Abstraction :-

```
interface Bank{
    void add();
}

class User { //Helper class
    private class SBI implements Bank{
        public void add(){
            Sop("Money is added");
        }
    }
    public Bank getUser(){ //Helper method
        Bank b= new SBI();
        return b;
    }
}
```

```
class Main{
    public sVm(){
        User u = new User();
        Bank b = u.getUser();
        b.add();
    }
}
```

XXVI

Loose Coupling :-

→ The worst feature of application is tight coupling.

→ To overcome that and improve our application features we prefer loose coupling mechanism

→ In tight coupling we directly create the object.

In case of loose coupling always we refer to common entity

Ex Bike is a vehicle
Truck is a vehicle.

→ vehicle is refered is a common entity for all types of vehicles. So, we never allow the user to create object of interface.

→ In above example if the user wants go for a date he requires bike object.

In same example if the user wants go for trip to goa he requires car object.

Based on the situation the user is prefering the different objects. So, always we have to declare variable using common entity ie, (vehicle)

```
vehicle v = ( new Bike();
              or
              new Car(); )
```

Applications :- with the help of core java we can create stand alone applications which doesn't required any internet or storing the information in Database like calculator, snake game etc.

→ to develop web applications we cannot adjust with only corejava we have to go for either web technologies or both web technology's & backend technologies.

→ we can develop static web applications with the help of frontend technologies (HTML, CSS, Javascript).

Static webapplications - wikipedia, blogges etc.

→ static webapplications is displaying the same information to each and every user.

Static web applications doesn't take more time to generate web pages.

In case of static webapplications the webpages are directly displayed which is stored as HTML document in webserver itself.

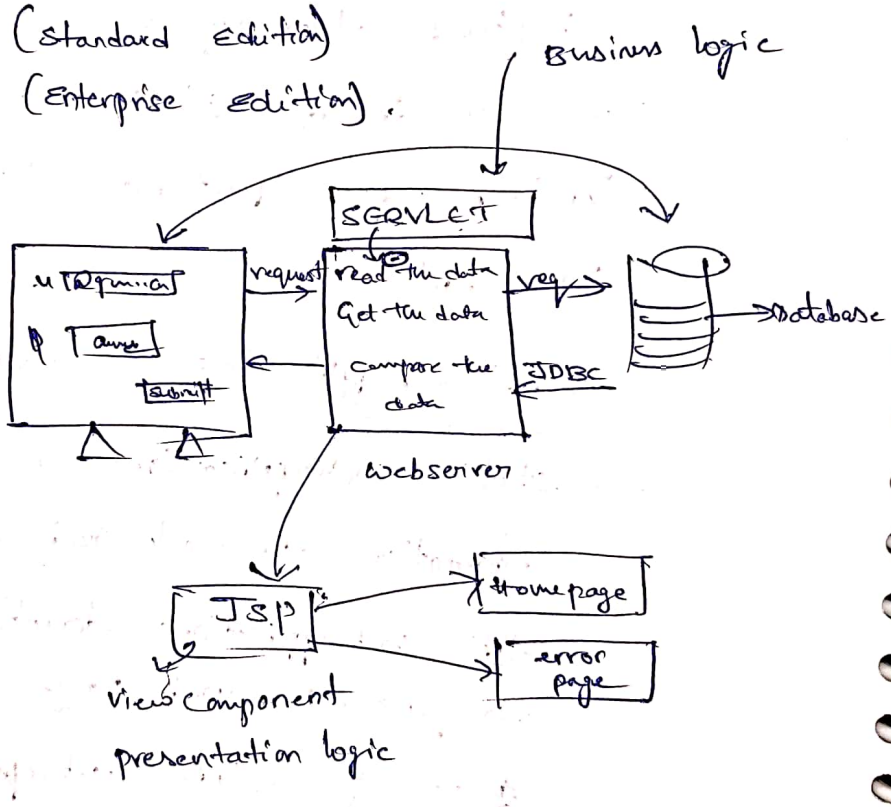
Dynamic webapplications :- the application which is generating dynamic webpages on the request of different users.

→ To develope those applications we will be using Backend technologies like JDBC, servlets, JSP - Java Server pages.

Advance Java technologies:

i) JDBC — Java database connectivity
ii) Servlet and JSP

→ i) Java SE (Standard Edition)
ii) Java EE (Enterprise Edition).



→ Java is classified into 3 editions (i) Java SE (Java standard edition)
(ii) Java EE (Java Enterprise edition)
(iii) Java ME (Java Micro Edition)

Advance Java technologies
i) JDBC
ii) servet and JSP

JDBC – (Java Database connectivity) :-

This is an a API which is used to make communication between Java applications and database applications.

→ All Api's is acting as a mediator. therefore API's are called Inter communication application.

Servlet :- It is also an API (Application program Interface) which is included in the webserver specially used to read and compile the data. ie, passed by client machine and database application.

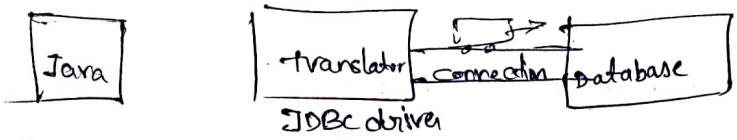
Jsp :- Jsp is an API which is used to display the data to the user. It is known as view Component Jsp is also known as presentation logic.

Both servelt and Jsp are included with the webserver so, ie, Servlet and Jsp is considered as webcomponents

Database :-

→ while creating webapplication, we prefare store the data in database because database have some standards

→ They are (i) It supports quary language
(ii) It avoid the duplicate or repeated the data by providing the primary key.
(iii) It provides security for the data
(iv) It store the data in structured format like tabler formate



→ we have make connection b/w Java application and the Database application.

In order to store the data Inserted the database using Java application we have to follow this steps.

(i) Load and register the driver software (translator)

(ii) establish the connection (Bridge or roadway)

(iii) Creating Statement object (assigning a vehicle).

