

**J2EE [ADVANCED JAVA]**

**Java EE at a Glance**

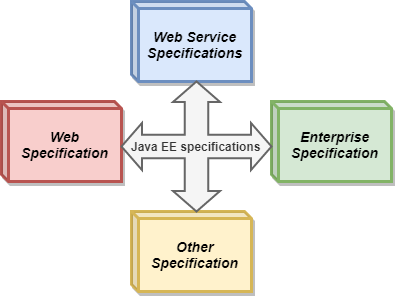
Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software. Java EE is developed using the [Java Community Process](https://www.jcp.org/en/home/index), with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals. Each release integrates new features that align with industry needs, improves application portability, and increases developer productivity.

**Java EE**

The **Java EE** stands for **Java Enterprise Edition**, which was earlier known as J2EE and is currently known as Jakarta EE. It is a set of specifications wrapping around Java SE (Standard Edition). The Java EE provides a platform for developers with enterprise features such as distributed computing and web services. Java EE applications are usually run on reference run times such as **microservers** or **application servers**. Examples of some contexts where Java EE is used are e-commerce, accounting, banking information systems.

**Specifications of Java EE**

Java EE has several specifications which are useful in making web pages, reading and writing from database in a transactional way, managing distributed queues. The Java EE contains several APIs which have the functionalities of base Java SE APIs such as Enterprise JavaBeans, connectors, Servlets, Java Server Pages and several web service technologies.



**1. Web Specifications of Java EE**

* Servlet- This specification defines how you can manage HTTP requests either in a synchronous or asynchronous way. It is low level, and other specifications depend on it
* WebSocket- WebSocket is a computer communication protocol, and this API provides a set of APIs to facilitate WebSocket connections.
* Java Server Faces- It is a service which helps in building GUI out of components.
* Unified Expression Language- It is a simple language which was designed to facilitate web application developers.

**2. Web Service Specifications of Java EE**

* Java API for RESTful Web Services- It helps in providing services having Representational State Transfer schema.
* Java API for JSON Processing- It is a set of specifications to manage the information provided in JSON format.
* Java API for JSON Binding- It is a set of specifications provide for binding or parsing a JSON file into Java classes.
* Java Architecture for XML Binding- It allows binding of xml into Java objects.
* Java API for XML Web Services- SOAP is an xml based protocol to access web services over http. This API allows you to create SOAP web services.

**3. Enterprise Specifications of Java EE**

* Contexts and Dependency Injection- It provides a container to inject dependencies as in Swing.
* Enterprise JavaBean- It is a set of lightweight APIs that an object container possesses in order to provide transactions, remote procedure calls, and concurrency control.
* Java Persistence API- These are the specifications of object-relational mapping between relational database tables and Java classes.
* Java Transaction API- It contains the interfaces and annotations to establish interaction between transaction support offered by Java EE. The APIs in this abstract from low-level details and the interfaces are also considered low-level.
* Java Message Service- It provides a common way to Java program to create, send and read enterprise messaging system's messages.

**4. Other Specifications of Java EE**

* Validation- This package contains various interfaces and annotations for declarative validation support offered by Bean Validation API.
* Batch applications- It provides the means to run long running background tasks which involve a large volume of data and which need to be periodically executed.
* Java EE Connector Architecture- This is a Java-based technological solution for connecting Java servers to Enterprise Information System.

**Setting up Java EE**

**Requirements**

-For the installation of latest SDK of Java EE which is Java EE 6 SDK on windows, you require to have a minimum memory of 1GB, minimum Disk space of 250MB free and JVM Java SE 6. For setting up Java EE, you require to have a JDK and then have an IDE preferably Eclipse as it is free.

Install a Java Development Kit

1. Browse to [Oracle's Java SE Development Kit downloads](https://www.oracle.com/technetwork/java/javase/downloads/index.html)
2. In the section titled **Java SE Development Kit 9.0.1**, read the license and, if you agree, click **Accept License Agreement**
3. Still, in that section, click on **JDK-9.0.1\_windows-x64\_bin.exe** (or the right download for your OS)
4. Run the downloaded JDK installer, using **Run As Administrator**
5. Add the Windows (or Linux) Environment Variable **JAVA\_HOME**. Set it to the root folder of your newly-installed JDK, which looks like C:\Program Files\Java\jdk1.8.0\_51.

**Install Eclipse for Java EE**

1. Browse to [Eclipse Downloads](https://www.eclipse.org/downloads/)

2. Click on the Download button under **Get Eclipse**.

3. On the resulting page, click on the Download button.

Note: The version of Eclipse (32-bit or 64-bit) which you download should match the version of your JDK. You installed JDK-9.0.1\_windows-x64 above, so download the 64-bit Eclipse.

4. Run the downloaded installer using **Run as Administrator**.

5. Choose the version of Eclipse you wish to install. **Eclipse IDE for Java EE developers** is preferable for Java work.

6. If the installation fails, try again with real-time virus scanning temporarily turned off. Remember to turn it on again when it's done.

|  |  |
| --- | --- |
| Java SE | Java EE |
| Java SE provide basic functionalities such as defining types and objects. | Java EE facilitates development of large scale applications. |
| SE is a normal Java specification | EE is built upon Java SE. It provides functionalities like web applications, and Servlets. |
| It has features like class libraries, deployment environments, etc. | Java EE is a structured application with a separate client, business, and Enterprise layers. |
| It is mostly used to develop APIs for Desktop Applications like antivirus software, game, etc. | It is mainly used for developing web applications. |
| Suitable for beginning Java developers. | Suitable for experienced Java developers who build enterprise-wide applications. |
| It does not provide user authentication. | It provides user authentication. |

**Application**

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

**Types of Java Applications**

There are mainly 4 types of applications that can be created using Java programming:

**1) Standalone Application**

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

**2) Web Application**

An application that runs on the server side and creates a dynamic page is called a web application. Currently, [Servlet](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), [Struts](https://www.javatpoint.com/struts-2-tutorial), [Spring](https://www.javatpoint.com/spring-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), etc. technologies are used for creating web applications in Java.

**3) Enterprise Application**

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, [EJB](https://www.javatpoint.com/ejb-tutorial) is used for creating enterprise applications.

**4) Mobile Application**

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

**Java Platforms / Editions**

There are 4 platforms or editions of Java:

**1) Java SE (Java Standard Edition)**

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, [String](https://www.javatpoint.com/java-string), Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

**2) Java EE (Java Enterprise Edition)**

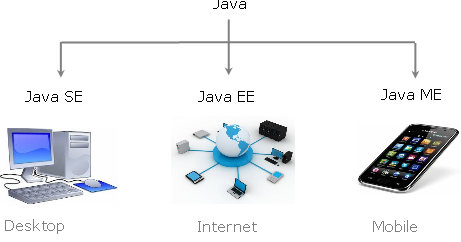
It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, [JPA](https://www.javatpoint.com/jpa-tutorial), etc.

