L:1 D:93

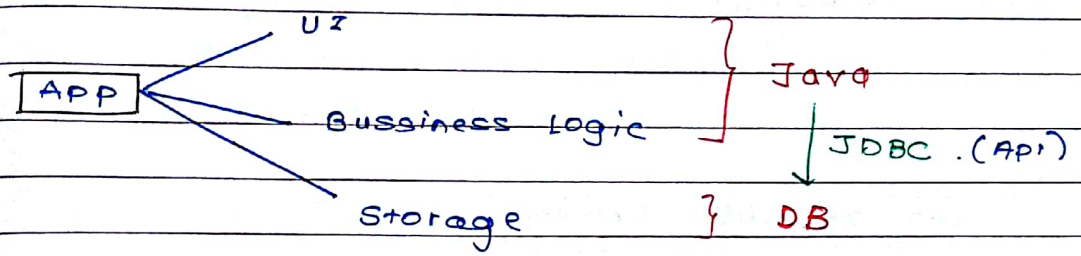
DATE 16 11 2017

ADVANCED JAVA

Desktop app: Standalone application (single pc) w/o client-server arch.

Web app: Client-server app.

Advanced Java: -> For web applications.



(Java DB connectivity)

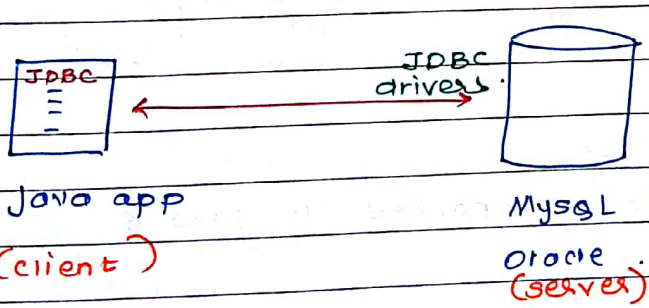
JDBC

(predefined library. (classes, interfaces ...))

- It is an API that make data transaction possible between java application & any RDBMS.

JDBC api is inside this package.

```
java.sql;
```



JDBC can interact with any DB.

But how?

for dyff. DB, there are drivers.

DB vendors (MySQL, Oracle) provides JDBC drivers.

PAGE

classmate

DATE

JDBC Drivers

JDBC driver -> it is a middleware S/W that is used to do communication between the java application & RDBMS

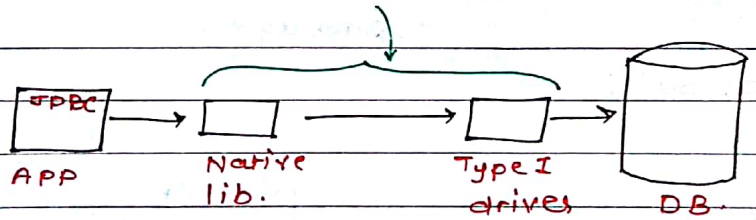
↳ Just like H/W drivers like keyboard. OS can interact with any keyboard manufactured by any vendor. because vendor provides drives for keyboard.

Types of JDBC Drivers

There are 4 types of JDBC drivers.

i) Type-I Driver / (JDBC odbc bridge) — Written in Native code (C/C++)

- Some native libraries are required on client m/c
- It is slowest of all



- Free of cost.
- It is not supported in java 8

Native lib. will interact with drives.
∴ Slow.

DATE

ii) **Type II Drives / (Java Native API Drives)**

- It also requires some Native code (lib) on client side.
- It is faster than Type I
- Not Free.

iii) **Type III Drives / (Java N/w protocol Drives)**

- No need of Native lib. on client m/c.
- With type 3 drives, you will only need 1 drives to access any DB on server. (MySQL, oracle, DB2, ...).
- ~ Universal adapter.
- It alone can communicate with any DB
- It can communicate with diff. DBs present in N/W

- It is fastes than Type I & II.

(If you are using multiple DBs, you will need separate drivers for separate DBs in case of Type I & II).

iv) **Type IV Driver [(Pure Java Drives / JDBC Protocol Drives)]**

- Type IV Drives is written in java.
∴ it is known as pure Java Drives).
-∴ It is fastest of all the drivers)

Type I, II & III are written in Native language.
we will use type IV drives.

PAGE

DATE

You'll need to download the drivers.
It is provided by the DB vendors

Steps to write JDBC code

I) **Loading Driver Class** (Configuring JDBC drivers)

use

Used to load class manually

```
Class.forName("driver_class_Name");
```

Built in class 'Class'

Static method

II) **Creating JDBC Connection**

use method

```
DriverManager.getConnection("url", "username", "password");
```

III) **Creating / writing query statements.**

Using one of the three ways.

* General Statement : Statement
* Prepared Statement : PreparedStatement
* Callable Statement : CallableStatement

} Interfaces

- query statement is represented as an object using above interfaces.
- Object will encapsulate the query.

DATE

IV) **Executing Query**

↳ Using one of the three ways:

- execute()
- executeUpdate()
- executeQuery()

execute(): for stored functions and procedures.

executeUpdate(): For DML queries.

executeQuery(): For DQL queries (SELECT)

↑ Recommended usage.

(e.g We can exe. DML using execute() but it is not recommended)

V) **Close the connection**

↳ close() method.

It is not compulsory but it is a good practice to close the resources.

Eclipse:

↳ Workspace :- Will contain all your projects

Project :- Will contain your java file.

- Create a workspace.
- Create a new project.
  - File menu -> Other -> java Project -> Next -> give proj. name -> next -> Finish -> Open Perspective / yes

↳ will create a proj with Src folder.

right click on Src -> create a class -> use package, give Name to package -> Give Name to your class -> check on psvm(...)

close the project afterwards.

printStackTrace(); -> will give the problem due to which exception occured

Connection is an interface which represents a logical link betn java app & DB.

DATE

Creating a simple JDBC code

- Download Drives file (Type IV) ..
- It is a .jar file.
  - ↳ mysql-connector-java-5.1.23.jar
- Create a new project
- click on Libraries tab.
  - -> Add external jar. & select the downloaded jar (Drives)

jar is a collection of .class files.

goto src & create a new class inside some package.

DEMO

```
psvm(...)
{
    try
    {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/MyDB",
             "root", "");  // creating connection
        Statement s = con.createStatement(); // creating stmt
        String q = "create table Student (rno int
                    primary key,
                    Sname varchar(10),
                    course varchar(10),
                    Fee float,
                    dob date);   // query
        s.execute(q); // Exe query.
        con.close(); // closing connection.
    }
```

Type 4 driver provided by MySQL

class

package.

class

throws checked exc. SqlException

Interface

port

url

Interface (it encapsulates query. With it the query is compiled & executed)

returns boolean

PAGE

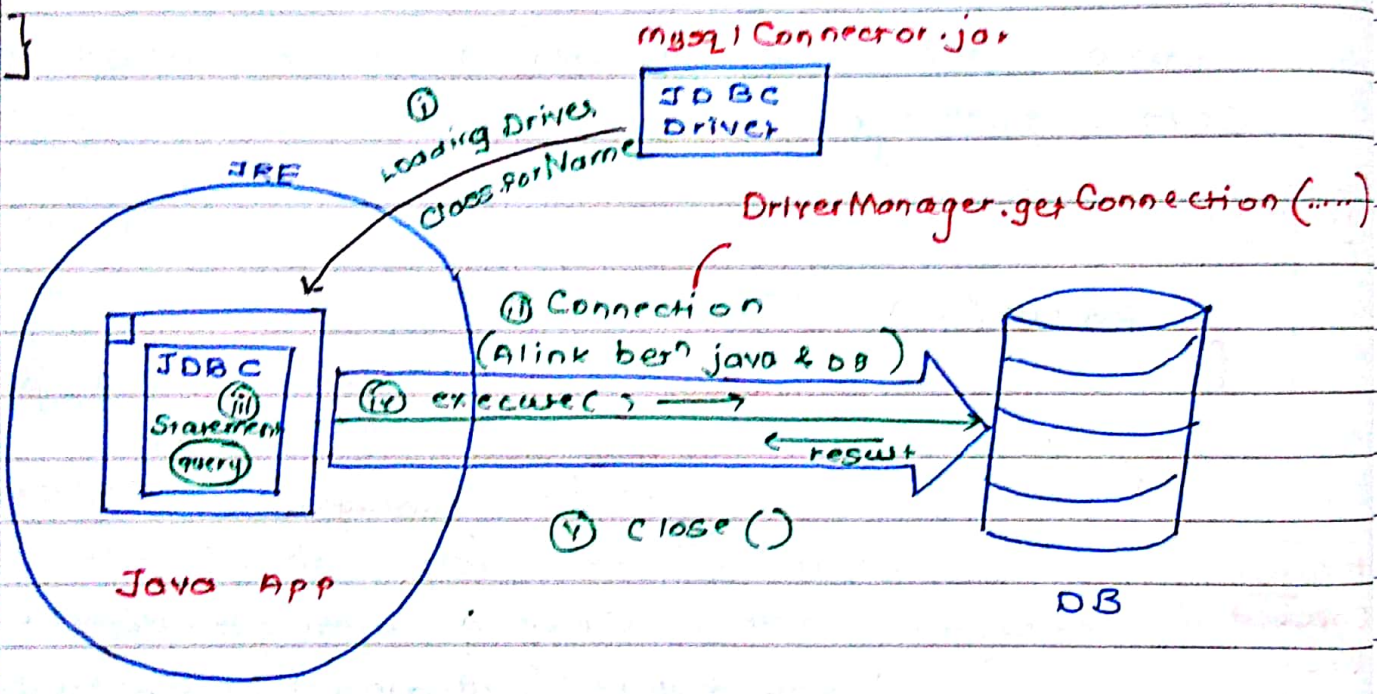
classmate

DATE

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

if Drives class is not available. (Checked)

getConnection(). if conn. is failed



DATE

Insertion

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

    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/MyDB", "root", "");
                                          //connection

    String q = "insert into student values
        (101, 'smith', 'pgdac', 90000, '1995-11-13')";
                                          //query

    Statement s = con.createStatement(); //statement

    int i = s.executeUpdate(q);          //Exe.
    Sopln(i + " record is inserted");

    con.close(); //close
  }

  catch(...)
  {
    ...
  }

  catch(...)
  {
    ...
  }
}
```

To exe. DML.
∵ DML returns: "x rows affected" ∴ return type is int.
(in our case it will return 1 ∵ we are inserting 1 row).

(update, delete are similar. only query will change.

PAGE

classmate

DATE

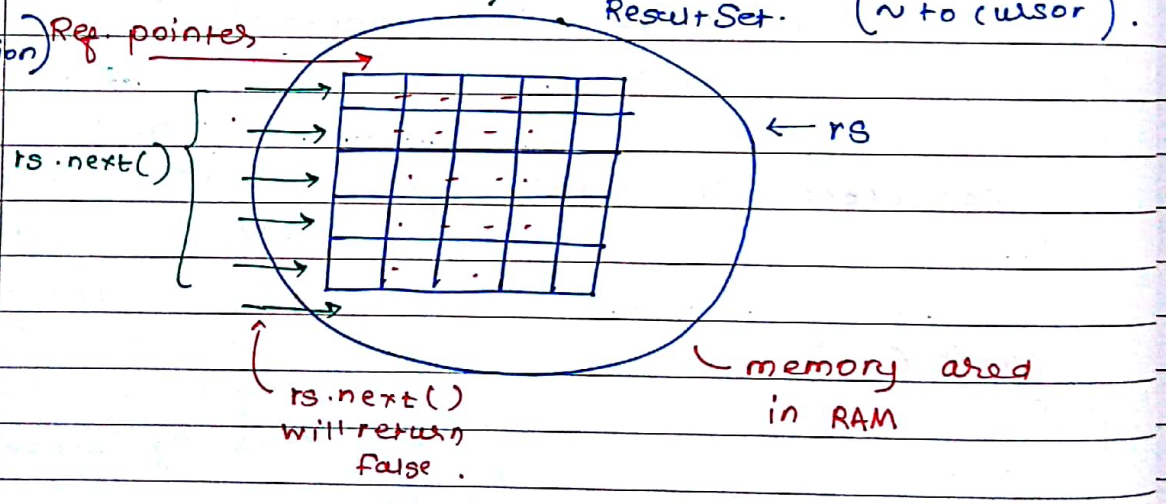
SELECT

query is first compiled & executed.

```
String q = "select * from student";
Statement s = con.createStatement();

ResultSet rs = s.executeQuery(q);

while (rs.next())
{
    Sop(rs.getInt(1)+"  "+rs.getString(2)+"  "+
        rs.getString(3)+"  "+rs.getFloat(4)+"  "
      + rs.getString(5)

}
```

interface (java.sql)

column



`rs.next();` → will move ref pointes to the next record & if the record is present in the curent pos^n it returns True if not present it will return false.

To move from ~~curser~~ one record to another record.

DATE

s executeQuery(q);

↳ will execute SELECT & will return all the rows. it will store all the rows fetched in a RAM in dedicated area. This area in RAM is called 'ResultSet'. (~ to cursor in Databases.)

← if after the while(), if we write. s.executeQuery(q); again,

Then the ref. pointer will be brought back to its initial position. i.e. above the first record.

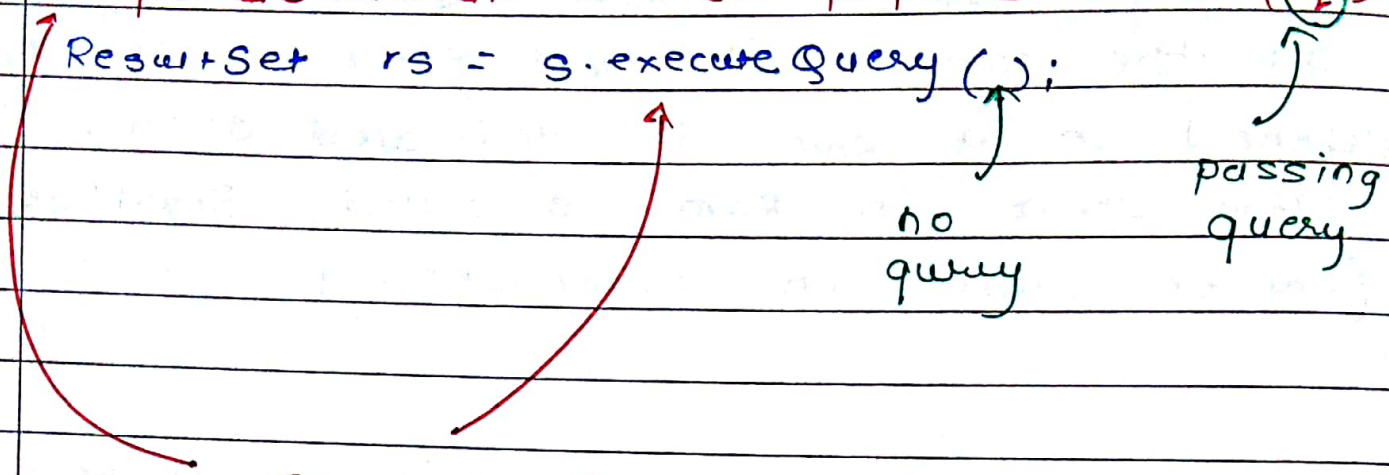
The query will compiled AGAIN & executed. (:: Statement).

(Not suitable if you want to exe. same query multiple times)

DATE

Using PreparedStatement :

```
String q = "select * from student";
PreparedStatement s = con.prepareStatement(q);
ResultSet rs = s.executeQuery();
```



In case of preparedStatement, the ~~eq~~ query is compiled only once - unlike Statement.

if we write s.executeQuery(); again, the query will be executed directly.

It is suitable if you want to execute same query multiple times.

PREpared :

Pre-compiled.

Compile Once

execute multiple times

DATE

Advantage of PreparedStatement.

```
String q = "insert into student values (?,?,?,?,?)";
```
(1 2 3 4 5)

```
PreparedStatement s = con.preparedStatement(q);
s.setInt(1, rno);
s.setString(2, name);
s.setString(2, course);
s.setFloat(4, fee);
s.setString(5, dob);
```

↑ Var name.

```
int i = s.executeUpdate();
```
↑ no query

⇑ Makes writing queries Easy.

Assignment:

1-9 Assigments

L : 2 D : 94

DATE 17 11 2017

CallableStatement

↳ It is used to call stored procedure/fun^n

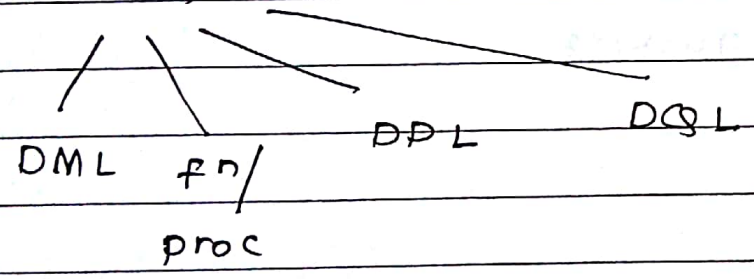
```
CallableStatement s = con.prepareCall("{call deleteStudent(?)}");
s.setInt(1, 104);
boolean b = s.execute();
```

Any query can be executed

universal method

execute() returns boolean value.

execute()



execute() methods can execute any query But, if the query returns a resultSet, execut() method returns 'true'.

executeQuery() returns ResultSet.
It is a dedicated method for dql.

DATE

```
q = "select * from student"

boolean b = s.execute(q);

if(b)
{
  ResultSet rs = s.getResultSet();
    .
    .
}
```

```
ResultSet rs = s.executeQuery(q);
```

execute()
(DQL, DML, DDL)

executeQuery()
(returns resultset).

executeUpdate().
(returns no. of rows affected)

DATE

ResultSet

Traversing methods
(next()...)

getter methods
(getInt(1) ...)

rs.getInt(1);

column no

OR

rs.getInt("rno");

Column Name As per table.

first column of the resultset returned by query.

rs.next(); → To traverse the ResultSet in Top to bottom direction

rs.preveous() → To traverse the ResultSet in Bottom to up direction.

rs.last() → cursor will jump to last record.

rs.first() → ... first record.

classmate

PAGE

DATE

PreparedStatement s = con.prepareStatement(q, ResultSet.

We can update the contents of ResultSet.

# ResultSet rs = s.executeQuery();
rs.last();
rs.updateInt(1, 777);
rs.updateString(2, "ABC");
rs.updateString(3, "NONE");
rs.updateFloat(4, 7000.77F);
rs.updateString(5, "2017-11-11");
rs.updateRow();

PreparedStatement s = con.prepareStatement(q, ResultSet.CONCUR_UPDATABLE);

It is a const. in ResultSet class.

Stands for concurrency

To make ResultSet compatible for updates

→ We are updating the last record in result set but the changes are also reflected in DB table. (∵ concurrency).

→ ∴ we are updating the table w/o "Query".

DATE

Transaction

Transaction -> set of queries (operation).

either all queries must be executed or none.

Transaction is 1 logical operation (Fund Transfer).

↳ But 1 Fund transfer involves to queries 1 for credit & other for debit.

Transaction can have just one transaction.

Transaction is a unit of work.
- It is an indivisible unit of work.

DDL commands are autocommit.
DML commands are not autocommit.

But w: in java, when you execute DML commands they are committed automatically.

There is a method
con.setAutoCommit(boolean);

↑ It is true by default.

DATE

As a developer you should control this behaviour.

∴ con.setAutoCommit(false);

↳ Now, we'll need to manage the query implicitly.

↓

con.commit();

```
// Fund Xfering from acc 100 to 101
q = "update acc set balance = balance -1000 where
                                accno = 100 " ;

con.setAutoCommit(false);

