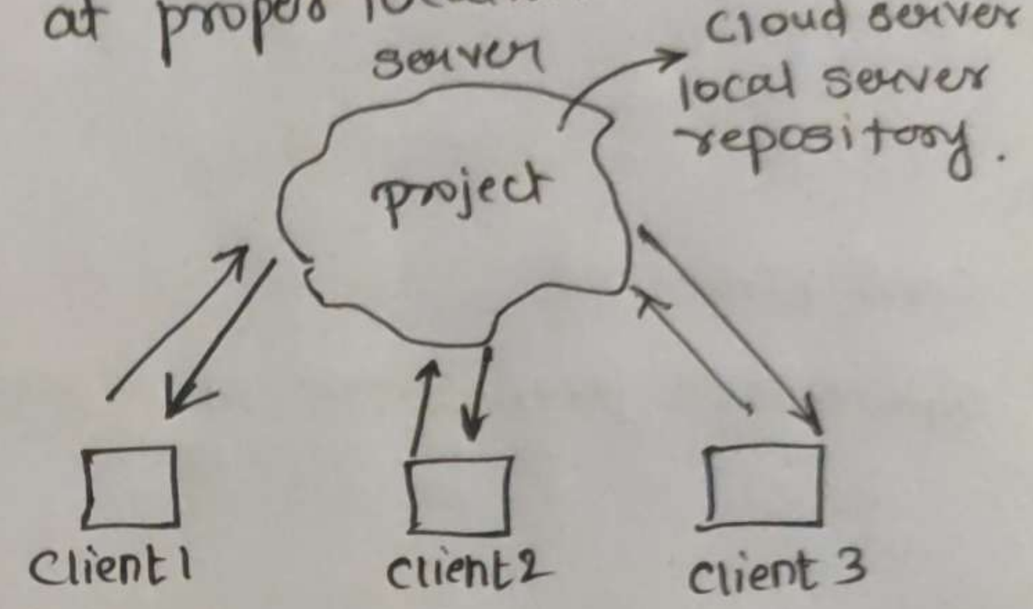
↳ unit testing.

↳ packaging in the form of jar / war / ear files.

jar ↓ java archive
war ↓ web archive
ear ↓ enterprise archiver

↳ deploying in right location

We provide the application to client properly not in multiple classes for that we deployed it in web server at proper location.



For deploying or uploading we require certain tools for uploading project at server.

To avoid

↳ human error.

↳ time - more required.

To avoid this we use tool like maven.

Maven is product of Apache Software Foundation.

Initially Apache ANT was used.

What is Maven?

1. Maven is a robus. project management tool based on the POM architecture (project object model).

2. It is used for project build, as well as for managing dependencies and documentation.

3. Maven is a tool that can be used to create and manage any Java-based project.

4. Maven is a build automation tool that is mostly used in Java projects but Maven can also be used to create and manage projects written in other languages such as C#, Ruby, Scala and others.

5. The Apache Software Foundation hosts the Maven project, which was previously part of the Jarkarta Project.

When should one use Maven

1. When the project has a large number of dependencies. Then, using Maven, we can easily manage those dependencies

2. When the version of a dependency changes frequently. To update dependencies simply update the version ID in the pom file.

3. Maven makes it simple to handle continuous builds, integration and testing.

4. When we need a quick way to generate documentation from source code, this is the tool we use. It helps in compiling source code and then packaging it into JAR or ZIP files.

Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

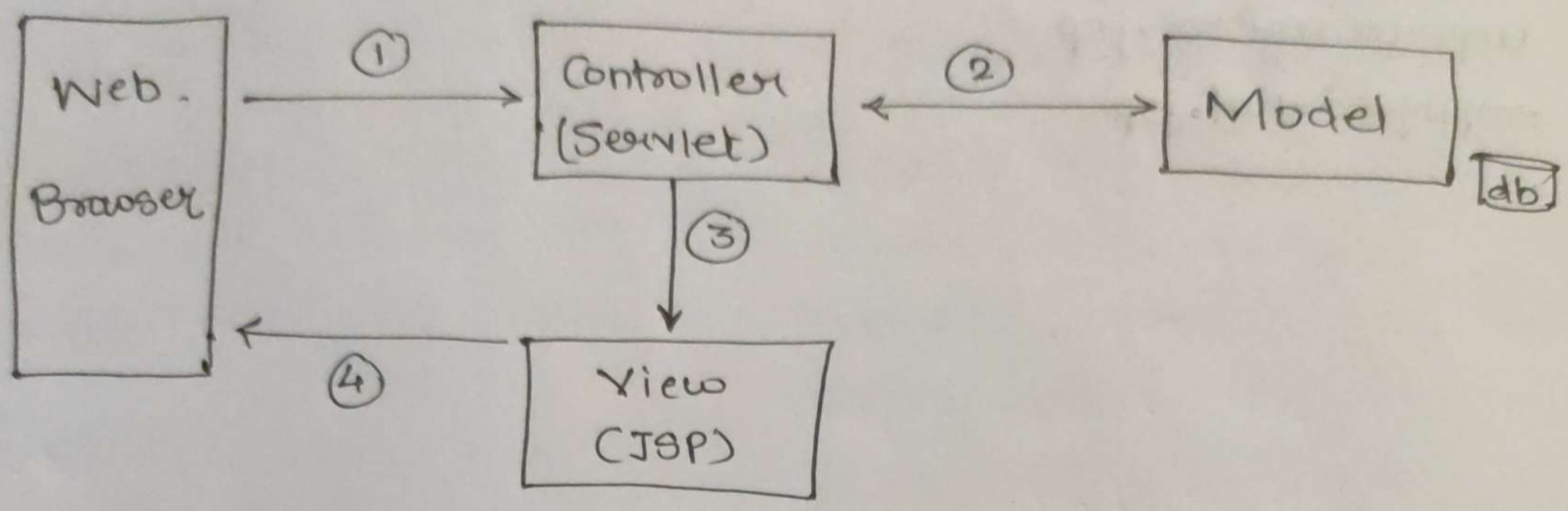
Maven provides developers ways to manage the following:

1. Builds
2. Documentation.
3. Reporting.
4. Dependencies.
5. SCMs.
6. Releases.
7. Distribution.
8. Mailing list.

Maven handles compilation, distribution, documentation, team collaboration.

Resigistration form Using JSP + Servlet + JDBC + MySql.

Model-View-Controller (MVC).



MVC Pattern.

1. MVC is a pattern used in software engineering to separate the application logic from the user interface.

Ans

2. MVC has 3 layers.
3. Model defines the business layer of the application.
4. The Controller manages the flow of the application
5. View defines the presentation layer of the application.

Employee Register Form.

First Name
Last Name
User Name
Password
Address
Contact No

Submit

Development steps.

↳ Create an Dynamic Web Project
↳ Add Dependencies.
↳ Project Structure
↳ MySQL Database Setup
↳ Create a Java Bean - Employee.java.
↳ Create a Employee Dao.java.
↳ Create a EmployeeServlet.java
↳ Create a employeeregista.jsp.
↳ Create a employee detail.jsp
↳ Demo

Java Bean → Employee.java.

EmployeeDao.java.

EmployeeServlet.java.

employeeregister.jsp

employeedetail.jsp

Need of build tool.

11/01/24

Along with development.

↳ source code.

↳ documentation.

↳ testing.

↳ creating distribution unit

↳ deploy it at right location.

Done automates all the phases

1. Due to which human error get avoided.

2. time required is lesser.

Agenda.

1. need of build tool
2. maven - intro
3. maven - basics.
4. using maven for console application.
5. using maven for web app
6. Intro to ORMapping
7. Intro to hibernate
8. Hibernate configuration
9. saving object.

Steps for Maven

↳ downloaded freely.

↳ installation.

↳ set the home path - command prompt

↳ mvn --version.

↳ eclipse - maven.

There are tools like maven:

1. Groovy
2. Gradle.

Maven

maven build tool.

↳ POM - Project Object Model.