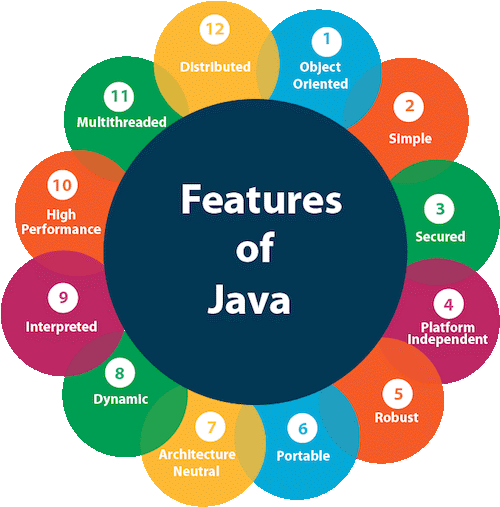
**3) Java ME (Java Micro Edition)**

It is a micro platform that is dedicated to mobile applications.

**4) JavaFX**

It is used to develop rich internet applications. It uses a lightweight user interface API.





**Project Structure**

Project Name

Javafiles

Sub\_packages

Source Folder

Base\_Package

Eclipse Structure

Src

>com.jspiders.cardekho\_case\_study

>>App.java

>com.jspiders.cardekho\_case\_study.entity

>>Car.java

> com.jspiders.cardekho\_case\_study.study.menu

>>CarDekhoMenu.java

> com.jspiders.cardekho\_case\_study.opration

>>CarOperation.java

Base Package Naming Convention

Format:

Domain.host\_name.project\_name

Example

com.jspiders.cardekho\_case\_study

Sub Package Naming Convention

Format:

Base\_package\_name.package\_name

Example:

com.jspiders.cardekho\_case\_study.main

-To understand the concept of the multithreading we need to understand what is Multitasking and multiprocessing.

**MULTITASKING**

-Performing more than one task at the same time is known as Multitasking.

**TASK**

-A task is the end goal that has to be achieve.

**P3**

**P2**

**P1**

**P4**

**P5**

**P6**

**P9**

**P8**

**P7**

**TASK**

-In order to achieve the completion, the task in lesser time or to reduce the time taken for the task completion, we need to implement multiprocessing.

**PROCESS**

-The Steps involved in the completion of the task are known as Processes of the task.

Hence it can be said that a task is the combination of Multiple (More than One) processes.

**MULTIPROCESSING**

-Executing More than One Process at a same time is called as Multiprocessing

-As Process is a step involved in achieving the task completion it is also divide into smaller Units.

**THREAD**

-A thread can be defined as Smallest Unit of Process or Light-Weight Process.

-The Smallest unit of the Process is known as a Thread

-hence it can be said that a Process is the Combination of more than one (Multiple) Threads

**Th3**

**Th2**

**Th1**

**Smallest Unit**

**Th4**

**Th5**

**Th6**

**Th9**

**Th8**

**Th7**

**Process**

**MULTITHREADING**

-Execution of more than one Threads is Known as Multithreading.

-In Java a Thread is considered as a special Class which can be created in two ways:

**1]By Extending the Thread Class**

**2]By Implementing the Runnable Interface**

**Extends Thread**

**extends** **Implements**

**Runnable**

**Interface**

**My Thread**

**Start()**

-It is a Non-Static Method Thread Class.

-To Start class as Thread we need to call with its Object Reference Variable.

**Run()**

-It is the method from Runnable Interface which is also overridden in Thread Class.

-The execution logic of a thread has to be declared in this Method.

-We do not need to explicitly call this method, the start() method calls it Implicitly

**1]By Extending the Thread Class**

**>>>>>>>>>>>>>>>> IMPORTANT >>>>>>>>>>>>>>**

-If the start() method is not called and the run() method called directly then the class behave has Normal Class

**MyThread1.java**

**package** com.jspiders.multithreading.thread;

**public** **class** MyThread1 **extends** Thread {

@Override

**public** **void** run() {

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

System.***out***.println("My Thread 1 is Now Running");

}

}

**ThreadMain1.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread1;

**public** **class** ThreadMain1 {

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

MyThread1 myThread1 = **new** MyThread1();

myThread1.start();

}

}

**================ OUTPUT ================**

My Thread 1 is Now Running

-The above programming is created by using **Thread** class which default class in Java present in java.lang package.

-By Using **start()** method which is Non-Static method of Thread class we call using Thread class Object and run the thread.

**2]By Implementing the Runnable Interface**

**MyThread2.java**

**package** com.jspiders.multithreading.thread;

**public** **class** MyThread2 **implements** Runnable {

@Override

**public** **void** run() {

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

System.***out***.println("MyThread 2 is Now Running");

}

}

**ThreadMain2.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread2;

**public** **class** ThreadMain2 {

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

MyThread2 myThread2 = **new** MyThread2();

Thread thread = **new** Thread(myThread2);

thread.start();

}

}

**================ OUTPUT ================**

MyThread 2 is Now Running

-When a Thread is created by implementing Runnable Interface then we do not have access of **start()** method with the object of Thread class.

-Hence we need to create the object of Thread Class Directly and Pass the Object of the user Defined Thread Class as an Argument to the Overloaded Constructor of the Thread Class.

-Now with the object of Thread Class , we can directly access the start() method which will implicitly call the run() method present in the User-Defined Thread Class.

Que.What is a Better Way for Creating a Thread and why?

Ans:-There are two ways to creating the thread in java

**1]By Extending The Thread Class**

**2]By Implementing the Runnable Interface**

-By Implementing The Runnable Interface is a better way to create thread in java because when we create a thread by extending class, all thread class methods are inherited while we can perform the task with the one method (run method) only. It result into Overhead Inheritance.

-Other reason java does not support Multiple Inheritance in case of classes .So if we create a thread by extending the thread class , we will not be able to extend any other class.

-Implementing the interface its better way to create a thread

-Because when we extend Thread class with another class at that time we cannot extends multiple class at a same time

-There should be diamond problem happen

-To avoid this restriction, we can implement the Runnable interface, in this scenario we can extend another class also

-That’s why the implement the Runnable interface is the better way to create a Thread

**LIFECYCLE OF THREAD**

**RUNNING**

**START**

**STOP**

**WAITING**

**Fig Of Life Cycle Of Thread**

-Whenever a thread is initialize it Moves The **Start** Phase.

-If the Thread is Executing then we can say that it is **Running** Phase.

-If the Thread is Started but not Executing then we can say that it is in **Waiting** Phase(State).