Statement s = con.createStatement();

int i = s.executeUpdate(q);
Sopln("debit");

q = "update acc set balance = balance +1000 where
                                accno = 101 ;

int j = 0; <— Assuming H/W failure ∴ credit fails
// j = s.executeUpdate(q);

if (i == 1 && j == 1)
{
    con.commit();
}

Sopln("credit");
```

↑ It will not reflect changes ∴ credit is failed.

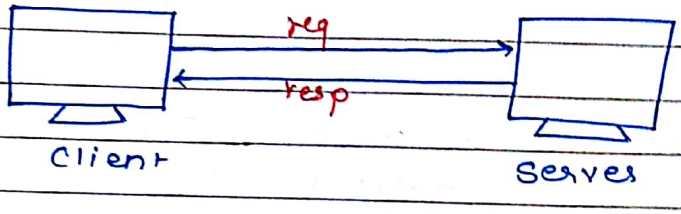
PAGE

classmate

DATE

Web Application:

- It is a client-server application.
- S/W running on server m/c can be used by client.
- Server is a service provider.
- Client is service consumer.



Server client app. is all about xferring data beth client & server.

All the client server app are not web app.

client server app which uses HTTP protocol for data xfer => Web application.

Client generates a request.
Server gives a response.

HTTP works on req.-res paradigm.

2 m/c must be connected physically (N/W)

logical connection is also necessary.
i.e client S/W & server S/W must be connected.

- Req is generated
- Res is given
- connection is closed!

classmate

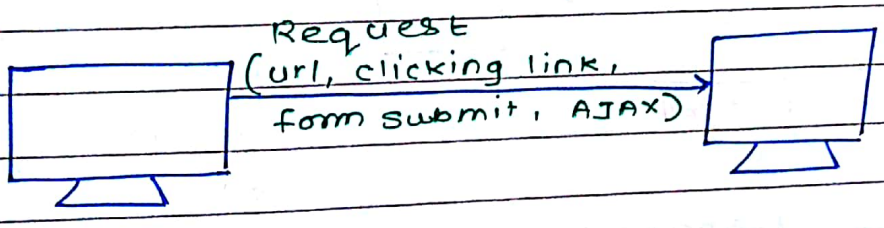
PAGE

m/c with web browsers ⟹ Web app client m/c.

m/c with web server s/w. (Apache, Glassfish) ⟹ Web Servers.

Web browser ——— web server.

↑ logical conn bet^n client s/w & server s/w.



Any request ultimately gets conv^d to URL!

What is url?

↳

https://domain name/path of service.

↑ name of your web app.

DATE

with req. we are establishing conn^n (logical) betn client s/w (browser) & server s/w (Apache)

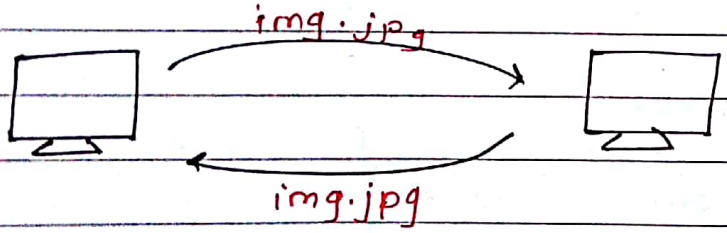
HTTP is a stateless protocol.

↳ ie. it doesn't remember the client
it will just accept request
send response &
destroy the connection.

Response generated by server is a static response.

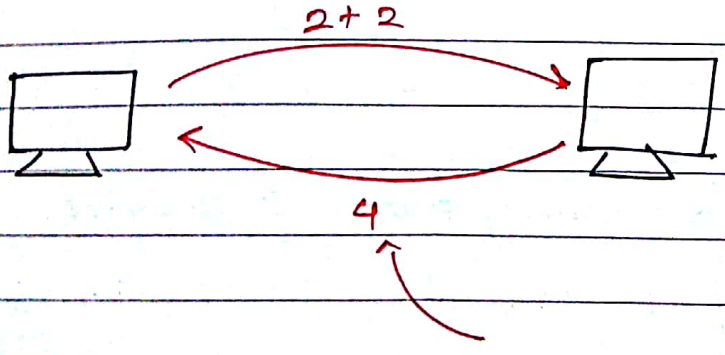
Static response.

↳ request for image.



Dynamic response.

↳ request for 2+2



Processing on server is req^d.

∴ Dynamic response.

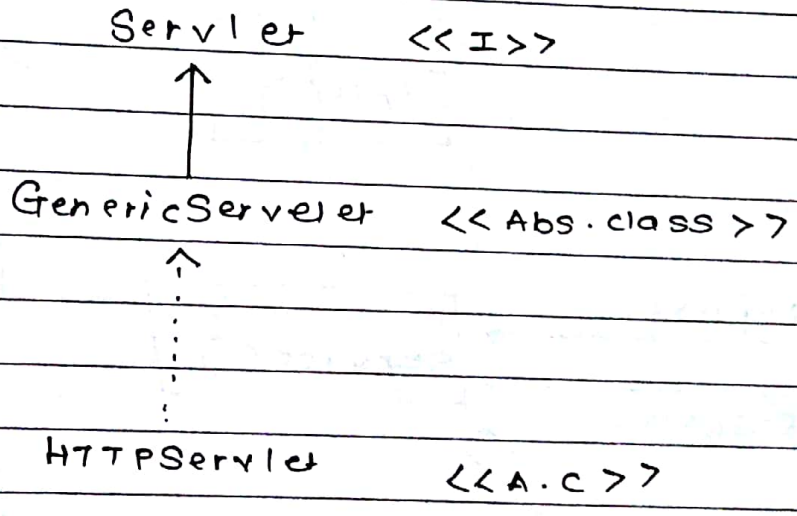
DATE

To generate dynamic response we need technologies like PHP, ASP, Java etc.

Java provides Servlet for generating dynamic response.

- It is a server side java component used to generate dynamic web page.

Servlet is an Interface in javax.Servlet.



There are 2 ways of creating servlet in java
i) by extending GenericServlet
ii) " " HTTPServlet

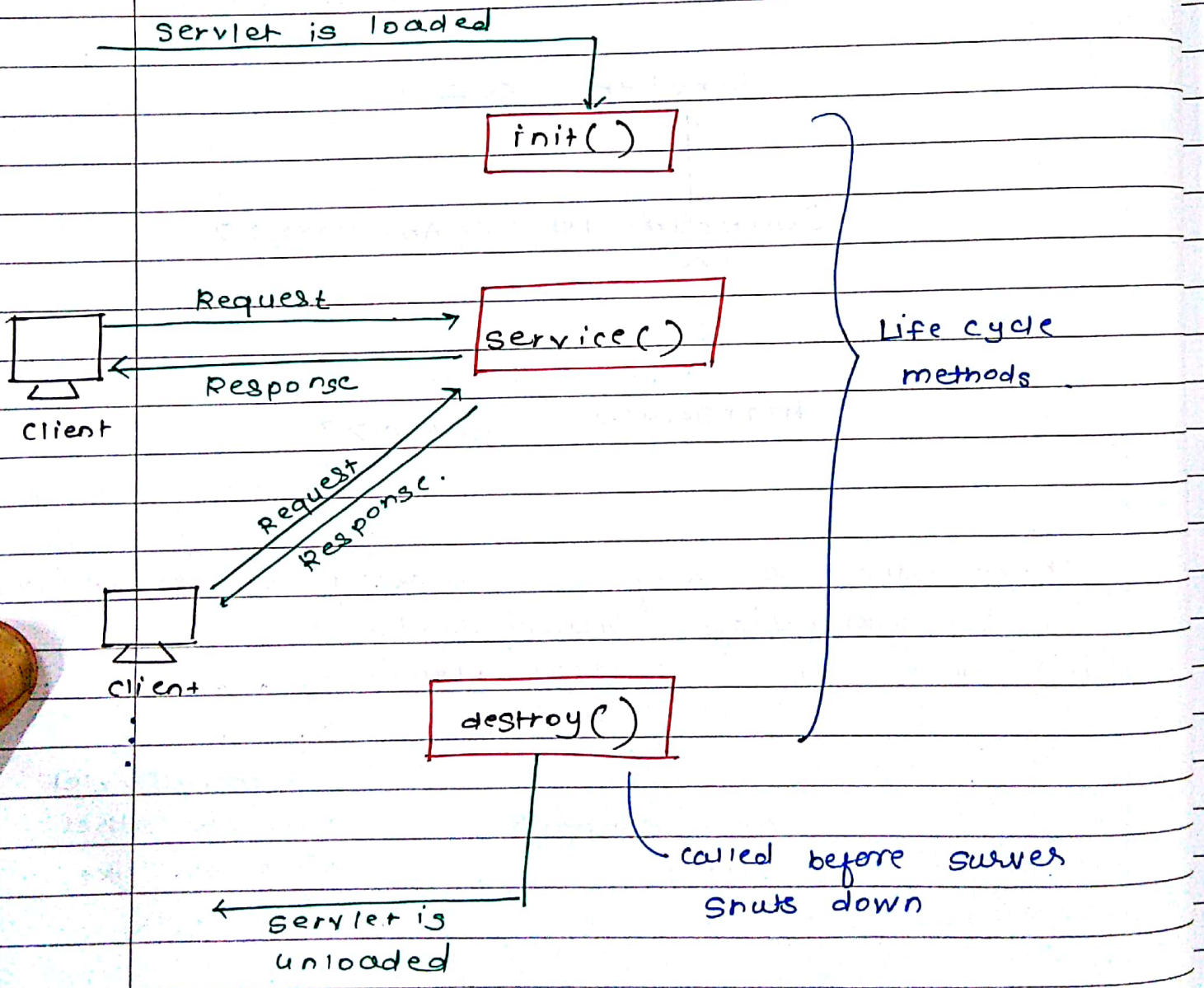
HTTPServlet: only supports HTTP

GenericServlet: Supports other protocols like ftp, HTTP, ...

main() is not required in servlet.
So from where the execution begins?

Servlet Lifecycle

instead of main we use methods ~~like~~:
init()
service()
destroy().



DATE

When servlet is loaded init() is called automatically just like main().

When client requests, service() will be executed. & service() method is responsible to provide response to client.

If there are multiple requests, ~~ex.~~ a ~~thread~~ is created & ~~thread~~ executes service()
∴ for every req, a ~~thread~~ is created. which executes service() method.

What is use of init()?

↳ To initialize connection.

destroy() → contains resource relieving code.
(like closing connection).

init() is called only once in servlet lifecycle. just after the servlet is loaded

destroy() is called only once throughout servlet life cycle. just after the servlet is unloaded.

But service() is called multiple times for each request.

DATE

init() service() destroy() are instance method.

↳ Then who creates an object, calls these methods?

↳ There is a pre-defined mechanism/ code that handles these events (servlet lifecycle)

⇑

It is called a Container / Servlet Engine.

Eclipse: (integrating Server in Eclipse)

i) Create a workspace

ii) file -> new -> other -> server -> server

next

↓

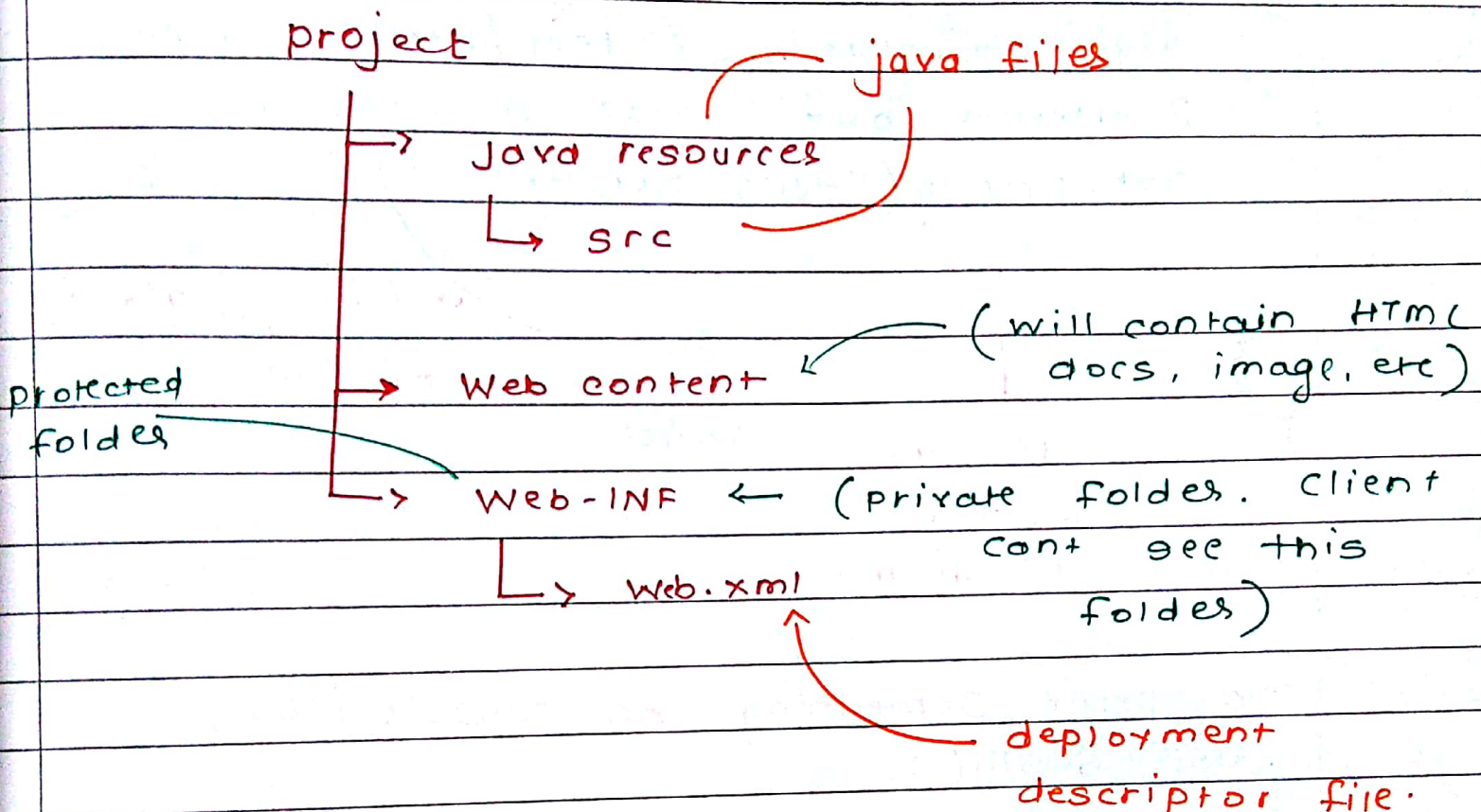
iii) apache -> Tomcat (Version based on your eclipse)

next -> browse -> select Server (Tomcat)

DATE

(creating web proj)

File -> new -> other -> web -> Dynamic web project.
-> Next -> proj.name -> next -> next ->
Generat with xml deployment descriptor -> Finish
-> Open perspective / yes



Src -> create new class in some package. ->
class name -> do not check on main()
method.

By extending HttpServlet, HelloServlet is now a servlet

DATE

```
public class HelloServlet extends HttpServlet
{
protected void service(HttpRequest arg0,
HttpResponse arg1)
{
    arg1.setContentType("text/html");
    PrintWriter out = arg1.getWriter();
    out.println("Hello Servlet");
}
}
```

Type of response

get obj of Writer

Writing/generating response.

To send Response to client.

To print something on console/log, use Sopln();

DATE

make an entry of your servlet in web.xml

open web.xml file.

after </welcome> write,

```
<servlet>
  <servlet-name> hello </servlet-name>
  <servlet-class> pj.HelloServlet </servlet-class>
</servlet>
```

pj → package name

HelloServlet → class name

```
<servlet-mapping>
  <servlet-name> hello </servlet-name>
  <url-pattern> /say </url-pattern>
</servlet-mapping>
```

/say → url pattern

Right click on your project -> Run -> Run on server. -> next.

-> type /say in url

http://localhost:8080/HelloServlet/say

↓

req. is forwarded to container.

↓

it will check web.xml

↓

will check <servlet-mapping> (hello).

↓

will check <servlet> & look for <servlet-name> = hello.

↓

will check <servlet-class> { HelloServlet }

↓

service() will be called.

classmate

PAGE

DATE

Annotations -> used to provide info to pre-written code.

e.g. @Override -> meant for compiler.

Here we are providing info to Container through web.xml. by manually adding url pattern.

We can do this with annotations.

file -> new -> others -> servlet

give class name (LifeCycleServlet)

⇓

next -> next -> uncheck constructors, do Post. Check on init, destroy & Service, inherited abstract methods.

⇓

Finish

↳ A servlet is created (w/o web.xml) with annotations.

Now. Annotations will provide:

url pattern (/LifeCycleServlet)

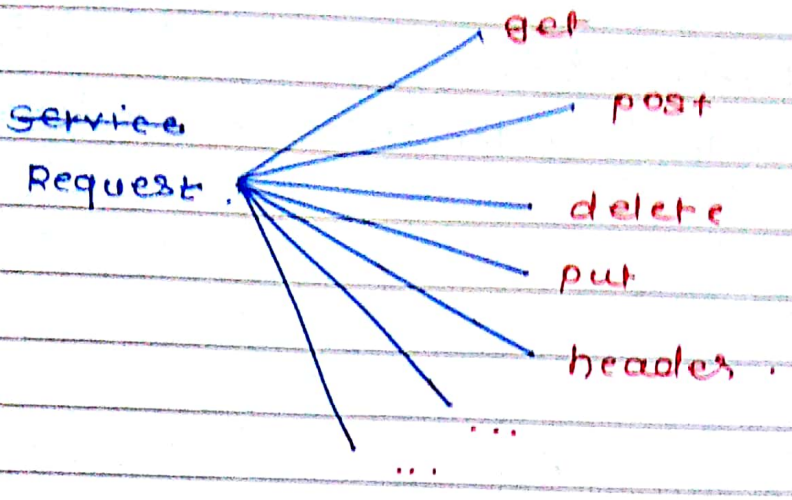
(@WebServlet ("/LifeCycleServlet")

Eclips will automatically add this annotation. DATE

```
@WebServlet("/LifeCycleServlet")
public class LifecycleServlet extends HttpServlet
{
    :

    init(....)
    {
      Sopln("In init()");
    }

    destroy()
    {
      Sopln("in destroy()");
    }

    Service(...)
    {
      Sopln("in service()");
    }
  :
}
```

-> @WebServlet("/LifeCycleServer") is an anotation meant for container (Servlet Engine). It specifies the url pattern to the container. Now, we dont need to make any entry for url-pattern in xml file!

DATE



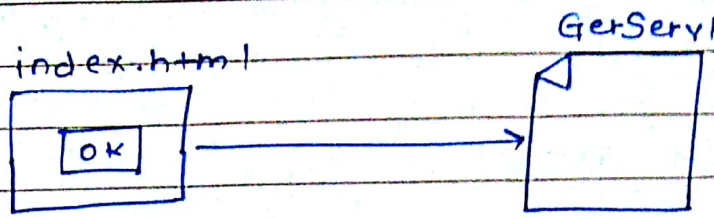
Service() method handles all kinds of request.

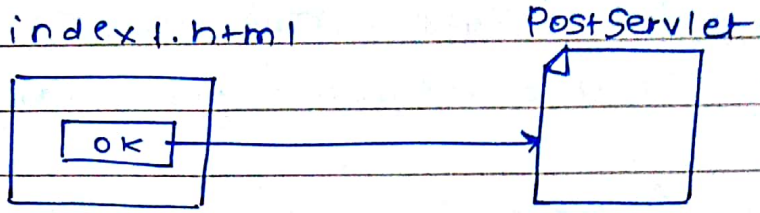
if you wan to handle specific requests,
-> doPost(), doGet(), ...

ger & post are most frequently used request

To create request specific Sewer.

DATE





right click on Web Content (folder) -> new -> html -> index.html

:

index.html

```
<form action = "GetServlet" method = "GET">
    <input type = "Submit" value = "ok" />
</form>
```

:

Src -> right click -> new -> Others -> Servlet

name -> GetServlet

:

check on doGet ...

⇓

Finish.

DATE

GetServlet.java

```
:
protected void doGet( ..... )
{
response.setContentType ("Text/html");
PrintWriter out = response.getWriter();
.
out.println ("Hello from doGet()");
:
```

Run index.html on server.

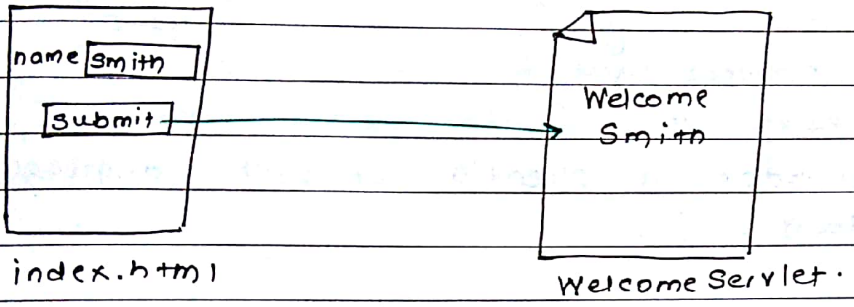
L:3 D:95

DATE 18 11 2017

Fetch Request parameters (Data) in Servlet

Methods used to fetch the request data (request parameter values).

- getParameter ("parameter_name") — gives one parameter
- getParameterValues ("parameter_name")



create index.html.