↓

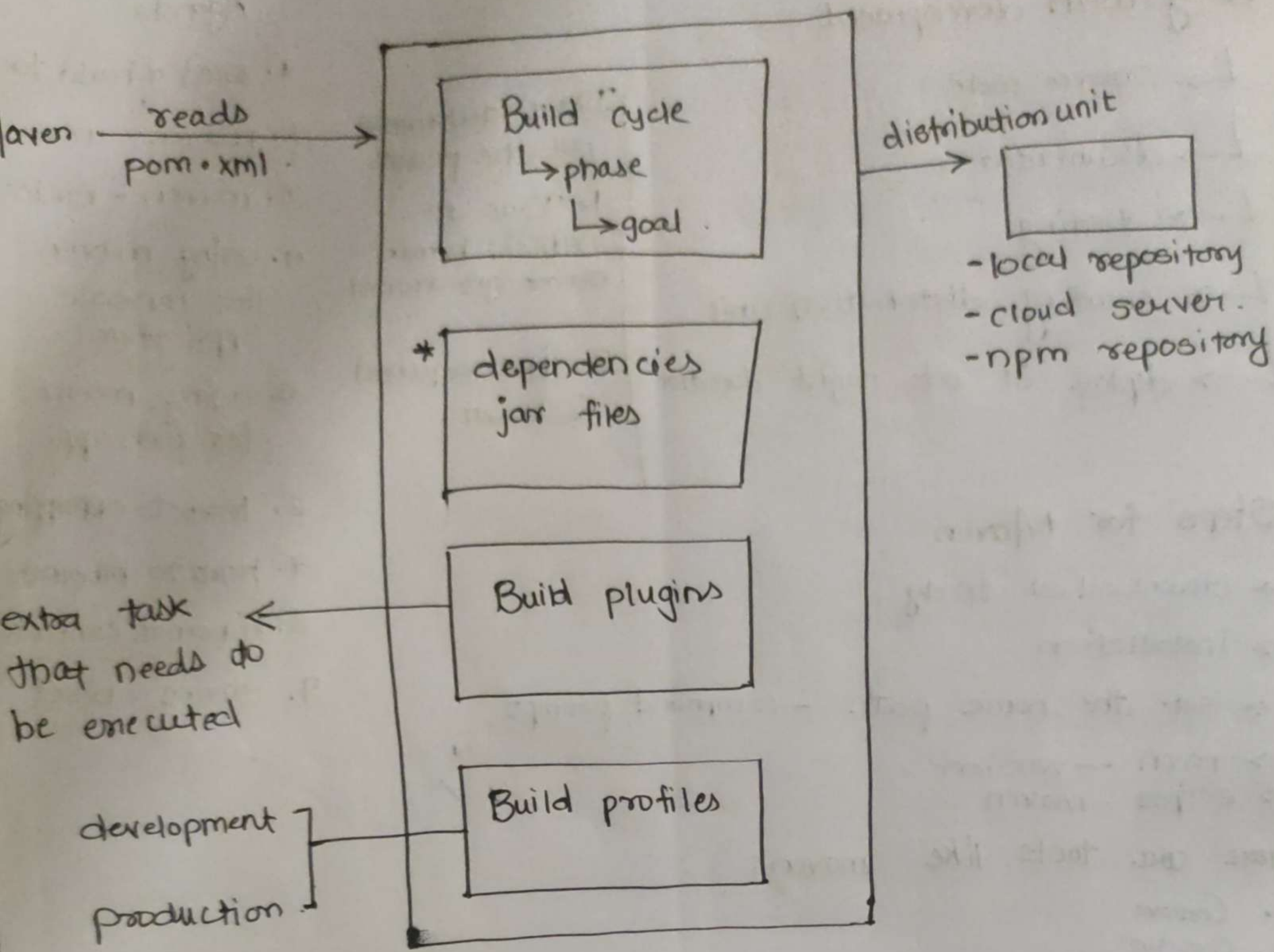
Pom.xml

↳ give all the requirements of the project.

↳ root folder of maven application.

