**04-03-2022**

**Phase 2 :**

Day 1

Maven :

**SQL** : Self learning using My SQL Database.

**JDBC** : Java Database Connectivity

**Hibernate** : Adv of JDBC

JEE :

**Servlet**

**JSP**

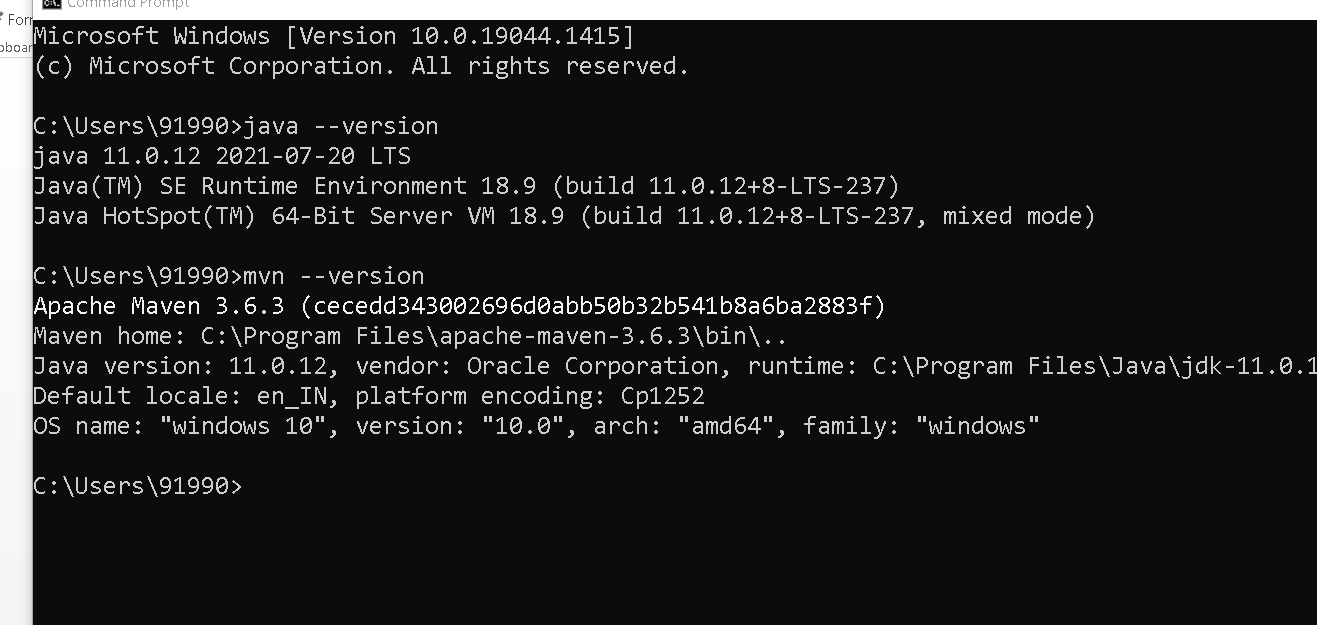
**Maven is known as Build tool. Build tools is responsible to compile, run, creating jar, war or ear file (Java Archive, Web Archive, or Enterprise archive), download the dependencies, creating documentation etc.**