```
<form action = "WelcomeServlet" method = "GET">
    <input type = "text" name = "username" /> <br>
    <input type = "submit" value = "Submit" />
</form>
```

Request parameters.

Creat WelcomeServlet. (use doGet())

```
... doGet ( .... request, ... response )
{
```

When the submit button is clicked,

... WelcomServlet ? username = Smith → to server

param. — Req. data

The container will create 2 objects of HttpServlet Request type & HttpServletResponse type.

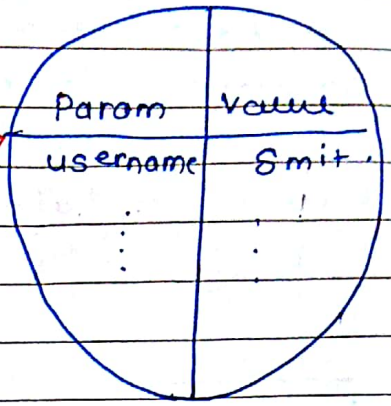
interface

DATE

doGet (HttpServletRequest request, HttpServletResponse response)

{

HttpServletRequest request →



Stores request data & http header info (info. about client), Specification of client's browser, language ... metadata.

∴ HttpServletRequest object contains,

Header info + req. data in key value pair.

like
- version of browser,
- language supported by browser.
- character set supported by web browser (utf8).

Data provided by user. (eg. form data).

When the url is forwarded to container, the container will create 2 objects, (request & response)

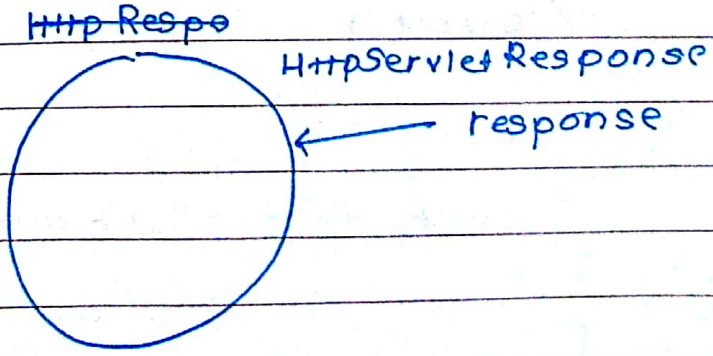
Parameters are always extracted in String.

DATE

To extract data from request object:
- getParameter("param_name");
- getParameterValues("param_name");

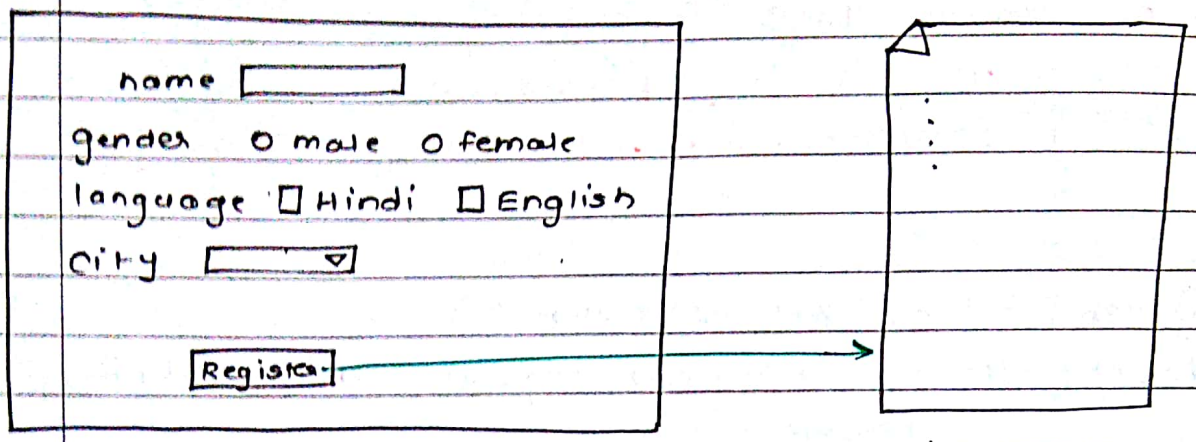
```
@WebServlet("/WelcomeServlet") ....
doGet(HttpServletRequest request, HttpServletResponse
        response)
{
    String nm = request.getParameter("username");
                    if data is not present for key "username"
                    it will return null
    respons.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("Welcome "+nm);
}
```

HttpServletResponse object has functionality to define response content-type, response-content-length, functionality to send response data.



DATE

Demo



reg_form.html

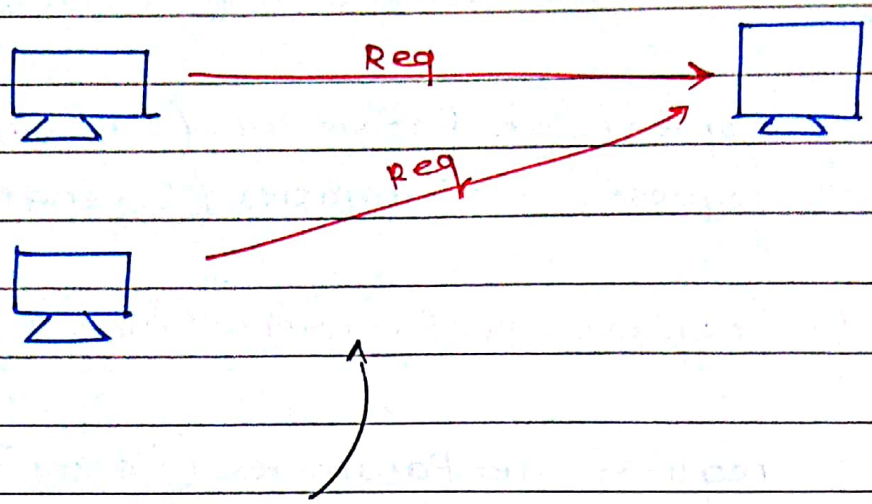
```
<form action="RegServlet" metno="POST">
name: <input type="text" name="username"/> <br>
gender: <input type="radio" name="gender" value="male">
<  "    "    "    "    "  value="female">F

lang: <input type="checkbox" name="language" value="hindi"/>
<  "    "    "    "    "  value="English"/>
<  "    "    "    "    "  value="marath"/>

city: <select name="city">
    <option value="mumbai"> Mumbai </option>
    <option value="pune"> Pune </option>
    <option value="nagpur"> Nagpur </option>
</select>

<input type="submit" value="Register"/>
```

DATE

Create Servlet RegServlet with doPost() method

```
...
doPost (HttpServletRequest request, HttpServletResponse response)
{
    String nm = request.getParameter("username");
    String gen = request.getParameter("gender");

String lang[] = request.getParameterValues("language");

    String city = request.getParameter("city");

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<h2>" + nm + "</h2>");
    out.println("<h2>" + gen + "</h2>");
    out.println("<h2>" + "Languages" + "</h2>");
        for(String s : lang)
        {
            out.println(s + "   ");
        }
    out.println("<h2>" + city + "</h2>");
}
```

Http is a stateless protocol



Server can not differentiate between the requests. It doesn't remember the client it will just accept req, send resp. & destroy the connection

⇑ The stateless behaviour makes it more efficient. (if there are millions of req. at a time).

In some applications, we'll need to track the client (req.).

↳ e.g. (flipkart)

So how to track the req (client) using http protocol?

⇓

java provides a way to achieve this.

↳ Session-tracking mechanism.

HttpSession -> Time period for which the server identifies the client

Session is a time spent by the client for certain period (between login & logout event)

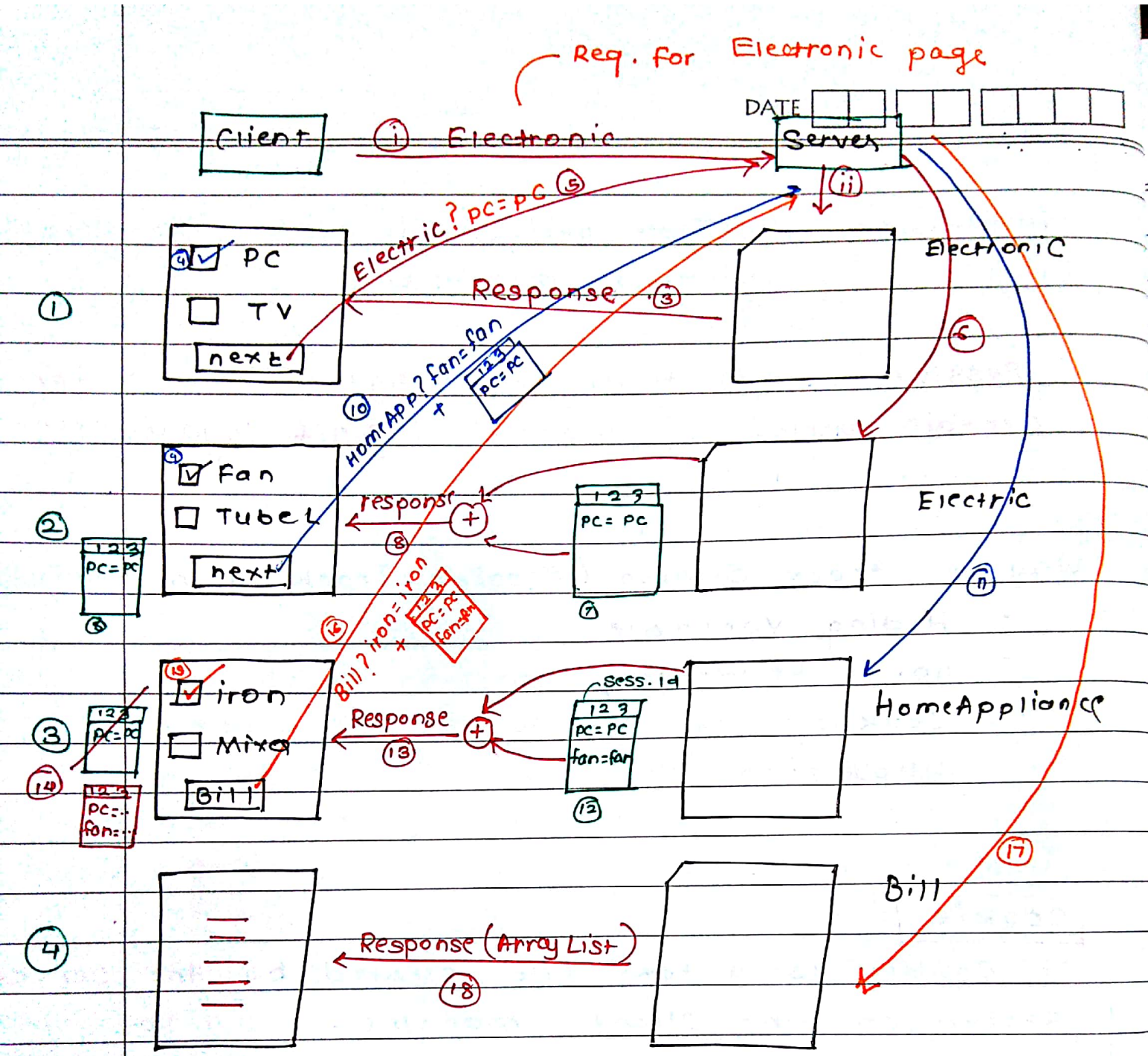
Way to track Session (Session Tracking mechanism)

- Hiding Variable
- URL Re writing
- Cookie
- HttpSession

Cookie

Cookie is a text file created by the server stored on the client machine.

SO, How to track session using Session ?



There are 4 web pages. We want to identify the user. i.e. if ~~th~~ whether the same user is visiting all the pages. (just like fipkart & amazon)

i.e We want to track client from Electronic to Bill

Electronic

DATE

```
@WebServlet("/Electronic")
:
doGet( ... , ... )
{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");   out.println("<head>");
    out.println("<title>");
    out.println("Electronic");
    out.println("</title>");

    out.println("</head>");

    out.println("<body>");
    out.println("<form action='Electric' method='GET'");
    out.println("<input type='checkbox' name='pc'
                value='pc'/> pc <br>");

    out.println("<input type='checkbox' name='tv'
                value='TV'/> TV <br>");

    out.println("<input type='Submit' Value='Next'/>");
    out.println("</form>");
    out.println("</body>");
    out.println("</html>");

}
:
:
```

Similarly create Electric, Home Appliance Servlets

DATE

Electric

```
;
doGet(...,...)
{
```

Adding cookie to response (storing cookie on client m/c)

Extracting parameters received from Electronic & adding them to cookie.

```
String p = request.getParameter("pc");
if(p!=null)
{
    Cookie ck = new Cookie("pc",p);
    response.addCookie(ck);
}

String t = request.getParameter("tv");
if(t!=null)
{
    Cookie ck = new Cookie("tv",t);
    response.addCookie(ck);
}
```

name value

※ write previous code from here.

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>");  out.println("<head>");
out.println("<title>");  out.println("Electric");
out.println("</title>");
out.println("</head>");  out.println("<body>");
out.println("<form action='HomeAppliance'>");
out.println("<input type='checkbox' name='fan'
        value='fan' /> fan <br>");
out.println("<input type='checkbox' name='tubelight'
        value='tubelight' /> tubelight <br>");
out.println("<input type='submit' value='next' />");
out.println("</form>");
out.println("</body>");
out.println("</html>");
}
```

Home Appliance Servlet is similar to Electric

DATE

Bill

```
doGet(..., ...)
{
    ArrayList<String> items = new ArrayList<>();
    Cookie cks[] = request.getCookies();
    if (cks != null)
    {
        for (Cookie ck : cks)
        {
            items.add(ck.getValue());
        }
    }
    String i = request.getParameter("iron");
    if (i != null)
    {
        items.add(i);
    }

    String m = request.getParameter("mixer");
    if (m != null)
    {
        items.add(m);
    }
    :
    //code to print items present
    // in items (ArrayList) as
    // response.
}
```

Will store all the items selected by client.

Adding items selected by client in Electronic, Electric HomeAppliance to Array list from cookies

Adding item to ArrayList selected by user in Home Appliance.

The items selected in the HomeAppliance are not put inside cookie. ∵ it is the last fom.

DATE

- Cookies are not secured. ~~for~~ user can see & manipulate the cookie ∵ cookies are text files stored on client m/c.
- Every cookie has an id. which is called Session id
- With session id, user is tracked with cookie.
- Session id is given by Containes. (Servlet Engine)

DATE

Session/client tracking using HttpSession

Electronic Servlet Will be some as prev.

→ HttpSession is an object created by Container and stored on the Server machine.

∴ Secure!
(Not available for client unlike cookies)

getSession (boolean flag)

true → Existing session object is given. If not, then new session object is given. → of current client

false → Existing session object is given. If not exist, no object is given. → of current client

For every client, an HttpSession object will be created. The session id is assigned to this HttpSession object by the Servlet Engine (Container).

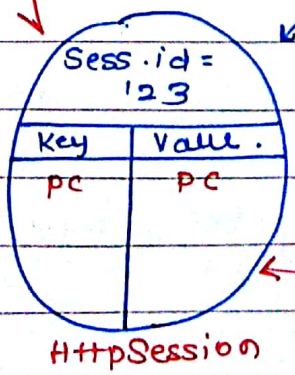
session obj is created per client.
i.e Every client has only 1 session obj.

DATE

Electric

```
doGet(....)
{
  String p = request.getParameter("pc");

  --> HttpSession session = request.getSession(true);
      if(p!=null)
      {
          session.setAttribute("pc", p);
      }

  String t =
      if(t!=null)
      {
          session.setAttribute("tv", t);
      }

  response.setContentType("text/html");
      :
      :
  // some code for
  // generating html

}
```

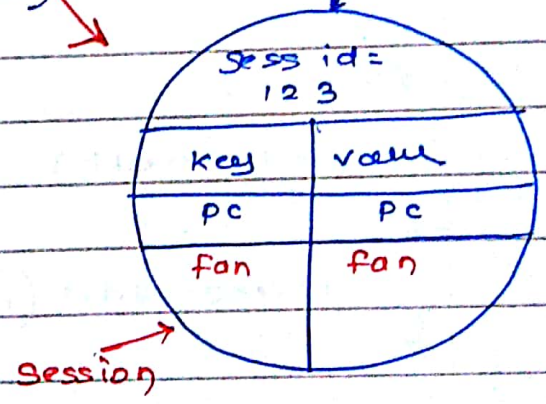
provides Session obj.



DATE

HomeAppliance

```
doGet(..., ...)
{

  HttpSession session = request.getSession(false);
  String f = request.getParameter("fan");
  if(f != null)
  {
     session.setAttribute("fan", f);
  }

  String TL = request.getParameter("tubelight");
  if(TL != null)
  {
     session.setAttribute("tubelight", TL);
  }

response.setContentType("text/html");

  :

  // some code for
  // generating HTML

}

.
.
```



DATE

Bill

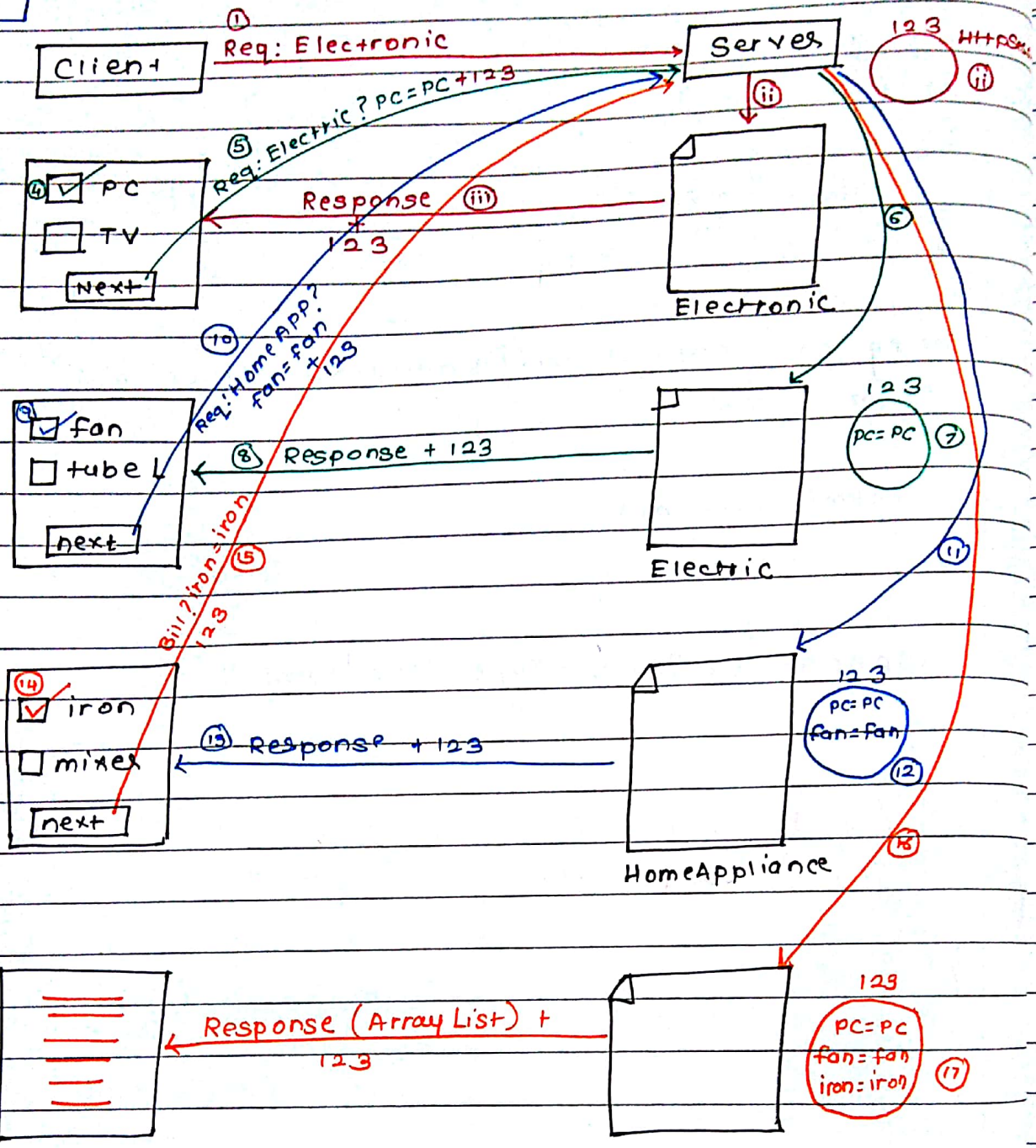
```
doGet(.... ,....)
{
ArrayList<String> items = new ArrayList<>();

HttpSession session = request.getSession(false);

String p = (String) session.getAttribute("pc");
if(p!=null)
{
    items.add(p);
}

String t = .... ... ... ("tv");
 if ..
{ .
    items.add(t);
}

String f= .... ("fan");
if(f!=null)
{
    items.add(f);
}

String tb= .... ("tubelight);
if(tb!=null)
{
    items.ddd(tb);
}
```

DATE

```
String i = request.getParameter("iron");
if (i != null)
{
    items.add(i);
}

String m = request.getParameter("mixer");
if(m != null)
{
   items.add(m);
}

response.setContentType("text/html");
  :
  :
  // code for
     generating html

}
:
```

DATE

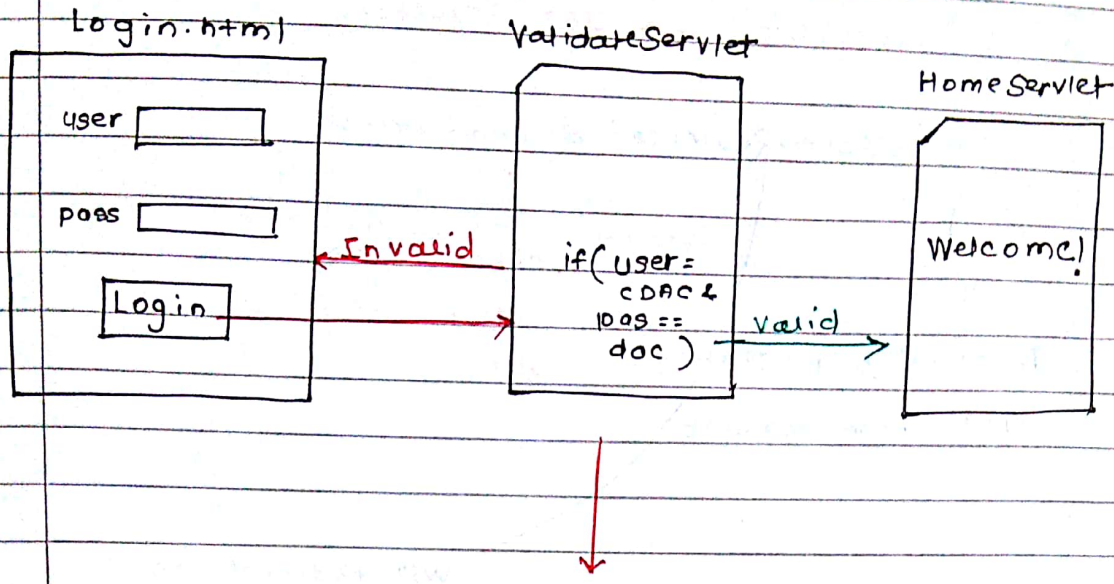
Working



client will req. server for 'Electronic'. ∵ it's client's first time, server (container), an HttpSession obj. is created for the client with a session id. (123). this object is stored on server unlike cookies. server sends response (html form) along with session id to client. From now onwards, the client will request server with session id (123). with this session id server tracks the HttpSession obj that belongs to current client.

DATE

Moving from one servlet to other Servlet




```
if(un.equals("cdac") && pass.equals("dac"))
{
    response.sendRedirect("HomeServlet");
}
else
{
    response.sendRedirect("Login.html");
}
```

This is url pattern not a file name

Assignment

Q5, Q6, Q7, Q8, Q9

Moving / passing data from servlet to another servlet

- Refer 'ServletToServlet' project from Lecture 3

  → or it can also be done using HttpSession or cookies

L:4 D:97

DATE 20/11/2017

Writing url → GET Request.

click (hyperlink) → GET Request.

response.sendRedirect("SecondServlet")

This is nothing but writing url.

Internally generating GET type request ∴ GET.