-When the Thread is Terminating or ended then we say it is in **Stop** Phase.

**TRANSITION OF THREAD**

**START**

**START**

**WAITING**

**RUNNING**

**WAITING**

**RUNNING**

Execution Complete

Resource

Allocated

Resource Taken Away

Waiting for To Long

Resource

Allocated

Resource

Not

Allocated

**WAITING**

**RUNNING**

**WAITING**

**STOP**

**STOP**

**RUNNING**

-There are six transitons during life cycle of a thread: -

1.Start to running phase: -: - When a thread scheduler gives CPU to a thread for its execution then,

that particular thread will go from start phase to running phase.

2.Start to waiting phase: -When thread scheduler doesn’t give a CPU to thread for its execution then

that particular thread will go from start phase to waiting phase.

3.Running to waiting phase: - When a thread scheduler takes back a CPU from the thread then that

particular thread will go from running phase to waiting phase.

4.Waiting to running phase: - When a thread scheduler gives a CPU to a thread which was in waiting

phase for its execution then that particular thread will go from waiting to running phase.

5.Running to stop phase: - When an execution of thread is completed than that particular thread will

go from running to stop phase.

6.Wai􀆟ng to stop phase: - When a thread is waiting for longer time then automatically that thread

will go from waiting to stop phase.

**MyThread3.java**

**package** com.jspiders.multithreading.thread;

**public** **class** MyThread3 **extends** Thread {

@Override

**public** **void** run() {

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

**for** (**int** i = 1; i <= 5 ; i++) {

System.***out***.println("MyThread 3 Is Now Running");

}

}

}

**EXECUTING MULTIPLE THREAD SIMULTENOUSLY**

**ThreadMain3.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread3;

**public** **class** ThreadMain3 {

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

MyThread3 myThread3 = **new** MyThread3();

myThread3.start();

}

}

**================ OUTPUT ================**

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 3 Is Now Running

Note:-Using Runnable Interface Executing Multiple Threads

**MyThread4.java**

**package** com.jspiders.multithreading.thread;

**public** **class** MyThread4 **implements** Runnable {

@Override

**public** **void** run() {

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

**for** (**int** i = 1; i <= 5; i++) {

System.***err***.println("MyThread 4 is Running Now");

}

}

}

**ThreadMain4.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread4;

**public** **class** ThreadMain4 {

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

MyThread4 myThread4 = **new** MyThread4();

Thread thread = **new** Thread(myThread4);

thread.start();

}

}

**================ OUTPUT ================**

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

NOTE:- If you below program the output is not constant.

**MyThread3$4.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread3;

**import** com.jspiders.multithreading.thread.MyThread4;

**public** **class** ThreadMain5 {

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

MyThread3 myThread3 = **new** MyThread3();

MyThread4 myThread4 = **new** MyThread4();

Thread thread = **new** Thread(myThread4);

myThread3.start();

thread.start();

}

}

**===== Actual Output =====**

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 3 Is Now Running

**===== Expected Output =====**

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 3 Is Now Running

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

MyThread 4 is Running Now

**==================== NOTE ====================**

-The Actual Output of the Above Program is Not Constant it is happing due to a Component Known As “**THREAD SCHEDULAR**”

**THREAD SCHEDULAR**

-It is a component in multithreading which is responsible to manage the Life Cycle of thread

-Whenever a thread initialize (Started) ,the Thread Schedular is responsible to allocate the a Dedicated Stack for the execution of that Thread.

**Thread Class**

-Thread is a special class in java

-We create a Thread in two Ways

**1]By Extending the Thread Class.**

**2]By Implementing the Runnable Interface.**

-Why is it special because there are two different Stacks allocated for the execution of Thread Class.Then the two Stacks Running the Simultaneously.

-That’s why the order of the Execution is getting in Different order.

**PROPERTIES OF THREAD**

-There are three Properties for every thread

**1]ID**

**2]Name**

**3]Priority**

-These properties can be access by using the Helpers Methods (Getters & Setters).

**ID**

**NAME**

**PRIORITY**

-The default ID for each thread is “**11**”.

-The default Name for each thread starts from “**Thread-0**”.

-The default Priority for each Thread is “**5**”.

: - Min is 1.

: - Max is 10.

**MyThread5.java**

**package** com.jspiders.multithreading.thread;

**public** **class** MyThread5 **extends** Thread {

@Override

**public** **void** run() {

System.***out***.println("MyThread 5 is Running Now");

}

}

**ThreadMain5.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread5;

**public** **class** ThreadMain5 {

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

MyThread5 myThread5 = **new** MyThread5();

myThread5.start();

System.***out***.println("Thread ID : "+myThread5.getId());

System.***out***.println("Thread Name : "+myThread5.getName());

System.***out***.println("Thread Priority :"+myThread5.getPriority());

System.***out***.println("Thread State : "+myThread5.getState());

}

}

**================ OUTPUT ================**

Thread ID : 11

Thread Name : Thread-0

MyThread 5 is Running Now

Thread Priority :5

Thread State : TERMINATED

id: - The value for this property will be assign by JVM.

**getId ():** - This method is used to access the id value of a thread, it is non-static in nature, present

inside thread class.

-There no method for set ID for Thread.

**getName()**

-It is Non-Static Method Present in Thread Class

-It is used to retrieve the Name Of the Thread.

**setName()**

-It is a Non-Static method present in thread class.

-It is used to modify the name of Thread.

**getPriority()**

-It is Non-Static Method from Thread class which is used to retrieve the Priority of a Thread.

**setPriority()**

-It is a Non-Static Method from Thread class which is used to Modify the Priority of Thread.

-We cannot set the Priority 0 (zero) to the Thread neither we get Runtime Exception Called Exception in thread "main" java.lang.IllegalArgumentException

-We can set a Thread Name and Priority using set Methods.

**MyThread6.java**

**package** com.jspiders.multithreading.thread;

**public** **class** MyThread6 **extends** Thread {

@Override

**public** **void** run() {

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

System.***out***.println("Name of Thread : "+getName());

System.***out***.println("Priority of Thread "+getPriority());

}

}

**ThreadMain6.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread6;

**public** **class** ThreadMain6 {

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

MyThread6 myThread6 = **new** MyThread6();

myThread6.setName("Thread-1");

myThread6.setPriority(1);

myThread6.start();

}

}

**================ OUTPUT ================**

Name of Thread : Thread-6

Priority of Thread 6

**currentThread():**

-This is a static method present in thread class

-It is used to return the reference of the currently executing Thread Object.

-When there is no directly relation between the Runnable Interface and User Defined Class then that time to return The Reference of the Current Executing Thread.