↳ root tag of pom.xml is <project>

maven will read the pom.xml and build the application based on it.



One build cycle has many phases and one phase has many goals.

Maven - 3 build cycle.

① default - normal development and testing.

② clean - remove previous and start fresh install & deploy.

③ site - creating installation unit along with documentation.

Build phases.

- validate.
- compile.
- test
- install - mvn install
- deploy.

if we gives command maven test then it will execute all the previous steps.

mvn install

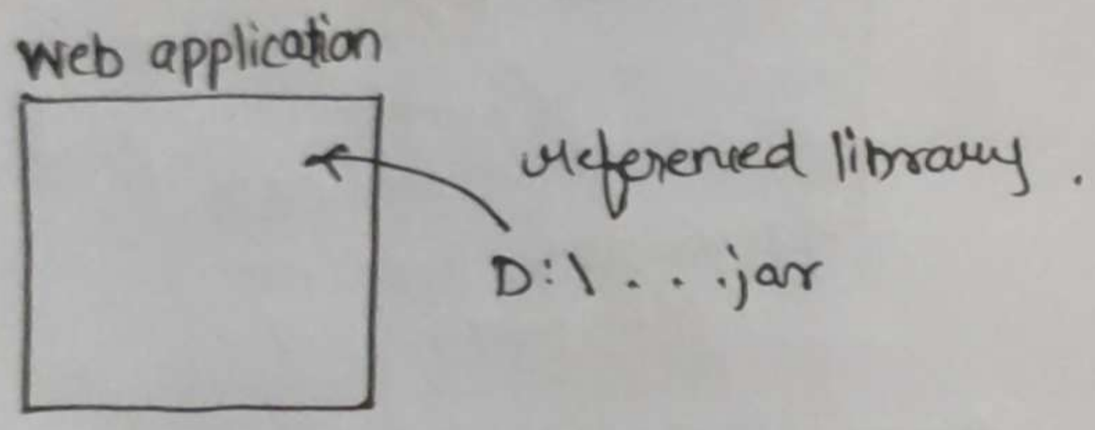
mvn clean deploy.

Dependencies Management

When we change the location of the dependency that project will not work.

Even when we want to share the project we have to share the whole project with all jar files.

All this problems are solved by maven.



- jar file requirement.
  - mentioned in pom.xml.

pom.xml.

```
<project>
  =
  <dependencies>
    <dependency>
      :
    </dependency>
  </dependencies>
</project>
```

specification of jar file

mvnrepository.com

mysql-connector 8.0.26.

Working of Maven

Searches jar files in:

1. local repository
2. Central repository
3. Remote repository.

*

1. Local Repository.

C:\users\abc\.m2\repository.

maven will check wheather dependency (jar) is available at repository or not, if not then it will download it from central repo.

If present than at first time will take time but after that doesn't require much time as dependency is present at local repository.

2. Central repository.
   1. Internet location.
   2. download jar files and saves it in the local repository.

3. Remote repository.

library which should only available for clients that lib are at remote repo.

eg. JBoss library.

* CCEE

Sequence:

Local → Central → Remote.

Application Structure in Maven.

1. uniform irrespective of its type

- application root
  - src
    - main
      - java — all business logic classes
      - resources — addition xml / properties
    - main
      - test — JUnit test classes
      - resources — support files
    - webapp
      - .jsp
  - target
  - pom.xml.

Maven Project.

Step1: New file → Maven Project
↓
maintaing pom.xml for specifying all jar file requirement.

Maven build - clean all project
Maven clean - clean all project
Maven test - test the code

Note: In Java all xml and dependencies are in source folder.

Note: Just like selenium JUnit is used for testing it is a test classes.