Will redirect to second servlet

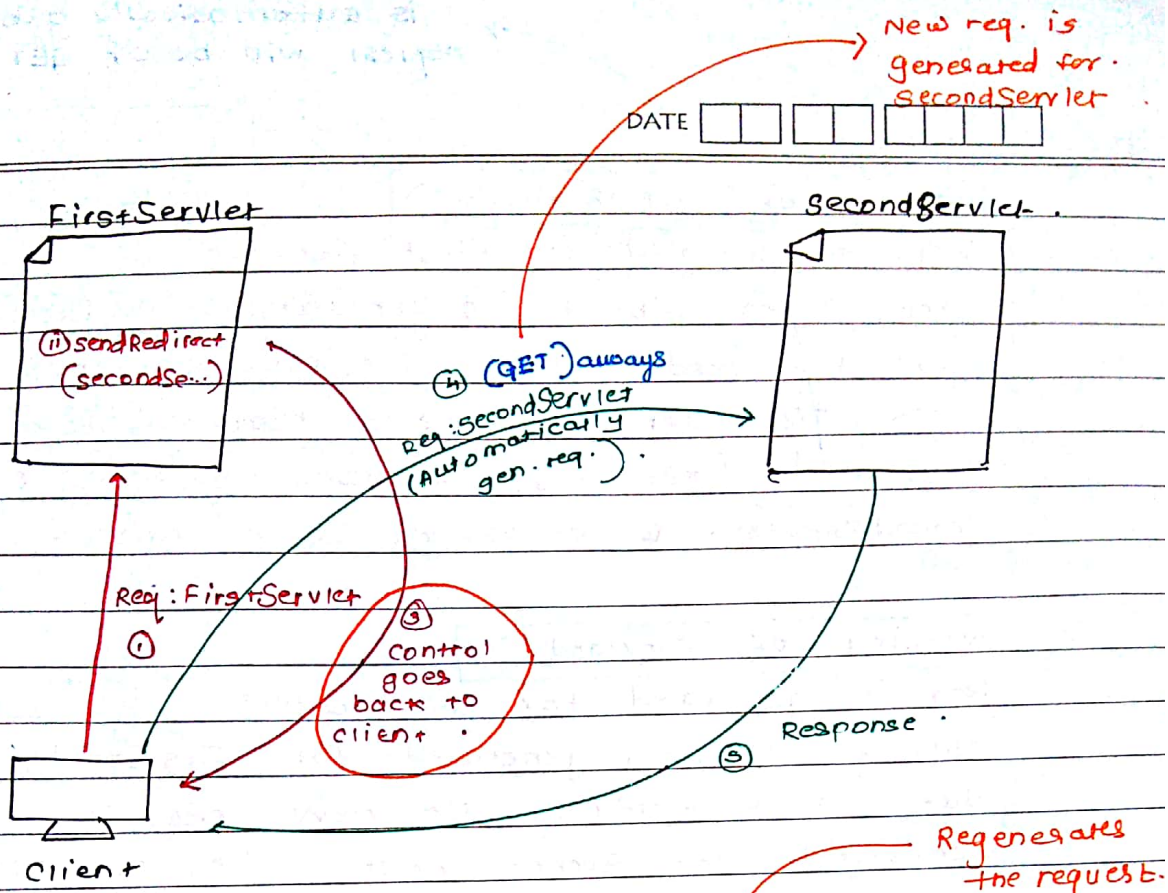
RequestDispatcher. (interface)

Above behaviour can be also implemented by:

```
RequestDispatcher rd = request.getRequestDispatcher
                       ("secondServlet");

rd.forward(request, response);
```

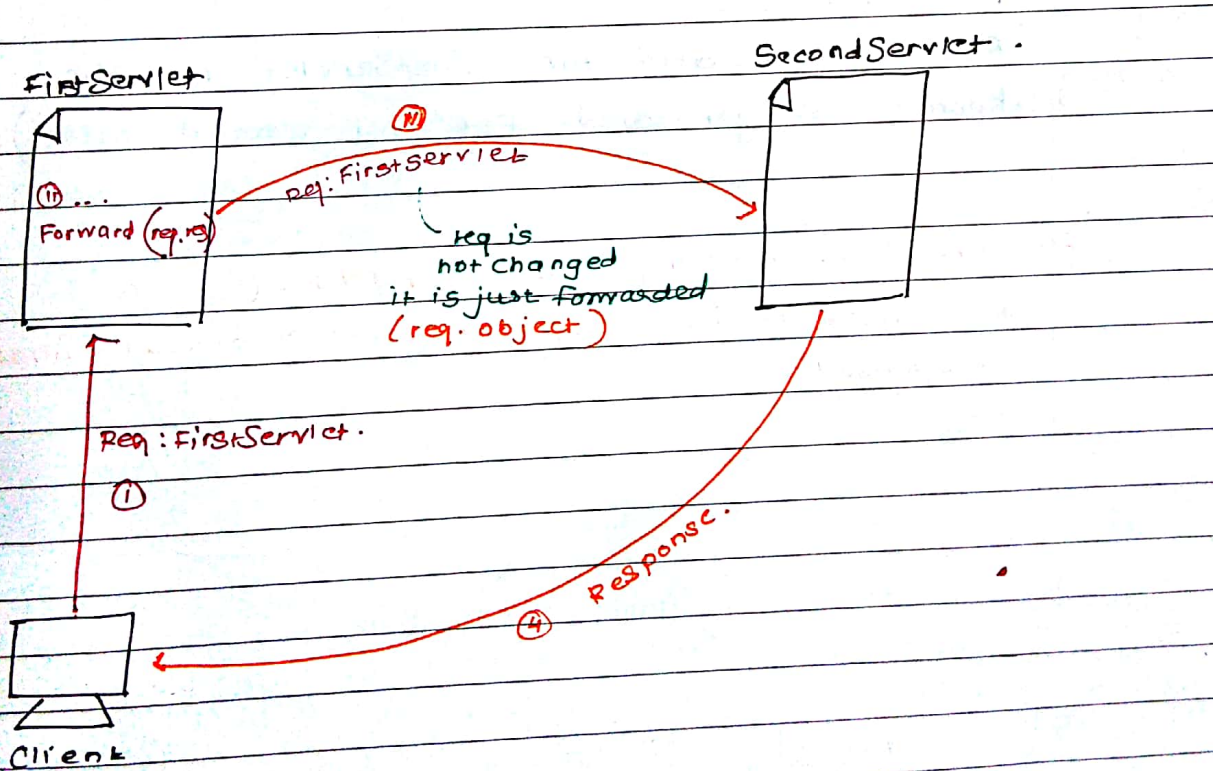
Forwarding request object/information

RequestDispatcher is an interface used for request chaining using forward() and including include() method



Regenerates the request.

(working of sendRedirect).



(working of forward).

forwards the req.
doesn't regenerate the req.

This automatically generated request will be of GET type

DATE

IMP

Working of sendRedirect()

Req. is generated for first Servlet.
req. & res object is generated for firstServlet.
due to sendRedirect, the control comes back to client a new request is generated (req req. & resp. objects) for SecondServlet & sent to SecondServlet.

Working of forward()

Req. is generated for first Servlet.
req., res obj. is generated for firstServlet.
due to forward(). No new req. is generated for SecondServlet. i.e no new req, & resp. object is generated. the same requested is forwarded to SecondServlet.

req Chaining

(Req. is generated for firstServlet but the response is generated for by secondServlet)

In the req. chaining using forward(), the last servlet is responsible to generate final response!

DATE

include()

FirstServlet

```
out.println("First");
...rd=...
rd.forward(request, response)
```

SecondServlet

```
Out.println("Second");
```

o/p will be "Second"

With forward() the second (last) servlet is responsible for generating final response.

Forward() will not include the response of FirstServlet

DATE

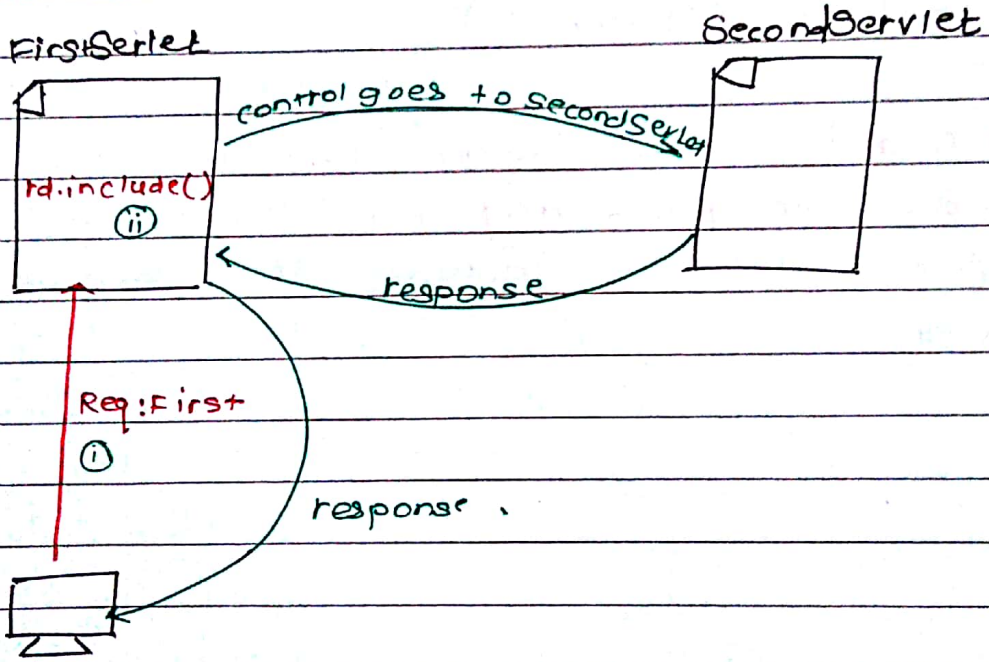
FirstServlet

```
:
out.println("First")
rd.include(request, response)
```

o/p will be "First Second"

SecondServlet

```
:
out.println("Second").
```



- The response from the second servlet is included in the response of FirstServlet.

- The FirstServlet is responsible for generating the response.

JDBC integration with Servlet

Right click on 'lib' folder inside 'Web-INF' paste jdbc connector there.

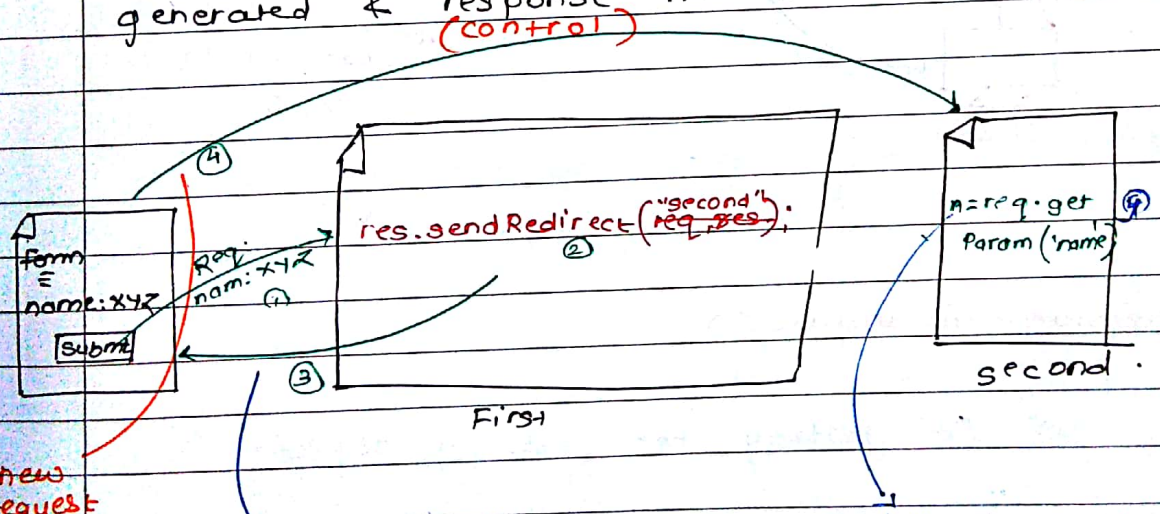
Assignment : Q 11, Q 9

Scopes in Servlet

i) **Request scope :**

↳ HttpServletRequest object.

- ~~HttpRequest~~ HttpServletRequest object is created by container per request.
- The object is alive throughout the $ request chaining (Forward() / include()).
  - ie it is alive bcz the time request was generated & response was sent back.



new request object will be created

∴ control is returned request object is destroyed (it goes out of its scope)

'n' will be null

- You can add extra info in the response by setAttribute()

DATE

JDBC integration with Servlet

Right click on 'lib' folder inside 'WEB-INF' paste jdbc connector there.

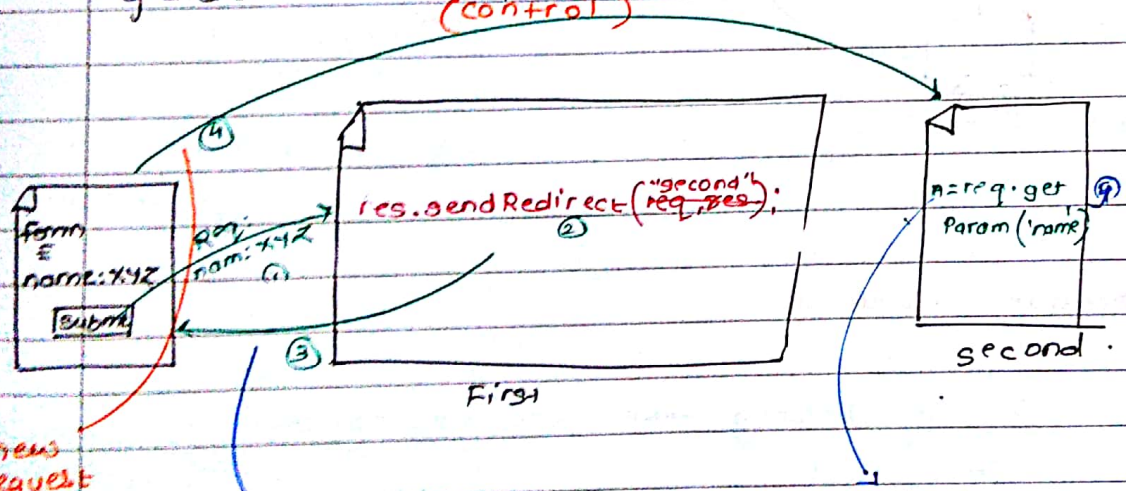
Assignment : Q 11, Q 9

Scopes in Servlet

i) **Request scope :**

↳ HttpServletRequest object.

- ~~HttpRequest~~ HttpServletRequest object is created by container 1 per request.
- The object is alive throughout the request chaining (Forward() / include()).
  ie it is alive born the time request was generated & response was sent back.



- You can add extra info in the response by setAttribute()

discuss

PAGE 59

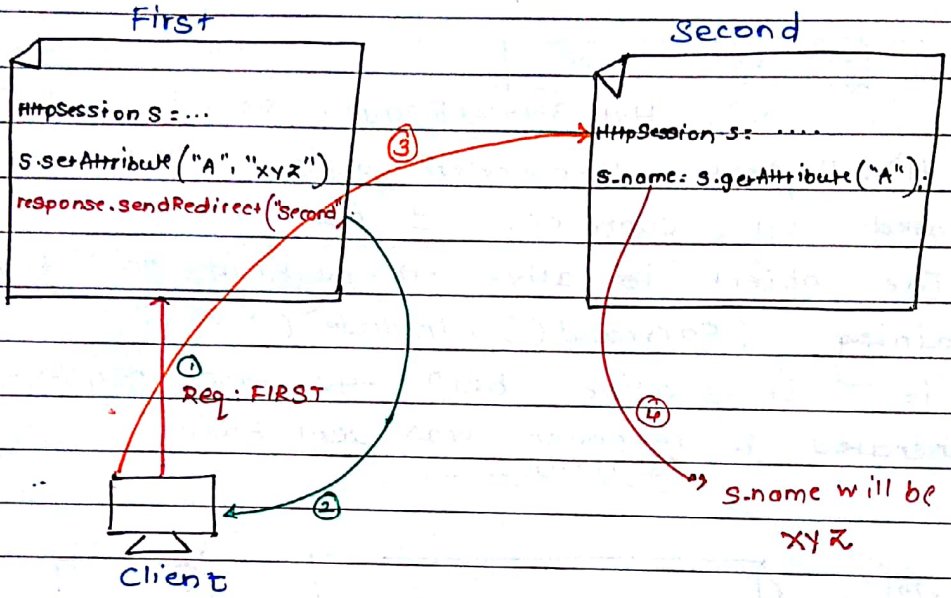
DATE

Session scope

↳ HttpSession object.

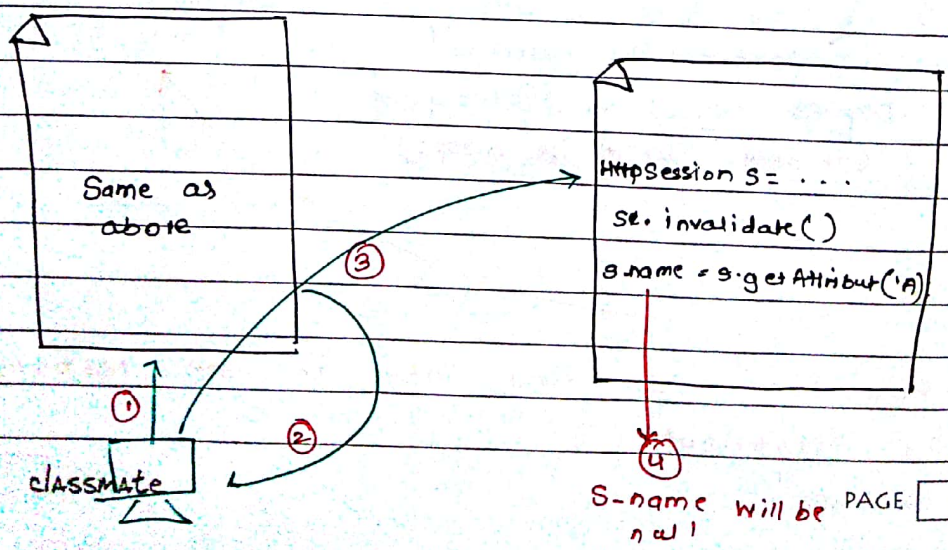
- it is created 1 per client
- It is alive from creation to invalidation (destroy) session object.

Client => instance of Web browser.



session.invalidate()

↳ To destroy the session object.



PAGE

Context :

→ ServletContext ~~ServiceContext~~ object.

- It is created 1 per application.
- Remains alive throughout the application.

If your application has thousands of clients & thousands of servlets. Only one object is created for all the servlets & all the clients.

Just like static var. which is shared between all the objects of same class.

```
ServletContext sc = request.getServletContext();
```

Will give the ServletContext obj which is shared with everyone (clients & servlets)

DATE

First

```
ServletContext sc =
request.getServlet()

sc.setAttribute("key", "value");
response.sendRedirect("Sec");
      OR
response.
    rd.forWord();
```

ng
echanism .

Sec

```
ServletContext sc =
request.getServlet();

ac
X = sc.getAttribute("key")
```

Will be "value"
(even if accessed
in diff. brousels
(clients))

creating context scope using Web.xml

DATE

context-param

- With web.xml, we are providing information to containes.
- Your java code uses info provided by web.xml.

web.xml

```
<context-param>                    Key
    <param-name> url </param-name>
    <param-value> jdbc:mysql://localhost:3306/MyDB </param-value>
</context-param>                   Value
<context-param>                    key
    <param-name> driver_class </param-name>
    <param-value> com.mysql.jdbc.Drives </param-value>
</context-param>                   value
```

servlet

```
ServletContext sc = request.getServletContext();
String dc = sc.getInitParameter("driver-class");
String u = sc.getInitParameter("url");
:
out.println(dc+"  "+u);
```

DATE

Web.xml provides info / data to java code.
Web.xml is helpful for maintainance in your code.
- You can make changes in your application W/o stopping your application. through web.xml

If you want to make changes in servlets W/o using web.xml, then you have to:
- Stop the servlet
- make changes to your code
- recompile the code
- & start the servlet.

L:8 D:98

DATE 21 11 2017

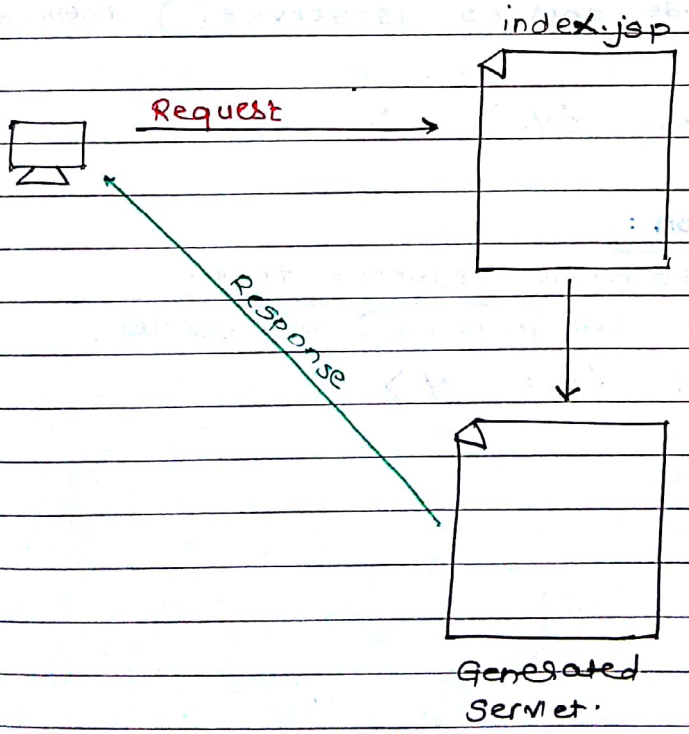
JSP

JSP - $ Java Server Pages.