**ANT :**

1. **Maven**
2. **Gradle**

**MAVEN set path**

**To check java and maven please open the command prompt**

****

**Creating simple maven project**

**Create folder with any name (maven project)**

**Open the command prompt in that location**

**mvn archetype:generate**

**wait it will download few dependencies from central repository.**

**mvn archetype : generate : after downloaded dependencies.**

**It will display some number ie 1870 :**

**Then click enter key**

**­Again hit enter key**

**It will ask the group Id : MyProjects (can be any name)**

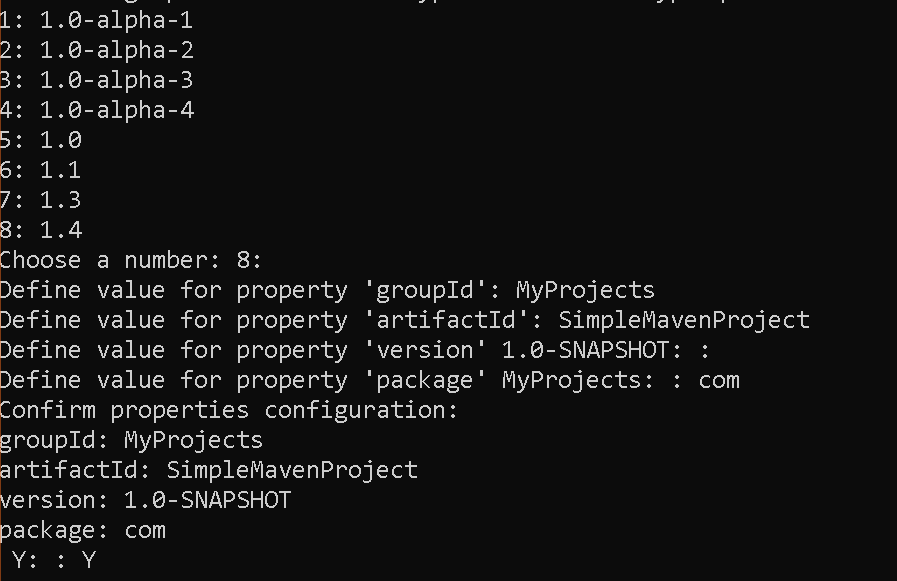
**It will ask the artefact id : SimpleMavenProject**

**Version : hit enter key**

**Package Name : com**

**Y**

**Then enter the key**

****

**pom.xml (Project Object Model). This file contains maven project description details.**

**Maven provide goals.**

**Maven goals are use to do specific task.**

**mvn compile**

**mvn clean**

**To run the maven project through command prompt we have use the command as**

**mvn exec:java -Dexec.mainClass="com.App"**

**My SQL Database in Virtual Lab**

**Open the command prompt**

**To connect the database you have to type**

**mysql –u root –p**

**password : Simplilearn**

**connect the my sql database through console.**

**Syntax to create the database.**

**create database databaseName**

**create database my\_database**

**command to view all the databases**

**show databases;**

**Command to switch into database.**

**Use databaseName**

**use my\_database**

**syntax to create the table**

**create table employee(id int primary key,name varchar(10), salary float);**

**Table is Employee**

**Id,Name,Salary**

**Id must be hold number value and it must be unique.**

**Name must be hold string value**

**Salary must be hold float value**

**desc employee : it display the table structure.**

**Insert the record in table.**

**insert into employee values(1,’Ravi’,12000);**

**To view the records from a table**

**Select \* from employee**

**Delete the records**

**delete from employee where id =4;**

**update query**

**update employee set salary = 24000 where id=1;**

**JDBC : Java Database Connectivity :**

**Java Program MySQL**

**JDBC**

**Or**

**Hibernate**

**JDBC : Java Database connectivity : Which provide set of classes and interfaces which help to connect the database through Java technologies.**

**JDBC throw (SQL exception) checked exception so whenever we write the JDBC code you have to write the code with exception handling.**

**In JDBC we have to load the Driver. Driver is a pre-defined class provided by vendor (database provider) which help to connect the database.**

**There are totally 4 types of driver**

**Type 1**

**Type 2**

**Type 3**

**Type 4 or jdbc pure or thin driver.**

**Class.forName(“driverName”)**

**In Java Class is a pre-defined class and name itself is a Class which contains forName() static method. which help to load the Driver.**

**Establish the connection**

**DriverManager is a pre-defined class which provide getConnection() static method. and we have to provide url, username and password. getConnection method return type of Connection interface reference.**

**Connection con = DriverManager.getConnection(url,username,password);**

**JDBC provided Statement and PreparedStatement interface which contains set of method which help to insert, delete, update and retrieve records from database.**

**executeUpdate() : insert, delete and update**

**executeQuery() : select query**

Day 2

07-03-2022

**PreparedStatement : PreparedStatement is a interface which provide set of method which help to dynamic SQL Operation. It support parameterized concept.**