Console Application

Project
↳ maven
↳ (✓ simple project)
→ Groupid
(organization's group id)
→ Artifact id
(followed for project name)
→ packing (jar)
↳ for console
(war)
↳ for web
→ Java class
(jdbc code)

download (mvn repository)
↳ mysql-connector Java
↳ version
↳ copy maven code
paste at <dependencies>
paste here
</dependencies>
↳ c
↳ users
↳ admins
↳ .m2
↳ repository
↳ mysql

Web Application

Project
↳ maven
↳ propeties
↳ Web project ↳ targeted Runtime
↳ ✓ Apache
↓
org.apache.maven. (3rd one)
archetype
↳ copy jstl
↳ past at pom.xml
<dependency>
</dependency>

OR Mapping Tool. (Hibernate)

Object Relational Mapping Tool.

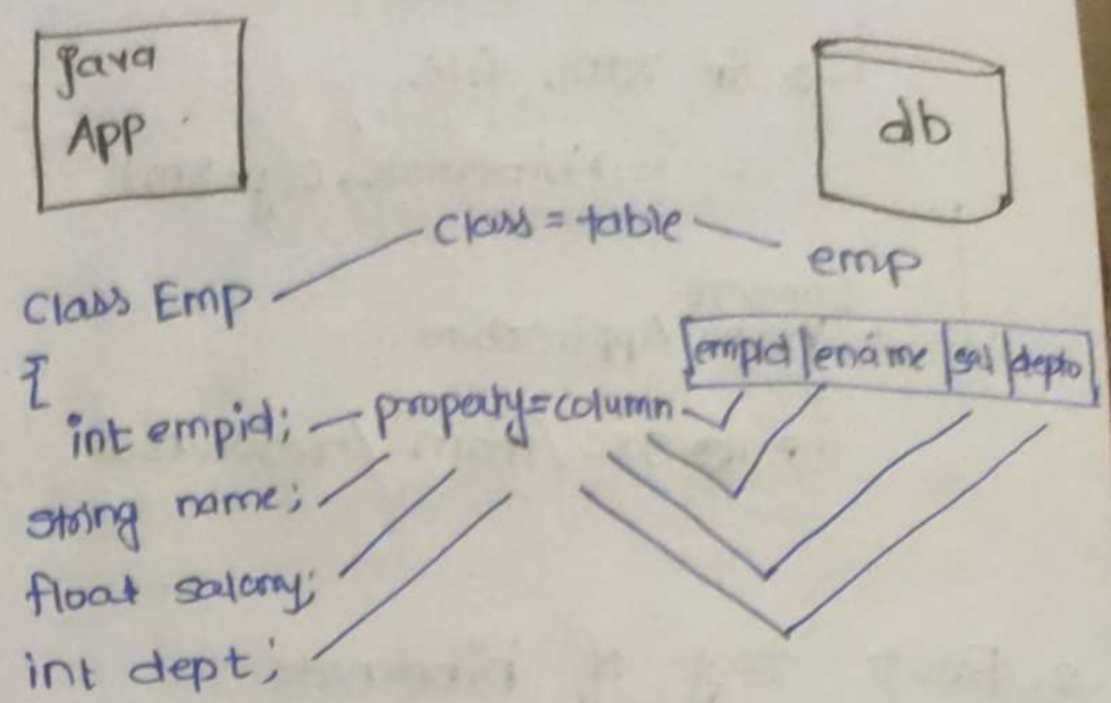
It is a tool which helps in without involving any use of code it helps in connecting with db.

It helps in connecting relational from database to the ~~classes~~ object.

Hibernate is one of the example of OR Mapping tool.