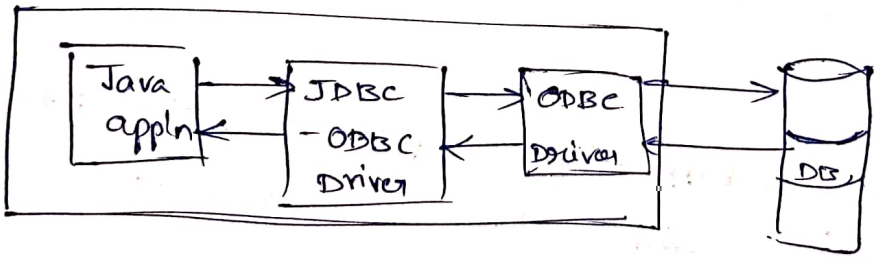
(iv) executing the Query. (or) retriving the data.
(Giving letter (or) recaining acceptance)

(v) closing the connection.

* Driver software :-

Driver Software is classified into 4 types. based on the specification.

Type 01 Driver → JDBC - ODBC Driver



→ This Driver is part of the JDK and it is supported till 1.7 version only. Internaly this driver takes support of ODBC Driver to communicate with the database.

→ It converts JDBC calls into ODBC calls and ODBC driver converts ODBC calls to Database Specific calls.

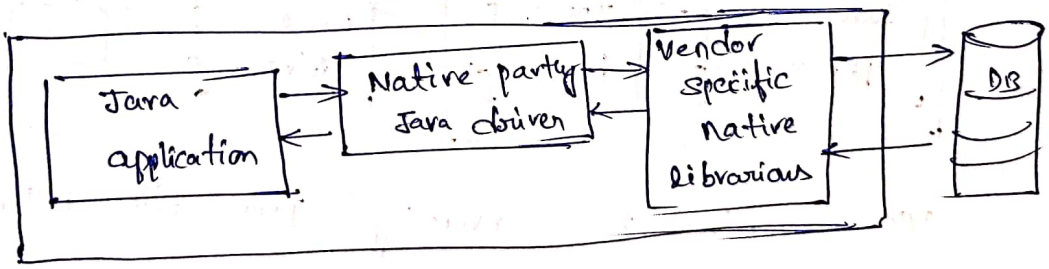
Adv :-

- It is very easy to use and maintain. and we don't need to install new softwares as it as part of JDK.
- It won't directly communicate with the database, So, It is easy to switch between different databases.

Disadvantages :-

- It is the slowest driver among all types of drivers because it converts JDBC calls into ODBC calls and ODBC driver converts ODBC calls into database specific calls.
- This is the depending on the ODBC driver. which works only on windows machines. and It is platform dependent driver.

(2) Native Driver (or) API partly Java Driver



→ Type-02 driver is similar to Type-01 driver. except the ODBC driver is ~~driver~~ replaced with vendor specific native libraries. this libraries are provided by respective databases.

→ This libraries are mostly written C(or) C++.

→ we have to install this libraries.

→ Type-02 driver converts JDBC calls into vendor specific database calls, So that the database can understand

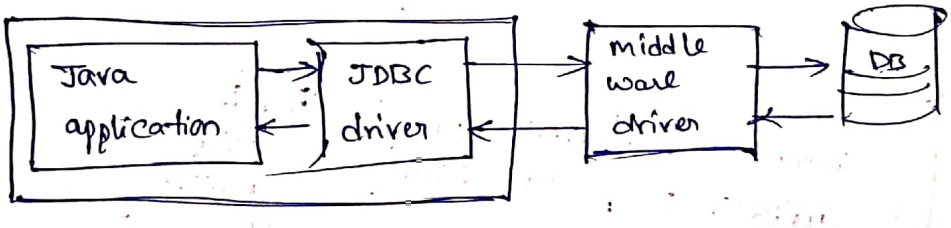
Those calls easily.

Advantages :- → type-02 driver is faster than type-01 driver because there is only one conversation. no need of any ODBC drivers.

→ The portability of type-02 driver is more when compared with type-01 driver because it works only windows machines.

Disadvantages :- →This driver is platform dependent driver and using vendor specific libraries where it becomes difficult to switch between different databases. Hence, it is database dependant driver.

3) Middle ware driver :-



type-03 driver is converting java calls into middle ware server calls and middle ware server calls into database specific calls.

→ Internally middle ware driver may use type-01, type-02 or type-04 drivers.

Adv:- →This driver can't communicate with database directly - so it is database Independent Language. and this driver is writtening java language. so, it is also platform Independant driver.

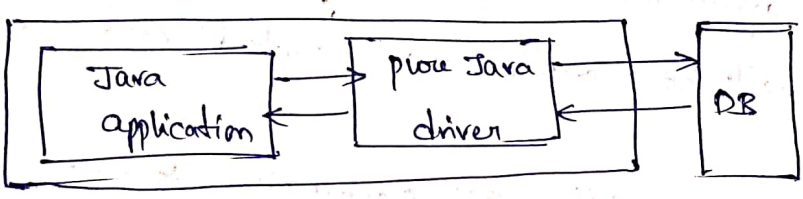
→ This driver doesn't required any ODBC driver or vendor Specific libraries.

Disadvantages:-

→ Internally type-03 driver using middle ware Server their might be performance issues.

→ middle ware Server is most costlier than other driver softwares.

(4) thin driver (or) pure Java driver :-



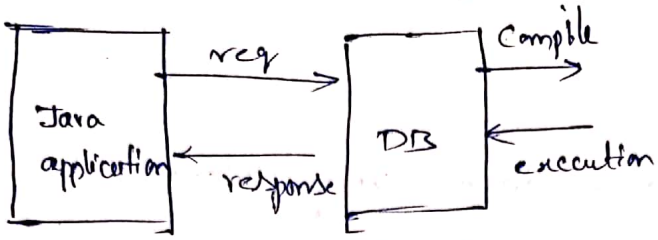
Type-04 driver is converting JDBC calls directly in to the database specific calls.

→ Type-04 driver doesn't required ODBC driver (or) native specific libraries (or) middle ware Server.