**If we use Statement each time query will compile on java side and execute on database side.**

**But if we use PreparedStatment it compile only once and execute n number of time.**

**So performance wise PreparedStatement is faster than statement interface reference.**

**JDBC with Maven with CRUD Operation on Employee table**

**Backend side Table must be match or map to Java Side Java Bean.**

**Table --🡪 Employee**

**class -🡪 Employee**

**column name must be map to variable name in java side.**

**As a good practise don’t; write JDBC code inside main method.**

**To write JDBC code we have to create DAO class**

**Ie Data Access Object.**

**This class contains pure JDBC or database Logic.**

**EmployeeDao etc**

**ProductDao**

**LoginDao**

**DAO class is not responsible to take the value through keywords. Means inside DAO class don’t’ create Scanner class object.**

**DAO class not responsible to interact with input device.**

**Service class :**

**This class is responsible to write business logic.**

**EmployeeService**

**CustomerService**

**ManagerService**

**Service layer not responsible to take the value through keyboard means don’t create the object of Scanner class. Service layer not responsible to display the output.**

**This class contains pure business logic.**

**Logic can be apply before call DAO method, or after called DAO method or Before as well as After DAO method base upon requirements.**

**Main class**

**This class is responsible to take the value through keyboard.**

**Then create the JavaBean class object**

**Set the value through constructor or setter method.**

**Then create service class object.**

**Pass the object service layer**

**And get the result type as message and display on console.**

**App.java**

**package** com.main;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Scanner;

**import** com.bean.Employee;

**import** com.service.EmployeeService;

**public** **class** App {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Scanner sc = **new** Scanner(System.***in***);

EmployeeService es = **new** EmployeeService();

**int** id;

String name;

**float** salary;

String con="";

**int** choice;

String result;

**do** {

System.***out***.println("1:Add, 2: Delete, 3: Update, 4:Display All");

System.***out***.println("Plz enter your choice");

choice = sc.nextInt();

**switch**(choice) {

**case** 1: System.***out***.println("Enter the id");

id = sc.nextInt();

System.***out***.println("Enter the name");

name = sc.next();

System.***out***.println("Enter the salary");

salary = sc.nextFloat();

Employee emp = **new** Employee(id, name, salary);

result = es.storeEmployee(emp);

System.***out***.println(result);

**break**;

**case** 2: System.***out***.println("Enter the id");

id = sc.nextInt();

result = es.deleteEmployeeInfo(id);

System.***out***.println(result);

**break**;

**case** 3:System.***out***.println("Enter the id");

id = sc.nextInt();

System.***out***.println("Enter the salary");

salary = sc.nextFloat();

Employee emp1 = **new** Employee();

emp1.setId(id);

emp1.setSalary(salary);

result = es.updateEmployeeInfo(emp1);

System.***out***.println(result);

**break**;

**case** 4:List<Employee> listOfEmp = es.getAllEmployee();

Iterator<Employee> li = listOfEmp.iterator();

**while**(li.hasNext()) {

Employee e = li.next();

System.***out***.println(e);

}

**break**;

**default**:System.***out***.println("Wrong choice");

**break**;

}

System.***out***.println("do you want to continue?");

con = sc.next();

}**while**(con.equals("y"));

System.***out***.println("Thank you visit again!");

}

}

**Employee.java**

**package** com.bean;

**public** **class** Employee {

**private** **int** id;

**private** String name;

**private** **float** salary;

**public** Employee() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Employee(**int** id, String name, **float** salary) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **float** getSalary() {

**return** salary;

}

**public** **void** setSalary(**float** salary) {

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";

}

}

**EmployeeService.java**

**package** com.service;

**import** java.util.Iterator;

**import** java.util.List;

**import** com.bean.Employee;

**import** com.dao.EmployeeDao;

**public** **class** EmployeeService {

**public** String storeEmployee(Employee emp) {

**if**(emp.getSalary()<12000) {

**return** "Salary must be > 12000";

}**else** {

EmployeeDao ed = **new** EmployeeDao();

**int** res = ed.storeEmployee(emp);

**if**(res > 0) {

**return** "Record inserted";

}**else** {

**return** "Record didn't insert";

}

}

}

**public** String deleteEmployeeInfo(**int** id) {

EmployeeDao ed = **new** EmployeeDao();

**if**(ed.deleteEmployee(id)>0) {

**return** "Record deleted successfully";

}**else** {

**return** "Record not present";

}

}

**public** String updateEmployeeInfo(Employee emp) {

EmployeeDao ed = **new** EmployeeDao();