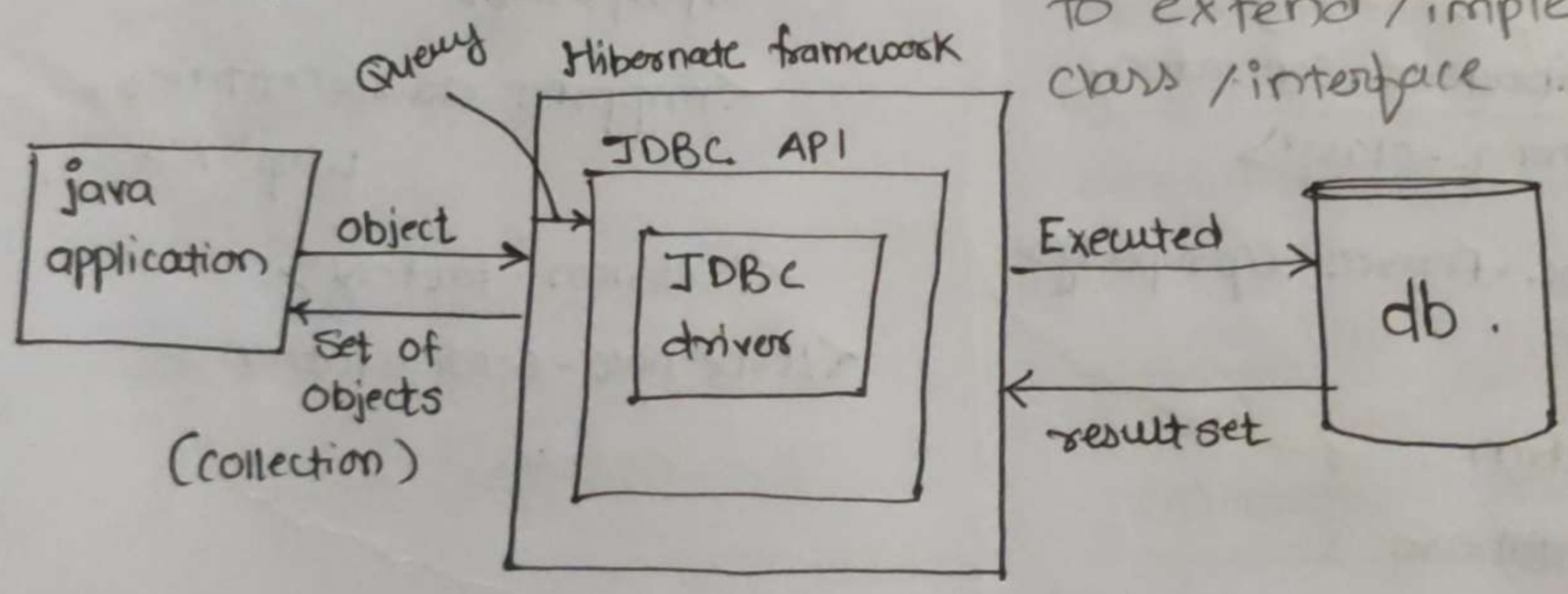
Other examples:

- Hibernate,
- toplink.
- eclipse link.



Note: JDBC driver is 'must required for the working of Hibernate.

Note: Hibernate is a non-invasive framework means it won't forces the programmers to extend / implement any class / interface.



Requirement of Hibernate.

① Connection related information.
↳ save in xml

② Mapping information between class & table
↳ in format — xml / annotations

Hibernate is invented by Gavin King in 2001.

1. Connection related information

↳ Web ~~Console~~ Application.

↳ in xml file

↳ hibernate.cfg.xml

↳ Console ~~Web~~ Application.

↳ src/main/resources.

2. Root tag of hibernate.

<hibernate-configuration>

</hibernate-configuration>

3. Child tag of <hibernate-configuration>

<session-factory>

child elements of session factory

4. Connection related information

* → no changes

<property name="hibernate.connection.driver-class">

com.mysql.cj.jdbc.Driver </property>

5. Dialect Information.

* ↳ Type of database

<property name="hibernate.dialect">

org.hibernate.dialect.MySQL5Dialect

</property>

6. Optional (Query Generate to display on Console)

<property name="hibernate.show-sql">

* true </property>

hibernate mapping

data defination language

<property name="hibernate.hbm2ddl.auto">

update </property>

<property - - - session context class>

7. Mapping class.

* ↳ class used for mapping object and entities.

<mapping class="entities.Emp"/>

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver-class">

<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>

</property>

<property name="hibernate.show-sql">true

</property>

<mapping class="entities.Emp"/>

</session-factory>

</hibernate-configuration>

Develop Mapping Class

↓

Bean class + annotations
(mapping with database)

The state of this is handled by hibernate.

```
@Entity
@Table (name ="emp")
class Emp
{
    @Id
    int empid;

    @Column (name ="------")
    String name;

    @Column
    float salary;

    @Column
    int deptno;

    // getter, setter

    // default Constructor.
}
```

annotation name start with capital letter

@column (name="--") ← if this object and db entity name are not same then we use it

JPA annotations.

