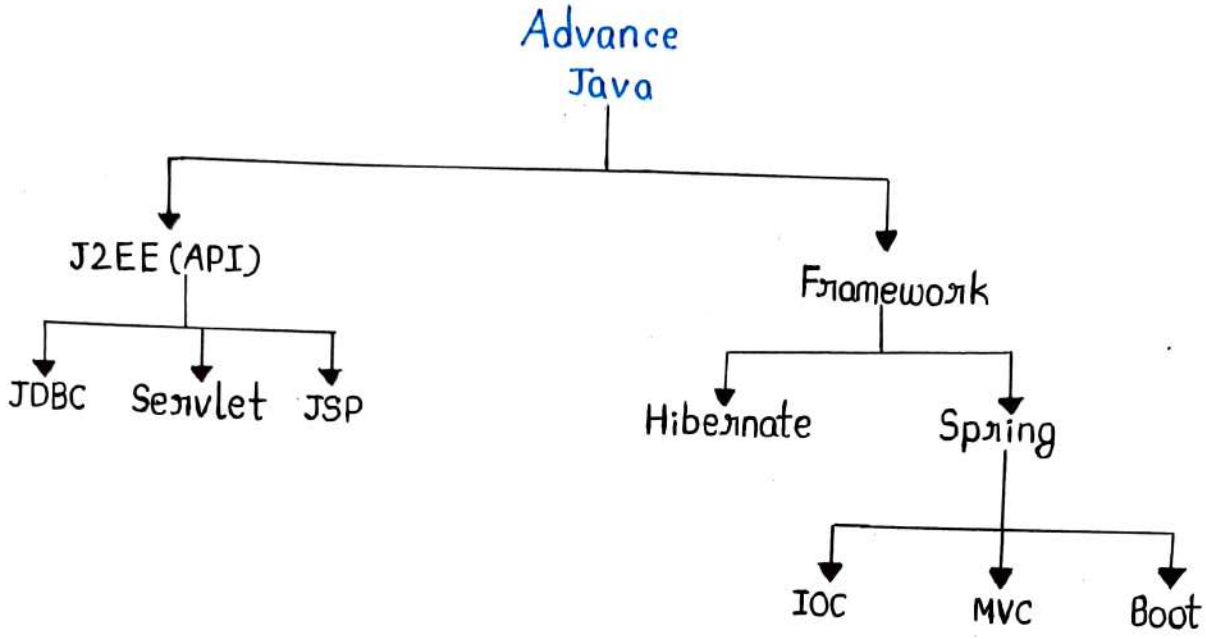
கிருஷ்ணா

கே.நவீன் குமார்

Advance Java



Project class (Boot + JS)
Advance JDBC
API Testing
Spring Data JPA
Spring JDBC

Advance Java

- What is JDBC?

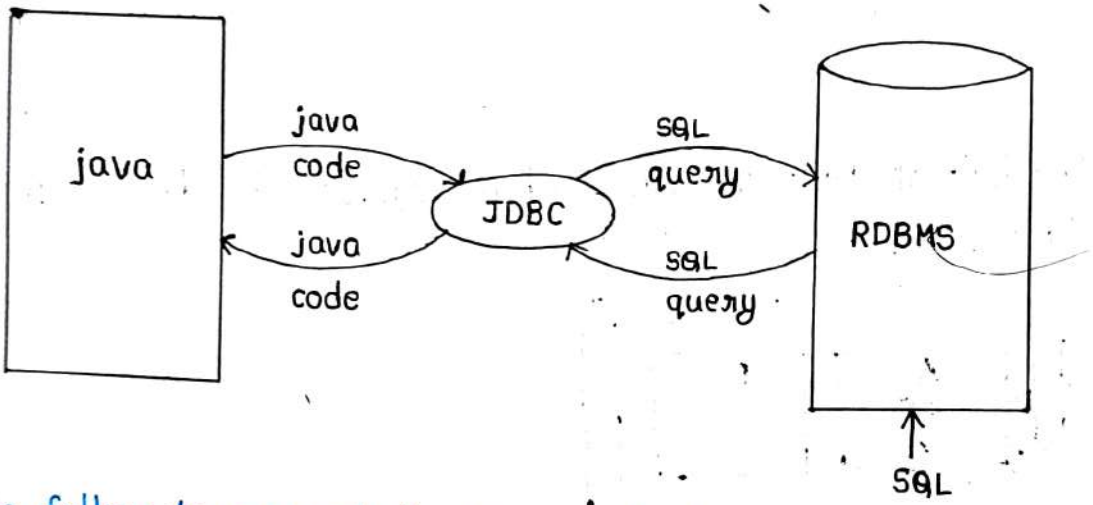
→ It is an Application programming interface (API) technology.

→ JDBC means Java Database Connectivity.

→ JDBC is used to connect java application with RDBMS software.

→ JDBC was implicitly installed into JDK from the JDK version 1.2.

→ So with the help of JDBC, we can perform CRUD operations.



- Steps to follow to connect the java application with RDBMS :-

(i) Load or register the drivers.

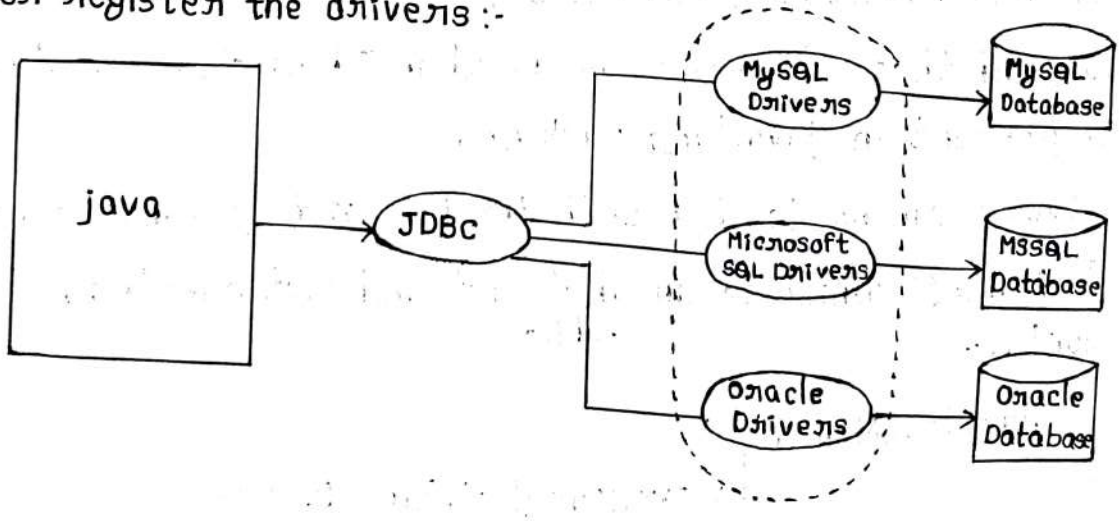
(ii) Establish the connection.

(iii) Establish the statements.

(iv) Execute the statements.

(v) close the connection.

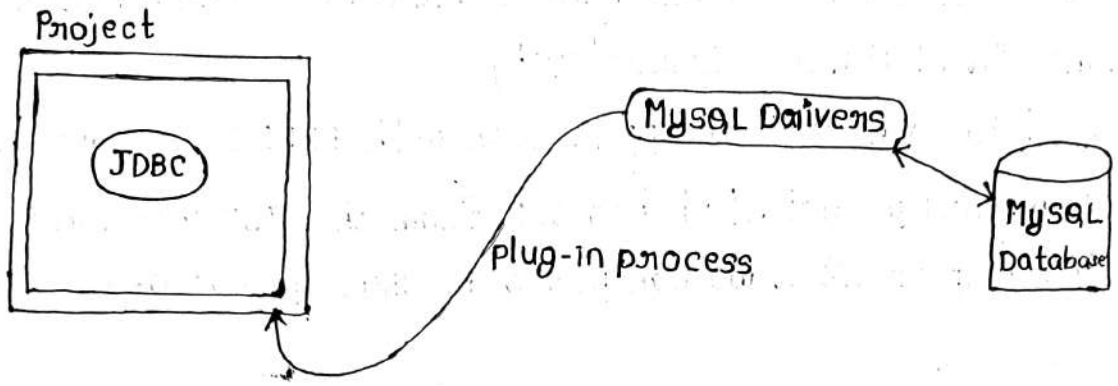
(i) Load or register the drivers :-



mvn repository.

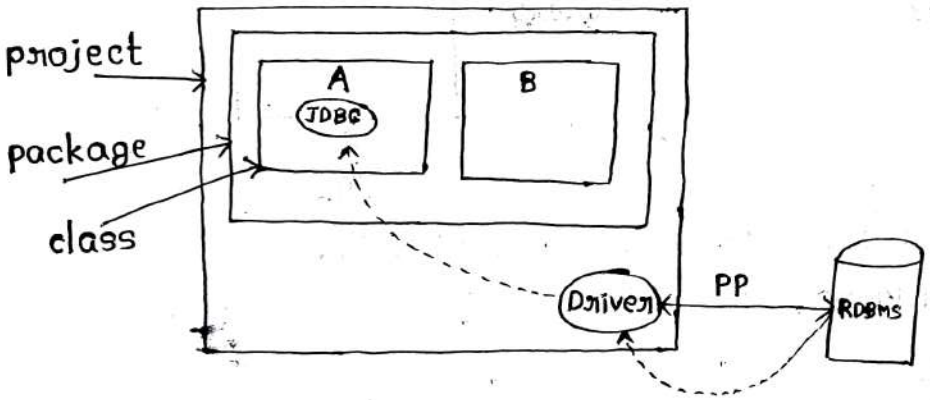
o Plug-in process :-

→ With the help of plug-in process, we can add the jar files into the project.



•step1 :-

→ With the help of step1 we can load or register the drivers into the particular java class files.



→ We can load or register the drivers into particular classes by two ways :

① Load Drivers :- With the help of load drivers we can load the drivers into particular class files.

- To load these drivers, we need an Identifier i.e., Class which is present in java.lang package.
- Drivers can be loaded with the help of helper method and i.e., forName() which is an abstract and static method.

syntax:- public

```
Class.forName("  ");
              ↳ com.Mysql.cj.jdbc.Driver
```

→ Here the helper method takes input as a fully qualified driver class name in the form of stringvalue.

Note :- We should follow the below format inorder to create the packages.

Ex:- com.Mysql.cj.jdbc.Driver

PT.CN.M.SM.C/I

where PT means ProjectType
CN means CompanyName
M means Module
SM means Sub-module
C/I means Class/Interface

(ii) Register Drivers :-

- With the help of register drivers, we can register the drivers into particular class files.
- To register the drivers we have to use registerDriver() method from DriverManager class which is present java.sql package and the registerDrive() method always throws SQLException.

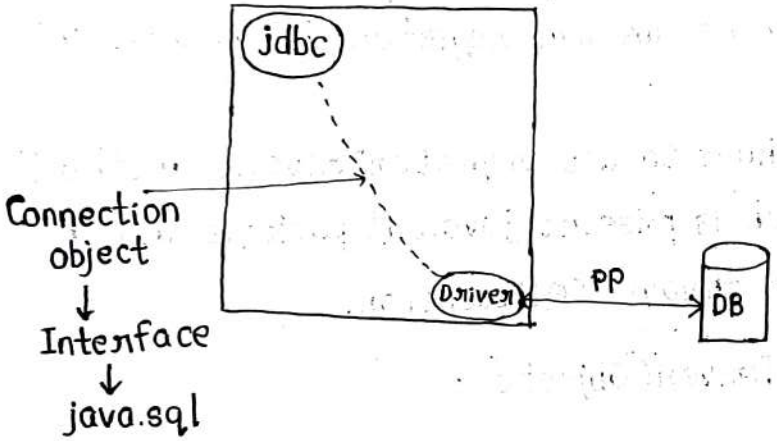
Ex:- DriverManager.registerDriver("Objecte");
↑ ClassName ↑ Helper method (public and static method)

Ex:-

```
import java.sql.DriverManager;
import java.lang.*;
import com.mysql.cj.jdbc.Driver;
public class Step2{
    public static void main(String[] args){
        try{
            // different ways to create objects of Driver class
            // Driver driver = new Driver();
            // ********** or **********
            // DriverManager.registerDriver(driver);
```

```
            //**********************
            // DriverManager.registerDriver(new Driver());
            //**********************
            DriverManager.registerDriver(new com.mysql.cj.jdbc.Driver());
        }
        catch(SQLException e){
            e.printStackTrace();
        }
    }
}
```

step2: Establish the connection



→ In step.2, we will be creating the connection between jdbc and the Drivers using Connection object.

→ Connection is an interface which is present in java.sql package helps in creating the Connection object.

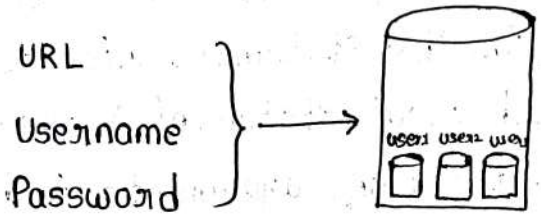
→ Interface objects will be created by using two ways i.e., upcasted reference type and other one is through helper methods.

→ We can create Connection objects in three ways with the help of three helper methods, which are present in DriverManager class and inside the java.sql package.

Connection c=
- → DM.getConnection(String1, String2, String3);
- → DM.getConnection(String);
- → DM.getConnection(String, Properties p);

→ And all these three helper methods throws SQLException, so that a suitable try and catch blocks will helps in handling those checked exceptions.

→ Inorder to access the database, we should pass the valid url, username and password. If any one of the parameter is invalid, we cannot access the database.



For example,

(i) Connection c = DriverManager.getConnection(String, String, String);
URL Username Password

where URL = jdbc : mysql : // localhost : 3306
api database
ip address → port number
Username = root
Password = root

(ii) Connection c = DriverManager.getConnection(String);
url, username, password

- Here we are performing URL rewriting, so that we can pass all the three parameters in the form of single statement.
- URL can be rewritten by the following syntax:

URL ? Key = Value & Key = Value .....

jdbc : mysql : // localhost : 3306 ? user = root & password = root

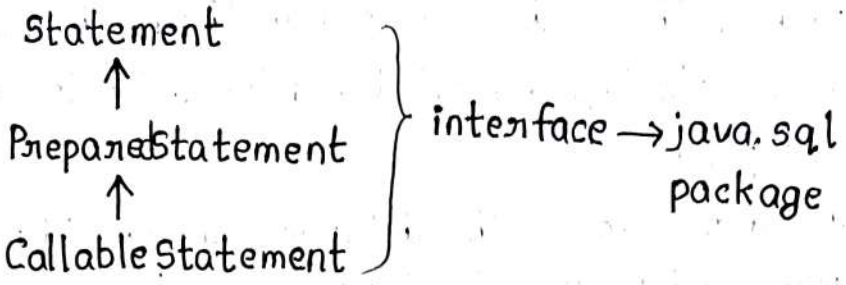
URL (protocol) Username Password

step3: Establish the statements

→ Using third step, we can transfer the data from java application to particular rdbms database in the form of statement objects.

→ There are three types of statements and all the statements are interfaces present in java.sql package.

- Statement
- Prepare statement which is the child interface of Statement.
- Callable statement which is the child interface of Prepare Statement.



• Statement :-

→ Statement is an interface present in java.sql package. Inorder to create the object for this Statement interface we have to use a helper method called createStatement(), which is a public and non-static method without any arguments. And this createStatement is present in Connection interface.

→ And this createStatement() method always throws SQLException.

syntax :-

```
Statement s = connection.createStatement();
```

→ And the statement object can be accessed with the help of Object reference variable of Connection interface.