**if**(ed.updateEmployee(emp)>0) {

**return** "Record updated successfully";

}**else** {

**return** "Record not present";

}

}

**public** List<Employee> getAllEmployee() {

EmployeeDao ed = **new** EmployeeDao();

List<Employee> listOfEmp = ed.findAllEmployee();

Iterator<Employee> li = listOfEmp.iterator();

**while**(li.hasNext()) {

Employee emp = li.next();

emp.setSalary(emp.getSalary()+1000);

}

**return** listOfEmp;

}

}

**EmployeeDao.java**

**package** com.dao;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.util.ArrayList;

**import** java.util.List;

**import** com.bean.Employee;

**public** **class** EmployeeDao {

**public** **int** storeEmployee(Employee emp) {

**try** {

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

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/my\_database", "root", "root@123");

PreparedStatement pstmt = con.prepareStatement("insert into employee values(?,?,?)");

pstmt.setInt(1, emp.getId());

pstmt.setString(2, emp.getName());

pstmt.setFloat(3, emp.getSalary());

**int** res= pstmt.executeUpdate();

**return** res;

} **catch** (Exception e) {

System.***out***.println(e);

}

**return** 0;

}

**public** **int** deleteEmployee(**int** id) {

**try** {

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

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/my\_database", "root", "root@123");

PreparedStatement pstmt = con.prepareStatement("delete from employee where id = ?");

pstmt.setInt(1, id);

**int** res= pstmt.executeUpdate();

**return** res;

} **catch** (Exception e) {

System.***out***.println(e);

}

**return** 0;

}

**public** **int** updateEmployee(Employee emp) {

**try** {

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

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/my\_database", "root", "root@123");

PreparedStatement pstmt = con.prepareStatement("update employee set salary = ? where id = ?");

pstmt.setFloat(1, emp.getSalary());

pstmt.setInt(2, emp.getId());

**int** res= pstmt.executeUpdate();

**return** res;

} **catch** (Exception e) {

System.***out***.println(e);

}

**return** 0;

}

**public** List<Employee> findAllEmployee() {

List<Employee> listOfEmp = **new** ArrayList<>();

**try** {

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

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/my\_database", "root", "root@123");

PreparedStatement pstmt = con.prepareStatement("select \* from employee");

ResultSet rs = pstmt.executeQuery();

**while**(rs.next()) {

Employee emp = **new** Employee(); // converting each record into java bean object.

emp.setId(rs.getInt("id"));

emp.setName(rs.getString("name"));

emp.setSalary(rs.getFloat("salary"));

listOfEmp.add(emp);

}

**return** listOfEmp;

} **catch** (Exception e) {

System.***out***.println(e);

}

**return** **null**;

}

}

**Day 3**

**08-03-2022**

**Resource layer : This layer is responsible to provide the resource details.**

**Like database connection.**

**ORM : Object Relation Mapping**

**Limitation of JDBC**

1. **Using JDBC we can’t store and retrieve object from database. We have to convert Java object into sql and vice-versa.**
2. **SQL is database dependent language. If we move from one database to another database like MySQL to Oracle we have to change the query. 70 to 80% query are same but 20% query different.**
3. **JDBC throw checked exception so we have to write the code with try-catch or throws mandatory.**
4. **JDBC doesn’t support is a(Inheritance) and has a(association, aggregation, and composition) relationship.**

**Object Relation**

**Programming side Database**

**Mapping**

**In ORM JavaBean class is known as**

**Entity class**

**class Employee { Employee 🡪Table id,name,salary ID,NAME,SALARY**

**setter and getter methods**

**}**

**Mapping**

**Employee(Entity class) --🡪 EMPLOYEE(Table)**

**id -🡪ID (PK)**

**name🡪NAME**

**salary 🡪SALARY**

**Old Version we were using xml file to do mapping**

**New Version Xml file replace by annotation**

**@Entity**

**@Table(name=”Employee”)**

**class Employee { Employee**

**@Id**

**private int id; id,name,salary**

**private String name;**

**private float salary;**

**}**

**@Entity**

**@Table(name=”EmployeeDetails”)**

**class Employee { EmployeeDetails**

**@Id**

**private int id; id,fname,salary**

**@Column(name=”fname”)**

**private String name;**

**private float salary;**

**}**

**Configuration**

**It hold database details like drivername,url,username,password,dialets class(responsible to convert java object into sql) and entity class details.**