- JSP is a server side java web component. (just like servlet)
- JSP is a document that separates view code (UI) and java code. (eg HTML)
- In JSP, java code is written in view code.

JSP life cycle

- In servlet, the designing code is written in java (HTML with out.println()).
- In servlet, the designing is complex.
- ∴ We have JSP. (it is just like servlet, a server side java web component)
- Designing becomes very easy.
- Jsp file is converted to Servlet. (Generated Servlet) (automatically)



gend Servlet is responsible for generating the respons.

DATE

JSP Components / Elements

- There are 4 JSP components.
- JSP components are used to write java code.

i) Scripting Elements

Declaration :

used to declare class variables or ~~the~~ to define methods.

Syntax: <%! %>

Scriptlet :

used to write request processing code.
It is just like Service() method of Servlet.
Its code goes to jspService() method.

Syntax: <% %>

Expression :

used to write response text.
just like out.println() of servlet.

Syntax: <%= %>

you can write view code (HTM directly (w/o out. printin

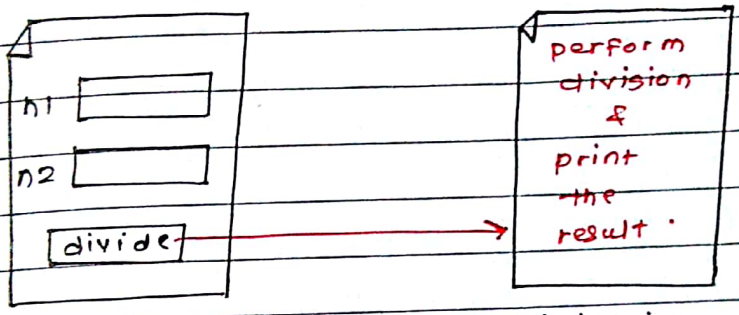
DATE

JSP is a document.

↓

you can write both view code (HTML) & java code in the same document.

Demo



index.jsp

divide.jsp.

jsp pages goes into Web Content folder not in java resources.

Right click on WebContent -> new -> Other -> Web -> jsp -> give name (index.jsp) -> next -> Finish.

index.jsp

DATE

```
<!DOCTYPE Html>
    :

<form action = "divide.jsp">
    <input type = "numbes" name="num1"> n1 </input>
    <input type = "numbes" name="num2"> n2 </input>
    <input type = "Submit" value="divide"> </input>
</form>

</body>
</html>
```

- When we request a jsp page, jsp engine generates an equivalent servlet for that jsp page.

divide.jsp

DATE

```
<%!
    int res;
    void div(int i, int j)
    {
        res = i/j;
    }
%>
```

Declaring var & methods.

```
<%
    String p1 = request.getParameter("num1");
    String p2 = request.getParameter("num2");

    int n1 = Integer.parseInt(p1);
    int n2 = Integer.parseInt(p2);
    div(n1, n2);
%>
```

Request processing

```
<%= res %>
```

Generating response

This is an HttpRequest object

But where is this object coming from?
& how are we able to use them directly?

-> when jsp is created the jsp engine provides this object by default. It is called implicit object.

There are many such objects provided by JSP Engine.

DATE

ii) JSP implicit Object

out
↳ object of type JspWriter

request
↳ of type HttpServletRequest

response
↳ of type HttpServletResponse

page
↳ represents current page (~ this)

pageContext
↳ of type PageContext

session
↳ of type HttpSession

application
↳ of type ServletContext

config
↳ of thepe ServletConfig

exception
↳ of type Throwable.

exception object is not available in all jsp files.
It is only available in error jsp.

DATE

iii) **Directive**

There are 3 directives

Syntax:

<%@ directive-name attribute = "value" %>

↳ eg:

<%@ page import = "java.util.date" %>

This is an attribute of page directive,

There are 3 directives:

- Page
- include
- taglib

type of response

-> <%@ page language = "java" contentType = "text/html"
↑ .... >

This directive is written automatically in your jsp. (by eclipse)

DATE

```
<%@ page import = "java.util.Date" %>

<!DOCTYPE Html>
  :
  :
<body>
  <%= new Date() >
  :
  :
</body>
</html>
```

→ will print current date.

Instead of writting exception handling code in every jsp, just make one jsp that will handle all the exceptions

⇓

error jsp

By writing this, this jsp becomes error jsp.

DATE

error.jsp

This attribute is false by default in all non-error jsps

```
<%@ page isErrorPage = "true" %>

<!DOCTYPE html>
    :
    :
<body>
    <h1> <%= exception.getMessage() %> </h1>
    :
    :
</body>
</html>
```

error.jsp will now handle Exceptions.

divide.jsp

```
<%@ page errorPage = "error.jsp" %>

/*
   Same
      Code
          */
```

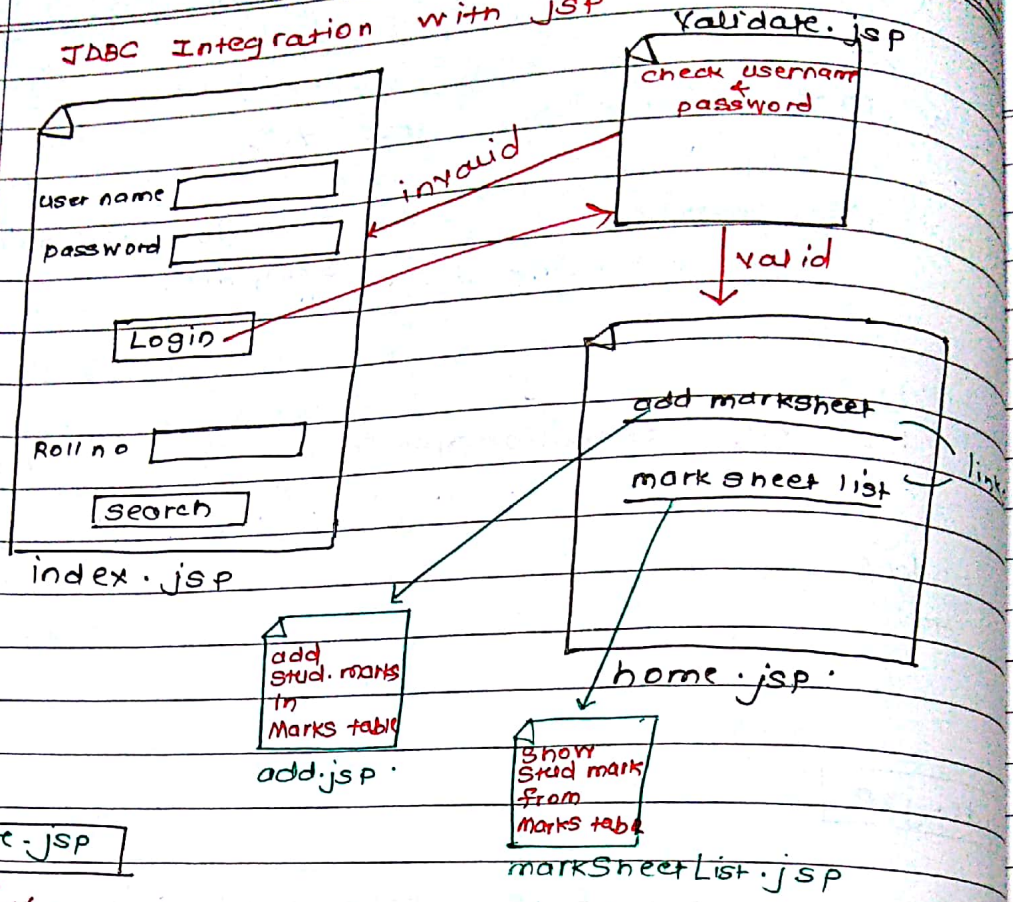
If there is any exception at run time, the control will go to error.jsp automatically & the response will be generated by error.jsp.

res = i/j ; statement might throw ArithmeticException at run time.

PAGE

DATE

JDBC Integration with jsp



Validate.jsp

```
<%
    Class.forName("com.mysql.jdbc.Drives");
    Connection con = Drives Manages.gerConnection(
        "jdbc:mysql://localhost:3306/MyDB",
        "root", "");
```

No need of Writing try catch. you can handle exceptions in error.jsp if you want.

```
PreparedStatement s = con.prepareStatment("
    Select * from my_admin where user_name=?
    and user_pass = ?);

String unm = request.getParomeres("userName");
String upass = request.gerParomeres("userPass");

    s.serString(1, unm);
    s.setString(2, upass);
```

classmate

DATE

```
ResultSet rs = s.executeQuery();

if (rs.next())
{
    response.sendRedirect("home.jsp");
}
else
{
    response.sendRedirect("index.jsp");
}

con.close();

%>
```

right click on lib folder inside WEB-INF & paste the jdbc-driver.

DATE

[c] add.jsp

```
<%
String r = request.getParameter("rno");
String sn = request.getParameter("sname");
String p = request.getParameter("phy");
float phy = Float.parseFloat(p);

String c = ..... ("chem");
float chem = .... (c);

String m = .... ("maths");
float maths = .... (m);

Class.forName("com.mysql.jdbc.Driver");
Connection con = ....

PreparedStatement s = con.preparedStatement("
insert into mark_sheet values(?,?,?,?,?,?);

s.setInt(1, rno);
s.setString(2, sn);
s.setFloat(3, phy);
s.setFloat(4, chem);
s.setFloat(5, maths);

int i = s.executeUpdate();
con.close();

response.sendRedirect("home.jsp");
%>
```

going back

classmate

PAGE

DATE

MarkSheetList.jsp

```
<%
    :
    /* JDBC connection */
    ResultSet rs = s.executeQuery();
    while (rs.next())
    {
%>
    :
<tr>
    <td> <%= rs.getInt(1) %> </td>
    <td> <%= rs.getString(2) %> </td>
    <td> <%= rs.getFloat(3) %> </td>
    <td> <%= rs.getFloat(4) %> </td>
    <td> <%= rs.getFloat(5) %> </td>
</tr>
<%
    }
    con.close()
%>
```

Java code

View code (HTML) can be written directly!

Java code

↑

- In above app, we are re-writing the same code (jdbc conn^n) 3 times.
- It becomes difficult to maintain such app. (if we want to make changes in your conn^n).
- Above app is not modular.
- Create a separate module for jdbc connection & reuse it whereever required

↓ → will reduce maintainance cost.

Modular App.

PAGE

classmate

DATE

Design Pattern

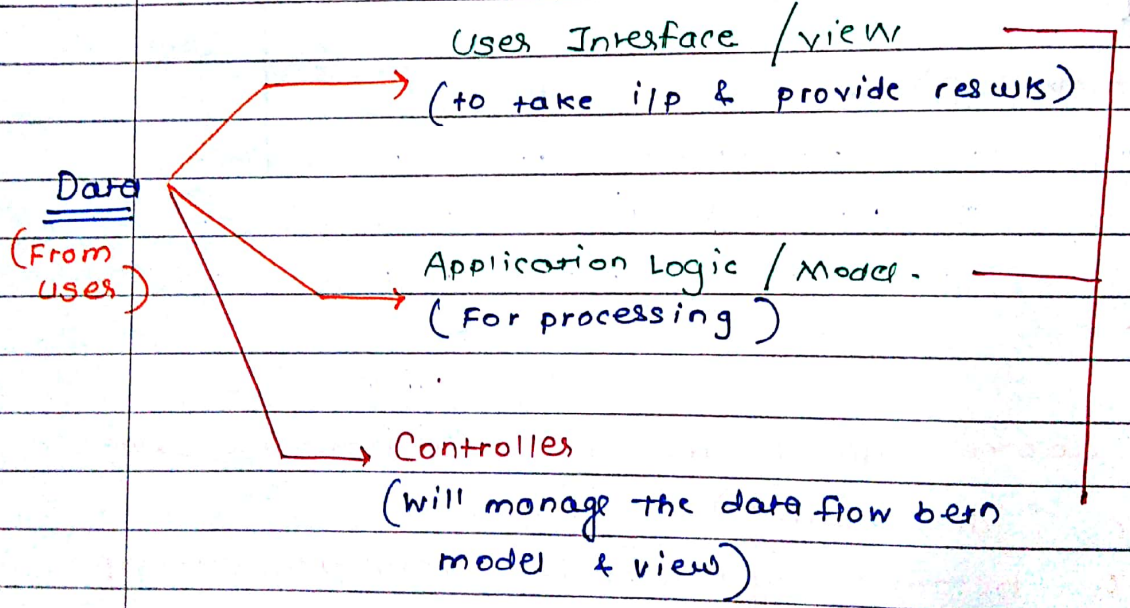
↳ Ways / Guidelines to develop an application. (efficient)

These guidlines are independent of language.
i.e it is not language specific.

- There are many design patterns available:
  - Singleton Object
  - Abstract Design Pattern
  - Abstract Factory Design pattern
  - MVC Design pattern

MVC Design pattern has 3 components:

i) Model
ii) view
iii) Controllers.



DATE

MVC model says;

↳ View should not have knowledge about model & model should not have knowledge about view.

↑ If we follow this guideline, our app will be Modular.

→ If View & model are not aware of each other, then how would the transfer data to each other?

↓

We have Controllers for this!

Java Bean

↳ POJO : Plain Old Java Object model

OR

DTO : Data Transfer Object.

Java Bean is a simple java class that has private properties, public no-argument constructor and public setter/getter methods for each property.

↓

This is must.

DATE

eg:

```
public class Employee
{
    private int empId;
    private String empName;
        :

public void setEmpId(int empid)
    {
        this.empId = empid;
    }

public int getEmpId()
    {
        return empId;
    }
    :
}
```

Assignment:

Q 11 - Q 16

Q 17 - Hill d)

L:6 D:99

DATE 22 11 2017

class represents a real time entity. (Student, Emp, ...)

↳ Not always.

e.g. Main class (class with main() only) is not a real time entity.

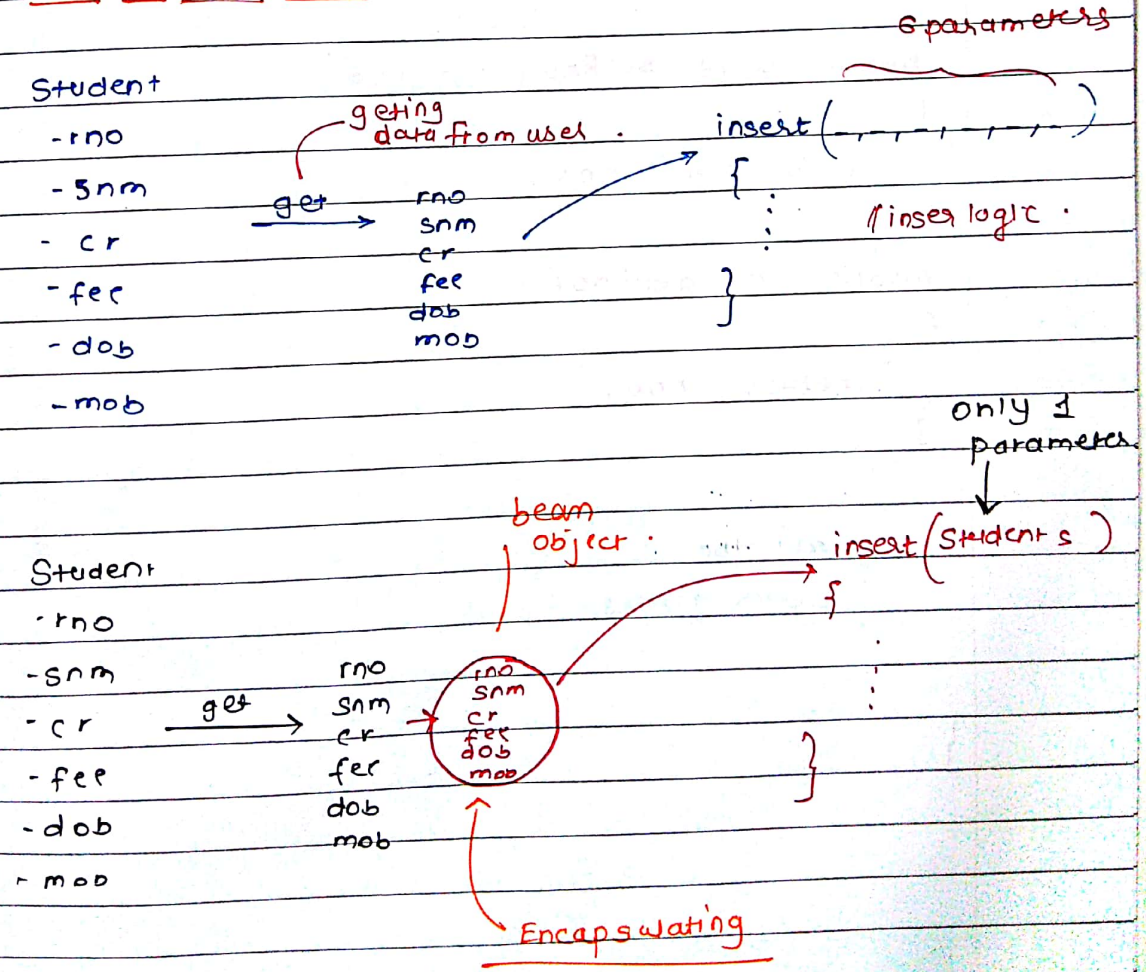
- Classes that represents real time entity are called data class.

- ₿★ data class should be bean class.

Why to create a class?

Creating class ⟶ Encapsulation.

Use of bean / data class.



Refer 22-NOV in Lecture 6 folder

DATE

Create a bean class

create a new dynamic web application.

name -> (javaBeanApp)

java resources -> src -> new class (Student)

inside package name dto (data Xfer object)

```
public class Student
{
    private int rno;
    private String studentName;
    private float fee;
    privark String course;

    public Student ()
    [ ]

    public void setRno ( int rno )
    {
        this. rno = rno;
    }
    public int getRno()
    {
        return rno;
    }

    // all the
        setter getters.

}
```

DATE

Source -> Generate Constructor using fields ->
deselect all (For no arg constr)

↳ Will automatically create no arg constr.

Source -> Generate getters & setters -> Select all
(will create all getters & setters).

Source -> toString

↳ to override toString method of class.

DATE

Web content -> Right click -> new JSP
(Student-Form)

Student_Form

```
<form action: "add.jsp"

RNO: <input type = "text" name = "rn"> <br>
Name: <input type = "text" name = "nm"> <br>
Fee: <input type = "text" name = "fe"> <br>
Course <input type = "text" name = "cr"> <br>

<input type = "submit">

</form>
```

∵ java executes on server & UI is generated at client.

When we req. any jsp, all the java code is executed first & then UI is created.

DATE

create add.jsp.

add.jsp

```
<%
    String rn = request.getParameter("rn");
    String nm = request.getParameter("nm");
    String fe = request.getParameter("fe");
    int r = Integer.parseInt(rn);
    float f = Float.parseFloat(fe);
    String cr = request.getParameter("cr");

    Student std = new Student();   //bean obj.
    std.setRno(r);
    std.setStudentName(nm);
    std.setCourse(cr);
    std.setFee(f);
%>
```

Setting values

Type conversion

storing values in bean object.

DATE

printing contents of bean obj.

```
:
<body>
   RNO: <%= std.getRno() %> <br>
   Name: <%= std.getStudentName() %> <br>
   Fee : <%= std.getFee() %> <br>
   Course: <%= std.getCourse() %>
</body>
:
```

DATE

Create a 2 jsps. Student-form1 & add1.

Student-form1.jsp

→ Same as student-form.jsp.
only change.

<form action: "add.jsp" > .

iv) Standard Actions (JSP Action) → 4th component of JSP.
JSP tags

i> **useBean:**

Syntax: <jsp:useBean id="" class=""></jsp:useBean>
(creating Bean Object
fully qualified name.

<jsp:useBean id="std" class="dto.Student"></jsp:useBean>

Student std = new Student();

→ Requesting jsp to provide the bean object.

Now, Bean obj. is provided by jsp.

Jsp will fist search, if there is any student object reffered by 'std'. (Searching based on key 'std') in all Scopes (Session, page application ...). If present it will return bean obj If not present it will create a new obj & will return it

classmate

PAGE

DATE

ii) **setProperty**

Syntax: <jsp:setProperty name="" property="" param="" /> </jsp:setProperty>

Used to set value

<jsp:setProperty name="std" property="rno" param="rn" /> </jsp:setProperty>

Value to be stored in bean property.

id of bean object

bean property name (as declared in class)

(Html parameter name).

Will store value of Html parameter "rn" to rno property of bean object std.

→ will internally perform type conv. if required & uses setter() of class.

DATE

add1.jsp

(Student std = new Student())

```
<jsp:useBean id="std" class="dto.Student">
</jsp:useBean>

<jsp:setProperty name="std" property="rno"
    param="rn"> </jsp:setProperty>

<jsp:setProperty name="std" property="studentName"
    param="nm"> </jsp:setProperty>

<jsp:setproperty name="std" property="Fee"
    param="fe"> </jsp:setProperty>

<jsp:setProperty name="std" property="course"
    param="cr"> </jsp:setProperty>
    :

<body>
  <jsp:getProperty name="std" property="rno"/> <br>
  <jsp:getProperty name="std" property="stud...."/> <br>
  <jsp:getProperty name="std" property="fee"/> <br>
  <jsp:getProperty name="std" property="course"/> <br>

</body>
```

setting values (much shorter code)

PAGE

DATE

iii) **getProperty.**

- To show the bean property values.

Syntax: <jsp:getProperty name:"" property:""/>
<//jsp:getProperty>

DATE

Student_form2.jsp

→ Same as student_form1.jsp
but
giving parameters names same as bean prop names)

```
<form action = "add2.jsp" >
    <input type = "text"  name = "rno" />
    <input type = "text"  name = "Student Name" />
    <input type = "text"  name = "fee" />
    <input type = "text"  name = "course" />
</form>
```

add2.jsp

```
<jsp:useBean id = "std" class = "dto.Student" ></jsp:...>
<jsp:setProperty name = "std" property = "*" />
```

This works only if the form parameters names are same as bean obj. property names.