Ex :-

```
import java.sql.Connection;
public class CreateStatement{
    public static void main(String[] args){
        try{
            //Step-1: Register the drivers
            DriverManager.registerDriver(new Driver());
            //Step-2: Establish the Connection
            Connection connection = DriverManager.getConnection("jdbc:
            mysql://localhost:3306?user=root&password=root");
            //Step-3: Establish the Statement
            Statement st = connection.createStatement();
```

className — registerDriver(new Driver()) — helper method

helper method — Interface — Object reference variable

interface — object reference of Connection interface — helper method without any input arguments

```
            S.o.pln(st);
        }
        catch(SQLException e){
            e.printStackTrace();
        }
    }
}
```

O/P:-

```
com.mysql.cj.jdbc.StatementImpl@15a34df2
```

Note:- If any error or missing arguments are passed inside the url, then we will get an Exception called CommunicationsException or it shows Communications link failure.

Step-4:- Execute the Statement

→ With the help of step4, we will execute the Statement type objects using below three methods.

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

→ All the above three methods are public and non-static methods. And all the methods are present in java.sql package. We have the overloaded methods of Statement interface and PrepareStatement interface.

(i) execute(" "); / execute();

(ii) executeUpdate(" "); / executeUpdate();

(iii) executeQuery(" "); / executeQuery();

String input

no arguments passed

String input

Statement interface

PrepareStatement interface

Ex:- statement.execute("//statements");

object reference of Statement interface

Step-5: Close the connection

→ We have to perform step5 inorder to keep the data as secured.

→ We can close the connection using close() method which is a public and non-static method which is present in Connection interface.

→ We can call the close() method using the Connection interface reference variable.

Ex:-

connection.close();

object reference of Connection interface

- Types of Queries:-

→ We have two types of queries

ⓘ Static Query

ⓘⓘ Dynamic Query

ⓘ Static Query:-

→ These are the queries that a user cannot pass the value, only the developer can pass the inputs to the queries.

Ex: Select

Create

ⓘⓘ Dynamic Query:-

→ These are the queries where the user can pass the input values to the query.

Ex: insert

delete

update

select based condition.

→ Inorder to provide the user input values to the query, we have to use PrepareStatement() method which is present inside the Preparestatement interface. As PreparedStatement is the sub-interface of Statement, we can access the properties of Statement interface.

Ex:- PreparedStatement prepareStatement = connection.prepareStatement("insert into table_name values(?,?,?)");
prepareStatement.execute();
connection.close();

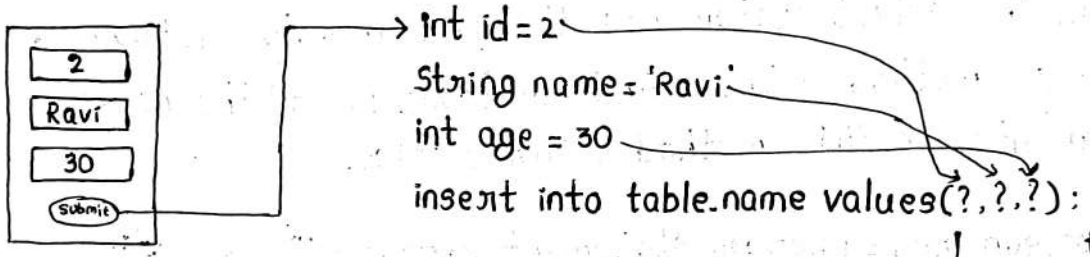
↑ single input i.e., sql statement.

° PreparedStatement :-

→ PreparedStatement is an interface which is present in java.sql package. with the PreparedStatement we can pass the dynamic query from java application to sql database.

→ Place holders (?) just acts like a container, stores the user given values.

Ex:-



table_name

id	name	age
1	Raju	50
3	Siri	25
2	Ravi	30

→ Inorder to pass the user provided inputs to the database, we have to use setter methods which are public and non-static methods. These set methods takes two inputs, the first input is an integer which represents the column number and the second input of set() method will be the type of the set() method (or) operation type of the set() method.

Ex:- setInt(column_number, int type value);
int value

→ Based on the operation, we have different types of set() methods:

- setInt(col_num, int value);
- setShort(col_num, short value);
- setDouble(col_num, double value);
- setLong(col_num, long value);
- setString(col_num, String value);
- setFloat(col_num, float value);
- setBoolean(col_num, boolean value);

→ So, while we are passing the inputs through set() methods, we should follow below syntax.

```
preparestatement.setInt(1,10);
```

preparestatement: object reference of PreparedStatement interface

1: column number of table

Note:-

→ execute() method of Statement interface always returns boolean type values.

→ Based upon the statements passed to the execute() method, it will returns either true or false.

→ For all the select statements, the boolean value returned by the execute() method is true. And similarly for all the non-select statements, execute method returns false.

Ex:-

```
boolean res = preparestatement.execute("select *");
S.o.pln(res); // true
boolean res1 = preparestatement.execute("insert...");
S.o.pln(res1); // false will be returned to insert, delete and update statements.
```

→ The limitation of execute() method is it doesn't clearly specify whether the table is affected or not by the input query. It always returns false to the execute() method for a non-select statement even if the input parameters are not present in the database.

→ To overcome the above limitation of execute() method, we use the ExecuteUpdate() method. If the input query inside the preparestatement method affected the table, then it returns an integer value other than 0, otherwise it returns 0.

→ If we pass the query in step3, then executeUpdate() method doesn't take query as input. But for the statement object in step3, we have to provide input query for executeUpdate() method in step4.

o DriverManager.getConnection(String, Properties P) :-

→ This getConnection(String, Properties P) is a helper method present inside the Connection interface and comes under java.sql package.

→ To the String input, url is passed for example "jdbc:mysql://local host:3306". And to the second input parameter is Properties object reference. The below syntax is used to create Properties object.

```
Properties pt = new Properties();
```

→ Create a new file, using FileInputStream and provide the properties i.e., username and password and save the file with extension .properties.

→ Now all the properties of FileInputStream ~~all~~ are loaded into the object of Properties type using load() method.

→ load() method is used to fetch the properties of FileInputStream as key and value pairs.

```
FileInputStream fis = new FileInputStream("asd.properties");
pt.load(fis);
```

Ex:-

```
import java.io.FileInputStream;
public class GetConnection1{
    public static void main(String[] args){
        try{
            Properties pt = new Properties();
            FileInputStream fis = new FileInputStream("asd.properties");
            pt.load(fis);
            DriverManager.registerDriver(new Driver());
            Connection connection = DriverManager.getConnection("jdbc:
            mysql://localhost:3306", pt);
            S.o.pln(connection);
        }
```

```
        catch(SQLException e){
            e.printStackTrace();
        }
        catch(FileNotFoundException e){
            e.printStackTrace();
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

asd.properties
user = root
password = root

o/p :-

```
com.mysql.cj.jdbc.ConnectionImp@5c80cf32
```

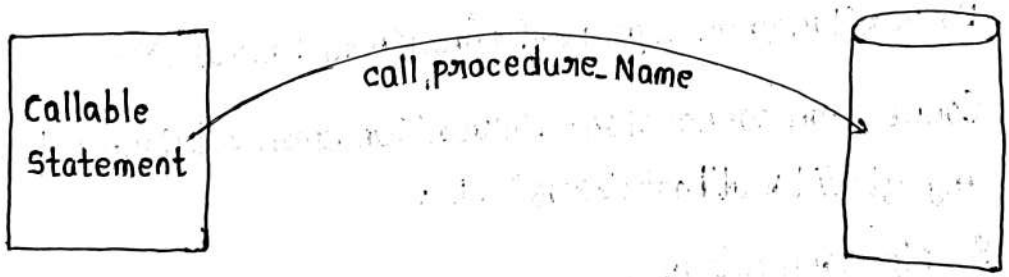
- **CallableStatement :-**

⟶ It acts like a container and is used to call the stored procedures and functions.

⟶ It is the child interface of PreparedStatement interface, present in java.sql package and inside the Connection interface.

⟶ Inorder to create the object of CallableStatement type, we need to use a helper method called prepareCall() method.

⟶ prepareCall() method is a public and non-static method present in the Connection interface and always returns CallableStatement type object.



prepareCall("call procedure_Name");

Ex:-

```
import java.sql.Connection;
public class CallStatEx{
    static int id=4;
    static String name="Nancy";
    public static void main(String[] args){
        try{
            DriverManager.registerDriver(new Driver());
            Connection connection = DriverManager.getConnection("jdbc:
            mysql://localhost:3306/jdbc_stored_procedures?user=root
            &password=root");
            CallableStatement callableStatement = (CallableStatement)
            connection.prepareCall("call last(?,?)");
            callableStatement.setInt(1,id);
            callableStatement.setString(2,name);
            callableStatement.execute();
            connection.close();
            S.o.pln("Data saved...");
        }
        catch(SQLException e){
            e.printStackTrace();
        }
    }
}
```

procedure_Name

O/P:-

Data Saved...

Database table:-

Table_Name = user

id	name
1	Dustin
2	Jane
3	Mike
4	Nancy

- Limitation of execute() and executeUpdate() method :-

→ The execute() method always return the boolean values instead of data for both select and non-select ~~statements~~ queries.

→ executeUpdate() method executes only select queries but not the non-select queries.

→ So to overcome these limitations of execute(), executeUpdate() methods we make use of executeQuery() method.

- executeQuery() method :-

→ executeQuery() method is a public and non-static method used to the (execute only) select queries.

→ It is used for (all) Statement, PreparedStatement and CallableStatement interfaces and always returns ReturnSet object. And this ReturnSet is an interface present in java.sql package.

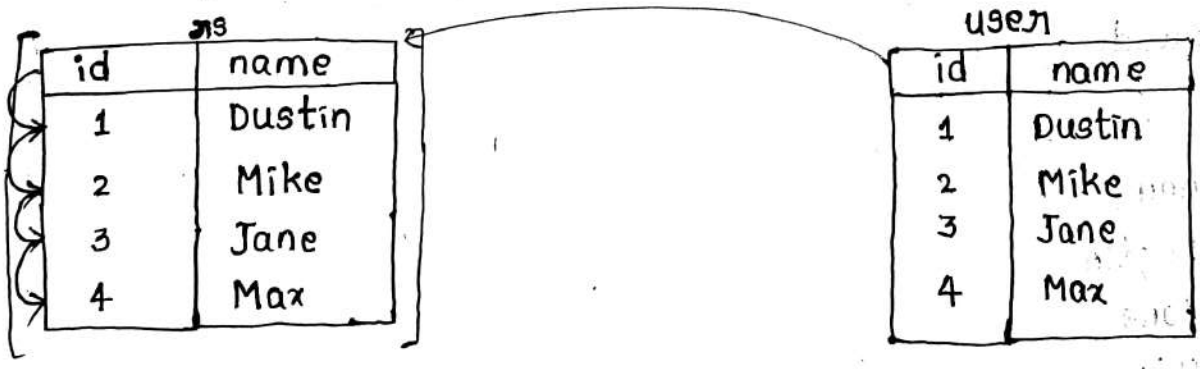
→ executeQuery() always throws SQLException.

Ex: statement.executeQuery("select * from user");

object reference of Statement type

ResultSet rs = statement.executeQuery("select * from user");

The object reference of ResultSet will store all the rows/objects of Table in the form of a temporary table. So (for) all the changes in the user table, the temporary table only changes if we again used the executeQuery() method again.



Ex: ResultSet rs = statement.executeQuery("select * from user");

```
while(rs.next()){
    S.o.p(rs.getInt(1)+" "+rs.getString("name")+" ");
    S.o.pln();
}
```

o/p :-

1 Dustin
2 Mike
3 Jane
4 Max

→ The next() method checks for the object or entry of the table, if there is an entry then the statements inside the loop will get executed and next() method pointer points to the immediate entry of the table.

- Performing JDBC_CRUD operations using layers of Spring Framework :-

→ The main disadvantage of jdbc driver is it will be complex, if it is used in large projects as it only provides the connection, everything else we need to do ourselves.

→ In spring data JDBC, we have different layers and it also provides a class called JDBC template which is used to handle the database-related logic as its methods helps to write the SQL queries directly, so it saves a lot of work and time.

→ These are the four layers present in Spring JDBC:

(i) DTO (Data Transfer Object) Layer.
(ii) DAO (Data Access Object) layer.
(iii) Service Layer.
(iv) Controller Layer.

(i) Data Transfer Object (DTO) layer:-

→ Data Transfer Object is used to transfer the data between different layers of the application.

→ The business class is present in this layer, all the properties of business class are made as private, so to initialize them we make use of getter() and setter() methods or by using constructors.

→ This business class is also called as POJO (Plain Object Java Object) class or bean class.

(ii) Data Access Object (DAO) layer:-

→ This data access object layer is responsible for encapsulating the bean class logic to interact with the database.

→ This dao layer provides methods to perform CRUD operations (Create, Read, Update and Delete) on the data.

→ DAO layer interacts with the database using frameworks such as JDBC, JPA (Java Persistence API) and other tools of ORM.

→ All the five steps of jdbc connection are written inside a class of the DAO layer.

(iii) Service Layer:-

→ The service layer acts as an intermediatary between the controller layer and the data access object layer.

→ This service layer encapsulates the business logic of the application. It contains methods or operations that represent the different functionalities or use cases of the application.

→ Service layer may also used to implement multiple interfaces with common logic and data.

→ Establishes a set of available operations and co-ordinates the application's response in each operation.

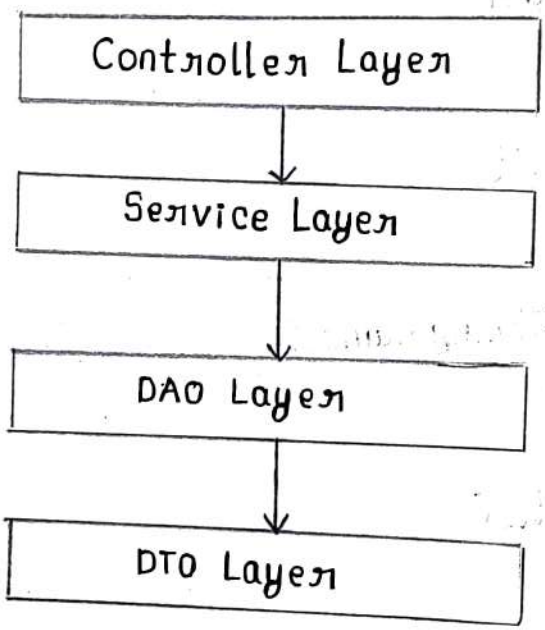
(iv) Controller Layer :-

→ Controller layer acts as an user interface for managing the communication and interaction with the database.

→ This controller layer manages the establishment and termination of connections to the database.

→ It provides the methods to establish a connection, authenticate with the database, and close the connection once it is no longer needed.

→ And the main() method is present only in the controller layer. The flow of execution from one layer to another is as follows:



Example :-

Write java codes to perform jdbc crud operations such as insert, update, delete etc.

① DTO Layer :-

```
package com.jsp.dto;
public class Student{
    private int id;
    private String name;
    private String email;
    private int age;
    private long phno;
```

```
public Student(int id, String name, String email, int age, long phno){
    super();
    this.id=id;
    this.name=name;
    this.email=email;
    this.age=age;
    this.phno=phno;
}
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 String getEmail(){
    return email;
}
public void setEmail(String email){
    this.email=email;
}
public int getAge(){
    this.age= age;
    return
}
public void setAge(int age){
    this.age=age;
}
```

```
    public void setPhno(long phno){
        this.phno = phno;
    }
    public long getPhno(){
        return phno;
    }
    public Student(){
        super();
    }
}
```

(ii) DAO Layer :-

```
package com.jsp.dao;
import java.sql.Connection;
public class StudentDao{
    public void runStudent(Student student){
        try{
            DriverManager.registerDriver(new Driver());
            Connection connection = DriverManager.getConnection("jdbc:
            mysql://localhost:3306/jdbc_crud?user=root&password
            =root");
            PreparedStatement preparedstatement = connection.prepare
            Statement("insert into student values(?,?,?,?,?)");
            preparedStatement.setInt(1,student.getId());
            preparedStatement.setString(2,student.getName());
            preparedStatement.setString(3,student.getEmail());
            preparedstatement.setInt(4,student.getAge());
            preparedStatement.setLong(5,student.getPhno());
            preparedstatement.execute();
            connection.close();
        }
```