**Configuration file can be xml, or properties file or java classes.**

**Hibernate : Hibernate is a open source framework which support ORM features.**

**Hibernate Test Application**

**First create the Object of Configuration class**

**Then using reference you have to load the xml file**

**con.configure(“hibernate.cfg.xml”);**

**Then we have to create the SessionFactory interface reference using syntax**

**SessionFactory sf = con.buildSessionFactory()**

**It is like a Connection in JDBC.**

**Then we have create the Session object which is equal to Statement or PreparedStatement interface reference.**

**Using Hibernate if you want to do any DML Operation ie insert, delete and update you have to do inside Transaction. So means by default through Hibernate it not auto commit. But by default auto commit option apply for JDBC.**

**SQL : Structure Query Language : Database dependent**

**Select \* from employee (retrieve all records from employee table ie SQL is not a case sensitive).**

**HQL : Hibernate Query Language : Database independent**

**Select emp from Employee emp (retrieve all properties from Employee entity class ie Employee is JavaBean or Entity class name)**

**Day 4**

**09-03-2022**

**Create the Maven Project**

**In pom.xml file**

**Add properties to change the version of java**

**Add two dependencies.**

**My sql connector**

**Hibernate**

**Create table -🡪 Employee**

**com.bean.Employee -🡪**

**com.dao.EmployeeDao 🡪**

**com.service.EmployeeService🡪**

**com.resource.DbResource 🡪**

**com.main.DemoTest 🡪**

**Hibernate relationship**

**Has a relationship : aggregation or composition**

**Is a relationship : inheritance**

**Has a relationship**

**One to many : primary key and foreign key**

**Trainer -🡪 Students**

**One to one : Primary key to FK or PK**

**Person has one Aadhar/Passport/PAN Card**

**Many to one : Employees working in one Project**

**Department**

**Many to many 🡪 Students more than one technologies**

**10-03-2022**

**Join using HQL**

**Join Using SQL**

**JPA**

**Java**

**J2SE J2EE J2ME**

**JavaSE JavaEE JavaME**

**JSE JEE JME**

**Standard Enterprise Micro**

**Edition Edition Edition**

**JEE : Java Enterprise Edition**

**JSE is use to create the standalone or desktop application.**

**JEE is use to create the web application.**

**https://**[**www.google.com**](http://www.google.com)

**req(http/https)--🡪**

**client Server**

**🡨--- res(http/https)-----**

**Html 5 css 3**

**JavaScript**

**HTML 🡪 display the content**

**CSS 🡪 how to display the content (presentation logic)**

**JS 🡪 programming or events on contents.**

**var a=10;**

**a=”Ravi”**

**JEE**

**Asp.net**

**Php**

**Python**

**CGI**

**Node JS**

**JEE : Servlet, JSP (Java Server pages) and EJB (Enterprise Java Bean).**

**Servlet : Servlet is normal Java program which help to create dynamic web page on server side.**

**API (Application programming interface).**

**import javax.servlet.\*;**

**servlet : servlet is a package which contains set of classes and interfaces.**

**import javax.servlet.Servlet;**

**Servlet : servlet is a interface which contains few abstract method ie 5 methods.**

**class MyServlet implements Servlet {**

**we have to override all five methods.**

**init**

**service**

**destroy life cycle methods.**

**getServletInfo**

**getServletConfig**

**}**

**GenericServlet : it is a type of abstract class which internally implements Servlet interface and override all methods except service methods.**

**class MyServlet extends GenericServlet {**

**we have to override only service methods.**

**}**

**HttpServlet : it is a type of abstract class which internally extends GenericServlet. It orverride service method also and It provided extra method in the form of doXXX like doGet and doPost.**