Advantages:- → Compared to type-03 driver type-04 driver is directly communicating with database, and it is not depending any other external softwares.

Disadvantages:- → This driver software directly communicate with database, so, it is database dependant driver.

Statement :-



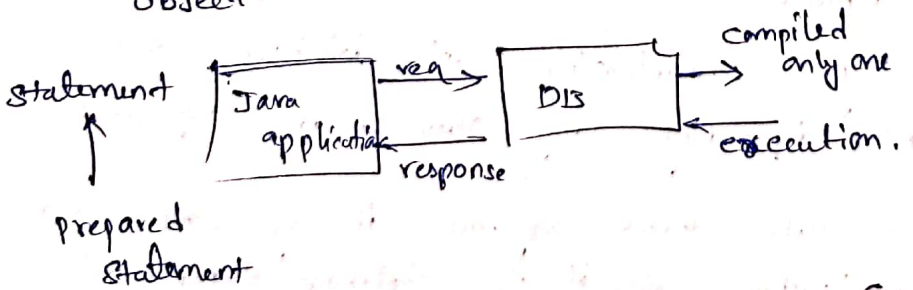
1 command = 1sec (req) + 1sec (compilation) + 1sec (exec) + 1sec (response)

1 lakh = 4lakh sec

→ While using object statement we write the sql commands as a actual argument, in execute update method It will be compile and executed each and every time.

→ In case of using 1 lakh SQL commands it takes long time to process the data. to overcome the compilation of each and every command every time they provided prepared statement.

→ prepared statement is the subclass of statement object.



1 command = 1sec (req) + 1sec (compile once) + 1sec (execution) + 1sec (response)

(3sec)

= 3sec

1 lakh commands = 3 lakhs second only compared to statement time

→ while using prepared statement the commands will be compiled only for one time and it will be executed for any no. of times.

→ prepared statement provides place holders (?) to pass the values during run time. where as statement object allows only static commands

→ to set the values to the respective place holders, the prepared statement will provide setter methods.

result set: It is an interface which is utilize to store the retrived data from the database. It is declared in written type for execute query method, in statement, prepared statement, call able statement.

→ It provides next method to iterate over the data provided by the database.

→ It provides getter() methods based on the datatypes which is use in database application.

→ to store alpha numeric data in java we have String. In database we have varchar, to store Integer values we have Integers in java - In database integer values have Int data type and for large no. of value we have big int datatype etc.

→ to store decimal type of data we have floating type in java - in datatype we have decimal

for example :-

get Int (1)

get string (2)

get double (3)

INT	VARCHAR	Decimal
Id	name	points
1	prasanth	30.0
2	sciKumar	21.0
3	Rajini	99.0

Statement :-

Statement object can be utilized to perform static querys.

→ SQL commands is passed after creation of statement object. we cannot pass dynamic values to the statement object during run time.

Prepared statement :- It is the child of statement which is excuted faster than statement object. it can be utilized to perform both the static & dynamic Quaries

→ while using prepared statement object we can pass the values to the prepared statement object during run time with the help of placeholders (?) - to set the values to placeholders prepared statement provides

Setter methods ( ).

Setter method ( ) :-

⇒ We write SQL commands as the actual arguments for the prepared statement object creation.

→ SQL commands are executed during prepared statement object creation.

→ Incase of statement object SQL commands is compiled & executed each and every single time.

→ In case of prepared statement SQL commands is compiled only once and can be executed any no. of times.

Callable statement :-

→ It is the child of prepared statement which is executed faster than prepared statement.

→ It can be utilized to execute static query's, dynamic queries and also to call store procedures

→ Store procedure is just like a Java method where we can write all the SQL commands b/w begin & End key words

→ Callable statement is specially used for calling Stored procedures., to call the stored producedure we follow the basic syntax i.e, stated below.

Ex:- { call procedureName & ( ) }

Methods :-

→ executeUpdate() :- It is used to DML commands

→ The return type is int.

Ex:- int i = statement.executeUpdate();

(*) → executeQuery() :-

It is used to perform DQL commands

→ like select * query.

Ex:- statement s = connection.executeStatement(
String "select * from employee");

→ resultset() :-

resultset r = s.executeQuery();

→ The return type of executeQuary Methods is resultset

(*) → executeMethod() :- It can perform DDL, DQL, DML Commands.

→ The return type is boolen type.

* Among statement, prepared statement & callable statement any of the object is carrying select query's, the execute method return the value as true ✓

→ If any of the statement object contains DDL/DML commands the execute method returns false ✗

→ execute Batch :- It is utilised to perform DDL, DML commands as a group of commands in one single Entity so, that all group of commands can be executed with one single request with the help of add batch () method.

→ The return type of execute batch method is Integer array.

Servlet :- Servlet is a web component which is utilized to generate dynamic webpages.

→ Servlet is an API which is playing important role in web applications

→ servlet is a web component as it is invoked by the web server and application server.

→ Servlet is mainly utilized to read the data from the user. and compare with data which is retried from the database.

→ after comparing the data servlet will process the logic and generate dynamic web page. (Dynamic HTML)