**MyThread7.java**

**package** com.jspiders.multithreading.thread;

**public** **class** MyThread7 **implements** Runnable {

@Override

**public** **void** run() {

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

System.***out***.println("Id Of Thread :- "+Thread.*currentThread*().getId());

System.***out***.println("Name Of Thread :- "+Thread.*currentThread*().getName());

System.***out***.println("Name Of Thread :- "+Thread.*currentThread*().getPriority());

}

}

**ThreadMain7.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread7;

**public** **class** ThreadMain7 {

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

MyThread7 myThread7 = **new** MyThread7();

Thread thread = **new** Thread(myThread7);

thread.setName("Thread-7");

thread.setPriority(7);

thread.start();

}

}

**================ OUTPUT ================**

Id Of Thread :- 11

Name Of Thread :- Thread-7

Name Of Thread :- 7

**Stop()**

**Deprecated 🡺No longer to use/Outdated/Not in Use.**

-It is a Non-Static method present inside the Thread class.

-It is used to stop the execution of currently executing thread forcefully.

-It is a deprecated method which is already mark for Removal.

**MyThread8.java**

**package** com.jspiders.multithreading.thread;

**public** **class** MyThread8 **extends** Thread {

@SuppressWarnings("deprecation")

@Override

**public** **void** run() {

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

**for** (**int** i = 1; i <=5; i++) {

**if** (i == 3) {

~~stop~~();

}

System.***out***.println(getName()+" Is Now Running");

}

}

}

**ThreadMain8.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.thread.MyThread8;

**public** **class** ThreadMain8 {

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

MyThread8 myThread8 = **new** MyThread8();

myThread8.setName("Thread-8");

myThread8.setPriority(8);

myThread8.start();

}

}

**================ OUTPUT ================**

Thread-8 Is Now Running

Thread-8 Is Now Running

**Resources Folder**

**Account.java**

**package** com.jspiders.multithreading.resource;

**public** **class** Account {

**private** **int** balance;

**public** Account(**int** balance)

{

**this**.balance = balance;

}

**public** **void** deposite(**int** money)

{

System.***out***.println("Trying To Deposite "+money+"Rs.");

balance+=money;

System.***out***.println("Deposite Successful");

System.***out***.println("Account Balance :- "+checkBalance());

}

**public** **void** withdraw(**int** money)

{

System.***out***.println("Trying To Withdraw "+money+"Rs.");

balance-=money;

System.***out***.println("Withdraw Successful");

System.***out***.println("Account Balance :- "+checkBalance());

}

**public** **int** checkBalance() {

**return** balance;

}

}

**Husband.java**

**package** com.jspiders.multithreading.thread;

**import** com.jspiders.multithreading.resource.Account;

**public** **class** Husband **extends** Thread {

**private** Account account;

**public** Husband(Account account)

{

**this**.account = account;

}

**public** **void** run()

{

account.deposite(5000);

account.withdraw(2000);

}

}

**Wife.java**

**package** com.jspiders.multithreading.thread;

**import** com.jspiders.multithreading.resource.Account;

**public** **class** Wife **extends** Thread {

**private** Account account;

**public** Wife(Account account) {

**this**.account = account;

}

@Override

**public** **void** run() {

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

account.deposite(2000);

account.withdraw(5000);

}

}

**AccountMain.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.resource.Account;

**import** com.jspiders.multithreading.thread.Husband;

**import** com.jspiders.multithreading.thread.Wife;

**public** **class** AccountMain {

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

Account account = **new** Account(1000);

Husband husband = **new** Husband(account);

Wife wife = **new** Wife(account);

husband.start();

wife.start();

}

}

**===== Actual Output =====**

Trying To Deposite :- 5000Rs.

Deposite Successful

Trying To Deposite :- 2000Rs.

Deposite Successful

Account Balance :- 8000

Trying To Withdraw :- 5000Rs.

Withdraw Successful

Account Balance :- 3000

Account Balance :- 6000

Trying To Withdraw :- 2000Rs.

Withdraw Successful

Account Balance :- 1000

**===== Expected Output =====**

Trying To Deposite :- 5000Rs.

Deposite Successful

Account Balance :- 6000

Trying To Withdraw :- 2000Rs.

Withdraw Successful

Account Balance :- 4000

Trying To Deposite :- 2000Rs.

Deposite Successful

Account Balance :- 6000

Trying To Withdraw :- 5000Rs.

Withdraw Successful

Account Balance :- 1000

**SHARED RESOURCE**

-The Resource which is access by more than One Thread Simultaneously is called as Shared Resource.

-A resource can be a Variable, a Method, an Object, Class or even Memory.

-If the Resource a Static member then it is Known as Static Resource Hence if it is a Non-Static member then it is known as Non-Static Resource.

**DATA INCONSISTENCY IN MULTITHREADING**

-When a Multiple Thread are Operating, each Thread is Allocated its own Separate Stack by the Thread Schedular.

-If this Thread are operating on shared resource, then the Threads are unaware of the operations performed on the Shared Resource by the Other Threads.

-Hence the Threads are operating On the Resource independently

Which leads “**Data Inconsistency**”.

**NOTE:** The data Inconsistency caused due to **Multithreading** can be avoided with the help of “**SYCHRONIZATION”.**

**SYNCRONIZATION**

-Synchronization is a process which is used to avoid the Data Inconsistency caused due to Multithreading while operating on a Shared Resource.

-Synchronization is implemented with the help of “**synchronized**” Keyword.

-When a Shared Resource is made Synchronized, then whenever a thread gets the access that Resource, it applies a Lock on That Resource.

-It means if a Shared Resource is Synchronized then Only one Thread Can Access at a time,

-All the Other Thread That want to Access that Resource will have to wait until that Lock on that Resource is Released.

Apply Lock Apply Lock

**T2**

**T1**

Release Lock

Release Lock

**Synchronized**

**Shared Resource**

Apply Lock Apply Lock

Release Lock Release Lock

T4

**T3**

**LOCKS**

**IN MULTITHREADING**

-In Multithreading there are two Types Of Lock

1]Class Lock

2]Object Lock

**Class Lock**

-If the Synchronized Shared Resource is a Static Member then the Lock Applied on it will be Class Lock.

**Object Lock**

-If the Synchronized Shared Resource is a Non-Static Member then the Lock Applied on it will be Object Lock.