↓

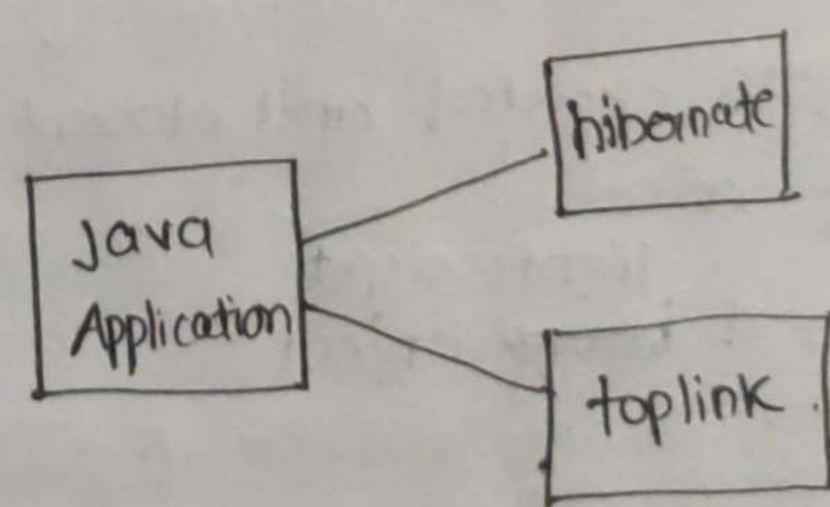
Java Persistance API

→ Suppose we are using Hibernate as ORM tool and suppose we want to chance the tool to toplink for example then how can we supply the information about connection from hibernate to toplink. for that some specification has to added that specifications given by Sun Microsystems is the JPA annotation.

→ Hibernate is implementation of JPA

→ Any ORM can understand mapping class.

→ All annotations in Mapping class are from JPA.



JPA naming / Jakarta naming.

Communicating with hibernate for getting the task done.

⇒ Hibernate API

1. Session factory

1. ↳ equivalent to Connection in JDBC.

= { create service
reads hibernate.cfg.xml
builds session factory.

2. Session factory can create one database connection.

3. Session factory can create n. number of session.

2. Session.

4. One session represents one dialog with database.

5. Session is short lived object.

6. It is equivalent with 3rd & 4th step of JDBC.

7. It gets created and closed only once.

8. It is a ~~heavy~~ lightweight object.

one session is equivalent to one dialog with db

3. Transaction

1. It start the transaction before executing anything in db.

2. commit (), rollback () (explicitly)

Note: Hibernate will convert check exception to unchecked exception and handle it by itself

Steps for Mvn

1. Maven - Skip architecture type selection.

2. Specify groupId, artifactId.

3. Open pom.xml - Specify dependencies - hibernate-core mysql-connector.

4. Add hibernate cfg.xml - src/main/resources

↳ edit as per our requirement.

5. Create mapping class - entity - provide basic annotations. ~~jaxx~~ javax.

for latest version jakarta.

*6. Modify hibernate.cfg.xml for adding mapping class.

7. Create main method
↓
Session Factors, Session, Transaction.

① Maven project
② mvn repo
↳ hibernate
③ paste it to pom.xml <dependency>
④ same for mysql connector
⑤ Create a class

} maven dependency

Util.Date vs sql.Date
↑ date + time
↑ only date

HQL

Hibernate Query Language.

1. HQL is an object oriented Query language, similar to SQL but instead of operating on tables and columns, HQL works with persistent objects and their properties.