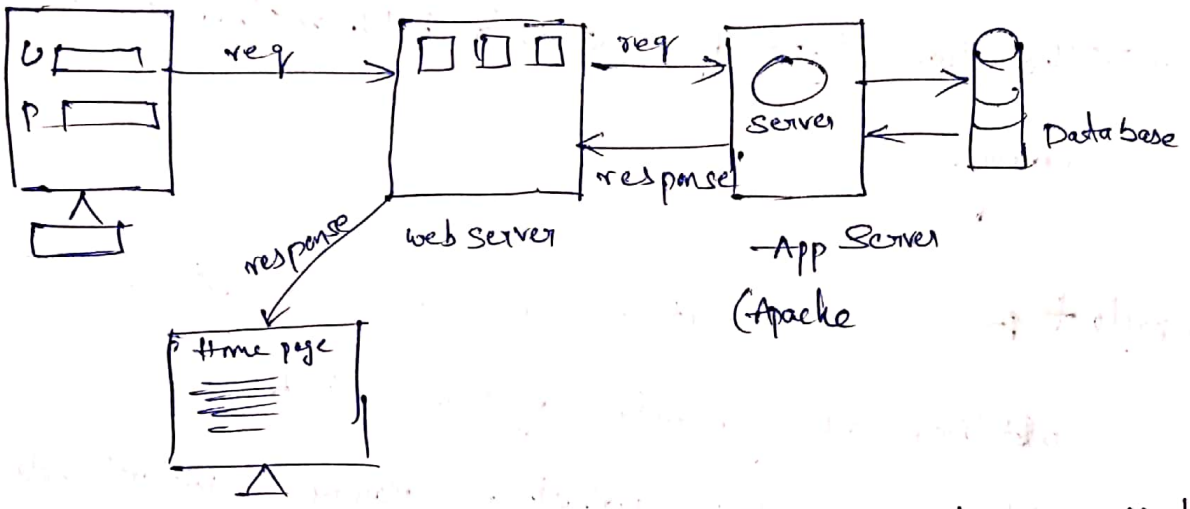
→ A servlet will process the data from user and data from database. So, it can be called as business logic or processing logic.

→ Servlet is built using abstract classes which are subclasses of Servlet Interface.

→ The servlet Interface is having two subclasses at different levels.

(i) Generic Servlet class (abstract class)

(ii) HTTP servlet (abstract class and subclass of Generic servlet class)



→ To run the servlet component we require application server like Apache-tomcat, Glassfish, Websphere etc

→ Servlet is total depends upon java language

→ Apache-tomcat server is also written in java language so, servlet are more familiar with apache-tomcat server, So, we can call Apache-tomcat server as web container. because it is storing all the web components like servelts, JSP etc.

→ A servlet component is invoked with a request which is passed by the user from the Html webpage connected to servlet java file with the help of deployment descriptor (web.xml)

→ The deployment descriptor contains the important information where the partical URL is mapped to a particular servlet java file.

→ In that deployment descriptor each and every servlet file is mapped to URL

→ always the URL should be mapped for only one servlet file. otherwise the tomcat server will be stopped.

→ To create a servlet we need to create a class and make it subclass of either Generic Servlet or HTTP servlet classes.

Creating a Servlet :-

We can create a servlet mainly using two classes which are abstract both Generic servet & HTTP servlet but only Generic servlet class is having abstract method.

i.e, - service method.

→ If we make any class as subclass of Generic servlet we have to override service method.

→ This service method can except any method-type of Data.

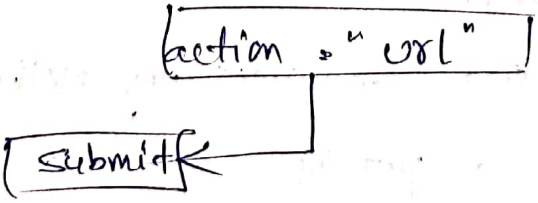
Eg :- Get type or post type etc.

→ creating a html and mapping a servlet class with the help of web.xml file (deployment descriptor)

```
<html>
    <head>
    </head>
    <body>
        <form action = "url">
            name : <input type = "text" name = "n">
            <input type = "submit">
        </form>
    </body>
</html>
```

```
<servlet-mapping>
    <servlet-name> Demo </servlet-name>
    <url-pattern> /url </url-pattern>
</servlet-mapping>
```

In html form submit invokes the URL



In html form we have action attribute which requires value the value which is given for the action attribute is a URL

→ The url is should be mapped to a servlet class using <URL-pattern>

→ All this mapping tags should be included web.xml (Deployment descriptor)

```
<servlet>
<servlet-name> Demo </servlet-name>
<servlet-class> Com.sk.D </servlet-class>
</servlet>
```

The URL is mapped to a servlet class file with the help of <servlet-name> tag.

→ Both <servlet> tags <servlet-mapping> tags are connected with the help of same servlet name which is included using <servlet-name>

```
class D extends GenericServlet {
Public void service (ServRequest req,
                     Serv response resp) {
String value = req.getParameter("n");
PrintWritten out = resp.getWriter();
out.println("<h1> name: "+value+ "</h1>");
}
}
```

→ In order to collect the information from the user we are utilizing HTML forms. We have to run html form using application server. i.e., tomcat server. which will be displaying the webpage in browser application.

→ Meanwhile when tomcat server is started it will create servlet object inorder to access the methods which are overrided. Inorder to invoke service method which is presented in servlet object. the tomcat server will create both request & response object. All this objects are created whenever this application server started.

→ The information which is collected from the user is loaded with in the request object. then the servlet class is executed by the JVM and accesing the information from the request object.

→ to access the information from the request object require getter() methods. i.e., req.getparameter(" ");

→ Always the data which is collected in the req object is in string format.

Ex:- String v = req.getparameter("n");

→ The information which is retrived from the request object we can displayed to the user using response object with the help of printwriter

Note :- Tomcat server will be stopped if the URL which is provided for html is not mapped properly to any of the servlets. In web.xml if the URL is not precided with "/" the html page might be loaded the user may get 404 not found.