**TO IMPLEMENT SYNCRONIZATION IN MULTITHREADING**

**Resources Folder**

**Account.java**

**package** com.jspiders.multithreading.resource;

**public** **class** Account {

**private** **int** balance;

**public** Account(**int** balance)

{

**this**.balance = balance;

}

**public** **synchronized** **void** deposite(**int** money)

{

System.***out***.println("Trying To Deposite :- "+money+"Rs.");

balance+=money;

System.***out***.println("Deposite Successful");

System.***out***.println("Account Balance :- "+checkBalance());

}

**public** **synchronized** **void** withdraw(**int** money)

{

System.***out***.println("Trying To Withdraw :- "+money+"Rs.");

balance-=money;

System.***out***.println("Withdraw Successful");

System.***out***.println("Account Balance :- "+checkBalance());

}

**public** **int** checkBalance() {

**return** balance;

}

}

-In this case as Synchronized Keyword is applied on the shared Resources the Deposit() method and Withdraw() Method ,it can be access either Husband or Wife Thread at Time.

-Hence if one Thread is Already operating on Shared Resources then the Other Thread cannot operate on it until the First Thread is done Executing.

-This will Help us to achieve a Consistent a final output.

**===== Actual Output =====**

Trying To Deposite :- 5000Rs.

Deposite Successful

Account Balance :- 6000

Trying To Withdraw :- 2000Rs.

Withdraw Successful

Account Balance :- 4000

Trying To Deposite :- 2000Rs.

Deposite Successful

Account Balance :- 6000

Trying To Withdraw :- 5000Rs.

Withdraw Successful

Account Balance :- 1000

**===== Expected Output =====**

Trying To Deposite :- 5000Rs.

Deposite Successful

Account Balance :- 6000

Trying To Withdraw :- 2000Rs.

Withdraw Successful

Account Balance :- 4000

Trying To Deposite :- 2000Rs.

Deposite Successful

Account Balance :- 6000

Trying To Withdraw :- 5000Rs.

Withdraw Successful

Account Balance :- 1000

**wait()**

-It is a Non-Static method which is used to put the currently executing thread from the running phase to the waiting phase forcefully.

-The Thread whose execution is forcefully stopped cannot resume its execution automatically.

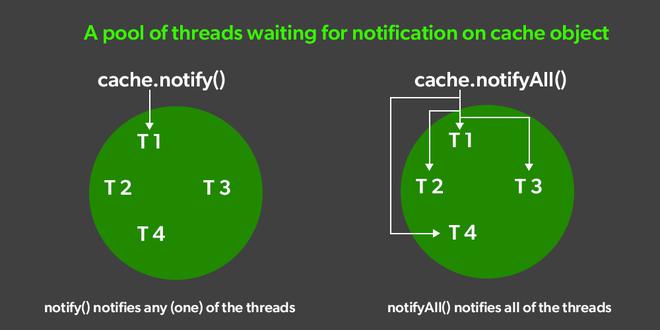
**notify()**

-It is Non-Static method which is used t to call a waiting Thread from the waiting Phase to Running Phase.

-This Method id capable to resume the only one waiting Thread.

**notifyAll()**

-It is similar to the notify() method but it is capable to Resume the Execution of all the waiting Threads.



QNo1.Write the Difference between between notify() and notifyAll()

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Key | notify() | notifyAll() |
| 1 | Notifications | In the case of the [multiThreading](https://www.geeksforgeeks.org/multithreading-in-java/), notify() method sends the notification to only one thread among the multiple waiting threads which are waiting for the send lock. | While notifyAll() methods in the same context send notifications to all waiting threads instead of a single thread. |
| 2 | Thread identifications | As in the case of notify() method, the notification is sent to a single thread among the multiple waiting threads, so it is sure that which of those waiting threads is going to receive the lock. | On the other hand, notifyAll() sends a notification to all waiting threads. Hence, it is not clear which of the thread is going to receive the lock. |
| 3 | Risk factor | In the case of notify() method, the risk of thread missing is high as notification is sent only a single thread, and if it misses that, then no other thread would get a notification and hence the lock. | While in the case of notifyAll(), it sends a notification to all the waiting threads, and hence if any thread misses the notification, there are other threads to do the job. Hence the risk is less. |
| 4 | Performance | [Memory](https://www.geeksforgeeks.org/introduction-to-memory-and-memory-units/) and [CPU](https://www.geeksforgeeks.org/central-processing-unit-cpu/)drain is less in notify() method as compared to notifyAll() method as notification is sent to single one thread so performance is better as compared to notifyAll(). | On the other hand, the cost of no notification is dropped and notification is sent to all waiting threads, the memory and CPU drain is more as compared to notify() and hence performance of notifyAll() is lesser. |
| 5 | Interchangeable | In the case of the notify() method, only a single thread is in the picture hence no concept of thread Interchangeable is possible. | While we should go for notifyAll() if all your waiting threads are interchangeable (the order they wake up doesn’t matter). |

|  |  |
| --- | --- |
| Wait() | Sleep() |
| Wait() method belongs to Object class. | Sleep() method belongs to Thread class. |
| Wait() method releases lock during Synchronization. | Sleep() method does not release the lock on object during Synchronization. |
| Wait() should be called only from Synchronized context. | There is no need to call sleep() from Synchronized context. |
| Wait() is non-static method. | Sleep() is a static method. |
| Wait() Has Three Overloaded Methods:   * wait() * wait(long timeout) * wait(long timeout, int nanos) | Sleep() Has Two Overloaded Methods:   * sleep(long millis)millis: milliseconds * sleep(long millis,int nanos) nanos: Nanoseconds |
| public final void wait(long timeout) | public static void sleep(long millis) throws Interrupted\_Execption |

**Resources Folder**

**Pizza.java**

**package** com.jspiders.multithreading.resource;

**public** **class** Pizza {

**private** **int** avaialablePizza;

**public** **synchronized** **void** orderPizza(**int** orderedPizza) {

System.***out***.println("Trying To Order : "+orderedPizza+" Pizzas");

**if** (avaialablePizza < orderedPizza) {

System.***out***.println(orderedPizza+" Pizzas Not Available "+" Please Wait");

**try** {

**this**.wait();

} **catch** (InterruptedException e) {

// **TODO**: handle exception

}

}

avaialablePizza -= orderedPizza;

System.***out***.println(orderedPizza+ " Pizzas Delivered ");

System.***out***.println(avaialablePizza+" Pizzas Available ");

}

**public** **synchronized** **void** bakedPizza(**int** bakedPizza)

{

System.***out***.println("Baking "+bakedPizza+" Pizzas");

avaialablePizza += bakedPizza;

System.***out***.println(bakedPizza+" Pizzas Baked");

System.***out***.println(avaialablePizza+" Pizzas Available ");

**this**.notify();

}

}

**Friends.java**

**package** com.jspiders.multithreading.thread;

**import** com.jspiders.multithreading.resource.Pizza;

**public** **class** Friends **extends** Thread {

**private** Pizza pizza;

**public** Friends(Pizza pizza) {

**super**();

**this**.pizza = pizza;

}

@Override

**public** **void** run() {

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

pizza.orderPizza(5);