2. HQL queries are translated by Hibernate into conventional SQL queries, which then perform action on db.

3. HQL is db independent.

4. HQL has class, properties, table, columns

5. All the clauses of SQL like select, from, group by, order by having are applicable as it is in HQL.

6. Keywords like SELECT, FROM and WHERE etc are not case sensitive but properties like table and column names are case sensitive in HQL

7. HQL (and JPQL) are loosely based on SQL and are easy to learn for anyone familiar with SQL

Advantages of HQL

There are many advantages of HQL.

They are:

1. database independent

2. supports polymorphic queries
   polymorphic → having or occuring in several distinct forms.

3. Easy to learn for Java Programmer.

HQL example.

1. It is an object oriented representation of hibernate Query.

2. The object of Query can be obtained by calling the createQuery() method Session Interface.

```
Query q = Session.createQuery("  ", )
```

" " → HQL query

( ) → result type

SQL	HQL
select * from emp	from Emp
select id, salary from emp	select empid, salary from Emp

SQL.

Select empid, salary from emp

Column name — table name.

HQL.

select empid, salary from Emp

properties — Class name.

Example

Query q = Session . CreateQuery (" from Emp", Emp.class)

This Query tells us it will a single object back or collection of objects

⇒ Example of HQL to get all the records.

```
Query q = session . create Query ("from Emp", Emp.class);
List <Emp> emps = q.get Result List();
for (Emp e : emps)
{
    System.out.println(e);
}
```

This will give us list of elements of Emp object.

⇒ Example of HQL query for retrieving few attribute.

It will return us array of object

record — column.

```
Query q = session . create Query ("select empid, ename from Emp
                                  where salary > 20000", Object[].
                                  class)
List <Object []> emps = q.getResultList();
for (object [] e : emps)
    System.out. println (e[0] + ":" + e[1]);
```

Query Interface Extended

Query Interface is extended further into two:

- 1. Selection Query (Select)
- 2. Mutation Query (DML)

1. Selection Query.

1. It is used for selection Query.
2. This is a generic Interface.
3. So instead of using Query & use of Selection Query for select operation.

2. Mutation Query.

1. It is used for DML operations.
2. This is a generic Interface.

Note: Hibernate also provided to use normal query usage that is usage of SQL. That means we can also use Normal SQL in hibernate.

```
                  HQL
createQuery(" ")
                  SQL
createNativeQuery(" ")
```

But SQL queries are db dependent whereas HQL are not.

Criteria Queries.

1. Criteria Queries are nothing but queries which are to be form dynamically.
2. SQL, HQL queries are write in the form of string.

Whereas Criteria Queries are generated by using method call.

3. This queries are prefered Now days.

Named Queries.

We can create the query and save it with the mapping class.

In other queries the problem is that Java + SQL or Java + HQL will mixed up together creates complexity

So in Named Queries instead creating queries in java code we can create the query and save it with the mapping class. And then get it extracted from that mapping class and then specify what is the query we want to retrieve and how that query ~~query~~ should be executed.

Summary

HQL → Java + HQL.

Native queries → Java + SQL.

Criteria queries → Method call

Named queries → saved at mapping class

HQL.

```
Query q = session.createQuery ("from Emp", Emp.class);
List<Emp> emps = q.getResultList();

for (Emp e : emps)
{
    System.out.println (e);
}
```

Native querie

```
String sqlQuery ="Select * from emp where id=1;

SQLQuery query = session.create SQLQuery (sqlQuery);

query.addEntity(Emp.class);
query.setParameter("param", "some value");

List<Emp>emp = query.list();
for (Emp e : emp)
{ System.out.println(e);
```

Criteria queries

```
transaction = session.beginTransaction();
Criteria criteria = session.create Criteria (Emp.class);
criteria.add (Restrictions.eq ("empid", 20));
List<Emp> resultList = criteria.list();

for (Emp empid : resultlist)
{
    System.out.println (empid);
}
transaction.commit();
```

Named queries.

```
@Entity.
@NamedQueries ({
@NamedQuery (nam ="findAllProducts",
    query ="FROM product"),
@NamedQuery (name ="findProductById",
    query ="FROM Product where id =
    :productId")
})
```

Criteria Queries.

1. Dynamically generated.

2.