Will set all the properties of std bean object automatically.

Instead of 4 lines now we just have to write 1 line.

DATE

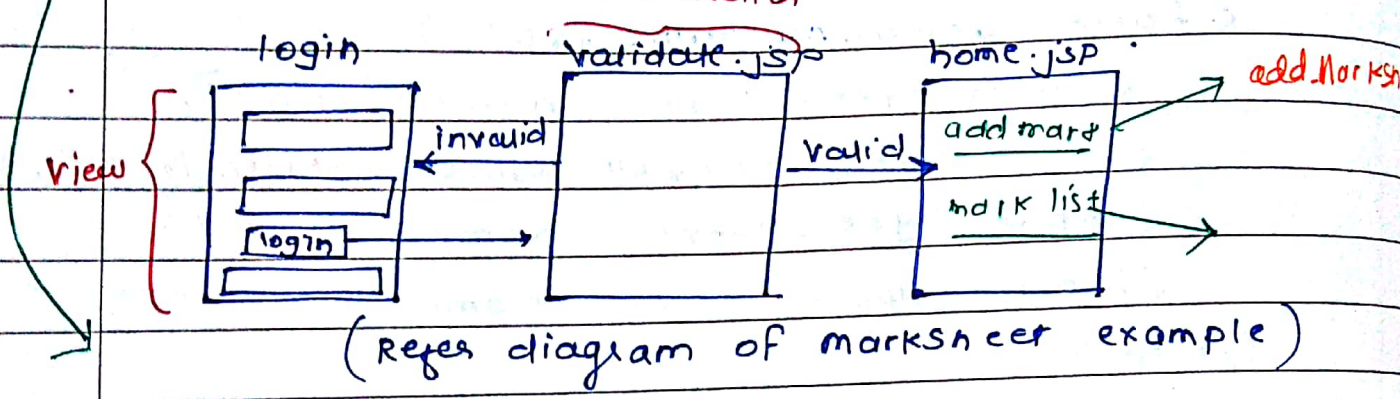
Creating an Mvc app

↳ Modularization : Separating code.

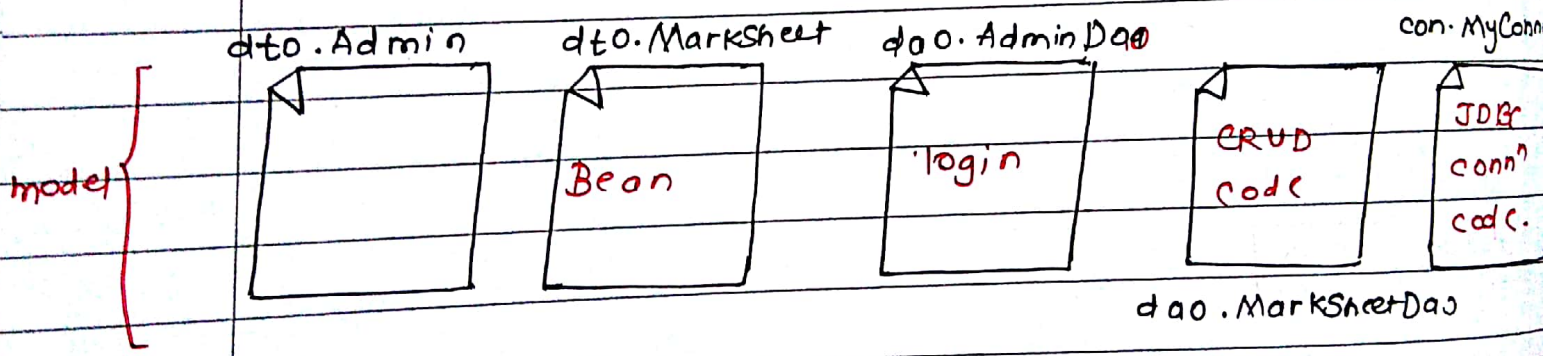
> class in which we write jdbc code (db oper

Dao -> Data Access Objects

dto -> Data Transfer object



(Refer diagram of marksheet example)



DATE

- Dynamic Webproject -> name (MarksheetGenerationSysMVC)
- paste jdbc jar (jdbc-connector) in lib folder. (inside Web-INF)

java resources -> src -> right click -> new java class

↳ create it inside package 'con.'
& name it 'MyConnection'

```
public class MyConnection
{
    private Connection con;
    public Connection getConn()
    {
      try
      {
        if(con == null || con.isClosed())   // checking if conn^n is not present or closed
        {
          Class.forName("com.mysql.jdbc.Driver");
          con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/MyDB","root","");
        }
      }
      catch(ClassNotFoundException e)
      {  }
      catch(SQLException e)
      {  }

      return con;
    }
}
```

DATE

Create a new class in src. in package 'dto'
With name 'Admin'

dto.Admin

```
public class Admin
{
    private int adminId;
    privat String userNome;
    privat String userPass;

    // Add constr (0 para)
    // Add Setters.
```

DATE

Create a class in src in package dao with name AdminDao ..... i..

dao.AdminDao

```
public class AdminDao
{                          ← class
   private MyConnection mcon;

   public AdminDao()
   private void myConf()
   {
      mcon = new MyConnection()
   }

   public boolean login(Admin admin)
   {
      boolean flag = false;
      try
      {
         Connection con = mcon.getConn(); //Reusing connection module

         PreparedStatement s = con.prepareStatement
         ("select * from my_admin where user_name = ?
          and user_pass = ? ");

         s.setString(1, admin.getUserName());
         s.setString(2, admin.getUserPass());

         ResultSet rs = s.executeQuery();

         if (rs.next())
         {
            flag = true;
         }
         con.close();
      }
      catch ....
      return flag;
   }
}
```

PAGE

login.jsp

DATE

Validate.jsp

```
<jsp:useBean id="admin" class="dto.Admin"></..
<jsp:setProperty property="*" name="admin" /><!

<%
    AdminDao adao = new AdminDao();
    boolean b = adao.login(admin);

    if(b)
    {
        response.sendRedirect("home.jsp");
    }
    else
    {
        response.sendRedirect("login.jsp");
    }

%>
```

DATE

```
public class MarkSheet
{
    private int rno;
    private String
    private float physics, chemistry, maths;

    // Add constructor
    // getters & setters.


public class MarkSheetDao
{
    private MyConnection mcon;
    public MarkSheetDao()
    {
        mcon = new MyConnection();
    }

    public int createMarkSheet(MarkSheet markSheet)
    {
        int i = 0;
        try
        {
            Connection con = mcon.getConnec();
            PreparedStatement s = con.preparestatement
            ("insert into mark_sheet values(?,?,?,?
            );
```

PAGE

classmate

DATE

```
s.setInt (1, marksheet.getRno());
s.setString(2, marksheet.getStudentName()),
s.setFloat(3, "   .getPhysics());
s.setFloat(4, "   .getChemistry());
s.setFloat(5, "   .getMaths());

  i = s.executeUpdate();
  con.close();
}
catch (SQLException e);
{ }

  return i;
}
```

In add_mark_sheet.jsp, the parameter names should be same as property names of class

DATE

Add-MarkSheet-jsp

```
<jsp:useBean id="markSheet" class="dto.MarkSheet">
</jsp:useBean>

<jsp:setProperty property="*" name="markSheet">
</jsp:setProperty>

<%
    MarkSheetDao mdao = new MarkSheetDao();
    int i = mdao.createMarkSheet(markSheet);
    response.sendRedirect("add-mark-sheet.jsp");
%>
```

L:7 D:100

DATE 23 11 2017

Working of :

Form . (UI) <input type="text" name="name" />

Submit

name = AB

(JSP) <jsp:useBean id = "a" Class = "Student" />

Student a = new Student();

<jsp:setProperty name="a" property="*" />

a.setName("AB");

This is how jsp internally calls setters when '*' is used with property attribute.

DATE

Expression Language (EL) (Explained Later)

JSP Scope:

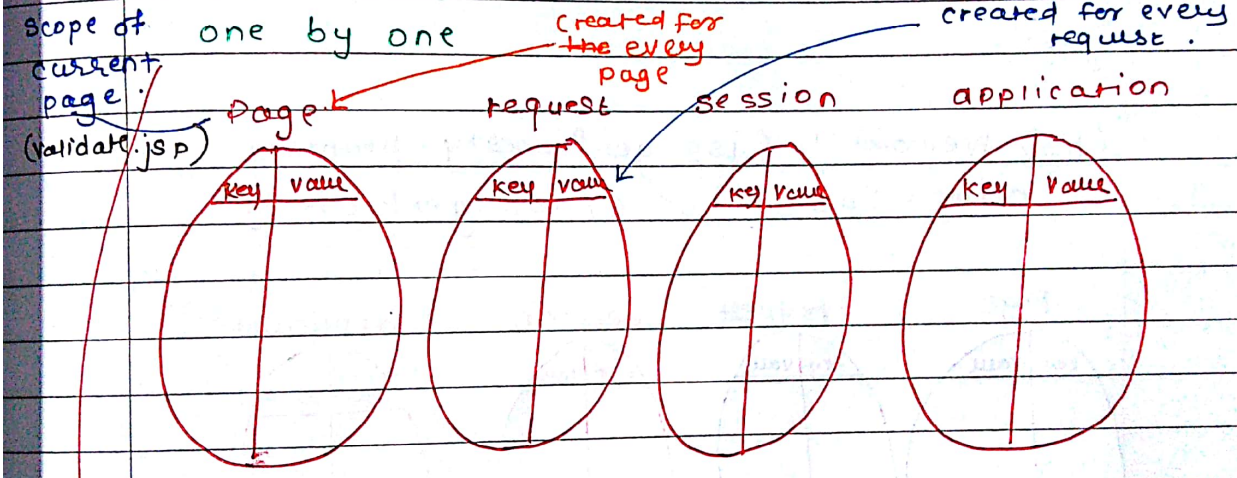
i) Page (Within current page only)
ii) request (through out request chaining)
iii) session (From creation to destruction of Session)
iv) application (Through out the application)

validate.jsp

`<jsp:useBean id="admin" class="dto.Admin" ></jsp:useBean>`

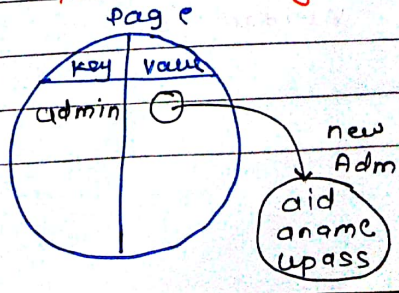
id → key

JSP engine will search all the scopes one by one



scopes are represented as objects.

∴ No obj. with name (key) 'admin' is found, jsp engine will create an object in default scope i.e Page.



classmate

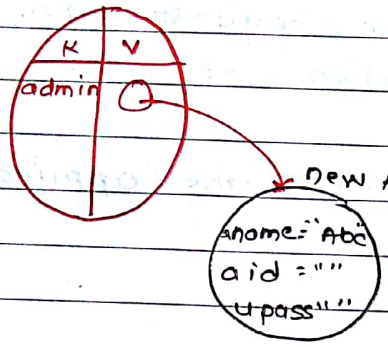
PAGE

DATE

coming from form lets say it is "Abc"

<jsp:setProperty name = "admin" property = "aname" param = "name" />
</jsp:setProperty>

Page



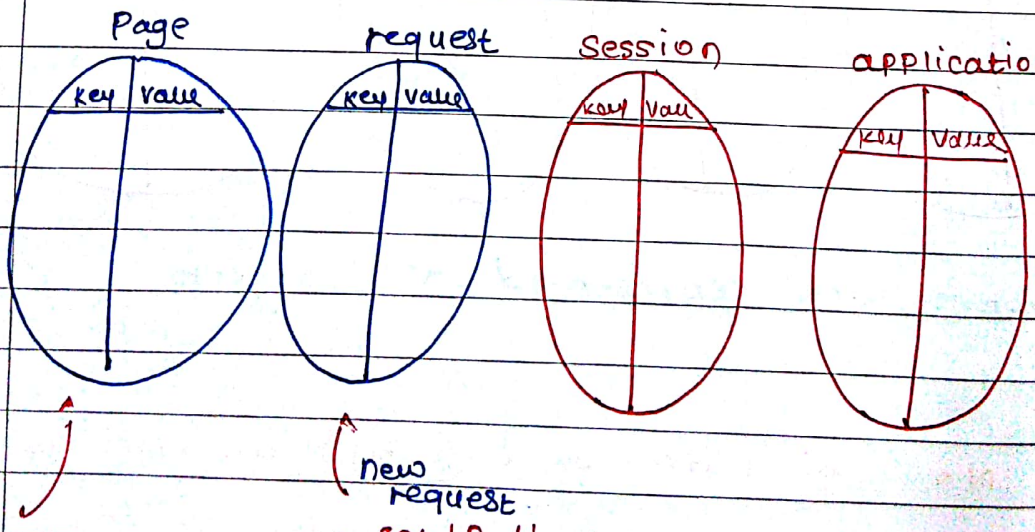
request.sendRedirect(Hom...

Home.jsp

<jsp:useBean id = "admin" class = "dto.Admin">...

</jsp.

<h3> Welcome !! <jsp:getProperty property = "userNam" name = "admin" /> </jsp:getProperty>



: new page

new request
:. sendRedirect was used in Validate.jsp.

DATE

O/P : Welcome null

Explanation :

For every jsp, page scope will be created. in Validate.jsp has its own page scope (obj) & Home.jsp has its own. ∴ changes made in Validate.jsp's session page, will not be reflected in Home.jsp's session page.scope

And since we are using request.sendRedirect(Home) in Validate.jsp, a new request will be generated Hence a new request scope will be created.

DATE

validate.jsp

```
<jsp:useBean id="admin" class="dto.Admin"
    scope="session" >
```

→ Jsp will search for admin in session scope only. if it is present, it will return the object if not, it will create new object in session scope.

```
<jsp:setProperty name="admin" property="aname
    param="name" >
```

DATE

Tracking the client.

logout.jsp

```
<%
    session.removeAttribute("admin");
    session.invalidate();
    response.sendRedirect("login.jsp");
%>
```

no-storage-cache.jsp

↳ Removing info. about the pages.
(Remove header info.)

```
<%
    response.setHeader("Cache-Control", "No-cache");
    response.setHeader("Cache-Control", "No-Store");
%>
```

Mark-sheet-list.jsp

```
<jsp:useBean id=
<%@ include file="no-storage-cache.jsp" %>
    :
<body>
    <a href="logout.jsp">LogOut</a>
```

↑ logout mechanism

DATE

What if user directly tries to visit this page w/o logging in?

Mark_sheet_list

<jsp:useBean id="admin" class="dto.Admin"

Retaining username in login.jsp if password is wrong.

DATE

validate.jsp.

if login fails

↓

```
cookie cookie = new Cookie("uname",
                admin.getUserName
                ());
cookie.setMaxAge(60*60); // 1HR.
response.addCookie(cookie);
response.sendRedirect("login.jsp");
```

login.jsp.

```
<%
String un="";
Cookie arr[]=request.getCookies();
if(arr!=null)
{
  for(cookie c : arr)
  {
    String name = c.getName();
    if name.equals("uname"))
    {
      un = c.getValue();
      break;
    }
}}
%>
```

:

<body>

:

<input type="text" name="userName"
value = <%= un %> />

Expression Language (EL)

- Used to express / print any expression value or to show any java Bean property value.

- Used to provide expression to the action tag

- Syntax:

${expression}

DATE

index.jsp

<body>

${2 + 4 * 2}
 // 10

2 + 4 * 2 // 2 + 4 * 2

</body>

${5 * 5} // false

Expression supports all the arithmetic operators (+, -, /, *, %.)

% or mod

${5 % 2} = ${5 mod 2}.

Relational operators:

< or lt

> or gt

<= or le

>= or ge

== or eq

!= or ne

Logical operators:

&& or and

|| or or

EL can also be used to display bean properties of bean object.

```
<body>
<%
    Student std = new Student (123, "ABC");
    pageContext.setAttribute ("stud", std);
%>
```

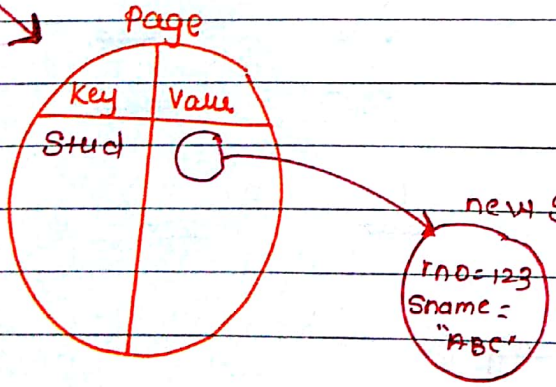
page context

```
<h2> ${stud.rno} </h2>          // 123
<h2> ${stud.studentName} </h2>  // ABC
<h2> ${"stud"} </h2>            // stud

<h2> ${stud} </h2>              //  123  ABC
```

Will internally call toString() of Student class.

(will search value in pageContext with key stud) &

</body>



OR

```
<h2> ${pageContext.stud.StudentName
```

DATE

```
<body>
<%
① application.setAttribute("data", "in application");
② session.setAttribute("data", "in sssion");
③ request.setAttribute("data", "in request");
④ pageContext.setAttribute("data", in page context");

%>

<h2> ${data} </h2>          // in page context
```

if ④ is commented,
o/p : in request.
if ④ & ③ are commented,
o/p: in session
if ④, ③ & ② are commented
o/p: in application.

To access attributes of a specific scope

```
<h2> ${pageScope['data']} </h2>.   // in pageScope
<h2> ${pageScope.data} </h2>.      // in pageScope

<h2> ${pageScope.
<h2> ${requestScope.data} </h2>    // in request scope
<h2> ${sessionScope.data} </h2>    // in session scope
<h2> ${applicationScope.data} </h2> // in applⁿ scop

</body>
```

Implicit objects of Expression Language.

DATE

EL also has emplicit objects:

- page Scope
- request Scope
- session Scope
- application Scope
- param
- cookie.

```
<body>
<%
ArrayList<String> list = new ArrayList<String>();
list.add("list element");
pageContext.setAttribute("val1", list);

HashMap<String, Object> m = new HashMap<String,
                                Object>();
m.put("item1","book");

pageContext.setAttribut("val1", m);  ~~list~~

%>


${val1[0]} <br>          // list element
${val2['item1']} <br>    // book
        OR
${val2.item1} <br>       // book
```

only works with java bean obj & Maps

Easy Syntax! (with java, we'll need to write multiple lines)

<body>

classmate

PAGE

DATE

Casting

```
<%
    HashMap mm = (HashMap) pageContext.getAttribute("val2");

    Object obj = mm.get("item");
// out.println(obj);
    out.println((String)obj);
%>
```

∴ Expression language makes designing/coding easier in jsp.

DATE

JSTL

- Jsp has predefined tags. (<jsp:useBean , etc)
- We can define our own tags.
- Tag library -> set of predefined Tags.

JSTL is one of such libraries.

↳ Jsp Standard Tag Library.

- has some tag for expressing value.
- It is not provided by java
- It is not an integral part of java
- It is created by a third party vendor.

download -> jstl 1.2 .jar. (could be any version.)

Contains tags.

- Dynamic web proj -> give name -> Finish.
- copy jstl 1.2.jar & paste it in lib folder.
- Create a jsp (index.jsp)

Any thing written in <> is considered as HTML tag

tag like <jsp:...> ⇒ jsp action tag.
tag like <%@...> ⇒ jsp directive tage.

- If you want to include prewriten code - include directive is used.
- Taglib directive is used for including external taglibrary in the jsp.

can be any thing

DATE

index.jsp

attribute of taglib

```
<%@ taglib  url = "http://java.sun.com/jsp/jstl/core"
    prefix = "jst" %>
    :
<body>
    <jst : out  value = " Hello JSTL " > </jst:out>
</body>
```

jstl tag
(present in jstl 12.jar)

Advantage of out tag:
you can write expression in out.

```
<jst : out  value = "${2+4}" > </jst:out>  // 6
```

jst:if tag

```
<jst : if  test = "${4 mod 2 == 0}" >
    <jst : out  value = " Even " > </jst:out>
```

jst : choose tag (if - else)

```
<jst : choose>
    <jst : when  test = "${4 mod 2 eq 0}" >
        <jst : out  value = " Even " > </jst : out>
    </jst : when>
    <jst : otherwise >
        <jst : out  value = "odd" > </jst : out >
    </jst : otherwise >
</jst : choose>
```

if →

else →

jst : forEach tag (For traversing collection)

```
<%.
ArrayList <String> l = new ArrayList <String> ();
l.add ("mumbai");
l.add ("pune");
l.add ("nagpur");
l.add ("delhi");

pageContext . setAttribute ("cities", l);

%.>

<jst : forEach items = "${ cities }" var = "ele" >
    <jsr : out value = " ${ele} " > </jst : out> <br>
</jst : forEach >
```

o/p

mumbai
pune
nagpur
delhi

DATE

```
<jst:forEach items="${cities}" var="ele"
             begin="1" end="2">
    <jst:out value="${ele}"></jst:out><br>
</jst:forEach>
```

O/P : pune
nagpur.