}

}

**Dominos.java**

**package** com.jspiders.multithreading.thread;

**import** com.jspiders.multithreading.resource.Pizza;

**public** **class** Dominos **extends** Thread {

**private** Pizza pizza;

**public** Dominos(Pizza pizza) {

**super**();

**this**.pizza = pizza;

}

@Override

**public** **void** run() {

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

pizza.bakedPizza(5);

}

}

**PizzaMain.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.resource.Pizza;

**import** com.jspiders.multithreading.thread.Dominos;

**import** com.jspiders.multithreading.thread.Friends;

**public** **class** PizzaMain {

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

Pizza pizza = **new** Pizza();

Friends friends = **new** Friends(pizza);

Dominos dominos = **new** Dominos(pizza);

friends.start();

dominos.start();

}

}

**================ Actual Output ==============**

5 Pizzas Not Available Please Wait

Baking 5 Pizzas

5 Pizzas Baked

5 Pizzas Available

5 Pizzas Delivered

0 Pizzas Available

**Assignment**

Assignment 1

QNo.1Write the details of all the methods used in multithreading in the below format.

**Method Name**

**Class/Interface**

**Method Signature**

**ReturnType**

**Modifier**

**Access Modifier**

**Exception**

Assignment 2

QNo1.Write the Difference between between notify() and notifyAll()

Assignment 3

QNo1.Write the java program to demonstrate the implementation of notifyAll()

**sleep()**

-It is Static Method which is used to pause the execution of the currently executing Thread for specific Time Interval.

-Unlike the **wait()**,the **sleep()** does not need notify() method call.

-It means if a Thread is paused with the help of **sleep()** then it will resume it execution automatically after the defined time period is completed.

-This method is an overloaded method.

-The most commonly used method signature is as follows:

**Method Signature:**

**sleep(long millisecond)**

**DEADLOCK**

-When the multiple Threads are Executed on Shared Resource and there execution is Interdepended and also each Thread has acquired(Applied)Lock on Resource which is required by another Thread and vice-versa then None of the Thread will be capable of Releasing Lock from the Resource and hence it cannot be given to any other Thread.

-In such situation None of the Thread will be able to complete it execution This is Known as deadlock Situation.

**SYNCHRONIZED BLOCK**

-Synchronized block allows us to make all the Resources (Variables,Objects,Classes) as Synchronized by default.

-It is a block of code which is having the keyword synchronized

Syntax of Synchronized Block is as Follows:

**Synchronized**

**{**

**------------------------------**

**------------------------------**

**------------------------------**

**}**

**Resources Folder**

**Resource.java**

**package** com.jspiders.multithreading.resource;

**public** **class** Resource {

**public** String res1 = "1st Resource";

**public** String res2 = " 2nd Resource";

}

**MyThread9.java**

**package** com.jspiders.multithreading.thread;

**import** com.jspiders.multithreading.resource.Resource;

**public** **class** MyThread9 **extends** Thread {

Resource resource;

**public** MyThread9(Resource resource)

{

**this**.resource = resource;

}

@Override

**public** **void** run() {

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

**synchronized** (resource.res1) {

System.***out***.println(getName()+" Applied Lock On "+resource.res1);

**synchronized** (resource.res2) {

System.***out***.println(getName()+" Applied Lock On "+resource.res2);

}

System.***out***.println(getName()+ " Released "+resource.res2);

}

System.***out***.println(getName()+ " Released "+resource.res1);

}

}

**MyThread10.java**

**package** com.jspiders.multithreading.thread;

**import** com.jspiders.multithreading.resource.Resource;

**public** **class** MyThread10 **extends** Thread {

Resource resource;

**public** MyThread10(Resource resource) {

**this**.resource = resource;

}

@Override

**public** **void** run() {

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

**synchronized** (resource.res2) {

System.***out***.println(getName()+" Applied Lock On "+resource.res2);

**synchronized**(resource.res1) {

System.***out***.println(getName()+ " Applied Lock On "+resource.res1);

}

System.***out***.println(getName()+" Released "+resource.res1);

}

System.***out***.println(getName()+" Released "+resource.res2);

}

}

**Deadlock.java**

**package** com.jspiders.multithreading.main;

**import** com.jspiders.multithreading.resource.Resource;

**import** com.jspiders.multithreading.thread.MyThread10;

**import** com.jspiders.multithreading.thread.MyThread9;

**public** **class** Deadlock {

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

Resource resource = **new** Resource();

MyThread9 myThread9 = **new** MyThread9(resource);

MyThread10 myThread10 = **new** MyThread10(resource);

myThread9.setName("T9");

myThread10.setName("T10");

myThread9.start();

myThread10.start();

}

}

**================ Actual Output ==============**

T10 Applied Lock On 2nd Resource

T9 Applied Lock On 1st Resource

**DAEMON TREADS**

-In java there are two types of Thread

**1]User-Defined Threads**

**2]Daemon Threads**

-The User defined Threads are the Threads which are created by the programmer with the help of the Thread Class or Runnable Interface.

-Such User-Defined Threads have to be called for execution explicitly by using the start().

-On the other hands Daemon Thread are Predefined in Java And are called for the Execution Implicitly by the JVM whenever required.

-It means the user is not responsible to call a Daemon Thread for Execution and also its execution can be seen.

**NOTE**: The Most Important Daemon Thread in java is **Garbage Collection**.

**GARBAGE COLLECTION**

-The Garbage Collection Daemon Thread helps in java to manage the Memory.

-The Garbage Collection Daemon Thread is Responsible to remove the unusable objects from the Memory.

-It is Implicitly call by JVM in two cases i.e

**1]When an object is DeReferred**

**@100**

**A a =new A();**

**@200**

**B a =new B();**

**a = b;**

**2]When an Object is Nullified.**

**@100**

**A a =new A();**

**A=null;**

**Advantages of Garbage Collection:**

1]It makes Java efficient in Memory management.

2]We as programmer do not need to worry about the Memory Management in java.

**FILE HANDLING**

-The process of performing some operation on file is called as File Handling.

-File is an entity where we can store the data as well as we can manipulate data.

-If we are reading some data from the file in that case that file will be considered as **Source Entity**. (Retrieving the Data from file is called as Source Entity)

-If we are writing some data to the file in that case that file will be considered as **Target Entity**.(When we data is sent to the file is called as Target Entity).

-To deal file in Java we need to create an Object of a class named **File.**

**================ NOTE ================**

-Performing operation on file (CRUD OPERATION) is called as File Handling.