Servlet chaining :- The process of connecting a one servlet to another servlet is known as servlet chaining

→ Servlet chaining can be done in two ways

(i) using the request dispatcher

(ii) using Send Redirect ("file/path")

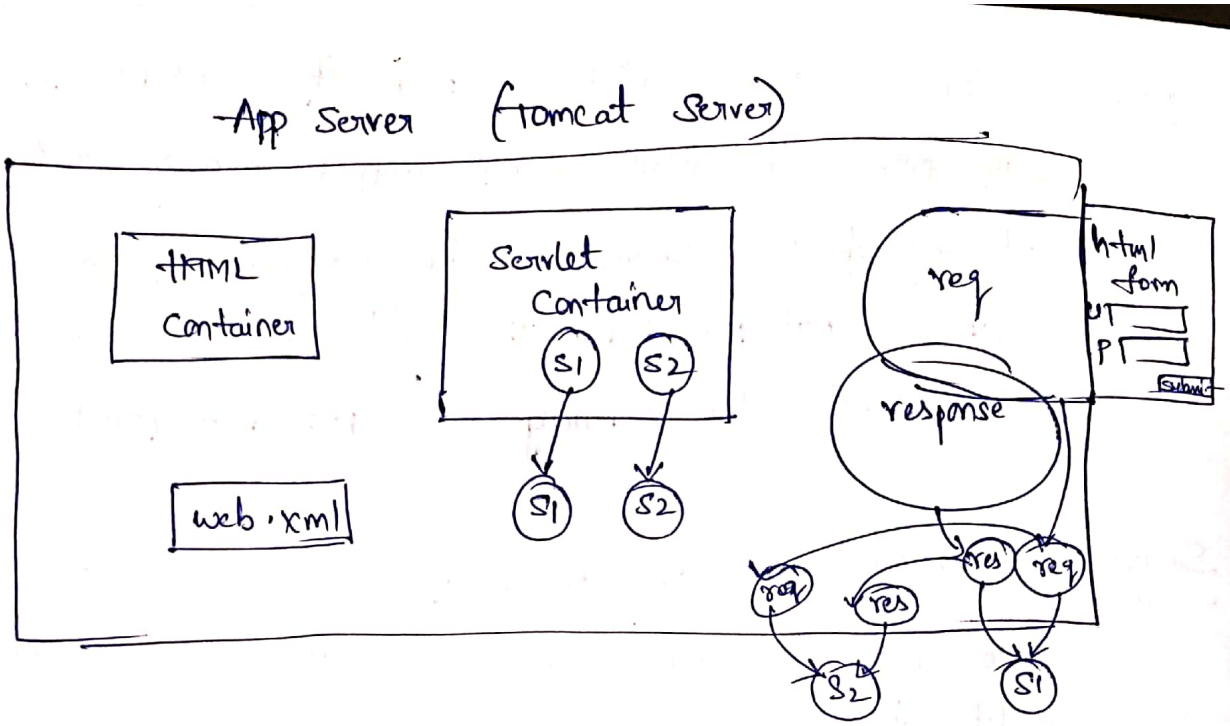
(i) Request Dispatcher :-

→ It is an interface which is utilize to perform servlet chaining.

→ The object of request dispatcher type is created with a help of get request dispatcher().

getRequestDispatcher("serdurl");

→ RequestDispatcher() Method is present is belongs to get servlet Request object */ In case of utilizing request Dispatcher will understand to transfer both request and response objects will be transferred to specified URL or any file page /*



Send redirect method :-

→ In order to utilize send redirect() method for servlet chaining first we need to create a servlet using HTTP servlet then our class becomes subclass of HTTP servlet

→ Then after making subclass of HTTP servlet we need to override do get() or do post()

→ we have to follow the above requirements inorder to use send redirect() method

→ If we are using requestDispatcher we can transfer the data utilizing request object itself because requestDispatcher forwards the same request object from one servlet to another servlet in servlet chaining process.

→ By using sendredirect() the control is displaying the response to the user in address bar as we are being forwarded to another webpage.

→ while using sendRedirect() when servlet 1 receives req & resp objects, sendRedirect() generates new request objects to send the new request to second servlet

→ to transfer the data from one servlet to another servlet by using sendRedirect we have 3 different ways.

(i) url rewritting
(ii) session tracking
(iii) cookies

url re-writting :- to perform this process we need to utilize the queryString (?variable name = value)

→ using url-re writting we can send only little amount of data

→ Increase if we want to send large amount of data we have to go for session or cookies.

Session tracking :-

This process is utilized to transfer the data by using http session. we can get hold of the session by calling getsession(). which belongs to Http servlet request object.

(*) Session :- It is a time interval b/w the users request till we get the responce as we login the application until logout of the application.

Syn :- HttpSession S = req.getsession();

→ we can utilize the session object as a temporary storage area where we don't need to store unnessary data in database.

→ we can store the data in session object by using setattribute() method, It can store large amount of the data and It can be retrived aswell.

Eg: HttpSession s = req.getsession();
s.setAttribute("identifier", "data of any kind");

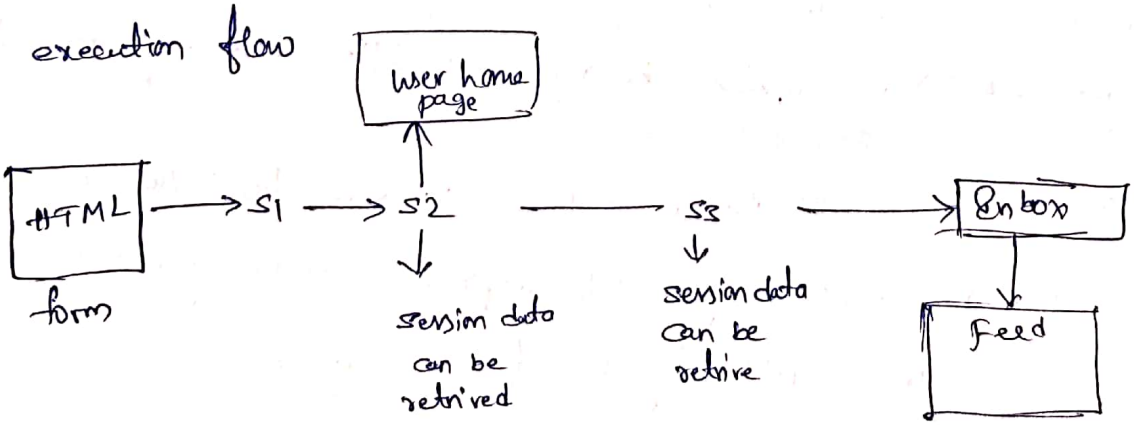
→ In the above statement the data which is stored in session object is Converted into object datatype.

→ whatever the data in the sesion object is generalised into object datatype.