★ JSTL tag can be used as an attribute of html tag

```
<h3 $ class='<jst:out value="odd">'> Hello </h3>
```

class attribute of Html tag <h3>

jstl out tag

★ Refer 23-NOV-17 → index.jsp

L:8 D:101

DATE 24 11 2017

HIBERNATE

Structure of RDBMS → Stores data in ROWS of table

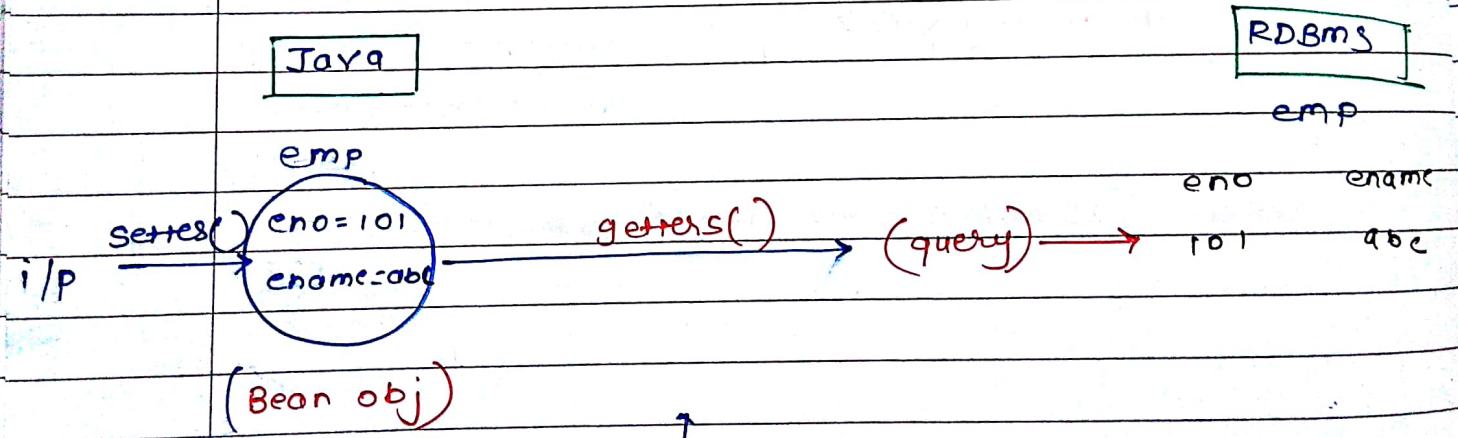
emp

eno	ename
101	ABC
102	BCD
:	:

Java → stores data in obj.

Employee

eno=101
ename=ABC



This is what we have done till now.

Java

1 record → 1 obj

RDBMS

1 Record → 1 row.

- We are storing data in bean obj
- We are extracting data first from bean obj
- then we create a query
- We fire the query & store data in table as a row

⇑ query based xaction.

↑ overhead . (extracting data + writing query)

We are storing query result in table.

ORM

It is a programming techniqu in which data is xferred betn object oriented programming language and and RDBMS

↓

> This data xaction is object based data xaction (Not query based).

We will save the object & a row will be created automatically.

(W/o Extracting & W/o writing query)

↑

To acheive this, there are many ORM tools available like iBatis, toplink & Hibernate

PAGE

DATE

(classes, interfaces, etc.)

- Hibernate is an ORM tool For your java app
- Whatever can be done using Jdbc can be done using Hibernate.
- Hibernate is a prewritten code (framework)

Features of Hibernate

- Provides ORM Functionality to your java app
- It is database portable.

  ↓

  - if app is developed for MySQL RDBMS using Hibernate
  - In future if the DB is changed to Oracle RDBMS, the same application works w/o making any changes in your application.

- Hibernate provides 3 fully featured query languages.
  - HQL - Hibernate Query Language.
  - Criteria API.
  - Native Query.

HQL → Similar to SQL

Criteria API → It is an API - contains predefined methods to do the task of queries

Native Query → DB dependent query
- ORACLE RDBMS queries.
- MYSQL queries

DATE

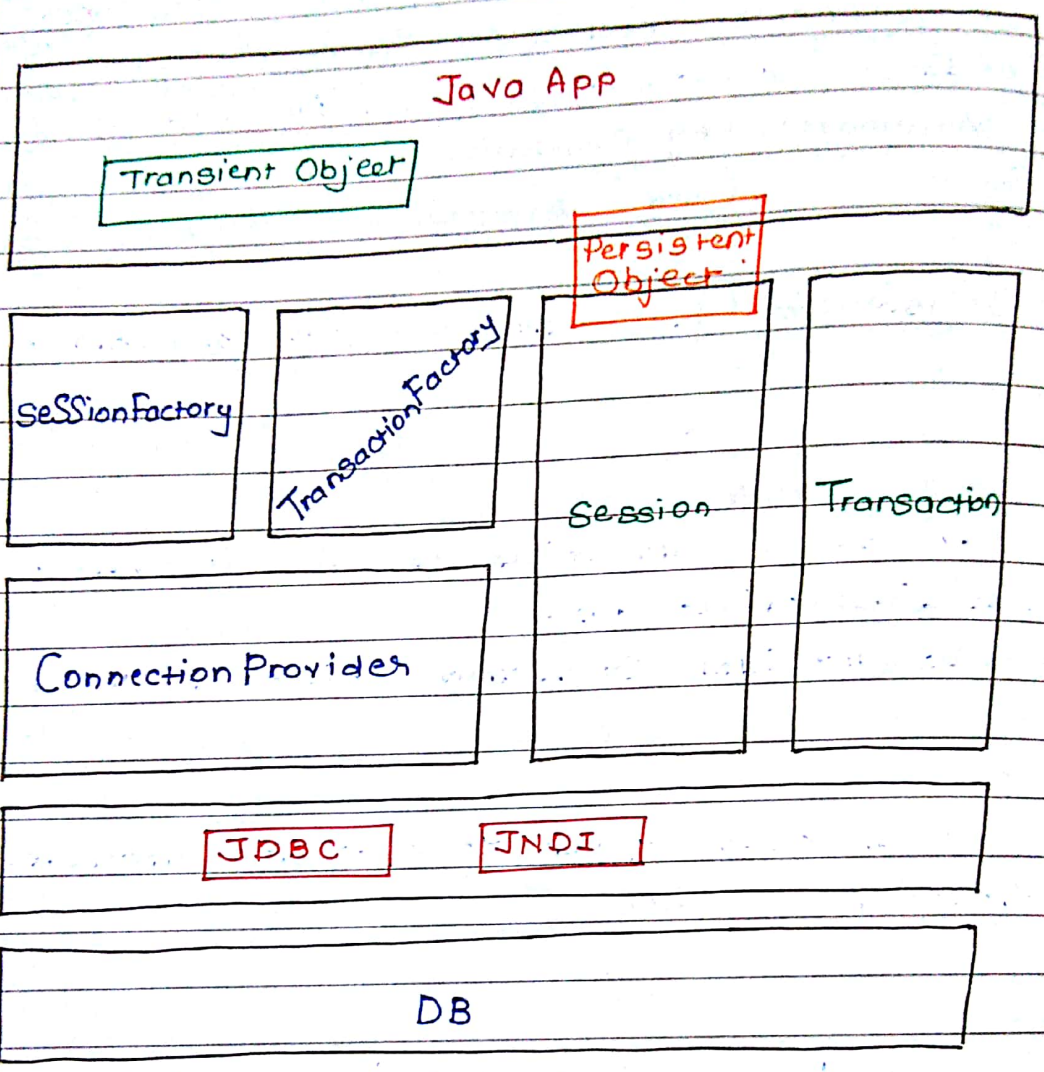
- Hibernate supports oops features like inheritance, polymorphism, association and it supports collection framework.

- Automatic key Generation.
  (eg.. Automatic primary key generation

- It is Free!

DATE

Hibernate Architecture



Session

↳ - Object that represents time period.

- This is an objection of org.hibernate.Session type that represents the time period for data transaction between java app and DB.

Transaction

- This is an object of org.hibernate.Transaction type that defines the unit of work that should be commited or roll backed.

group of queries.

DATE

SessionFactory

- It is a class that is responsible for providing Session obj
- This is the factory or pool of Session Objects.
- It is client for ConnectionProvides.

TransactionFactory

- This is the factory or pool of Transaction objects

ConnectionProvider → interface.

↳ This is an object of org.hibernate.connection.ConnectionProvider type.

- It provides Connection objects to the SessionFactory

Transient Object

- A bean object that is not attached to the hibernate session.

Persistent Object

- A bean object that is attached to the Session.

DATE

instead of using phpMyadmin as a DB Client, use Eclipse as a DB Client.

Connecting DB / integrating DB in Eclipse

Window -> Show view -> Data Source Explorer (if not available, click on other -> (Data managment -> Data Source Explorer) -> one window will appear -> Database connections -> Right click -> new -> MySQL -> give connection name if you want -> next -> Click on button, New Drives Definition -> Select Drives version as per your jdbc jar -> jar list -> remove existing jar -> Add jar/zip -> select the jdbc connector jar from your pc -> OK -> provide your database name -> change URL as per your database -> username = "root" -> password = "" -> Test connection -> "ping succeded!" if connection is successful -> next -> Finish

- Now the DB is connected to Eclipse. Database will be added to "Data Sources"

↑ Do this For Every Workspace.

DATE

Developing a Hibernate application.

i) Create a Java bean class. (inside some package (like dto)

ii) Create a Hibernate Configuration file. which is a XML file.

e.g hibernate.cfg.xml.

↳ It contains information to map your java app to DB.

iii) Create a Hibernate Mapping file.

e.g: hibernate.hbm.xml.

↳ It contains information to map bean class to data base table and bean property to table column.

New -> other -> java Project -> next -> give it a name -> next
-> libraries -> Add -> External jar -> select all hibernate jar files -> finish ->
Open perspective / yes.

create bean class

Src -> right click -> class -> pkg name "dto"
class name "Employee" -> Finish

Employee.java

```
public class Employee
{
    private int empId;
    private String empName;
    private float salary;
    private Date hireDate;

    // constructor (No arg) (must!)

    // parameterized constr (for all fields). (4para)

    // getters() , setters()

    // toString()  (for all fields)

}
```

Check if your eclipse has hibernate plugin.
File -> new -> other. -> look for 'Hibenate'.

if not available, go to Help -> Eclipse Marketplace
-> Search for Hibernate Plugin. (JBoss Hibernate plugin)
(JBoss Tools 4.5.1 Final)
↑ depends on your eclipse version

classmate

PAGE

DATE

configuration file

configuration file can be stored anywhere. but store it in root path. (src)

src -> new -> other -> Hibernate -> 'Hibernate Configuration file cfg.xml' -> next -> let the name as it is (hibernate.cfg.xml)) -> next -> get values from connection -> select your connection. -> select MySQL from "Database Dialect" (from dropdown list) -> Finish -> configuration file will be created. select source tab from bottom left. -> will show XML.

Session factory tab -> Expand Properties tag -> Hibernate -> Show SQL -> True -> Hbm2ddl Auto -> update -> Save the file (ctrl+s)

Mapping file

Can be stored any where. but store it inside the package that contains bean classes. (dto).

dto -> Right click -> new -> other -> Hibernate -> "Hibernate XML Mapping File (hbm.xml)" -> next -> Do not change the name -> next -> Finish

open cfg file -> click on source -> check if there is an entry of mapping file.

goto Session Factory tab -> Mappings -> Add button -> Resource -> Browse -> Select the (...hbm.xml) file from dto. -> OK -> Finish -> Save it (ctrl+s) -> Entry for mapping file (...hbm.xml) will be made in cfg.xml

PAGE

cfg.configure() -> By taking info from cfg file & creating all the required objects. (connection, session etc) in background
SessionFactory

DATE

Open hibernate.cfg.xml -> Remove name tage -> save it

Now Hibernate is ready to use.

$ Creating main class

src -> new -> class -> check for psvm -> store it in some package. -> give class name (Main)

Main.java
org.hibernate.cfg package

```
public class Main
{
    psvm(...)
    {
        Configuration cfg = new Configuration();
        cfg.configure();

//creating sessionfactor  SessionFactory sf = cfg.buildSessionFactory();
        Session s = sf.openSession();

        Employee emp = new Employee(101,"smith",
                                    9000.50f, new Date());
        ...
        Transaction t = s.beginTransaction();
        s.save(emp);
        t.commit();
        s.close();
        sf.close();
    }
}
```

Will search 'hibernate.cfg.xml' on the root path & reads info provided in the file (configuration info)
hibernate is getting ready (initializing db conn, session obj, xaction obj, etc.)

SessionFactory provides Session Object

starting Xaction

object based Xaction

Saving the object.

closing resources.

classmate

PAGE

In jdbc, the transaction is started explicitly. In Hibernate, we have to start the xaction explicitly.
↳ execute(); → execute + commit (Autocommit).
In hibernate we need to commit xaction Explicitly.

DATE

Observ hibernate.cfg.xml file.

with hibernate.cfg.xml, we are providing configuration info to Hibernate.

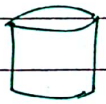
↳ <session-factory> tag.

↓

Hibernate will create a session-factory object.

Hibernate is DB portable → How?

↳ Hibernate has 'hibernate-dialect' mechanism. which is responsible for making hibernate DB portable.

Hibernate query → MySQL Dialect → MySQL sql query → 

↳ for Diff DB's there are diff hibernate dialects.

See 'hibernate.cfg.xml' file

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

<property name="hibernate.show_sql"> true </property>

↳ Hibernate will show the query generated by Hibernate (dialect) on console.

PAGE

Hibernate mapping info.

DATE

open this file & observe

Employee.hbm.xml

```
<class name="dto.Employee" table="Employee">
```

mapping of bean class & table.

Employee bean is mapped to Employee table.

Table name is optional if not given, a table with the name of bean class will be created

Bean class has properties

Table has columns -

property ——— column.
Mapping

Employee

empId

empName

Salary

hireDate

bean property of name

DATE

<id name="empId" type="int">

If not provided, it is taken from bean prop (empId)

Bean property which represents primary key in table.

<column name="EMPID" />

mapping bean (empId) property to EMPID col^m.

remove <generator> tag.

</id>

<property name="empName" type="java.lang.String">
<column name="emp-name" />
</property>

mapping other properties.

<property name="salary" type="float">

Just like this, all the properties are mapped.

L:9 D:102

DATE 25 11 2017

Mapping files (hbm) file are not used Now adays.

The same work can be done using Annotation.

New -> Java Project -> proj.Name (AnotationApp) -> next -> libraries -> add ext. jars -> add all hibernate lib. & sql connector -> finish.

Src -> new -> class -> Employee .java in "dto"

dto

Employee.java

```
public class Employee                    auto inc.
{
    private int empId;
    private String empName;
        "      "   dept;
        "    float salary;
        "    Date hireDate;

// 0 para const^r
// 4 para const^r (:: empId is auto inc)

//setter getters

// to String() (For all the fields)
```

root path of proj

DATE

↓

src -> new -> others -> hibernate -> cfg file -> keep name as its -> get values from connection -> database dialect -> mySQL -> Finish.

properties -> hibernate -> Hbm2Ddl auto -> update
Show SQL -> true.
Mappings -> add -> class -> dto.Employe
Donot create Hbm file. We'll go for Annotations.

```
public class Employee
{
    @Entity
    @Table(name = "emp")
public class Employee
{
    @Id
    @Column(name="empId")
    private int empId;

    @Column(name="ename")
    private String empName;

    @Column(name = "deptno")
    private String dept;
        .
        .
        .
}
```

Telling hibernate that this is the bean class (entity) that will be mapped to the db table.

Table to which bean object will be mapped.

used to make define identifies property that represents PK of table

To provide column Name (Mapping obj prop. to table column)

Mapping bean obj. property to table column.

DATE

(insert)

Main.java Same

```
class main {
  psvm (...)
{
  SessionFactory sf = new AnnotationConfiguration().
                      configure().buildSessionFactory();

  Session s = sf.openSession();
      ⋮   //insert (save())
  sf.close();
 }

}
```

Prior way of creating SessionFactory object wont work with delete, update

DATE

Main1.java (delete)

DML operations in hibernate are performed on PK.

```
class Employee
{
    :
    :
    Employee (int empId)
    {
        this.empId = empId;
    }
    :
}
```

To delete a row from table, we'll need an object with PK ∴ we only need PK to delete the row.

```
class Main1
{
    psvm(...)
    {
        :
        :
        Employee e = new Employee (1);
        S.delete(e)

        :

        sf.close()
    }
}
```

PAGE

classmate

DATE

Main2.java (Select) → 1 Row

```
Class Main 2
{
    psvm(...)
    {
        :
        Employee e = (Employee) s.get (Employee.class, 1);
        Sop In (e);
        :
        Sf.close();
    }
}
```

PK

Returns an obj

will call toString().

Main3.java (update)

```
Class Main 3
{
    psvm(...)
    {
        :
        Employee e = new Employee (5, "Five", "math",
                                   9800.50f, new Date());
        S.update(e);
        :
        Sf.close();
    }
}
```

To locate the row to be updated

Primary key (will go in WHERE of UPDATE query)

Values to be updated.

Will update all the columns except the primary key of the row.

DATE

Employee.java

```
public class Employee
{
    :
    :
    //Add const' with all parameters
}
```

What if we do not provide all the values?

- Still it will update all the columns. the column value that is not provided, that column will get updated with the default value of that property. (e.g String with "null" int with 0...)

To overcome this problem, next page >>

PAGE

DATE

Main4.java (updating w/o update())

```
Class Main4
{
    psvm(...)
    {
        Employee e = (Employee) s.get(Employee.class, 1);
        Sop ln(e);
        e.setEmpName("ABC");
        t.commit();

        e.setDept("civil");
        Sop ln(e);
        s.close();
        sf.close();
    }
}
```

selecting record to be updated

making changes in the obj. returned by get()

we are getting the obj. from hibernate. with get().

This object is attached to the session.

Such object is called "persistent object"

Any changes made to this object, are reflected to db table.

due to commit(), e is now not attached to the session. Changes made in this obj. will not be reflected in db.

This Object is called "Transient Object" or "detached object"

Object that was ~~not~~ attached to Hibernate Session but now they are not.

eg

DATE

```
Employee e = new Employee (5, "five", "math", 75,
                           new Date ());
S.save(e);
t.commit();
```

e is persistent.

e is now a transient object.

e is transient.

e is now a persistent object ∵ it is attached to the Hibernate session.

If we make any changes in 'e' here, they will be reflected in DB.

get() v/s load()

just like get(), we have load() method to fetch object (row) from db.

if the record is not present, it returns "null"

if the record is not present, it throws ObjectNotFoundException

```
Employee e = (Employee) s.get(Employee.class, 1)
// sopln(e);
```

we are fetching the obj but we are not using it. Still Hibernate will fire SELECT query. (selection is done in advance.) = Eager Fetching!

```
Employee e = (Employee) s.load(Employee.class, 1);
// sopln(e);
```

∵ we are not using 'e', Hibernate will not fire SELECT query ⇒ Lazy Feching!

Hibernate will fetch no matter if you are using it or not.

Hibernate will fetch only if you are using the fetched object.

PAGE

DATE

HQL - Hibernate Query Language

- It is similar to SQL
- Statement based query language.
- Instead of table name, entity name (bean name) is used.
- Instead of column names, property names are used.
- Entity name & property names are case sesitive and all of the other tokens are case insensitive.

Src -> new - class -> name it -> put it in a pkg

public class HqlDemo
{

If you want to select all the columns from the table in HQL,
SELECT is not used.
SELECT is not required in HQL to fetch all the cols.

~ Select * from Empr

Query q = s.createQuery("from Employee ");

org.Hibernate.Query

not a table name. It is a bean name.

Selecting All the rows & all the columns.

DATE

```
List<Employee> l = q.list();
for(Employee e : l)
    sopln(e);

t.commit();
```

* `Query q = s.createQuery("from Employe orderby empId");`

Bean Name (Not table name) — Bean property (Not column Name)

```
Query q = s.createQuery("from Employee Where salary > ?");
q.setFloat(0, 8000f);
```

0th index arg. (1st '?')

```
Query q = s.createQuery("from Employee Where ename
                         like ? ");
q.setString(0, "S%");
```

name starting with 'S'

```
Query q = s.createQuery("select ename from Employee");
List<String> l = q.list();
```

selecting one column.

PAGE

classmate

L:10 D:04

DATE 27 11 2017

```
Query q = s.createQuery("select sum(salary)
          From Employee");

List<Double> l = q.list();
```

```
Query q = s.createQuery("select dept, sum(salary)
          from Employee group by dept");

List<Object[]> l = q.list();
```

Criteria API

→ many classes & interfaces are present.

- API is given to write query.
- API contains
  - Criteria (Interface that defines query)
  - Projections (defines SELECT clause)
  - Projection List (Group of SELECT clause)
  - Restrictions (defines WHERE clause)

To select multiple columns (but not all)

Whatever that can be done using HQL can also be done using criteria.

Projection ⇒ What you want to SELECT? (columns)

DATE

```
Criteria q = s.createCriteria(Employee.class);
List<Employee> l = q.list();
```

~ select * from Employee

```
Criteria q = s.createCriteria(Employee.class);
q.addOrder(Order.desc("salary"));
list<Employee> l = q.list();
```

~ order by salary des;

```
Criteria q = s.createCriteria(Employee.class);
q.add(Restrictions.gt("salary", 1000f));
List<Employee> l = q.list();
```

~ where Salary > 1000

```
Criteria q = s.createCriteria(Employee.class);
q.add(Restrictions.like("empName", "s%"));
List<Employee> l = q.list();
```

~ where empName like s%

```
criteria q = s.createCriteria(Employee.class);
q.ProjectionList plist = Projections.projectionList();
plist.add(Projections.property("empName"));
q.setProjection(plist);
List<String> l = q.list();
```

bean prop.

~ collection of SELECT clause (columns)

Select empName from Employee.

adding property to projectionList

getting ProjectionList object.

: we are only selecting one column (property) empName of type String.

PAGE

DATE

```
Criteria q = s.createCriteria(Employee.class);
ProjectionList plist = Projections.projectionList();
plist.add(Projections.groupProperty("dept"));
plist.add(Projections.sum("salary"));
List<Object[]> l = q.list();
```

select dept, sum(salary)
from Employee, groupby
dept;

column used with group by clause must be present in SELECT clause.

groupProperty(); takes care of this rule.