```
    catch (SQLException e){
        e.printStackTrace();
    }
}
public void updateStudent(Student stu){
    try{
        DriverManager.registerDriver(new Driver());
        Connection connection = DriverManager.getConnection("jdbc:
        mysql://localhost:3306/jdbc_crud?user=root&password=
        root");
        PreparedStatement preparedStatement = connection.prepare-
        Statement("update student set name=? where id=?");
        preparedStatement.setString(1,stu.getName());
        preparedStatement.setInt(2,stu.getId());
        preparedStatement.execute();
        connection.close();
    }
    catch(SQLException e){
        e.printStackTrace();
    }
}
public void deleteStudent(Student stu){
    try{
        DriverManager.registerDriver(new Driver());
        Connection connection = DriverManager.getConnection("jdbc:
        mysql://localhost:3306/jdbc_crud?user=root&password=
        root");
        PreparedStatement preparedStatement = connection.prepare-
        Statement("delete from student where id=?");
        preparedStatement.setInt(1,stu.getId());
        preparedStatement.execute();
        connection.close();
    }
```

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

(iii) Service Layer :-

```
package com.jsp.service;
import com.jsp.dao.*;
public class StudentService{
    StudentDao sd = new StudentDao();
    public void saveStudent(Student stu){
        S.o.pln("Student Service Layer...");
        sd.runStudent(stu);
    }
    public void updateStudent(Student stu){
        S.o.pln("Student Service Layer...");
        sd.updateStudent(stu);
    }
    public void deleteStudent(Student stu){
        S.o.pln("Student Service Layer...");
        sd.deleteStudent(stu);
    }
}
```

(iv) Controller Layer :-

```
package com.jsp.controller;
import java.util.Scanner;
public class StudentController{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        S.o.pln("Enter your id");
        int id = sc.nextInt();
```

```
        S.o.pln("Enter your name");
        String name = sc.next();
        S.o.pln("Enter your email");
        String email= sc.next();
        S.o.pln("Enter your age");
        int age = sc.nextInt();
        S.o.pln("Enter your phno");
        long phno = sc.nextLong();
        Student s = new Student(id, name, email, age, phno);
        StudentService ss = new StudentService();
        ss.saveStudent(s);
    }
}
```

```
package com.jsp.controller;
import java.util.Scanner;
public class StudentUpdateController{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        S.o.pln("Enter the id");
        int id = sc.nextInt();
        S.o.pln("Enter the name");
        String name = sc.next();
        Student s = new Student();
        s.setId(id);
        s.setName(name);
        StudentService ss = new StudentService();
        ss.updateStudent(s);
        S.o.pln("Data updated...");
    }
}
```

```
package com.jsp.controller;
import java.util.Scanner;
public class StudentDeleteController{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        S.o.pln("Enter the id");
        int id = sc.nextInt();
        Student s = new Student();
        s.setId(id);
        StudentService ss = new StudentService();
        ss.deleteStudent(s);
        S.o.pln("Data deleted...");
    }
}
```

Disadvantages of JDBC :-

→ Developer has to write lot of code, in case the backend database changes, the application developer may need to purchase and deploy a new type driver specific to the new database.

→ It is not good for large projects.

→ It requires database specific queries. If database change we need to change all SQL statements of queries.

→ Opening, handling and closing the database connections is not always easy. User may face many errors while doing these operations.

→ Performance is not good when having multiple connections.

→ Handling exceptions with JDBC is a big issue. You need to write multiple nested try-catch blocks.

→ Transactions and concurrency must be hand-coded.

Hibernate :-

→ Hibernate is database-independent java framework that simplifies the development of Java application to interact with the database.

→ It is open source, lightweight, ORM (Object Relational Mapping) tool.

→ Hibernate implements the specifications of JPA (Java Persistence API) for data persistence. JPA is a java specification that provides certain functionality and standard to ORM tools. JPA classes and other interfaces are present in javax.persistence package.

→ Hibernate reduces lines of code by maintaining object-table mapping and returns result to the application in the form of Java objects.

→ Hibernate is more-efficient and object-oriented approach for accessing a database.

Advantages of Hibernate :-

→ Hibernate automatically generates the queries.

→ It is flexible and powerful ORM to map java classes to database tables.

→ It reduces the development time and maintenance cost.

→ Makes an application portable to all SQL databases.

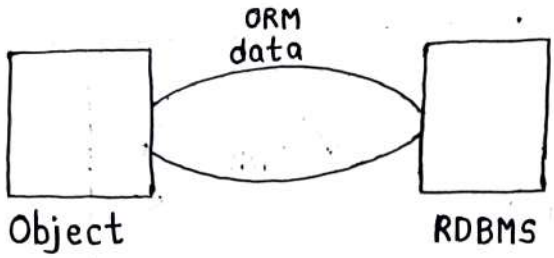
→ Handles all create-read-update-delete (CRUD) operations using simple API.

→ Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.

→ Hibernate provides a powerful query language Hibernate Query Language - HQL (independent from type of database).

→ Fast performance.

→ Automatic table creation.



javax.persistence
- EMF (EntityManagerFactory)
- EM (EntityManager)
- EntityTransaction(ET)

What is an ORM framework?

Object Relational Mapping (ORM) is a programming technique which is used to convert data from relational database and object-oriented programming.

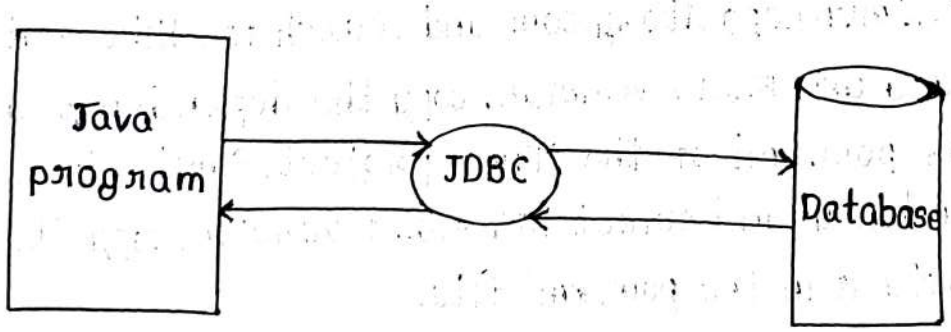
JPA :-

→ JPA stands for Java Persistence API.

→ Through JPA the developer can map, store, update and retrieve data from relational databases to Java ~~projects~~ objects and vice versa.

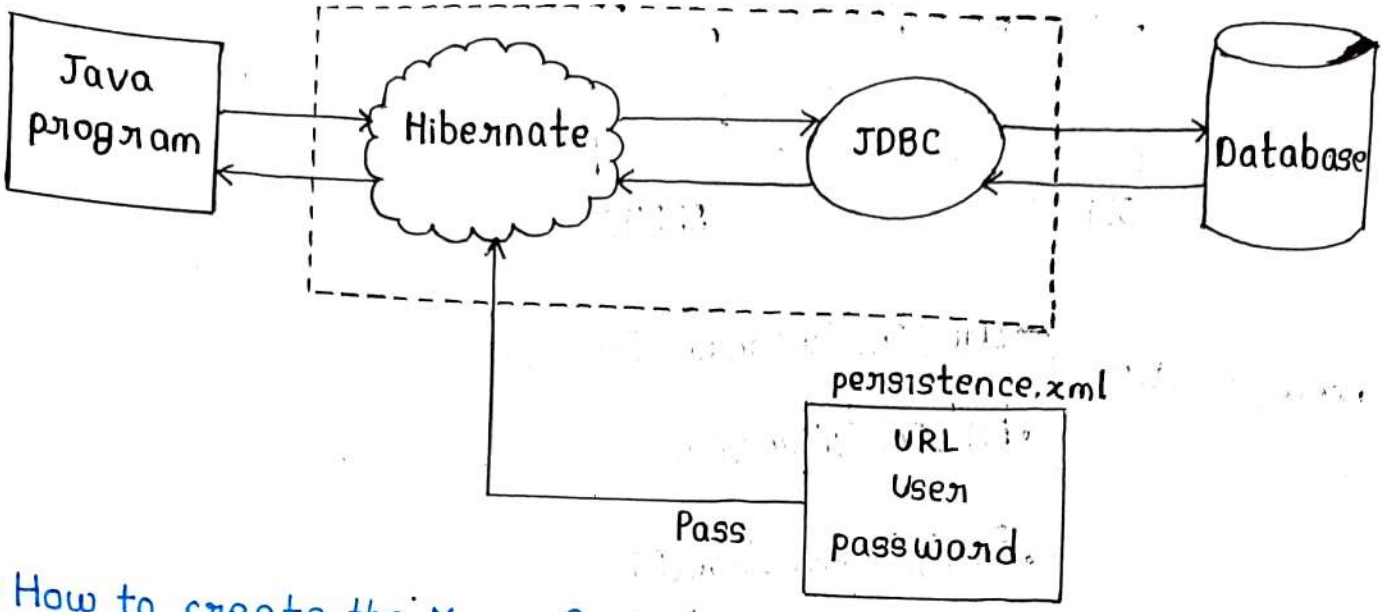
→ JPA can be used in Java-Enterprise Edition and Java-Standard Edition applications.

Before using the hibernate, the data flow is:



But in hibernate, the data flow changes. As if we are using a normal java project then we are using plugin process.

If we are using maven project then we are using dependency tags.



- How to create the Maven Project?

→ Click on the Project Explorer.
→ Ctrl+N
→ In the wizards type Maven.
→ Select the Maven project and click the next.
→ Check the checkbox for create a simple project.
→ Click the next button
→ Group Id: give your basic package name.
Artifact Id: give your Project name.
→ Click the finish button. After some time Maven project will be built in the project explorer.
→ Go to https://mvnrepository.com and search for Hibernate Core Relocation and use 5.6.10 version, copy the dependency and paste it inside the pom.xml of the Maven project. Similarly search for MySql dependency and select for 8.0.31 version, copy the dependency and paste it in the pom.xml file.
→ Persistence file:-
It is a file with the extension of .xml; which is used to store the configurable data.

Steps to create persistence.xml file:

- Right click

- Select Resource file, press ctrl+N.
- Create a folder with name as META-INF.
- Inside the META-INF create a file named persistence with an extension of .xml.
- Go to the www.github.com/Prashi974386 and click on supporting file and then select persistence.xml, copy the code and paste it inside the created persistence.xml file of META-INF folder.

o What is Entity Class?

→ A JPA entity class is a POJO class, i.e., an ordinary Java class that is annotated as @Entity, makes the class to represent objects in the database.

→ @Entity annotation makes the particular class as an entity bean.

Requirements for Entity class:-

- The class must be annotated with the javax.persistence.Entity annotation.
- The class must have a public or protected, no-argument constructor.
- The class and members of those class must not be declared final.
- All the data members should be prefixed with private access modifier and all the data members having setter and getter methods.

→ @Id is used to make the data member as a primary key.

→ @GeneratedValue() :- It is used to automatically set the value to primary key and it takes two parameter Strategy and generator.

If you want to perform CRUD operations with the help of Hibernate we use three elements:

- EMF (EntityManagerFactory)
- EM (EntityManager)
- ET (EntityTransaction)

- **EntityManagerFactory (EMF):-**

→ It is an interface present in javax.persistence package, which is used to create connection between java application to Database and in this level only Load or Register Driver is done.

```
EntityManagerFactory entityManagerFactory = Persistence.
createEntityManagerFactory("persistence filename.xml");
```

→ Persistence is a helper class and it is present in javax.persistence package and createEntityManagerFactory() is a method present inside the Persistence class.

→ Through this createEntityManagerFactory only we can create an object of the EntityManagerFactory.

→ Name of the persistence file is passed as input to the createEntityManagerFactory() method.

→ EntityManagerFactory is used to provide an EntityManager.

- **EntityManager :-**

→ EntityManager is an interface and it is present inside javax.persistence package which is used to perform a CRUD operation.

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

reference of the EntityManagerFactory

→ entityManagerFactory.createEntityManager() returns object of EntityManager.

→ EntityManager is used to provide an EntityTransaction.

- **EntityTransaction :-**

→ EntityTransaction is an interface and it is present inside the javax.persistence package which is used to manage the transaction.

→ With help of the EntityManager we get EntityTransaction object.

```
EntityTransaction entityTransaction = entityManager.getTransaction();
```

reference of the EntityManager

→ When we are calling Non-select query method on that EntityTransaction object is mandatory.

- Inbuilt methods in Hibernate :-

(i) persist(Object entity) :-

It is used to save the object into the database.

(ii) remove(Object entity) :-

It is used to remove the record from database based on the primary key of that database.

(iii) find(class<T> entityclass, Object entity) :-

It is fetching the record from the database based on the primary key of that database.

(iv) merge(T entity) :-

Merge method is used in two ways. when the record found with the help of find(), then object is existing that time merge method works as update the object else it acts like save new object to the database.

(v) createQuery(String s) :-

With the help of this method we can write our own query and it returns Query interface object.

Query is an interface present in javax.persistence package.

Note :-

→ EntityTransaction interface consists of some methods such as :

* void commit() :- It is used to commit the current resource transaction.
* void begin() :- It is used to start a resource transaction.

- Query Parameter :-

→ These are the way to build and execute the parameterized queries.

→ With the help of this Query Parameter, we can pass our own dynamic query into the Hibernate.

→ Similar to JDBC prepared statement parameters, JPA specifies two different ways to write parameterized queries by using:

- PPQP (Positional Parameter Query Parameter)
- NPQP (Named Parameter Query Parameter).

- Positional Parameters :-

With the help of this Positional Parameter Query Parameter we can pass our own dynamic query into Hibernate with the help of position we are passing values for query. We declare these parameters within the query by typing a question mark, followed by a positive integer number.

Ex:-

```
Query query = entityManager.createQuery("select a from User
where a.email = ?1 and a.password = ?2");
query.setParameter(1, values);
```

- Named Parameters :-

With the help of this Named Parameter Query Parameter we can pass our own dynamic query into Hibernate with the help of key name we can pass values for query. We declare these parameters within the query by typing a semicolon, followed by a reference name.

Ex :-

```
Query query = entityManager.createQuery("select a from User
where a.email = :email and a.password = :password");
query.setParameter("email", values);
```

Example on Hibernate :-

```
package com.jsp.dto;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class User{ //Entity class
```

```
    @Id
    private int id;
    private String name;
    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;
    }
}
```

Test Program :-