→ to retrive the data from the session object we use below statement.

object obj = s.getAttribute("identifier");

→ we can retrive the data whenever we want throughout the execution flow



Cookies :-

→ cookie is a piece which will be stored in browser application of clients machine. always cookies are created by web-sites the users regularly visit

→ By accepting the cookies of that website that user easily can we identified by the website. and It makes online experience easy and faster.

→ By storing those cookies the user will be signed in with that website and It always gives you a relevant content that users references.

→ Cookie are classified into two types

① non-persistence cookie → the cookie which is stored only when the user is logged in application and they disapper when user logout of the application. this are known non-persistence cookies.

② persistence cookie → The cookies which will be stored in the client browser for a specified amount of time is known as persistence cookie.

→ these cookies will be appered in the clients browser application. even after the user logged out of the application.

→ to check out the cookies in browser application.

Step-01 :- go to settings

Step-02 :- check out privacy & Security

Step-03 → Checkout see all the cookies & data.

Step-04 → Search localhost

Non-persistence-cookies :- cookie is a class, which is used to collect the Information in the form of String using cookie Constructor.

Syntax:- Cookie c = new cookie ("Identifier", "value");

Eg:- Cookie c = new cookie ("yuva", "My name");

Cookies works on clients site but created at Serverside and this cookies will be added to the response object and stored in the browser application.

persistence Cookies :- The cookies which is stored which specific amount of time using a method which belongs to cookie class

```
Cookie c= new Cookie ("my cookie", "my data");
    c.setMaxAge (6*60);
```

→ This cookies will be stored only for 6 minutes in browser application and after tym expires. The cookies will be removed automatically.

Note :- Servlet - chaining is also known as servlet - collaboration.

Servlet-config :- Servlet config object is always created for each and every servlet class to Initilize the values using parameters.

→ servlet config object will work only for one servlet, So, It is local scope . we Initilise the values using servlet config object by providing the <Init-parem> in web.xml.

```
<Init-param>
    <param-name> email </param-name>
    <param-value> sss@gmail.com </param-value>
</init-param>
```

→ servlet config object Initilasing the values to the particular servlet object using getInitparameter.

→ This servlet config object Is Initialising the values to only one particular servlet object by following statements. where we can retrive

```
ServletConfig Config = getServletConfig();
String email = Config.getInitparameter("email");
```

Servlet context :- A web container (tomcat server) will create a one global object for each and every web application separately.

→ In our words in the eclipse workplace If we have 10 web application, the tomcat server will create 10 servlet context objects.

→ Hence, servlet context is of global scope

→ we can initilise the values to any servlet class in our web application using Context parameter

→ In web.xml

```
<context-param>
    <param-name> password </param-name>
    <param-value> class143 </param-value>
</context-param>
```

→ servletcontext context = getservletContext();

String password = context.getInitparameter("password");

FRAME WORKS :-

Hibernate with JPA (Java persist Apis) :-

→ In order to create any web applications we might utilise database to store the data.

→ So, we need JDBC to communicate with database application

→ But, In order to make relations between two different or more than 2 tables. It is increasing the complexity (difficult) to code while using JDBC

→ using JDBC it becomes complex to perform commands like joints, to make foreign key relationship etc... to overcome this complexity (difficult) we have hibernate framework.

→ frameworks is nothing but collection of utilities. (API's).

→ hibernate framework will perform all types of SQL Commands like (joints, foreign key relationships etc) by providing inbuilt methods.

→ ORM tools with JPA :- (object relational mapping).

we have ORM tools like hibernate, toplink, iBatis. where it becomes difficult if we create and ORM application like hibernate application. because if we want switch to another ORM tool : we have to change the entire code.

→ To overcome this problem they have introduced a Specification which is known as "JPA - Java persistance API"

→ JPA provides as common configuration for all the ORM tools like hibernate, toplink, iBatis.

→ we cannot create JPA application because it is a specification. But we can create hibernate with JPA / Toplink with JPA etc.....

→ JPA provides Entitymanagerfactory, Entitymanager, EntityTransaction.

→ EntityManagerfactory is an interface which is utilized as return type in persistance class for a static method. i.e.,

Create. emf (string " ")

→ The object of Entity manger factory is created with the help of helpermethod which is static in persistance class.

EntityManagerfactory emf = persistance. createEMF ("persistance unit name");

→ The persistance unitname for persistance. createEMF() is provided by persistance. xml file.

→ This file consists of all database related information.

→ Inorder to save the data in database Entity Manger provides persist method ().

→ EntityManager object is created with the help of EntityManager factory referance variable.

Syntax :-

```
EntityManager entityManager = entityManagerFactory.createEntityManager();
```

Note :- we have to create a class which starts with uppercase letters and variable names should be similar to class name. under naming convention which carries 5 marks.

→ EntityManagerFactory handels the connections b/w Java application and database application with the help of persistance.xml file.

→ If we don't have persistance.xml file (or) if the persistance unit name is wrong we get persistance exception. and display's info like META-INF/persistance.xml not found

→ In order to create persistance.xml first we have to create a source folder belongs to Java → select the project > cntrl+N > In the wizard search folder > select source folder which is present and name the source folder name as "src/main/resources" > then select src/main/resources > control+N & search for folder which is belongs to general. and name the folder name as "META-INF".

Then select "META-INF" ⇒ control+N search for file which belongs to general & select normal file. and name the file name as "persistance.xml".

Source folder ← src/main/resources

folder ← META-INF

file ← persistence.xml

↓

Database configuration file.

Search maven-archetype-webapp

Group-Id : com.sk

Artifact Id : servlets-demo.

Hibernate relationships:- relationships are defined to makes connection between different entity's.

entity's represents tables in the database.

→ In order to make foreign key relationships b/w two or more tables, Hibernate framework makes it very easy by providing the relational mapping annotations.

(i) @ one-to-one

(ii) @ one-to-many

(iii) @ many-to-one

(iv) @ many-to-many

one to one :- In one to one we are mapping a one table to another table, but Incase of one-to one mapping only one object data in mapped to only one objects.