**class MyServlet extends HttpServlet {**

**service or doGet or doPost**

**}**

**doGet and doPost not life cycle method they wrap service methods.**

**To run the server side technologies we require server.**

**Server is a engine responsible to execute jee application.**

**2 types of server**

1. **Web server : apache 🡪 tomcat**

**We can run only servlet and jsp**

1. **Application server: weg logic, jboss etc.**

**Servlet, jsp and EJB.**

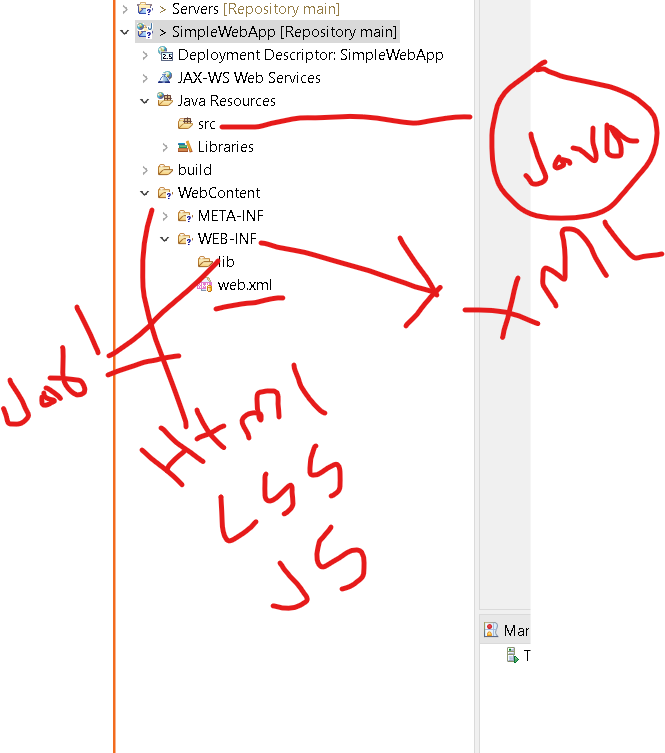
**Application server provide extra features.**

**Connection pooling, thread management, security etc.**

**Inside server we will get container(container also known as engine or run time environment for web application). Container is part of server responsible to execute servlet, jsp and ejb program. Inside Servlet, jsp and EJB no main method. container is responsible to load the classes, creating object, calling life cycle method, maintain the life the object.**

**If server is type of web server it contains only one type of container ie web container.**

**If server is type of application server it contains different types of container ie web container, ejb container, jms container etc.**

****

**protocol://hostname:portNumber/projectName/urlPattern**

[**http://localhost:8080/SimpleWebApp/Demo**](http://localhost:8080/SimpleWebApp/Demo)

**11-03-2022**

**Create table in database as login**

**With emailId and password (email id is primary key).**

**create table login(**

**emailid varchar(25) primary key,**

**password varchar(25));**

**insert into login values('raj@gmail.com','123');**

**select \* from login**

**Every form contains action and method attribute.**

**Action attribute use to send the data.**

**Method attribute contains value as get or post**

**By default it consider as get method. if method is get it will send the data through URL in form of query param concept. Like URL?key=value&key=value.**

**If you want security then you have to use method is post. If method is post then data will send through body. Which we can’t see.**

**But performance wise get is faster than post.**

**If form method is get then it will call doGet method of Servlet.**

**If form method is post then it will call doPost method of Servlet.**

**RequestDispatcher : it is a interface which provide set of methods which help to navigate from one page to another page base upon the condition.**

**Syntax to create the reference of RequestDispatcher**

**RequestDispather rd = request.getRequestDispatcher(“path”);**

**If target page is servlet then path must be target page url pattern.**

**If target page is html or jsp then path must be target pageName.html or pageName.jsp.**

**rd.include(request,reponse);**

**//source page content + target page content display as one page**

**or**

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

**// we will get only target page output.**

<servlet>

<description></description>

<display-name>Demo</display-name>

<servlet-name>A</servlet-name>

<servlet-class>com.Demo</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>A</servlet-name>

<url-pattern>/Hi</url-pattern>

</servlet-mapping>

**These two tag use to map the request.**

**If url end with URL pattern ie Demo it can be any name(Hi). then container check the ServletName part of servlet mapping tag and servlet name part of servlet tag must be match.Then container load the com.Demo and after loaded successfully it will create the object servlet. Then it will call the life cycle methods. Init() it will call only once. Service it will call again and again whenever client send the request and destroy it will call at the last if no one accessing that application.**

**If we create another servlet program IDE create another servlet and servlet-mapping tags.**

**Number of servlet equal to number of servlet and servlet-mapping tags.**