```
package com.jsp.demo;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
public class Demo1{
    public static void main(String[] args){
        User user = new User();
        user.setId(3);
        user.setName("Naveen");
        //user
        EntityManagerFactory entityManagerFactory = Persistence.create
        EntityManagerFactory("god");
        EntityManager entityManager = entityManagerFactory.create
        EntityManager();
```

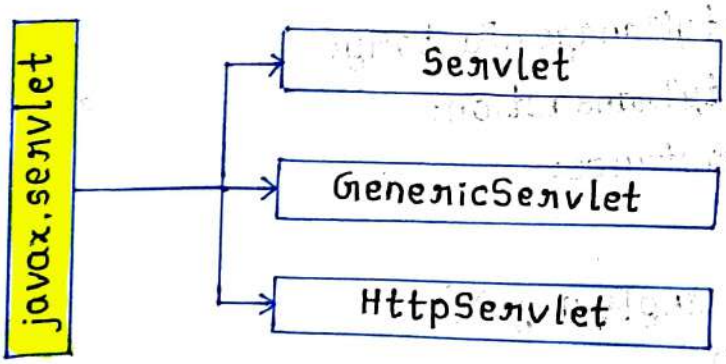
```
        EntityTransaction entityTransaction = entityManager.getTransac-
        tion();
        entityTransaction.begin();
        entityManager.persist(user);
    // entityManager.merge(user);                    );
    // entityManager.remove(user);
        entityTransaction.commit();
        S.o.pln("Data Saved");
    }
}
```

Servlet :-

→ Servlet is a piece of program and is one of the API, with the help of servlet we can run the server.

→ With the help of servlet we can handle the request obtained from the webserver, process the request, produce the response, then sends a response back to the webserver.

Servlet Hierarchy :-



- Abstract methods of Servlet interface

(i) init() :- initialize the servlet objects and this is life cycle method of servlet and it is invoked by the web container only once.

(ii) service() :- provides the response for the incoming request and it is invoked at each request by web container.

(iii) destroy() :- It is invoked only once and it indicates that servlet is being destroyed.

(iv) getServletConfig() :- It returns the object of ServletConfig.

(v) getServletInfo() :- It returns information about servlet such as writer, copyright, version etc.

- **Deployment Descripton :-**

→ Deployment Descripton is the file used by the servlet container to map particular URL and particular classes.

→ It is a web.xml file and it is present inside a WEB-INF folder.

Ex:- `@WebServlet("/hello");`

annotation, deployment descripton

- **GenericServlet :-**

→ It is an abstract class and it implements Servlet, Serializable interface, it is present in javax.servlet.GenericServlet package.

→ GenericServlet is used to create a servlet.

→ It is protocol-independent as GenericServlet class can handle any type of request.

→ We can create Servlet by inheriting GenericServlet class. And GenericServlet has one abstract method, i.e.,

public void service(ServletRequest req, ServletResponse res);

For this above abstract method, we can provide implementation through inheritence.

- **HttpServlet :-**

→ It is an abstract class and it extends the GenericServlet class which is used to create a Servlet and it is present in javax.servlet. HttpServlet package.

→ It is the child class of GenericServlet class that handles an Http request and provides an Http response usually in the form of HTML page.

→ And it doesn't have any abstract methods, because the implementation of key methods have to be provided by a custom servlet class. At present everyone are using HTTP servlet.

Some important methods of HttpServlet :-

- void doGet(HttpServletRequest req, HttpServletResponse res)
- void doPost(HttpServletRequest req, HttpServletResponse res)
- void doPut(HttpServletRequest req, HttpServletResponse res)
- void doDelete(HttpServletRequest req, HttpServletResponse res)
- void doOptions(HttpServletRequest req, HttpServletResponse res)
- void service(ServletRequest req, ServletResponse res)
- void service(HttpServletRequest req, HttpServletResponse res)

Difference between doGet() and doPost()

doGet()	doPost()
i) It is a default method for any Http request.	i) It is not a default request.
ii) Parameter is not encrypted.	ii) Parameter is encrypted here.
iii) doGet() method triggers by itself.	iii) doPost() cannot be triggered by itself.
iv) It is not suitable for Encrypt operations.	iv) It is suitable for encrypt operation.
v) Not suitable for file uploading operation.	v) It is suitable for file uploading operation.
vi) We can send maximum size of 240 bytes data.	vi) We can send unlimited data.
vii) With the help of this doGET() method we can pass video, audio and image etc.	vii) With the help of doPost() method we can pass video, audio and image etc.

o PrintWriter :-

PrintWriter is a class that extends Writer class, used to print the objects in the webpage rather than printing in the console.

Ex:-

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException{
    PrintWriter printWriter = res.getWriter();
    printWriter.write("<html><body><h1>Hi</h1></body></html>");
}
```

- RequestDispatcher:-

→ RequestDispatcher is an interface and it is providing the facility of dispatching the request to another resource.

→ RequestDispatcher is present in javax.servlet package used to transfer the request from one page to another page.

methods of RequestDispatcher:-

- void forward(ServletRequest req, ServletResponse res)
- void include(ServletRequest req, ServletResponse res)
- void forward(ServletRequest req, ServletResponse res):-
  Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
- void include(ServletRequest req, ServletResponse res):-
  Includes the content of a resource (servlet, JSP page, or HTML file page) in the response. It is used to return the response with a message.

→ calling html to html/servlet class - use anchor tag.

→ calling html/form to html page/servlet - action attribute

→ servlet to html page - RequestDispatcher

→ servlet to servlet - RequestDispatcher.

```
<form action="login" method="post">
    Email: <input type="email" name="email"><br><br>
    Password: <input type="password" name="password"><br><br>
    <input type="submit" value="login">
</form>
```

Note :-

→ The doGet() and doPost() methods of HttpServlet class automatically triggers and calls the service() methods.

→ The Get() and Post() methods of Javascript are also used retrieve or send data to a server.

Steps to create static web pages :-

(i) Select resources and click Ctrl+N.

(ii) Create a folder named META-INF.

(iii) Click Ctrl+N on META-INF and create an xml file named persistence.

(iv) Go to www.github.com/Prashi974386 and select persistence.xml file of supporting files, copy the code and paste inside the created xml file.

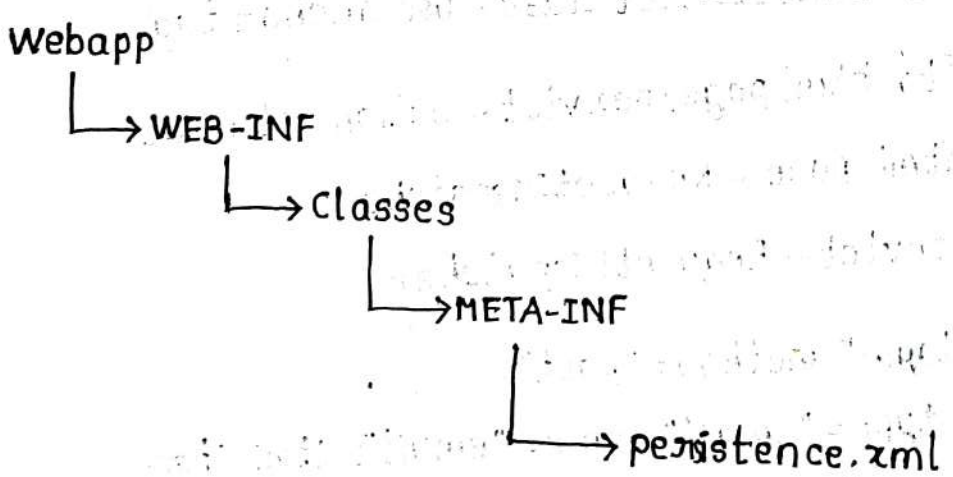
Steps to create dynamic web pages :-

(i) Go to webapp of src folder.

(ii) Open WEB-INF folder and create a new folder with a name "Classes", inside Classes folder create another new folder with a name META-INF

(iii) Create a new xml file inside META-INF and name it as persistence.xml.

(iv) Go to www.github.com/Prashi974386, open supporting files and select persistence.xml, copy the code and paste it inside the created xml file.



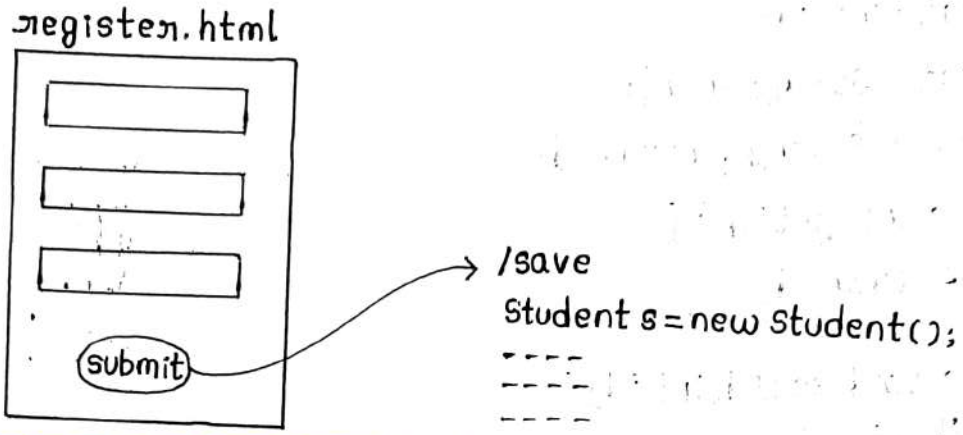
Ex:-

```
@Entity
public class Student{
    @Id
    private int id;
    private String email;
    private String password;
    public int getId(){
        return id;
    }
    public void setId(int id){
        this.id = id;
    }
    public String getEmail(){
        return email;
    }
    public void setEmail(String email){
        this.email = email;
    }
    public String getPassword(){
        return password;
    }
    public void setPassword(String password){
        this.password = password;
    }
}
```

- Select webapp
- press ctrl+N
- Create a new html file with name as register.html

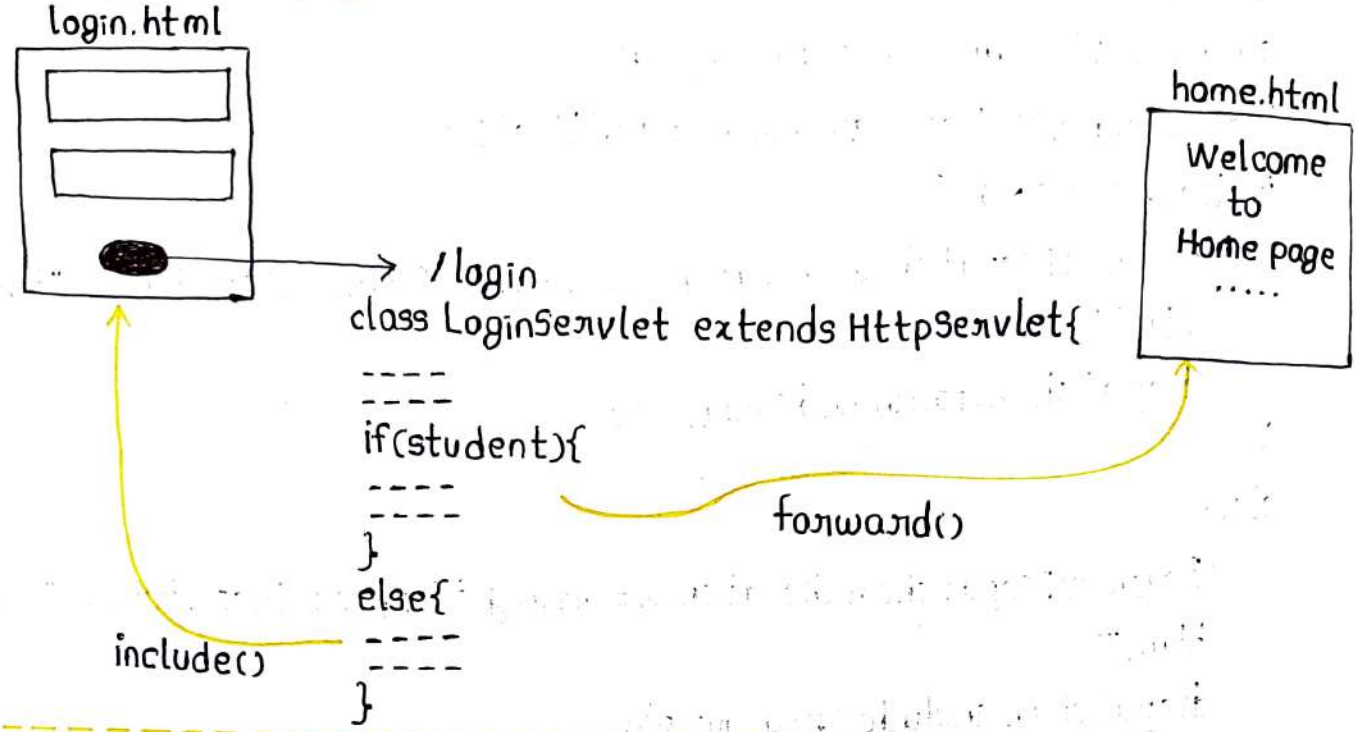
```
<body>
    <form action="save">
        ID: <input type="text" name="id"><br><br>
        Email: <input type="email" name="email"><br><br>
```

```
Password : <input type="password" name="password"><br><br>
<input type = "submit" value = "Register">
</form>
```




```
@WebServlet("/save")
public class SaveStudent extends HttpServlet{
    protected void doPost( HttpServletRequest req, HttpServletResp-
    onse res) throws ServletException{
        int id = Integer.parseInt(req.getParameter("id"));
        String email = req.getParameter("email");
        String password = req.getParameter("password");
        Student s = new Student();
        s.setId(id);
        s.setEmail(email);
        S.setPassword(password);
        EntityManagerFactory emf = Persistence.createEntityManager
        Factory("god");
        EntityManager entityManager = emf.createEntityManager();
        EntityTransaction et = entityManager.getTransaction();
        if(s! = null){
            et.begin();
            entityManager.persist(s);
            et.commit();
            S.o.pln("Data Saved");
        }
    }
}
```

// To open home page



// login.html

```
<form action="login" method="post">
    Email: <input type="email" name="email"><br><br>
    Password: <input type="password" name="password"><br><br>
    <input type="submit" value="login">
</form>
```

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet{
    protected void doPost(HttpServletRequest req, HttpServletRespon-
    se res) throws ServletException{
        String email=req.getParameter("email");
        String password = req.getParameter("password");
        EntityManagerFactory entityManagerFactory = Persistence.create
        EntityManagerFactory("god");
        EntityManager entityManager = entityManagerFactory.create
        EntityManager();
        Query query = entityManager.createQuery("select b from Student
        b where email=?1 and password=?2");
```

```
        query.setParameter(1, email);
        query.setParameter(2, password);
        List<Student> list = query.getResultList();
        if(list.size()>0){ // calls home.html page
            RequestDispatcher dispatcher = req.getRequestDispatcher("home.
            html");
            dispatcher.forward(req, res);
        }
        else{ // calls login.html page
            RequestDispatcher dispatcher = req.getRequestDispatcher("login.
            html");
            dispatcher.include(req, resp);
        }
    }
}
```

```
// home.html
<title> Home.html</title>
<body>
    <h1> Welcome to web page</h1>
</body>
```

What is Session Tracking?

It is a process where we can transfer objects from one page to another page.

Why we use Session Tracking?

To recognize the user and we have four session tracking techniques:

- URL Rewriting
- Hidden Form Field
- Cookies
- HttpSession

URL Rewriting :-

It is a process of appending or modifying any url structure while loading a page. Here, we append a token or identifier to the url of the next servlet or the next resource.

⟶ We can send parameter name/value pairs using the following format:

url ? name1 = value1 & name2 = value2

- A name and value are seperated using an equal sign.
- A parameter name/value pair is seperated from another parameter using ampersand (&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a servlet, we can use getParameter() method to obtain a parameter value.

Advantages of URL Rewriting :-

- Browser independent.
- Extra form submission is not required on each page.

Disadvantages of URL Rewriting :-

- It works only with the links.
- It can send only text information.

Hidden Form :-

⟶ It is an invisible text flield which is used for maintaining the states of the user.

⟶ In this type of case, we store the information in the hidden field and get it from another servlet.

syntax :

```
<input type="hidden" name="uname" value="Jspiders">
```

Here, uname is the hidden field name and Jspiders is the hidden value.

Advantage :-

- Browser Independent.

Disadvantages :-

- It is maintained at server side.
- Only text type information can be used.

Cookies :-

→ Cookies are client-side files on a localcomputer that hold user information.

(or)

→ Cookies are small pieces of information that is stored in key-value pair format to the client's browser during multiple requests.

→ Cookies are present inside the javax.servlet.http package.

→ Basically, the server treats every client as a new one so as to avoid this situation cookies are used.

Advantages:-

→ Simplest technique of maintaining the state.

→ Cookies are maintained at client side.

Disadvantages:-

→ It will not work if cookie is disabled from the browser.

→ Only text type information can be set in Cookie object.

HttpSession :-

→ It is an interface present in javax.servlet.http package used to create a session between the client and the server.

→ The session is created between an HttpClient and HttpServer by the servlet container using HttpSession interface.

→ HttpSession object is used to store the entire session with a specific object. And we can store, retrieve and remove attribute from HttpSession object.

→ Any servlet can have access to Httpsession object throughout the getSession() method of the HttpServletRequest object.

→ And the session information of HttpSession such as the properties are stored on the Web client as a cookie.

syntax to create HttpSession object:-