Eg:

```
@Entity
class Student {
@Id
private int id;
@ one to one
private Address address
```

```
@ Entity
class Address {
@Id
private int id;
private string name;
}
```

Student

Id	name	Add-id
1	yuva	101

Add-Id	city
101	Hyd

→ Incase of one-to one mapping, we can take the above example, where student entity is the parent entity which depends on the address Entity

student - dependent class
address - Independent class

→ If we try to persist (save) student object without saving address object we get transient exception

→ Because, the student table requests the primary key of the child object, where student table maintains the foreign key relationship with Adress table

→ So, always we have to save the Independent object first, i.e, address object

→ after saving the address object we can save student object.

→ The above rules will be applied for all the remaining CRUD operations. Except read & update operations.

→ we face problems while saving (or) deleting the data.

→ parent object is depending upon the child reference object.

→ Incase of deleting the data, we cannot delete child object directly, because child reference is present in parent table.

→ to perform delete operation, first we have to delete dependent object first, i.e., student entity.

→ After deleting parent object we can easily delete child object.

(4) One to many :- This kind of relationship is used to represent where one entity is related to many entity objects.

```
@Entity
Class University {
@Id
private Int id;
@one-to-many
private List<candidate> Candidate();
}
```

```
@entity
Class candidate {
@Id
private Int id;
}
```

university

id	university
1	AU

Candidate

id	name
101	Raj
102	Ram

university -- candidate

univ-id	can-id
10	101
1	102

→ In this relationship it will be having three tables, because university table cannot have two primary keys has a foreign key in university table.

→ An extra table is created and it is the combination of both university and candidate where 2 primary keys are mapped in this extra table.

→ The same rules are applied which is similar to one to one relationship.

(*) Many to one :- In this kind of relationship we mentioned many to one annotation where all entity objects referred to one entity object.

Eg:
```
@Entity
class Institute {
@Id
private int id;
}
```

```
@Entity
class Branch {
@Id
private int id;
@ManyToOne
private Institute institute;
}
```

Institute

id	name
1	jspiders

Branch

id	name	Institute id
101	punjagutta	1
102	Jntu	1

→ Incase of many to one relationship it creates only two tabels. , Because in child table it creates foreign key column by representing the primary key of the parent table.

→ Even incase of many to one relationship save method & delete methods works same.

→ we have to save both parent & child data at a time otherwise we get a transiant exception.

→ To perform delete operation, first we have to delete the table which is having foreign key data. then we can delete the other table data which is related to it.

Many to many :- In many to many relationship two different entity's having many to many relationship is represented by three different tables.

→ In many to many relationship one object data is connected to many objects as well as many objects are connected to the many of objects.

Eg :- @Entity
Class FoodOrder {

@many to many
private List <FoodItem> foodItems
}

@Entity
class FoodItem {

@many to many
private List <FoodOrder> foodOrder
}

Many to Many :-

FoodOrder

Id	order	name
1	meal	vikas
2	snacks	mahesh

FoodItem

Id	Item	price
101	pizza	250
301	burger	150
302	noodels	100

FoodOrder_FoodItem

FoodOrderId	FoodItemId
1	101
1	301
2	101
2	302

food-ItemId	food order Id
101	1
101	2
301	1
302	2

→ In the above example, one order consists of List of Items
eg:- vikas ordered pizza, burger, noodels. In the same way pizza is belongs to food Item the same pizza is ordered by mahesh also. So, one Item belongs to two different orders and one order has many Items.

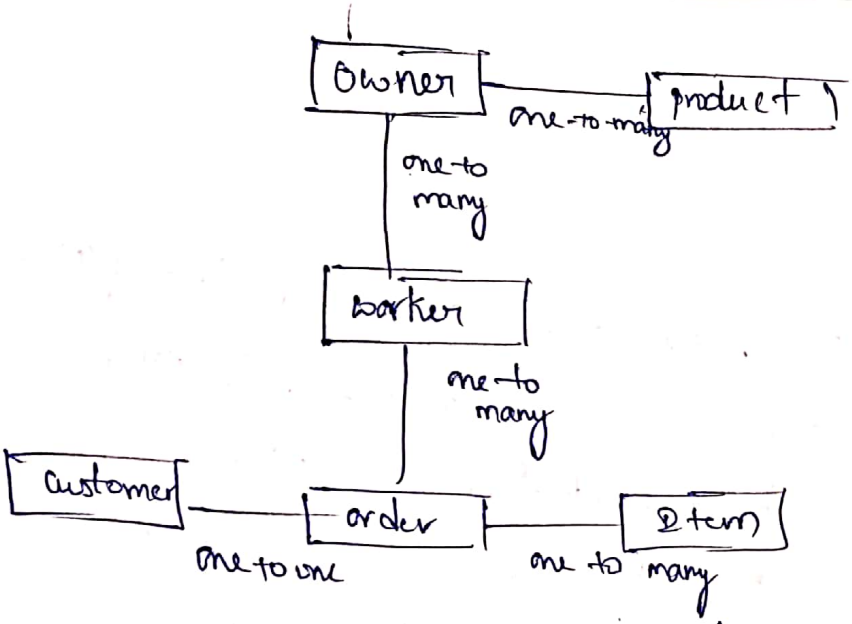
→ In the above scenario both the food order & food Item tables depends on each other, So, we are making at bi directional by cash cading

@manyTomany (cascade = CascadeType.ALL)

ER Model :- In order to create any project first we need to make a strategy to store the data.

→ ER models are where help full in order to make connection b/w different entitys. i.e., represented as tables in the database.

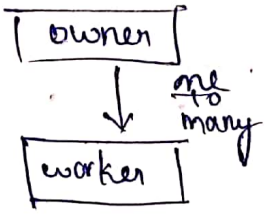
→ Before creating any project first we need to make a clear picture of ER Model.
eg:- we are taking food project model mentioned below.



→ In the above ER model first we need to make relationship between any two important Entity's.

→ As, the above example is related to real-time ~~Application~~ human example.