**OPRATIONS ON FILE**

**1]Create a File**

**2]Fetch The Information about The File**

**3]Delete The File**

**4]Write a Data to a File**

**5]Read Data From the File**

-The operation that can be perform on file in java, are as Follows:

**Get Info A File**

**Delete The File**

**Create The File**

**Read From File**

**Write to File**

-The operation of creating, deleting and getting information of a file are not depended upon the data of a file where as the operation of Read and Write depend upon the data which the file holds or can hold.

**FILE CLASS**

-File is a concreate class present inside the **java.io** package.

-We need to create an object for the File Class In order to perform

Operations on a file.

-The File Class has Overloaded constructor

-The Most commonly used constructor is one which accept a String arguments.

-The String argument is considered has the “**Path Name**” of the File.

-Generally used constructor is as below:-

**Syntax:-**

**File(String path\_name)**

**{**

**}**

**Path Name:-**

-The value for the path name can be given in two ways:-

**1]By mentioning the File Name along with its extension.**

-In this case the path will be Project Folder.

-It is also called as **Default Path** name.

**2]A User can Provide a folder path along with a File Name and Extension.**

-This Path is called as **Absolute Path.**

**1]Create a File**

**createNewFile()**

-It is Non-Static method present inside the File Class.

-This method is used to create the File based on the Path Name given in the File Class Constructor.

-This method override the Existing files when a file Same name with Exception being created.

**CreateFile.java**

**package** com.jspiders.filehandling.operations.create;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** CreateFile {

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

File file = **new** File("Demo.txt");

**try** {

file.createNewFile();

System.***out***.println("File Is Created");

} **catch** (IOException e) {

// **TODO**: handle exception

}

}

}

**================ Actual Output ==============**

File Is Created

**Checks If the File is Already Present or Not**

**CreateFile1.java**

**package** com.jspiders.filehandling.operations.create;

**import** java.io.File;

**public** **class** CreateFile1 {

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

File file = **new** File("Hello.txt");

**if** (file.exists()) {

System.***out***.println("File Is Already Exists");

}

**else** {

**try** {

file.createNewFile();

System.***out***.println("File Is Created");

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

}

}

}

**================ Actual Output ==============**

File Is Created

**Create Multiple Types Of Files**

**CreateFile1.java**

**package** com.jspiders.filehandling.operations.create;

**import** java.io.File;

**public** **class** CreateFile1 {

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

File file = **new** File("Hello.txt");

**if** (file.exists()) {

System.***out***.println("File Is Already Exists");

}

**else** {

**try** {

file.createNewFile();

System.***out***.println("File Is Created");

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

}

}

}

**================ Actual Output ==============**

File Is Created

**READING AND WRITING OPERATIONS ON FILE**

-Reading and Writing operations on a file depend on the Type of data that particular file is handling.

-The data is categorized in two types

**1]Character Stream Data**

**2]Byte Stream Data**

**1]Character Stream Data**

-To perform read operation on a file which handles Character Stream Data we used Reader Classes.

-To perform write operations on file which handles Character Stream data we use Writer Classes.

**2]Byte Stream Data**

-To Perform Read Operation on a file which handles Byte Stream Data we use Input Stream Classes.

-To perform write operations on file which handles Byte Stream data we use Output Stream Classes.

**FILE**

**Character**

**Stream**

**BYTE**

**Stream**

**Read**

**Write**

**Write**

**Read**

FileOutputStream

FileInputStream

FileReader

FileWriter

Scanner

**1]Character Stream Data**

**FileWriter**

**FileWriterDemo.java**

**package** com.jspiders.filehandling.CharacterStreamData;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** FileWriterDemo {

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

File file = **new** File("Demo.txt");

**if** (file.exists()) {

FileWriter fileWriter = **new** FileWriter(file);

fileWriter.write("Data From The Java Program");

System.***out***.println("Data Is Written To The File");

fileWriter.close();

}

**else** {

file.createNewFile();

System.***out***.println("File Is Created");

FileWriter fileWriter = **new** FileWriter(file);

fileWriter.write("Data From The Java Program");

System.***out***.println("Data Is Written To The File");

fileWriter.close();

}

}

}

**================ Actual Output ==============**

File Is Created

Data Is Written To The File

-To perform Write operation on a file we can make use of Writer Class.

**createNewFile()**

-It is Non-static Method present inside the File Class.

-It is used to create a New file in the specified folder.

**exists()**

-It is a Non-Static method present inside the file class it is used to check whether a specified file is Present or Not.

-If the File is Present it return True otherwise it return false.

FileReader

**FileReaderDemo.java**

**package** com.jspiders.filehandling.CharacterStreamData;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**public** **class** FileReaderDemo {

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

File file = **new** File("Demo.txt");

**if** (file.exists()) {

FileReader fileReader = **new** FileReader(file);

// int res = fileReader.read();

System.***out***.println("Data Is Fetched Using File Reader : "+fileReader.read());//Returns ASCII Value of the First Character

fileReader.close();

}

**else** {

System.***out***.println("File Is Not Exists");

}

}

}

**================ Actual Output ==============**

Data Is Fetched Using File Reader : 68

-To perform Read operation on a file we can make use of reader Class.

-If you want print the data from the File then look below program.

**FileReaderDemo1.java**

**package** com.jspiders.filehandling.CharacterStreamData;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.util.Scanner;

**public** **class** FileReaderDemo1 {

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

File file = **new** File("Demo.txt");

**if** (file.exists()) {

FileReader fileReader = **new** FileReader(file);

// int res = fileReader.read();

System.***out***.println("Data Is Feteched Using File Reader : "+fileReader.read());//Returns ASCII Value of the First Character

fileReader.close();

Scanner scanner = **new** Scanner(file);

// for(;scanner.hasNext();)

// {

// System.out.println("Data Fetched Using Scanner :"+scanner.nextLine());

//

// }

**while**(scanner.hasNext())//Checking the Multiple lines

{

System.***out***.println("Data Fetched Using Scanner :"+scanner.nextLine());

//String res=scanner.nextLine();

}

scanner.close();

}

**else** {

System.***out***.println("File Is Not Exits");

}

}

}

**================ Actual Output ==============**

Data Is Feteched Using File Reader : 68

Data Fetched Using Scanner :Data From The Java Program

**Byte Stream Data**

**FileOutputStream**

**FileOutputStreamDemo.java**

**package** com.jspiders.filehandling.ByteStreamData;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**public** **class** FileOutputStreamDemo {

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

File file = **new** File("ByteData.txt");

**if** (file.exists()) {

FileOutputStream fileOutputStream = **new** FileOutputStream(file);

fileOutputStream.write(1101);

System.***out***.println("Data Is Written To The File");

fileOutputStream.close();

}

**else** {

file.createNewFile();

System.***out***.println("File Is Created");

FileOutputStream fileOutputStream = **new** FileOutputStream(file);

fileOutputStream.write(20);

System.***out***.println("Data Is Written To The File");

fileOutputStream.close();

}

}

}

**================ Actual Output ==============**

File Is Created

Data Is Written To The File

**FileInputStream**

**FileInputStreamDemo.java**

**package** com.jspiders.filehandling.ByteStreamData;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**public** **class** FileInputStreamDemo {

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

File file = **new** File("ByteData.txt");

**if** (file.exists()) {

FileInputStream fileInputStream = **new** FileInputStream(file);

System.***out***.println("Data Fetched From The File : "+fileInputStream.read());

fileInputStream.close();

}

**else** {

System.***out***.println("File Is Not Present");

}

}

}

**================ Actual Output ==============**

Data Fetched From The File : 20

**SERIALIZATION AND DESERIALIZATION**

**SERIALIZATION**

It is the process of converting the Java Object into Byte Stream format is called Serialization.