```
HttpSession session = req.getSession();
session.setAttribute("email", email);
```

Note:

setAttribute() is a method present in HttpServletRequest and it

takes two inputs as key and value pair.

- **JSP :-**

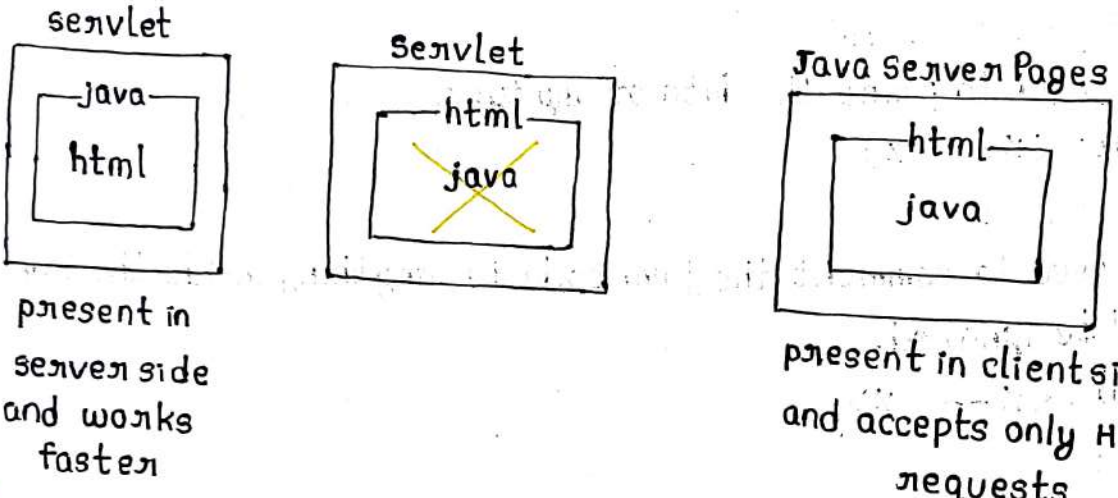
→ It stands for Java Server Pages.

→ It is a server-side technology.

→ It is used for creating web application.

→ It is used to create dynamic web content.

→ In this with the help of JSP tags are can insert java code into HTML pages.



present in server side and works faster

present in client side API. and accepts only HTTP requests.

Advantages of JSP over Servlet :-

→ Easy to maintain.

→ Recompilation or redeployment is required.

→ JSP has access to entire API of Java.

→ JSP are extended version of Servlet.

Features of JSP :-

→ Simple coding : Just adding Java code to html/xml.

→ Reduction in the length of code with the help of action tags, custom tags.

→ Connection to Database is easier.

→ We can create interactive dynamic web pages.

→ Extends to Servlet, so all the features of servlets, implicit objects and custom tags can be used in JSP.

Tags of JSP :-

There are 5 tags of JSP those helps us to write the java code.

(i) Declaration Tag :

It is used to declare variables and methods.

syntax : <%! %>

(ii) Java Scriplet Tag :

It allows us to add any number of java codes, local variables and logics.

syntax : <% %>

(iii) Java Expression Tag :-

It is used to print the data of any type.

syntax : <%= %>

(iv) Comment Tag :-

It is used to comment the java code i.e., anything inside the comment tag will be ignored.

syntax : <%-- %>

(v) taglib :

We can import the external/internal libraries.

syntax : <%@ %>

```
query.setParameter(1,email);
query.setParameter(2,password);
List<Student> list = query.getResultList();
if(list.size()>0){
    req.setAttribute("list",list.get(0));
    //calls to home.jsp page
    RequestDispatcher dispatcher = req.getRequestDispatcher("home.jsp");
    dispatcher.forward(req,res);
}
```

user (or) student (or) list

home.jsp :-

```
<body>
    <% Student student = (Student) req.getAttribute("list"); %>
    <% = student.getId() %>
    <% = student.getEmail() %>
</body>
```

→ To print many objects use the below code:

```
List<Student> list = query.getResultList();
if(list.size()>0){
    req.setAttribute("list", list);
    RequestDispatcher dispatcher = req.getRequestDispatcher("home.jsp");
    dispatcher.forward(req, res);
}
else{
    RequestDispatcher dispatcher = req.getRequestDispatcher("page.jsp");
    dispatcher.include(req, res);
}
```

home.jsp :-

```
<%
 List<Student> list = (List)<Student>) req.getAttribute("list"); %>
<% for(Student student : list){ %>
<% = student.getId() %>
<% = student.getEmail() %>
<% = student.getPassword %>
<%
 }
%>
<table border="3px" style="border-collape:collapse;">
 <tr>
    <th>Id</th>
   <th>Email</th>
   <th>Password</th>
</tr>
```

```
<%
 for(Student student : list){
%>
<tr>
    <td><% = student.getId()%></td>
    <td><% = student.getEmail()%></td>
    <td><% = student.getPassword()%></td>
</tr>
<%
 }
%>
</table>
```

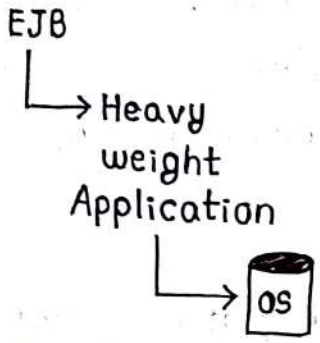
Differences between JSP and Servlet :-

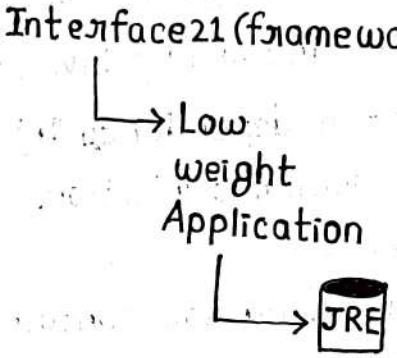
JSP	Servlet
• JSP is a HTML based code.	• Servlet is a java code.
• JSP accepts only Http requests.	• Servlet can accepts all protocol requests.
• In JSP we cannot override the service() method.	• In Servlet, we can override the service() method.
• In JSP session management is automatically enabled.	• In Servlet, the default session management is not enabled, user have to enable it explicitly.
• In JSP, inbuilt implicit objects are present.	• In Servlet, inbuilt implicit objects are not present.
• Packages can be imported into the JSP code (i.e., top, middle, bottom or client-side).	• Packages are to be imported on the top of the program.
• JSP is slower than Servlet because the first step in the JSP lifecycle is translation of JSP to java code and then compile.	• Servlet is faster than JSP.

Spring Framework:-

→ Spring is a light weight framework and it is called as framework of frameworks since based on the spring many sub-frameworks were created.

→ Interface21 framework was the old name of Spring framework. It was developed by Rod Johnson in 2003.

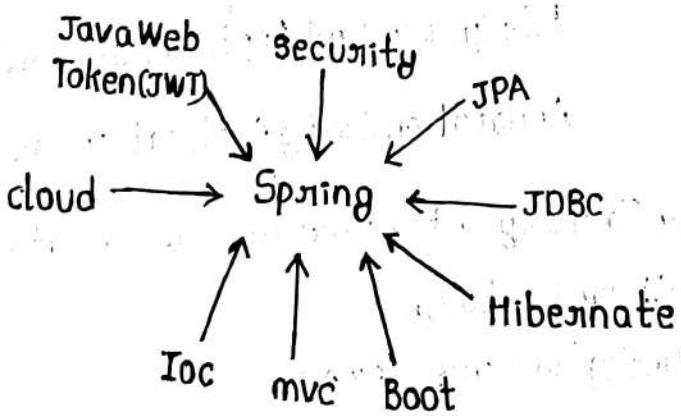




Why we go for Spring framework:-

- We have two types of Servers:-
  - (i) Web Server
  - (ii) Application Server → Web container
    → EJB container (Enterprise Java Bean)
- EJB container is used to create heavy weight application and application server.
- So through the EJB we cannot fetch the data, so we go for Spring framework.

→ The subframeworks of Spring framework are:



Advantages of Spring :-

→ We can create stand-alone applications.

→ We can create dynamic applications.

→ We can create light-weight applications.

Spring IOC :-

→ It is a framework and is used to get objects from the containers, it will only create and assign the objects for particular variable.

→ Spring Ioc means Spring Inversion of Controller

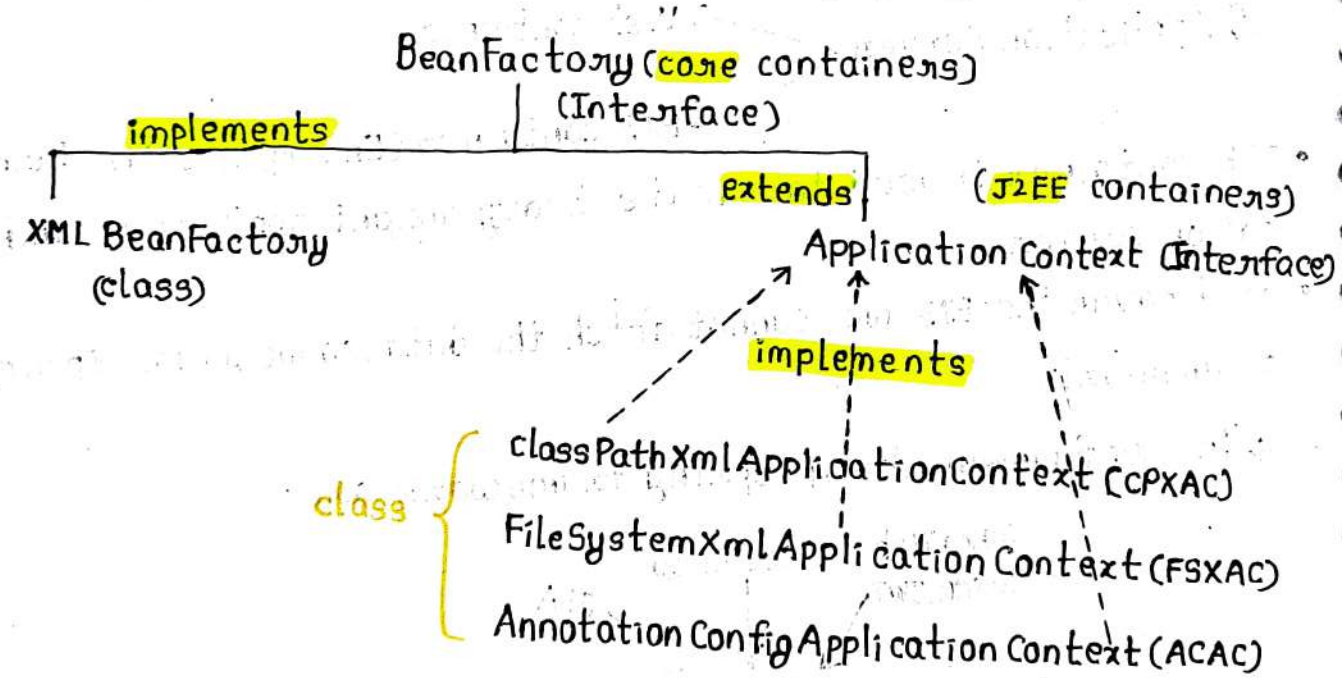
→ With the help of dependency Injection of Spring Ioc we can get the value injection or variable injection.

IOC (Inversion of Controller) :-

→ To use Ioc we need Ioc container. with the help of core container or J2EE container we can create Ioc container.

→ With the help of Bean factory object. we can create Core container.

→ BeanFactory is an interface present in org.Spring.beanpackage



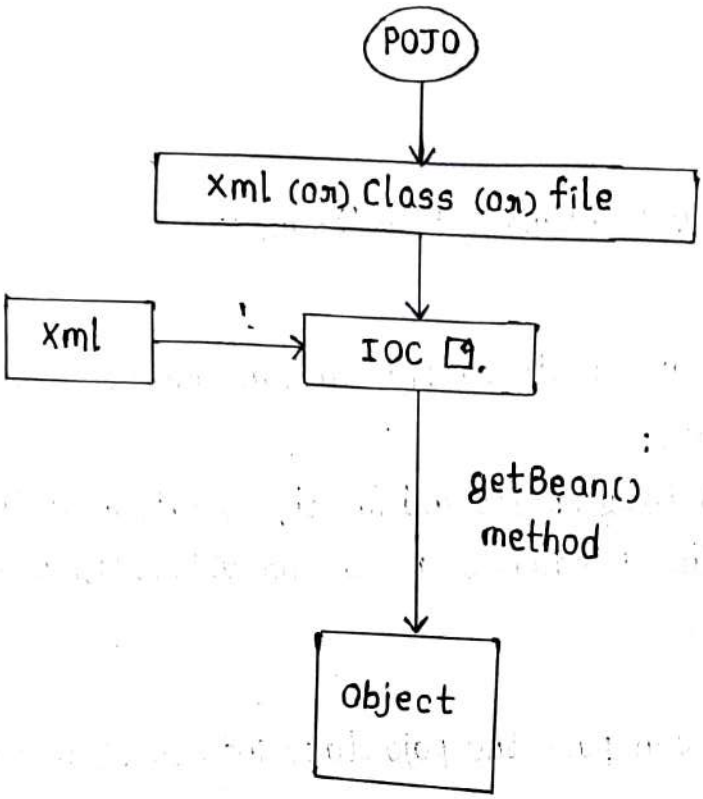
→ With the help of BeanFactory interface we can create core containers by implementing xmlBeanFactory.

→ When we have xml config we use CPXAC.

→ When we have files we use FSXAC.

→ When we have annotation config we use ACAC.

- Creation of object :-



- With the help of getBean() method, we can retrieve the object.

- Creation of Maven Project :-

i) Create a Maven project.

ii) Go to mvnrepository and search for spring context dependencies, select 5.3.18 version and copy the maven dependency.

iii) Go to maven project, open pom.xml and paste the maven dependency.

iv) Create a pojo class named as Test inside a new package.

```
public class Test{
    public void demo1(){
        S.o.pln("Hai");
    }
}
```

v) Create an xml file with a name myspring.xml, add the configurations such as bean tags.

vi) Go to www.github.com/Prashi974386, select supporting files, open springxsd.xml, copy the code and paste it inside the myspring.xml file.

myspring.xml

```
<beans>
   ----
   ----
   ----
  <bean id="test1" class="com.jsp.demo.Test"></bean>
</beans>
```

Note: With the help of beans we can create configuration connection file.

vii) Create a Core container class:

Create an XmlBeanFactory object and for the creation of the Container we need object of the Resource, Resource object is generated using ClassPathResource(" ");

Note:

With the help of bean tag we can pass the pojo class information into the xml file.

```
public class CoreContainer{
    public static void main(String[] args){
        ClassPathResource classPathResource = new ClassPathResource(
        "myspring.xml");
        BeanFactory beanFactory = new XmlBeanFactory(classPathResou-
        rce);
        Test test = (Test) beanFactory.getBean("test1");
        test.demo1();
    }
}
```

o/p :- Hai

Note :-

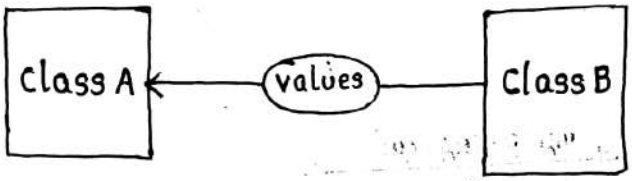
To create ApplicationContext (J2EE) container we just need to modify the Core class but same POJO class can be used.

viii) Create a J2EE container: Create a new class with a name as J2eeContainer.