→ A person wants to establish a hotel, so, first owner hire the workers




```
@entity
class owner {
@

@one-to many (mapped By = 'owner')
  private List<worker> workers;
}
```

```
@ entity
  class worker {

  @manytoone
    @ owner owner;
  }
```

→ In the above scenario the parent entity (owner) maintains one to many and at the same time in the child entity (worker). also maintains many-to one, Then we have to mention mappedBy in the parent entity (owner) it will create only two tables..

→ we are using mappedBy just reduce the no. of tables.

@one-tomany (mapped By = "owner")

owner

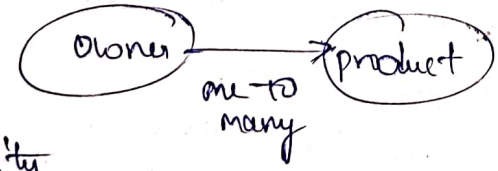
id	name
1	rajesh

worker

@manytoone
private owner owner;

id	name	owner-id
101	pullaye	1
102	pavani	1

(2) → we have to make relationship b/w owner and product



@entity
class owner {

@one-tomany (mappedBy = "owner")
private List <worker> workers;
@one to many (mappedBy = "owner")
private List <foodproduct> products;
}

@entity
class foodproduct {

@ manytoone
Private Owner owner;
}

→ owner, worker, foodproduct are main entity's in the scenio.

→ first, we need to ~~save worker object~~ Create both the worker, owner objects, and we have to set the owner object to the worker.

→ Owner o = new Owner();
Worker w = new Worker();
w.setOwner(o); // setting owner object to the worker

→ Create the food product object and retrive the existing Owner object.

OwnerDaoImpl owerDao = new OwnerDaoImpl();
Owner o = ownerDao.getOwnerById(1);
FoodProduct p = new FoodProduct();
p.setOwner(o); // setting Owner obj to the product

→ After completion of three main entity's i.e, owner, worker, food products as these are three core entity's for food product in real time scenio aswell.

→ The worker takes orders from the customers. customer after ordering the list of items worker first take the customer details and then order will be executed.

→ So, first we need to save the customer and then order should be saved.

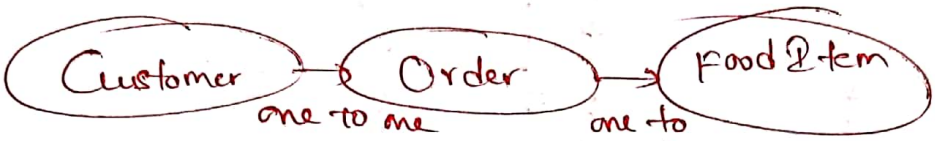
Order | one to many → FoodItem

class Order {
@OneToMany(cascade = "CascadeType.ALL")
private List<FoodItem> items;
}

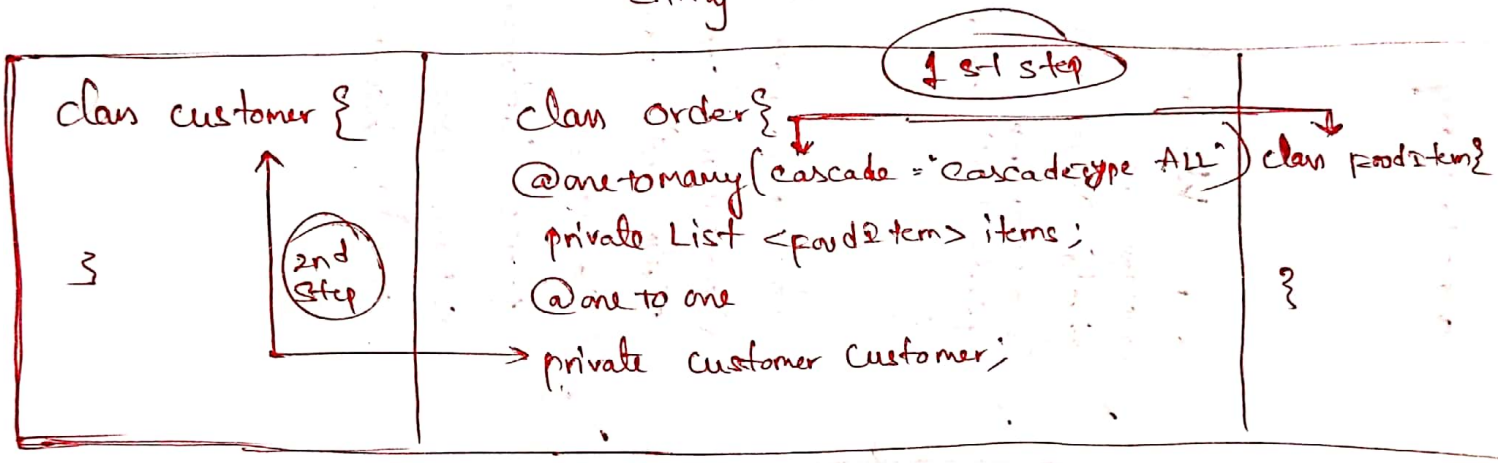
class FoodItem {
}

→ In the above relation the parent entity (order) is mentioned cascade. so, that the Entity's which are child of order will be saved automatically, No need persist the foodItem objects.

→ Then make the connection of the customer Entity to the order entity as one to one.

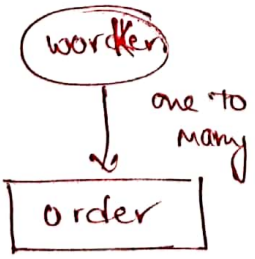


→ order is the parent for the customer Entity



→ After making this connection b/w three different Entity's which are interrelated.

→ In realtime scenrio orders are taken by the workers So, workers has relationship with foodorder.




```
class worker {
@many-to one
private Owner owner;
@one-to many (mappedBy = "worker")
private List<order> orders;
}
```

```
class order {
@one-to many (cascade =
"cascadeType.ALL)
P List <foodItem>
-foodItems;
@one-to one
private Customer
Customer;
@many-to one
private worker worker;
}
```