Criteria does all the tasks that can be done using HQL.

- you can only use DQL with Criteria.
- There's no way of writing HAVING clause.

DATE

AKA Named Queries

Native Queries

→ Database specific queries. (Oracle, MySQL).

Query can be written anywhere. but write it in bean class.

we can use this query with this name from anywhere.

(∴ it is Named query)

query name.

```
@Entity
@Table (name = "employee")
@NamedNativeQuery (name = "selAll", query = "select * from employee", resultClass = Employee.class)
public class Employee
{
    .
    .
    .
}
```

DB table name. Not bean class.

(∴ it is a Native query not an HQL Query)

```
Query q = s.getNamedQuery ("selAll");
List<Employee> l = q.list();
```

PAGE

Inheritance:

Beans

Account
- accNo
- accHolder
- amount
- rateOfInterest.

SavonigAccount
- noOfTran

FdAccount
- duration

There are 3 inheritance strategies provided by Hibernate.

i) # table per inheritance class (hierarchy).
- Only one table is created for all of the classes.
- Specific property column can not have NOT NULL constrains.
e.g: For above 3 classes, there will be only 1 table with 6 cols.

noOfTran
duration

Inheritance Type. SINGLE-TABLE

An extra column accType will be added to table which will differentiate (discriminate) both objects of subclasses.

will create only one table for the whole hierarchy.

DATE

Account

```
@Entity
@Table(name="acc")
@DiscriminatorColumn(name="accType", discriminatorType=DiscriminatorType.String)
@Inheritance(Strategy=InheritanceType.SINGLE_TABLE)
public class Account
{
    @ID
    private int accNo;          // PK
    private String accHolder;
    private double amount;
    private float rateOfInterest;

    // constructors  0 para & 4 para

    //getters()  setters()  toString()

}
```

create a Subclass

If you create an obj of SavingAc., the descr. colum. (accType) will have value "sa".

```
@Entity
@DiscriminatorValue(value="sa")
public class SavingAccount extends Account
{
    private int noOfTransactions;

    // constructor. 0 para.

    // constructor 5 para.

    // getters()  setters()  toString()
}
```

↑

SavingAccount

DATE

FdAccount

create a subclass FdAccount.

```
@Entity
@DiscriminatorValue(value = "fd")
public class FdAccount extends Account..
{
    privare int duration;

    // constr  0 para  5 para

    // getters()  setters()  toString()
```

hibernate.cfg.xml -> Mappings -> Add -> Account
-> Add -> Saving Account
Add -> fdAccount
-> Save

& include properties -> ...

Main.java

DATE

```
.
.
.

SavingAccount sa = new SavingAccount(101, "Smith",
10000f, 4f, 5);
                    ↑
                noOfTran
FdAccount fa = new FdAccount(102, "mark",
7000f, 8f, 5)
                    ↑
                duration

s.save(sa);
s.save(fa);
t.commit();
s.close();
sf.close();
```

We are not providing value for accType. it will be automatically added by Hibernate.

accType	AccNo	accHolder	amount	rateOfInterest	no.Tran	duration
sa	101	smith	10000	4	5	NULL
fd	102	mark	7000	8	NULL	5

We can't provide NOT NULL constraint with specific properties (properties of subclass)

PAGE

DATE

Main1.java

class (entity) name

```
Query q = s.createQuery("from Account");
List<Account> l = q.list();

for(Account a:l)
    Sopln(a);

t.commit();
s.close();
Sf.close();
}
```

OR

```
Query q = s.createQuery("from SavingAccount");
List<Account> l = q.list();
```

OR

```
("from FdAccount");
```

DATE

ii) **Table per Subclass**

- separate tables are created for every bean class.

Account.java

Uses JOINS internally

```
@Entity
@Table(name="acct")
@Inheritance(Strategy = InheritanceType.JOINED)
// we dont need discriminator now.
public class Account
{
    :
}
```

SavingAccount.java

```
@Entity
@Table(name="sa_acct")
public class SavingAccount extends Account
{
    :
}
```

FdAccount.java

```
@Entity
@Table(name="fd_acct")
public class FdAccount extends Account.
{
    :
}
```

DATE

Main.java

```
:
:
:
// insertion is exactly same as prev.
// selection is same...
```

iii) Table per concrete class

- Tables are created for concrete sub classes.
  ↳ used when superclass is abstract class or interface.

Account.java

```
@Entity
@Inheritance (Strategy = InheritanceType.TABLE_PER_CLASS)


public abstract class Account
{
    :
    :
}
```

SavingAccount.java

```
@Entity
@Table (name='Sa_acc2')
public class SavingAccount extends Account
{
   :
}
```

FdAccount.java

```
@Entity
@Table (name: "Fd_acc2")
public class FdAccount extends Account
{
   :
}
```

Main.java

```
Query q = s.createQuery("from Account");
   :
```

→ records will be selected from both the tables.

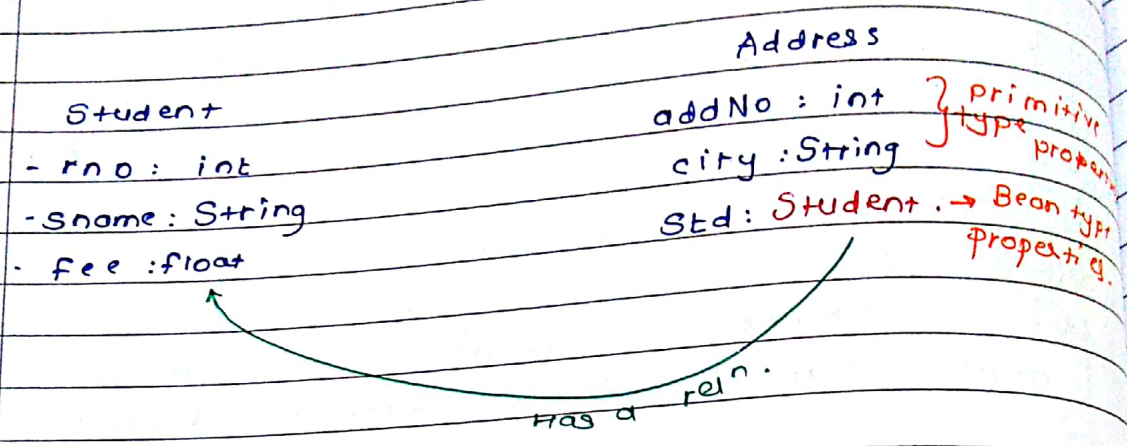
```
Query q = s.createQuery("from SavingAccount");
   :
Query q = s.createQuery("from FdAccount");
```

rel^n bern classes

DATE

Association

Basic DTs (primitives & String) are present in DB.



Address has a Student.

Association

→ one to one
Many to one
one to many
Many to many. } known as Collection Mapping

→ Will require collection.
One dept. many employees

ONE-TO-ONE

DATE

Student.java

```
@Entity
@Table(name = "std");
public class Student
{  @Id
   private int rno;
   private String sname;
   private float fee;

   // 0 para constr , para constr
   // getters() setters() toString()

}
```

Address.java

```
@Entity
@Table(name = "addr")

public class Address
{
   @Id
   @Column(name = "add_no")
   private int addNO;
   private String city;
   private Student std;

   // 0 para constructor, para constr
   // setters() getters() toString()
```

unique fk

@OneToOne @JoinColumn (name="rno", unique=true)

hibernate.cfg.xml

Mappings
-> Add -> Student
-> Add -> Address

PAGE

classmate

INSERT

Main.java

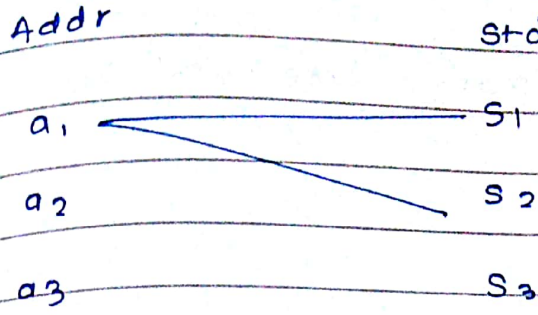
```
.
.
.
Student st = new Student (1001, "smith",
                          8989F);

Address ad = new Address (101, "mumbai", st);

s.save(st);
s.save(ad);
t.commit;
s.close();
sf.close();
```

SELECT

```
Query q = s.createQuery("from Address");
List<Address> l = q.list();

for (Address ad:l)
{
    Sopln (ad);
    Sopln (ad.gerStd());
}

t.commit();
s.close();
sf.close();
```

Many-to-one

DATE



if Addr is owner -> one to many

if Std is owner -> many to one.

Address.java

```
@Entity
@Table(name: "addr2")
public class Address {.....
@ManyToOne
@JoinColumn(name: "rno")
private Student std;
.
.
.
}
```

unique constraint is not necessary ∵ many to one

Student.class is exactly same as prev.

D:105 L:11

DATE 28 11 2017

one to many

one dept has many employees.

dto — Employee.java

```
@Entity
@Table(name="emp")
public class Employee
{
    @Id
    @Column(name="emp_Id")
    private int empID;
    private String empName;

    // constr 0 para & 2 para
    // getters() setters() toString()

}
```

dto — Department.java

```
@Entity
@Table(name="dept")
public class Department
{
    @Id
    @Column(name="dept_Id")
    private int deptId;
    @column(name="dept_name")
    private String deptName;
    @OneToMany  @JoinColumn(name="dept_Id")
    @Fetch(FetchMode.SELECT)
    private Set<Employee> emps = new HashSet<>();

    // 0 para constr . 2 para constr (deptId & name)

    //setters() getters() toString()
```

Fk

not necessary it is default mode.

classmate

PAGE

∴ Department class has collection of other bean (Employee)
it is the owner. ∴ one to many.
The table structure in DB is exactly same in both
one to many & many to one.

DATE

hibernate.cfg.xml

Mappings → add → Employee & Department.

e1 Main.java

```
public class Main
{      static {
    psvm(...)
    {

public class Main
{
    private static SessionFactory sf;

    static
    {
      sf = new Annota.....;
    }

    psvm(...)
    {
        :

    Employee  e1 = new Employee (101, "Smith");
              e2                (102, "mark");
              e3                (103, "Steve");

    Department dept = new Department (10, "cs");
    dept.getEmps().add(e1);
                   (e2);
                   (e3);

    t.commit();
    s.close();
    sf.close();
```

getter of Hashset.

PAGE

DATE

```
s.save(dept);
s.save(e1);
s.save(e2);
s.save(e3);

t.commit();
s.close();
sf.close();
}
}
```

Main.java

```
psvm(...)
{
    :

Department
Query q = s.createQuery("from Department");
List<Department> l = q.list();
for(Department d:l)
{   Sopln(d);
    for(Employee e : d.getEmps())
    {
        Sopln(e);
    }
}
}
```

Name of bean

print dept.details

print employees present in current dept (d)

fetching Employees from department.

```
o/p   10  cs
      101 Smith
      102 mark
      109 Steve
```

DATE

If Instead of @Fetch(FetchMode.~~SUB~~SELECT) if we use @Fetch(FetchMode.SUBSELECT)

Will fire one query for deportment & for every employee, a separate query is fired.

It will fire 2 queries. Always! 1st for department & 2nd will be a subquery. that will fetch all the employees present in the current dept.

@OneToMany → Eager
@OneToMany → Lazy

@OneToMany(Fetch = FetchType.EAGER)

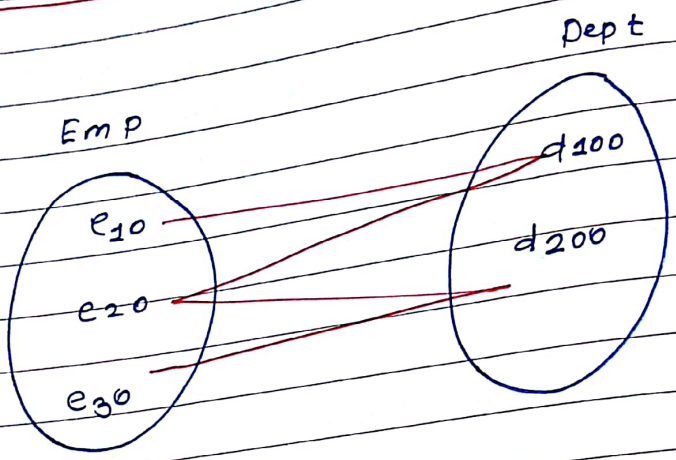
@OneToMany
@BatchSize(size=2)
private Set<Employee> emps = ...;

PAGE

Fetching Strategies.

Select (uses simple Select)
SubSelect (uses sub query)
Join
BatchSize

Many to Many



emp

eid	ename
10	a
20	b
30	c

dept

did	dnm
100	m
200	p

We'll need an additional table to establish many to many rel^n.

Intersection Table. →

emp-dept

eid	did
10	100
20	100
20	200
30	200

DATE

Department.java

:

@ ManyToMany

@ JoinTable (name = "emp ; joinColumns = { @JoinColumn =

"

@ Many ToMany

@ JoinTable (name = "emp-dept" , joinColumns = {@J

@ JoinTable (name = "emp-dept" , joinColumns = { @JoinColumn

(name = "dept_Id"

Employee.java

:

@ ManyToMany(mappedBy = "emps")

private Set<Department> depts = new HashSet<Department>();

DATE

Caching

In one session, if you are fetching the same record multiple times, or the SELECT query is fired only once, for the consequent fetches, the same object is provided.

(Refer Main4.java from CacheAPP)

→ Caching. (Session Caching / First level caching)

When a record is fetched from the database for the first time, The record is stored in the cache for that Session. For the subsequent fetches, the cached result is returned instead of executing the SELECT query.

⇓

(First level caching / session caching / Default caching)

- Object is fetched just once in the session & same object is provided for other call for the same object.

DATE

Second Level caching:

You'll need to add a class in hibernate.cfg.xml.

```
@Entity
@Table(name = "emp1")
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Employee
{
   :
}
```

Now, if you fetch the data for the first time in session 1, the SELECT query is fired & result is saved in Hibernate cache just like first level caching. But Now, if you want to fetch the same record in some other session, the cached object is provided. No Fetching is done.

- It is also known as SessionFactory Cache
- Object fetched by a session is provided to the other session if request.

PAGE

DATE

SPRING

- Design pattern is a way of developing any App.
- Frame work is a set of components used to develop any app on the basis of any design pattern e.g: Spring
  → (pre-written code)

- Spring is a Framework to develop bean based application.
- It works on IOC programming technique. (Inversion of Control)

Create a new java proj w/o any libraries.

new class -> MyTemplate

MyTemplate.java

```
public class MyTemplate
{
    public void service()
    {
        Sopln("done");
    }
}
```

data bean class : To store the data. to encapsulate data.
dao class : Service provider class. processing is done.

DATE

MyDao.java

```
public class MyDao
{
    public void getService()
    {
        MyTemplate m = new MyTemplate();
        m.service();
    }
}
```

-> getService() is dependent on MyTemplate
i.e MyTemplate is a dependency in getService()

Main.java

```
public class Main
{
  psvm(...)
  {
    MyDao m = new MyDao();
    m.getService();
  }
}
```

Lets say in future, new features are added to Mytemplate. By creating its subclass.

DATE

```
class MyTemplatImple extends MyTemplate
{

  @Override
  public void service()
  {
     Syso.("done!!");
  }
}
```

If we execute main(), the above service() wont be called.
Due to dependencies, we'll need to make changes everything where.

∴ Yous class is controlled by the dependency.

DATE

MyDao.java

```
class MyDao
{
    public void getService(MyTemplate myTemplate)
    {
        myTemplate.service();
    }
}
```

Main.java

```
psvm(...)
{
    MyDao m = new MyDao()
    m.getService(new MyTemplate());
    m.getService(new MyTemplateImple());
```

→ o/p: done
done!!

IOC:

A programming technique in which the code is not controlled by dependency.
i.e sending the control out of the code.

We are injecting/providing the dependency to the code. → Dependency & injection. (DI)

DATE

IoC is achieved using DI.
- IoC is a technique
- DI is a process.

Types of Dependency Injection

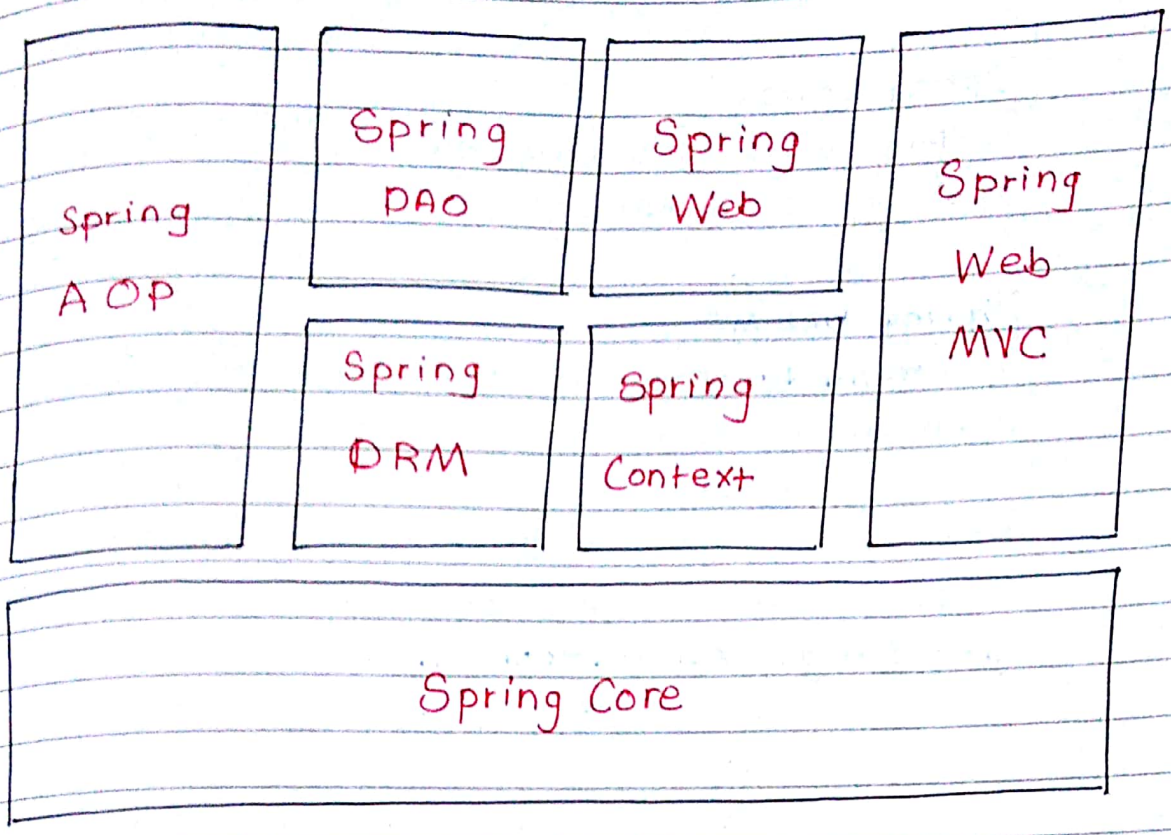
i) Setter Method
ii) Constructor
iii) interface.

Spring supports on Setter DI and constructor DI.

Spring is a layered framework.
↳ Diff layers (tools) are provided for different purpose.
- You dont need to use all the layers.
- You can select the layers as per your requirements.

DATE

Spring Architecture



Spring Core

It is a basic layer of the Spring framework that is responsible for bean instantiation, bean initialization and Dependency Injection

Spring DAO

This layer provides JDBC functionality in the Spring application

Spring web

This layer contains functionality to develop web app. in Spring

DATE

Spring ORM

This layer provides ORM tools like Hibernate functionality in your application

Spring Context

This layer has functionality for Messaging, Mailing, validation.

Spring Web MVC

Has functionality to develop MVC based application.

Spring AOP (Aspect Oriented Programming)

It integrates AOP programming technique in Spring Application.

DATE

You'll need to download a plugin.

Help -> Marketplace -> Search -> "STS Spring plugin" -> install it.

File -> new -> java project -> name -> HelloSpring -> next -> Add external jars -> "Add all Spring libraries" -> Finish.

src -> new -> class. -> inside pkg "comp" -> name your class -> "HelloBean"

HelloBean.java

```
public class HelloBean
{
    private String fname;

    // constructors
    // getters() setters()

    public String getfname()
    {
        sopln("getfname");
        :
    }

    public String setfname()
    {
        Sopln("setfname");
    }
}
```

PAGE

classmate

DATE

root path

Src -> new -> others -> Spring -> Spring Bean Configuration File -> give it a name "cfg" -> next -> don't check anything -> next

-> cfg.xml will be created
(provides configuration info to Spring)

Make an entry of bean.

cfg.xml

```
<bean id = "hello" class = "comp.HelloBean" >
<property name = "fname" value = "HelloSpring">
</property>
</bean>
```

bean property

Key of bean

Will call setter()

Main.java

DATE

```
public class Main
{
    psvm(...)
    {
        ApplicationContext cntx = new ~~Cla XMLClass~~
                                  ClassPathXMLApplicationContext("cfg.xml");
        HelloBean b = (HelloBean) cntx.getBean("hello");
        SOPln(b.getFname());
    }
}
```

Represents the Application. It is a Spring IOC container.

Spring doesn't know which object will be ~~returned~~. it returns Object type obj.

Bean id provided in cfg.xml

It contains all the bean objects

o/p

setfname
getfname
HelloSpring

Spring will read cfg.xml.
∵ there is only 1 bean tag, Spring will create obj. of HelloBean bean class.
this object can be accessed using the key provided ("hello")
due to <property> tag, Spring will automatically call the setter method. The obj. is stored in (cntx) IOC container. ∴ we can use bean created by Spring through IOC container. (getBean())

PAGE

DATE

Application Context cntx