```
public class J2eeContainer{
    public static void main(String[] args){
        ApplicationContext applicationContext = new classPathXmlApplicat-
        ionContext("myspring.xml");
        Test test = (Test)applicationContext.getBean("test1");
        test.demo1();
    }
}
```

- **Dependency Injection :-**

It is a process where we can inject the values from one class to another class.



We have three types of dependency injections :

ⓘ **Constructor Dependency Injection :-**

It is a process where we can inject the values through the constructors.

ⓘⓘ **Setter Dependency Injection :-**

It is a process where we can inject the values through the setter methods.

ⓘⓘⓘ **Field Dependency Injection :-**

It is a process where we can inject the values through the annotations.

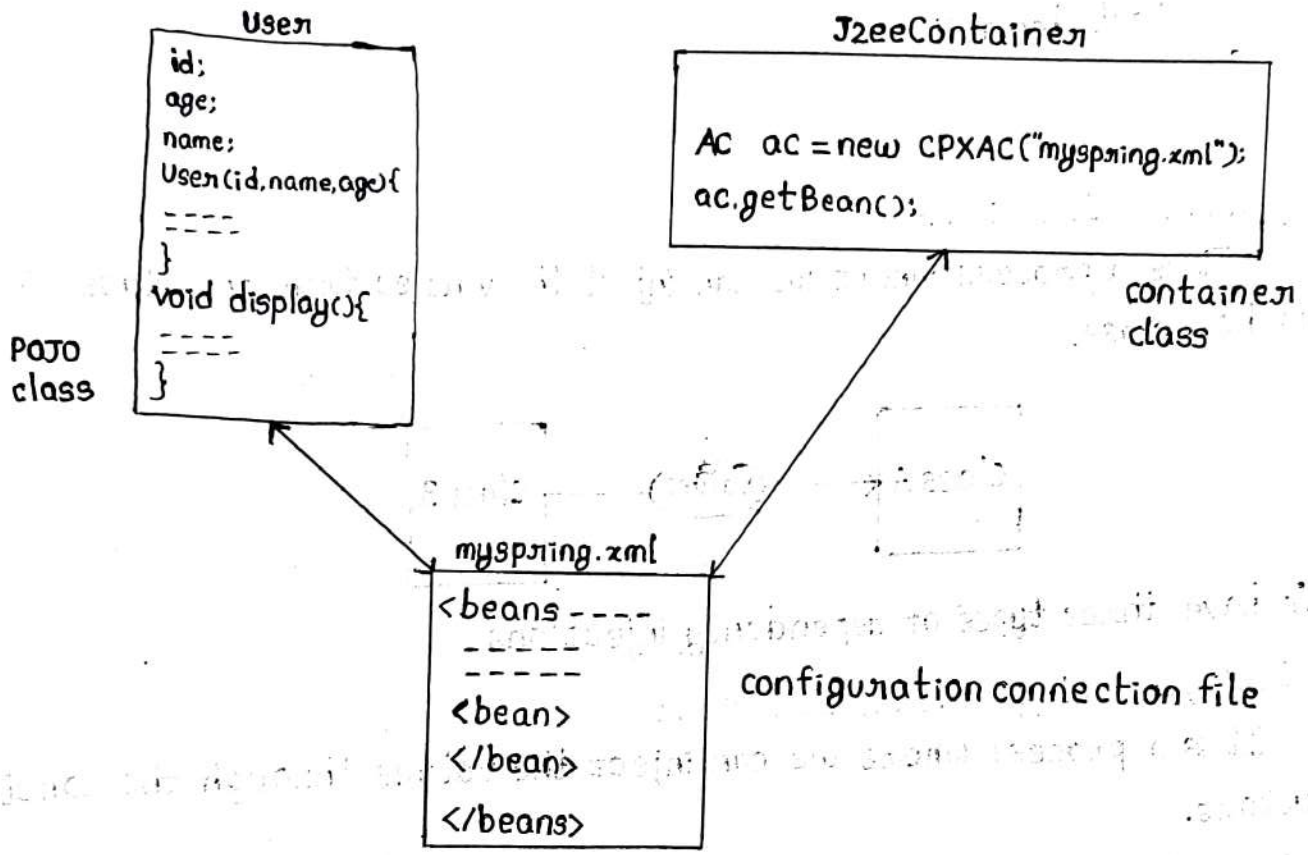
Advantages of dependency injection :-

- Improved modularity.
- Increased testability.
- Enhanced flexibility.
- Simplified maintenance
- Easily extendable.

ⓘ Constructor Dependency Injection :-

→ It is a process where we can inject the values through the constructors.

→ We need an xml file to create configuration connection between the two classes.



configuration connection file :-

```
//myspring.xml
<beans>
  <bean id="test" class="com.jsp.demo.Test"></bean>
  <bean id="user" class="com.jsp.cdi.User">
  <constructor-arg value="1"></constructor-arg>
  <constructor-arg value="Raju"></constructor-arg>
  <constructor-arg value="21"></constructor-arg>
  </bean>
</beans>
```

POJO class:-

```
public class User{
  private int id;
  private String name;
  private int age;
  public User(int id,String name,int age){
      super();
      this.id=id;
      this.name=name;
      this.age=age;
  }
  public void display(){
      S.o.pln(id);
      S.o.pln(name);
      S.o.pln(age);
  }
}
```

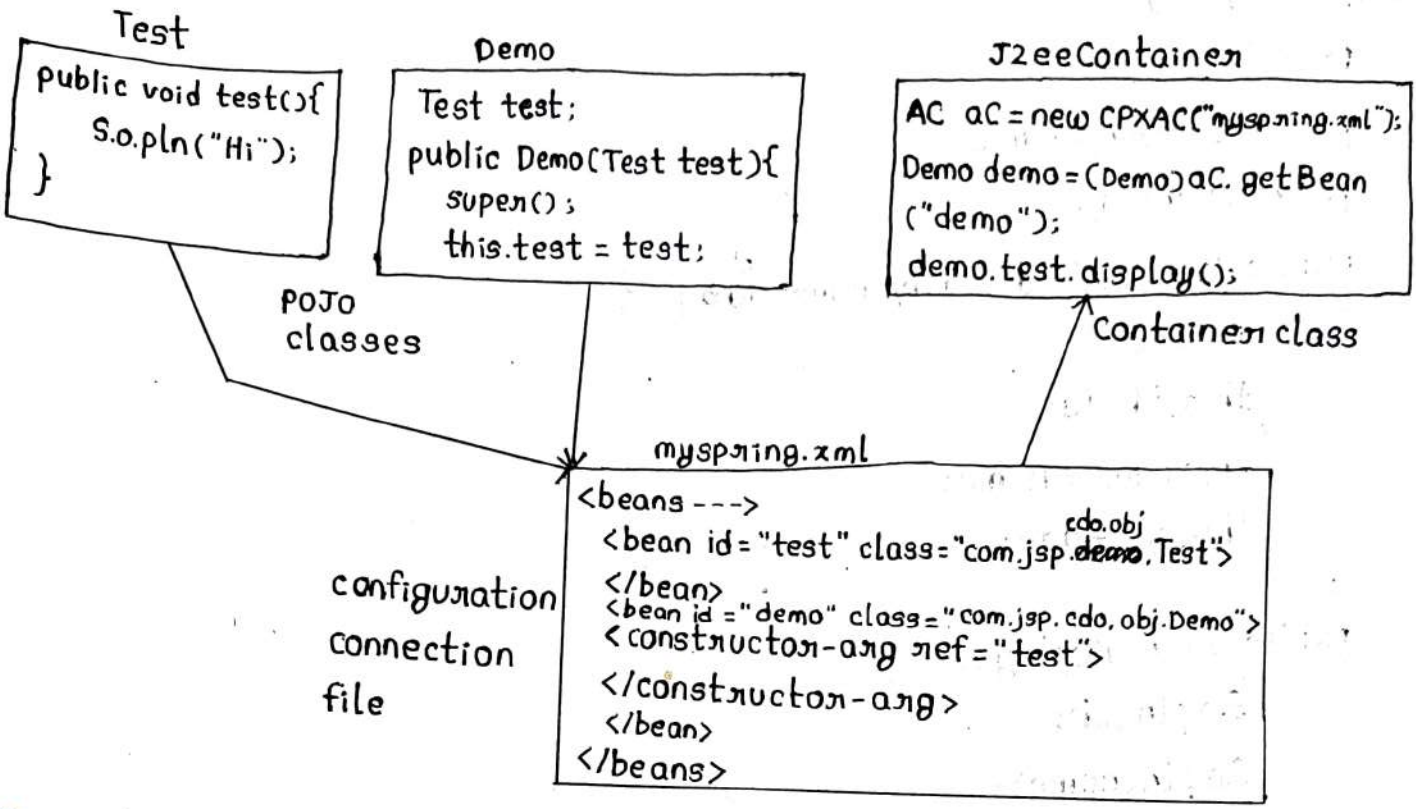
Container Class:-

```
public class J2eeContainer{
    public static void main(String[] args){
        ApplicationContext applicationContext=new ClassPathXmlApplicati-
        onContext("myspring.xml");
        User user=(User)applicationContext.getBean("user");
        user.display();
    }
}
```

O/P:-

```
1
Raju
21
```

Constructor Dependency Object Injection :-



POJO class1 :-

```
public class Test{
    public void display(){
        S.o.pln("Hello");
    }
}
```

POJO class2 :-

```
public class Demo{
    Test test;
    public Demo(Test test){
        super();
        this.test = test;
    }
}
```

configuration connection file:

```
<beans>
  <bean id="test" class="com.jsp.cdo.obj.Test"></bean>
  <bean id="demo" class="com.jsp.cdo.obj.Demo"></bean>
```

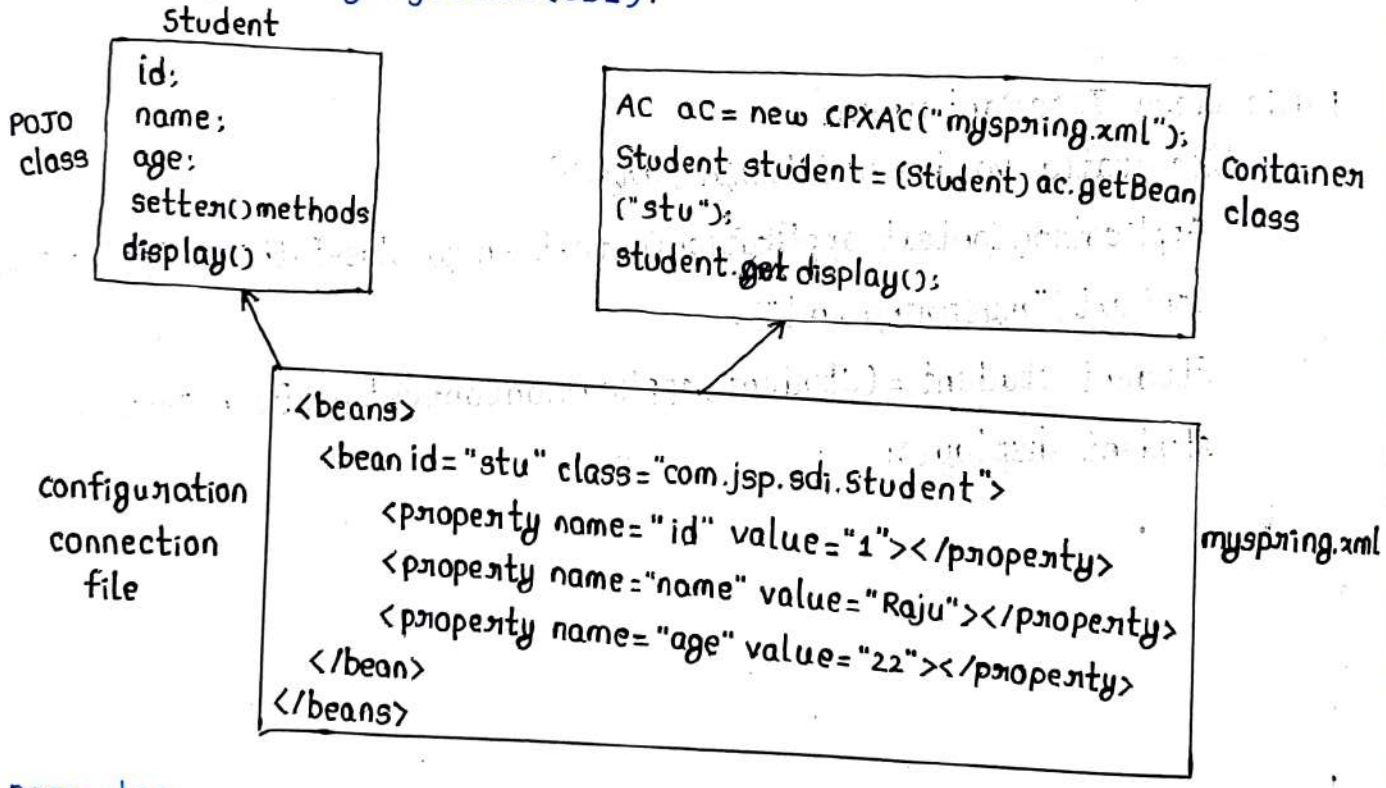
```
<constructor-arg ref="test"></constructor-arg>
</bean>
</beans>
```

Container class :-

```
public class J2eeContainer{
    public static void main(String[] args){
        ApplicationContext applicationContext = new ClassPathXmlApplication
        Context("myspring.xml");
        Demo demo = (Demo) applicationContext.getBean("demo");
        demo.test.display();
    }
}
```

o/p :- Hello

ⓘⓘ Setter Dependency Injection (SDI):-



POJO class :-

```
public class Student{
    private int age;
    private String name;
    private int id;
```

```
    Public void setId(int id){
        this.id = id;
    }
    Public void setName(String name){
        this.name = name;
    }
    Public void setAge(int age){
        this.age = age;
    }
    public void display(){
        S.o.pln(id);
        S.o.pln(name);
        S.o.pln(age);
    }
}
```

```
public int getId(){
    return id;
}
public String getName(){
    return name;
}
public int getAge(){
    return age;
}
```

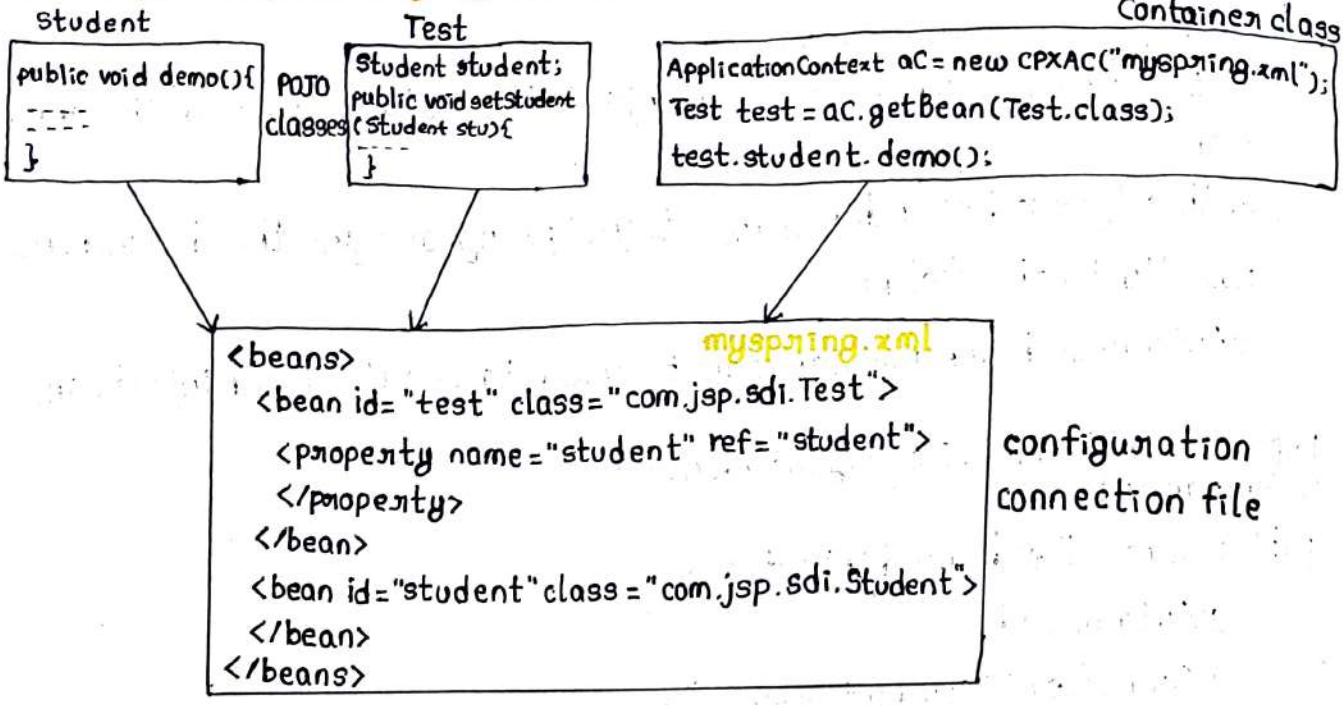
Container class :-

```
public class J2eeContainer{
    public static void main(String[] args){
        ApplicationContext applicationContext = new ClassPathXmlApplication
        Context("myspring.xml");
        Student student = (Student)applicationContext.getBean("stu");
        student.display();
    }
}
```

O/p :-

```
1
Raju
22
```

Setter Object Dependency Injection:-



POJO class1 :-

```
public class Student{
    public void demo(){
        S.o.pln(" Hello");
    }
}
```

POJO class2 :-

```
public class Test{
    Student student;
    public void setStudent(Student student){
        this.student = student;
    }
}
```

Container class :-

```
public J2eeContainerClass{
    public static void main(String[] args){
        ApplicationContext applicationContext = new classPathXmlApplication
        Context("myspring.xml");
        Test test = applicationContext.getBean(Test.class);
        test.student.demo();
    }
}
```

o/p:- Hello

(iii) Field Dependency Injection:-

→ It is a process where we can inject the values through the annotations.

→ In this type of Dependency Injection, Spring assigns the dependencies directly to the fields.

→ @Autowired annotation can be used to create Field Injection.

Ex:-

```
import com.jsp.fdi.MessageStudent;
@Component
public class FieldBasedInjection{
    @Autowired
    @Qualifier("StudentService")
    private MessageStudent messageStudent;
    public void processMsg(String message){
        messageStudent.sendMsg(message);
    }
}
```

MessageStudent

```
public interface MessageStudent{
    public void sendMsg(String message);
}
```

StudentService

```
@Service("StudentService")
public class StudentService implements MessageStudent{
    public void sendMsg(String message){
        S.o.pln(message);
    }
}
```

Configuration File

```
@Configuration
@ComponentScan("com.jsp.fdi");
public class AppConfiguration{

}
```

Container class :-

```
public class JzeeContainerClass{
    public static void main(String[] args){
        ApplicationContext applicationContext = new AnnotationConfig
        ApplicationContext(AppConfiguration.class);
        FieldBasedInjection fieldBasedInjection = applicationContext.get
        Bean(FieldBasedInjection.class);
        fieldBasedInjection.processMsg("Come to the class");
    }
}
```

o/p :-

Come to the class

Note :-

@Component

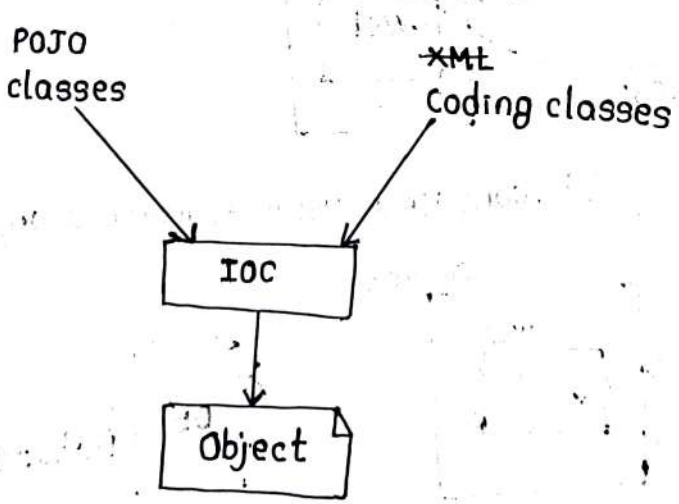
- It is a class level annotation.
- We can easily create object for the class.
- Ioc container creates an object

@Configuration

- With the help of it we can config the component class.

@ComponentScan

- With the help of it, we are configuring the base package.



@bean

- With the help of it we can store the object into the Ioc container.

@value

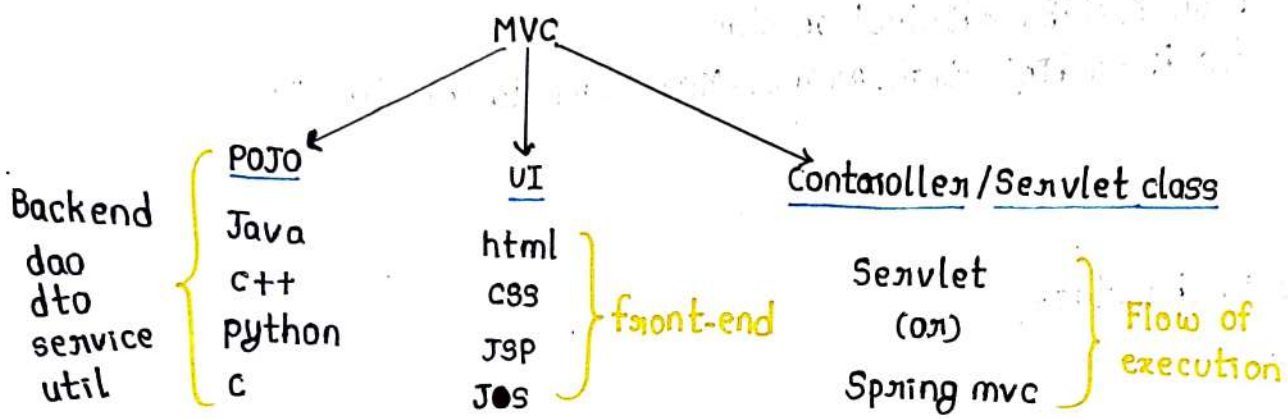
- With the help of it, we can inject the values within the same class.

@Component Autowired

- It is used to inject the object of one class to another class.

Model View Controller (MVC) :-

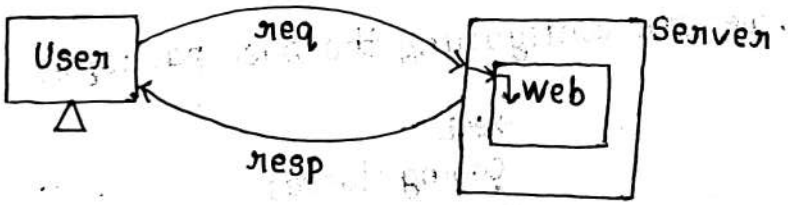
→ Spring MVC is a Model View and Controller based web framework mainly used to develop dynamic web pages and RESTful web services.



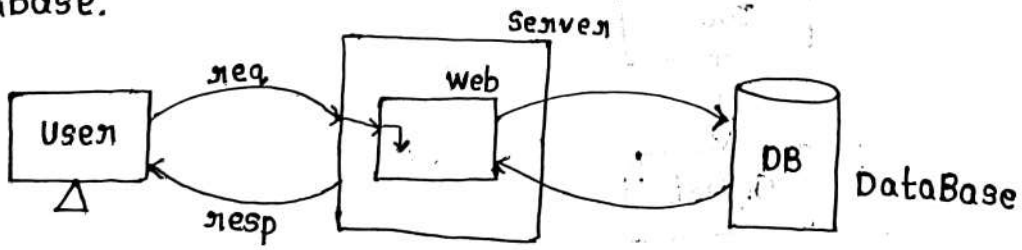
→ Spring mvc is an architecture and is used to light weight applications.
→ To overcome the limitations of spring IOC we are using Spring mvc.
→ With the help of Spring-mvc we can get dependency injection feature.

Types of architecture layers :-

(i) 2-Layer :- It is a static application with 2-layered architecture.

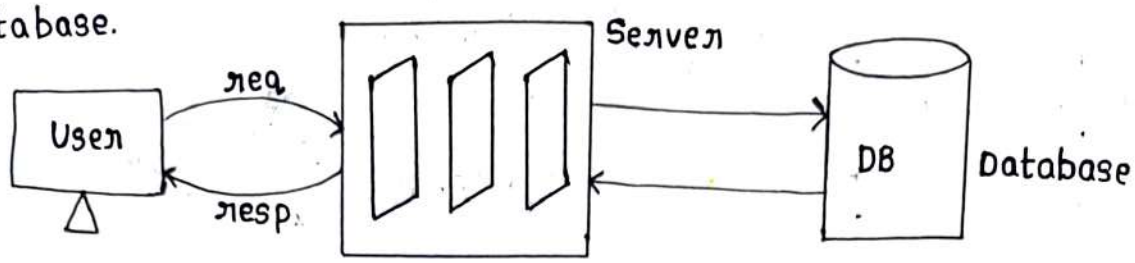


(ii) 3-Layer architecture :- It contains user, server along with the database.



Ex: Hibernate crud

(iii) N-Layer architecture :- It consists of user, server, filter and Database.



Ex: All the websites such as google.com

Why Spring mvc ?

To overcome the limitation of Servlet.

- It is built on the top of the Servlet and Spring core.
- We can use Spring IOC and servlet too.
- We can create dynamic web app, enterprise application etc.

Github
↳ Supporting files
↳ pom.xml (mvc)
- spring-web
- spring-webmvc
- servlet-api
- jsp-api
- jstl

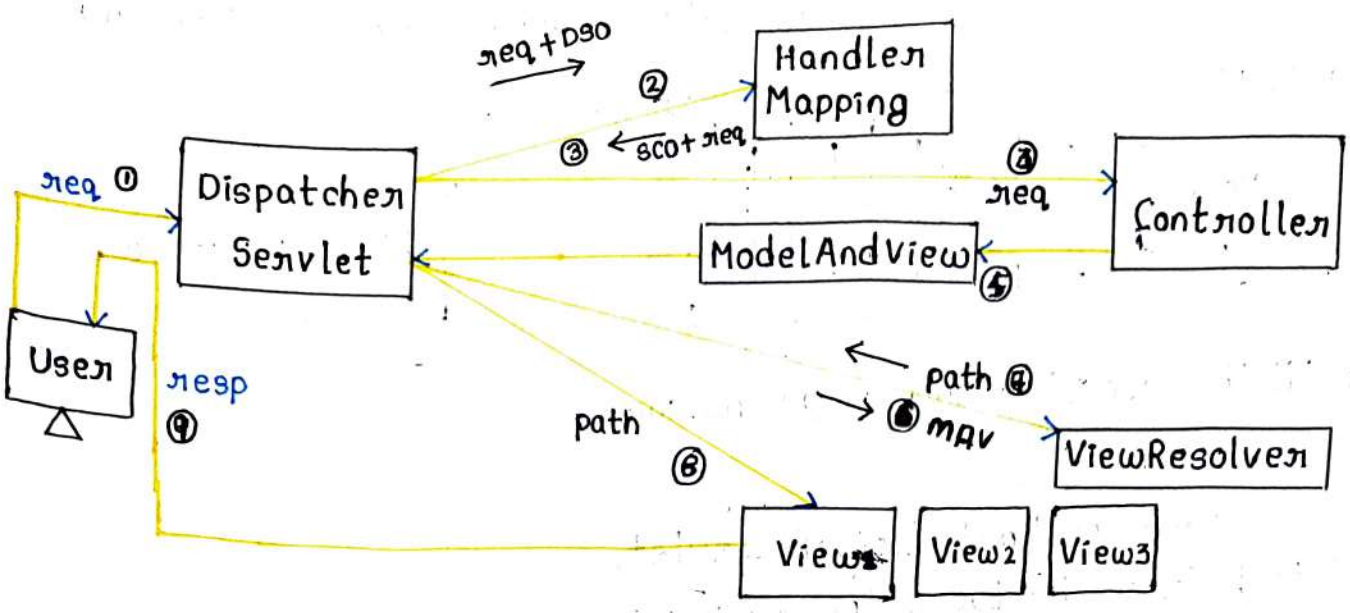
Web-app
↳ WEB-INF → mymvc-servlet.xml
↳ classes
↳ META-INF
↳ persistence.xml

- With the help of Dispatcher Servlet class we can control the flow of program execution of Spring MVC application. It is also called Front Controller.
- Model :- A model contains the data of the application. Here data means a single object or collection of objects.
- Controller :- A controller contains the business logic of an application. Here, @Controller annotation is used to mark the class as the controller.
- View :- A view represents the provided information in a particular format. A JSP+JSTL is used to create a view page.

Monolithical application :-

→ Within a single project, we are creating front-end and backend, then it is called monolithical application.

→ DispatcherServlet here acts as the front controller and all the requests are mapped to this class.



req means request
resp means response
SCO means SubController Object
SC means SubController
DSO means Dispatcher Servlet object

→ All incoming requests are intercepted by the DispatcherServlet that works as a front controller.

→ The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller

→ The controller returns an object of ModelAndView

→ The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

context:component-scan
It helps to identify the java classes which are registered as spring beans.

context: ~~component~~-annotation-config
It is used to activate annotations in beans already registered in the application context.

@RequestMapping
It is used to create an url for the methods.

@Controllen

It provides the object to the dispatchen servlet.

@model Attribute

We can fetch the object from the JSP page and with the help of this annotation we are passing the object to the method argument.

Ex:-

```
public class Student{
    int id;
    String name
    int age;
    public Student(int id, String name, int age){
        super();
        this.id=id;
        this.name=name;
        this.age = age;
    }
    public Student(){
        super();
    }
    public int getId(){
        return id;
    }
    public String getName(){
        return name;
    }
    public int getAge(){
        return age;
    }
    public void setId(int id){
        this.id = id;
    }
    public void setName(String name){
        this.name=name;
    }
```

```
    public void setAge(int age){
        this.age = age;
    }
}
```

JSP files :-

first.jsp

```
<body>
  ${time}
  ${msg}
</body>
```

Student.jsp

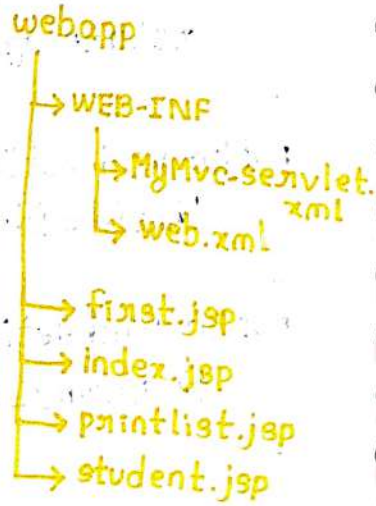
```
<body>
    ${stu.getId()}
    ${stu.getName()}
    ${stu.getAge()}
</body>
```

printlist.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<body>
    <c:forEach var="std" items="${list}">
      ${std.getId()}
      ${std.getName()}
      ${std.getAge()}
    </c:forEach>
</body>
```

MyMvc-servlet.xml

```
<beans ----
    ----
    ----
    ---->
  <context:component-scan base-package="com.jsp">
  </context:component-scan>
</beans>
```



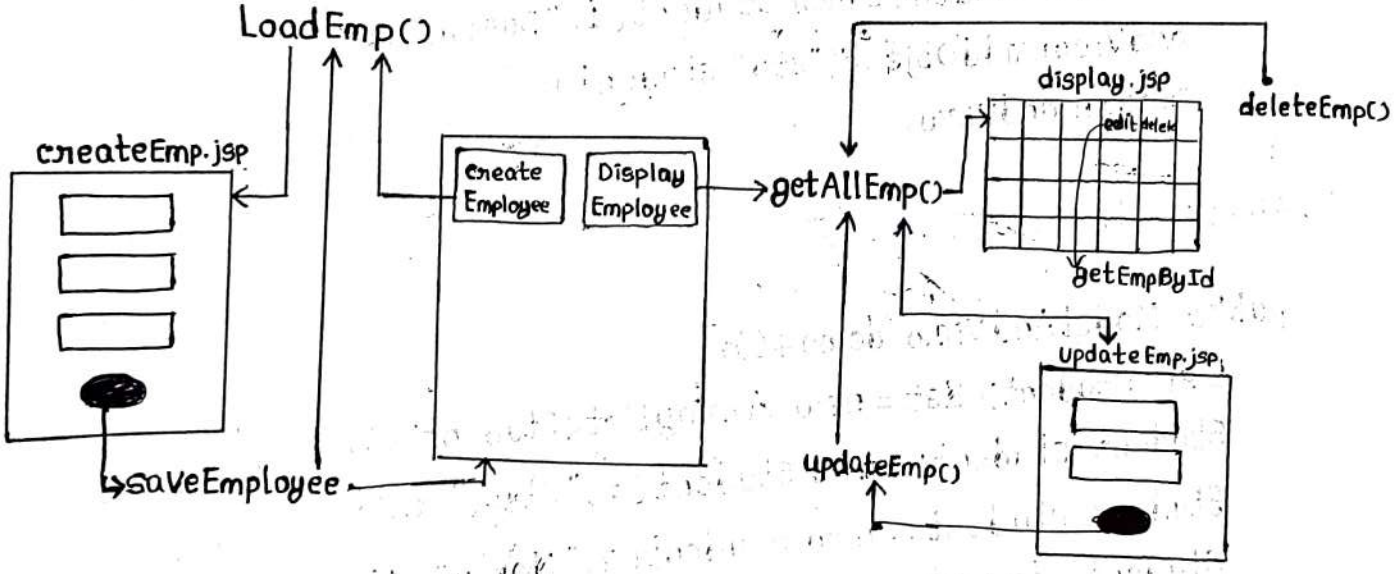
```
@Controller
public class Test{
    @RequestMapping("/demo1")
    public ModelAndView demo1(){
        ModelAndView andView = new ModelAndView("first.jsp");
        andView.addObject("msg", "<---Naveen-->");
        return andView;
    }
    @RequestMapping("/demo2")
    public ModelAndView demo2(){
        ModelAndView andView = new ModelAndView("first.jsp");
        andView.addObject("time", LocalDateTime.now());
        return andView;
    }
    @RequestMapping("/demo3")
    //for a single student object
    public ModelAndView demo3(){
        ModelAndView andView = new ModelAndView("student.jsp");
        Student student = new Student(1, "Dustin", 16);
        andView.addObject("stu", student);
        return andView;
    }
    @RequestMapping("/demo4")
    //for list of student objects
    public ModelAndView demo4(){
        List<Student> list = new ArrayList<Student>();
        Student student1 = new Student(1, "Steve", 22);
        Student student2 = new Student(2, "Nancy", 21);
        Student student3 = new Student(3, "Jane", 16);
        Student student4 = new Student(4, "Max", 16);
        list.add(student1);
        list.add(student2);
        list.add(student3);
        list.add(student4);
```

```
        ModelAndView andView = new ModelAndView ("printlist.jsp");
        andView.addObject("list", list);
        return andView;
    }
}
```

Limitations of Spring mvc :-

→ It creates only the main controllen.
→ We have to create the projects manually.
→ Manually we have to add the dependencies and servens.
→ Multiple methods are needed to perform same operation.
→ Requires more configuration files.
→ Not possible to create stand-alone applications.
→ We cannot send JSON objects from frontend to backend.
→ Always returns the ModelAndView objects.

To overcome all the above limitations we go for Spring boot.




```
<td><button><a href="get?id=${emp.getId()}">Edit </a></button></td>
<td><button><a href="delete?id=${emp.getId()}">Delete </a>
    </button></td>.
```

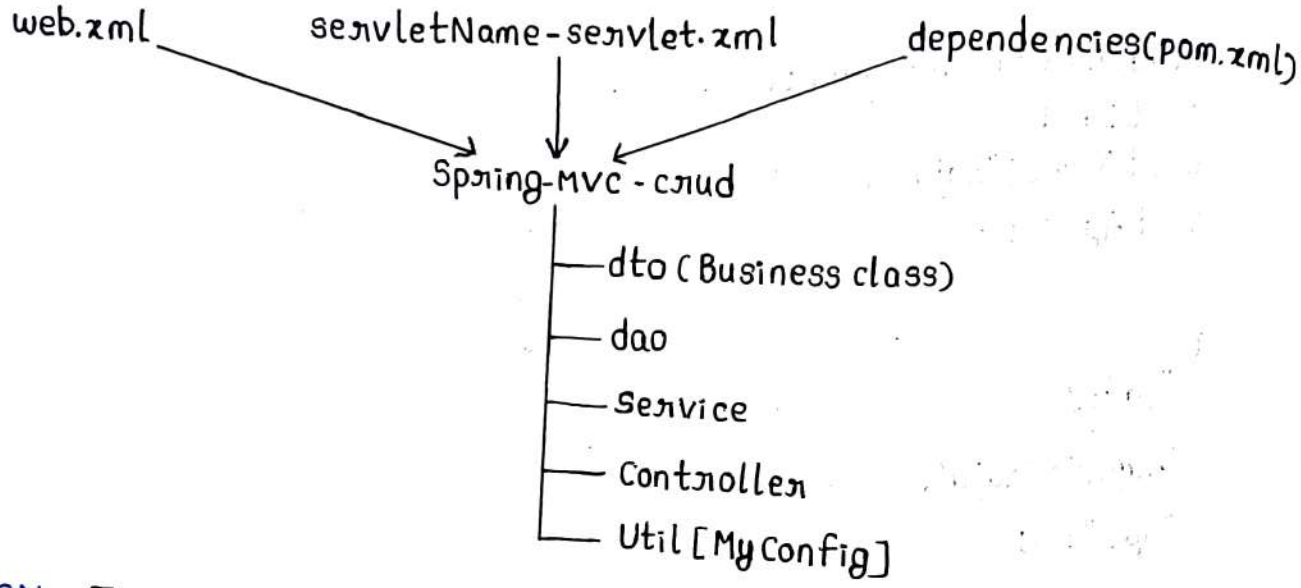
Note :-

jsp to jsp/method ⟶ anchor <a> tag.

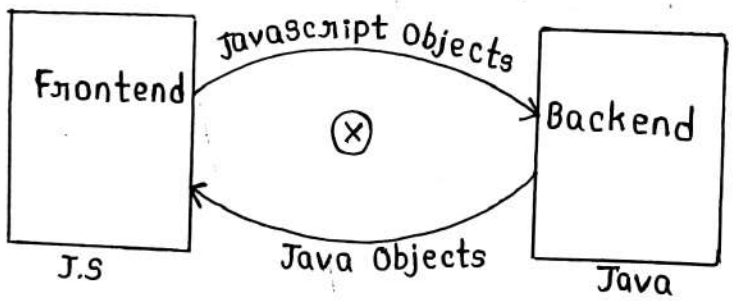
jsp/forms to jsp/method ⟶ action attribute

method to jsp ⟶ ModelAndView [JSP file name]

method to method ⟶ ModelAndView [/method URL]

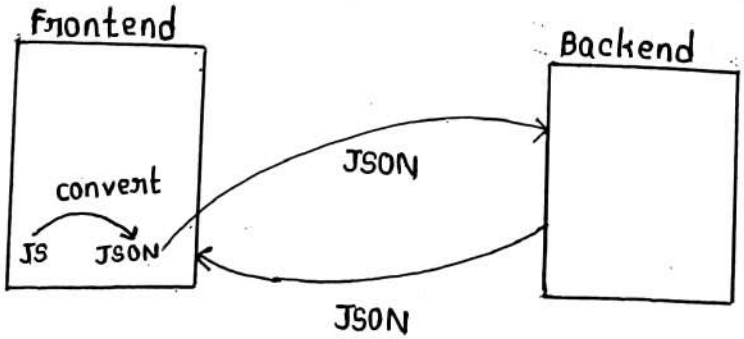


- JSON :- JavaScript Object Notation



⟶ It is a lightweight and an open standard data-interchange format whose objects are understandable by every programming languages.

⟶ JSON objects includes data structures such as arrays and objects.



Ex: How to create JSON object for a class

```
class Student{
    int id;
    String name;
    int age;
}
```

Normal Object:-

```
Student s=new Student();
S.setId(1);
S.setName("Raju");
S.setAge(21);
```

JSON object:-

```
{
  "id":1,
  "name":"Raju",
  "age":21
}
```

Ex2: Creating JSON object for a list

```
class Student{
    int id;
    String name;
    int age;
} List<String> l;
```

JSON object:-

```
{
  "id":1
  "name":"Dustin",
  "age":21,
  "l":["Nancy",
       "El",
       "Steve",
       "Max"
  ]
}
```

{} → obj

[] → list

Ex3:-

```
class Student{
   int id;
   String name;
   int age;
   List<Teacher> t;
}
class Teacher{
    int id;
    String name;
}
```

JSON object:-

```
{
  "id": 1,
  "name": "Dustin",
  "age": 22,
  "t": [ { "id": 2,
           "name": "Steve"
         },
         { "id": 3,
           "name": "Erica"
         }
    ]
}
```

Spring Boot :-

→ Spring Boot is an auto-configured framework and it is also a sub-framework of Spring framework mainly used to create standalone applications, dynamic webpages etc.

→ With the help of Spring Boot we can get the Spring IOC, Spring mvc and dependency injection.

→ With the help of Spring Boot, we can create

- stand-alone applications.
- Dynamic applications.
- Dynamic webpages
- Restful API's and Soap API's
- Enterprise Edition
- Standard Edition

→ Here, the projects are automatically generated by Spring initializers.

→ To have direct access of Spring initializers use the following url: start.spring.io.

→ Using @RestController we can create the controller class.

→ We are not creating xml files, projects and not adding dependencies, servers etc. Hence this Spring Boot is called auto-configured framework.

Advantages of Spring Boot :-

- Auto configuration
- No multiple methods for single operation.
- Automatically adds the dependencies.
- Execution is faster when compared to mvc.
- Easy to test (Postman tool)
- Embedded servers.

Note :-

PostMan is a ~~toot~~testing tool, with the help of PostMan we can test API's, developers can easily create, test, share and document API's. Also used to change the path also.

? How to create project in Spring Boot :-

→ Go to the url start.spring.io.

→ Select project Maven.

→ Select language as Java.

→ Select version as 2.7.13 for Spring.

→ Make group-id as com.jsp.

→ Provide artifact-id as spring-boot-first.

→ Select packaging as jar.

→ Select Java version : 8, it is the stable version.

→ Add dependencies

→ Web

→ Click on Generate

↓ Download

spring-boot-first.zip

boot-crud

web
data-JPA } mandatory

Mysql
Devtool } optional

→ Extract the zip files and copy the project file and paste it in the workspace.

Note :-

Using RequestParam we can fetch the object from the URL.

RequestMapping

→ PostMapping → for "save"

→ GetMapping → getById, getByName etc

→ DeleteMapping → Delete

→ PutMapping → For updating entire object.

→ PatchMapping → For updating a single object.

@Component

@Repository

@Service

@Controller

@RestController

MVC

① Model attribute

② RequestParam

Boot

① RequestBody

② RequestParam → url rewriting

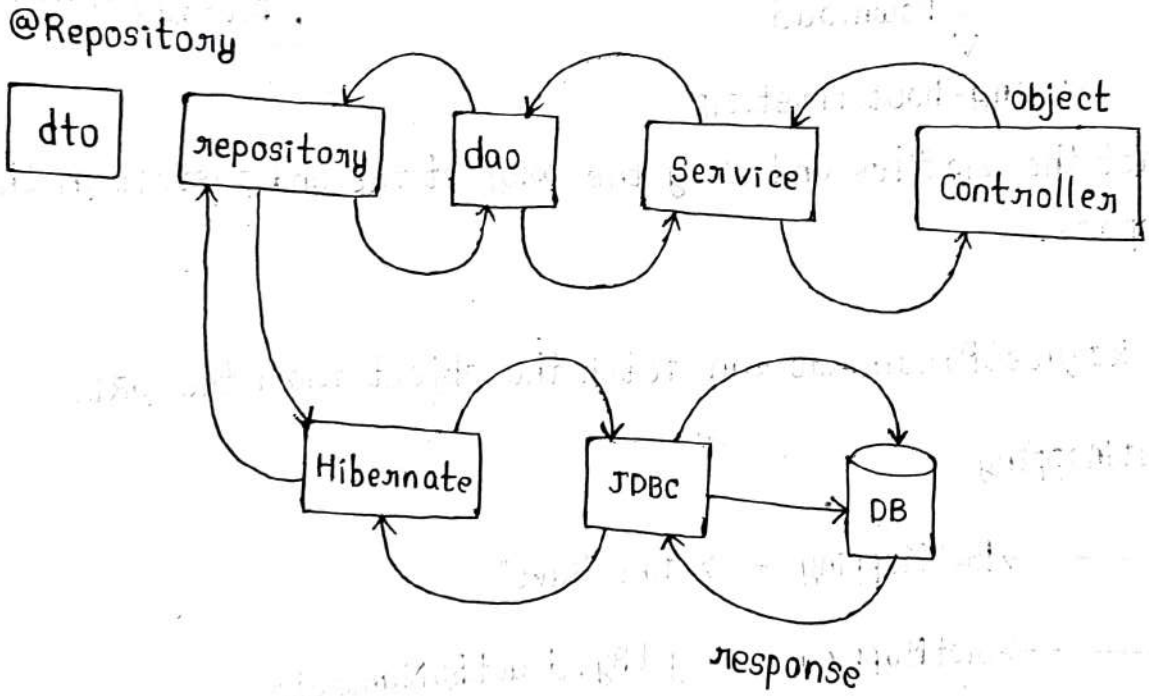
↓

getById? id=1

PathVariable → value for url

↓

getById/1



@Webservlet → @model attribute → @RequestBody

@RequestParam [url/?1 = value]

@PathVariable [url]

Ex:-

```
public class User{
    String email;
    String password;
    public String getEmail(){
        return email;
    }
    public String getPassword(){
        return password;
    }
    public  void setEmail(String email){
        this.email = email;
    }
    public void setPassword(String password){
        this.password = password;
    }
}
```

```
@RestController
public class Demo{
    @RequestMapping("/test1")
    public String test1(){
        return "All are heroes";
    }
    @RequestMapping("/test2")
    public LocalTime test2(){
        return LocalDateTime.now();
    }
    @RequestMapping("/test3")
    public User test3(){
        User user = new User();
        user.setEmail("Dustin@gmail.com");
        user.setPassword("Dustin123");
        return user;
    }
```

```
@RequestMapping("/test4")
public List test4(){
    User user1 = new User();
    user1.setEmail("Nancy@gmail.com");
    user1.setPassword("Nancy123");
    User user2 = new User();
    user2.setEmail("Maxine@gmail.com");
    user2.setPassword("Max123");
    List<User> list = new ArrayList<User>();
    list.add(user1);
    list.add(user2);
    return list;
}
@PostMapping("/save")
public void saveUser(@RequestBody User user){
    S.o.pln(user.getEmail());
    S.o.pln(user.getPassword());
}
}
```