-To perform serialization we use **writeObject()** method of **ObjectOutputStream Class**.

-To perform serialization we need to have object of FileOutputStream.

**DE-SERIALIZATION**

-It is the Process of converting Byte Stream data format into original Java Object.

-For De-Serialization **readObject()** method **ObjectInputStream** class will be used.

**Client Side**

**Application**

**Network**

**Server Side**

**Application**

**Byte Stream**

**Data**

**Java**

**Object**

**SERIALIZATION**

**Byte Stream Data**

**DE-SERIALIZATION**

**Importance Of Serialization and De-Serialization**

-When a Java Object travel through some network it cannot travel in its original format that Java Object has to be Converted into the Network supported format of the particular Network

-Byte Format will be consider has Network Supported format

**SERIALIZABLE INTERFACE**

-Serializable it is an **Marker** Interface.

-The Class whose object under goes Serialization and Deserialization process that class should implement Serializable interface.

**MARKER INTERFACE**

-It is an **Empty** interface which can be implemented to provide some **Special** abilities to a class.

-Marker interface is an **Empty** Interface.

QNo1.Find out All the Marker Interface present inside the java and write their purpose.

**Java Object**

**User.java**

**package** com.jspiders.Object;

**import** java.io.Serializable;

**public** **class** User **implements** Serializable {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**int** id;

String name;

String email;

String password;

**public** User(**int** id, String name, String email, String password) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.email = email;

**this**.password = password;

}

@Override

**public** String toString() {

**return** "User [id=" + id + ", name=" + name + ", email=" + email + ", password=" + password + "]";

}

}

**SERIALIZATION**

**Serialization.java**

**package** com.jspiders.Serialization;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectOutputStream;

**import** com.jspiders.Object.User;

**public** **class** Serialization {

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

File file = **new** File("Serialization.txt");

**if** (file.exists()) {

System.***out***.println("File Is Already Exist");

FileOutputStream fileOutputStream = **new** FileOutputStream(file);

ObjectOutputStream objectOutputStream = **new** ObjectOutputStream(fileOutputStream);

objectOutputStream.writeObject(**new** User(24, "Amol", "24amolpawar@gmail.com", "Amol2404@"));

System.***out***.println("Object Is Written To The File");

fileOutputStream.close();

objectOutputStream.close();

}

**else** {

file.createNewFile();

System.***out***.println("File IS Created");

FileOutputStream fileOutputStream = **new** FileOutputStream(file);

ObjectOutputStream objectOutputStream = **new** ObjectOutputStream(fileOutputStream);

objectOutputStream.writeObject(**new** User(24, "Amol", "24amolpawar@gmail.com", "Amol2404@"));

System.***out***.println("Object is Written To The File");

fileOutputStream.close();

objectOutputStream.close();

}

}

}

**================ Actual Output ==============**

File IS Created

Object is Written To The File

**DE-SERIALIZATION**

**DeSerialization.java**

**package** com.jspiders.DeSerialization;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** com.jspiders.Object.User;

@SuppressWarnings("unused")

**public** **class** DeSerialization {

@SuppressWarnings("resource")

**public** **static** **void** main(String[] args) **throws** IOException, ClassNotFoundException {

File file = **new** File("Serialization.txt");

**if** (file.exists()) {

FileInputStream fileInputStream = **new** FileInputStream(file);

ObjectInputStream objectInputStream = **new** ObjectInputStream(fileInputStream);

//Here Dynamic Method Dispatch Method Overidding(Run Time polymorphism happing)

//Object class contain toString() and child class has also toString() method

// Object object = objectInputStream.readObject();

// System.out.println(object);

//Downcasting from ObjecInputStream to user Class

User user =(User) objectInputStream.readObject();

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

System.***out***.println("Object is Read From The File");

objectInputStream.close();

fileInputStream.close();

}

**else** {

System.***out***.println("File Does Not Exists");

}

}

}

**================ Actual Output ==============**

User [id=24, name=Amol, email=24amolpawar@gmail.com, password=Amol2404@]

Object is Read From The File

**DESIGN PATTERNS**

-Design patterns are the predefined structure which can be used to avoid the recurring development issues.

-Using Design Patterns, we can avoid the Development issues but we unable to solve Development issues.

-Based on the type of issue that the Design Patterns are dealing with they are classified into two types

**DESIGN PATTERNS CATLOG**

-It is place where Design Patterns are available.

-A newly discovered design pattern can we added to the Design Patterns catalog.

-Based on the Type of issues with which a particular Design Pattern is dealing with, Design patterns are classified into several type.

-Widely Used Design Patterns are

**1]Creational Design Pattern**

-This Design Pattern deal with the Requirements or the issues associated with object Creation in an Application.

**A]Singleton Design Pattern**

**B]Factory Design Pattern**

**C]Builder Design Pattern**

**2]Structural Design Pattern**

-This Design patterns the requirements or the issues associated with the Structure of Classes an interfaces in an application.

-Widely used Structural Design Pattern is

**A]Adapter Design Pattern**

**1]Creational Design Pattern**

-This Design Pattern deal with the Requirements or the issues associated with object Creation in an Application.

-Widely used Creational design patterns are:

**A]Singleton Design Pattern**

**B]Factory Design Pattern**

**C]Builder Design Pattern**

**A]Singleton Design Pattern**

-Singleton design pattern can be used to restrict the user from creating Multiple Objects of a Particular class.

-In Singleton design pattern we are going to make the constructor of particular class as **Private** and in order to access that Private Constructor we define One **Helper Method**.

-To access the that Constructor outside the class we can implement one Helper method.

**Singleton.java**

**package** com.jspiders.designpatterns.creational.Singleton;

**public** **class** Singleton {

//Here we Created the Object Reference static

**private** **static** Singleton *singleton*;

**private** Singleton() {

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

}

// Here we declare the Static method because we did not create the Object

// Constructor is Private

**static** **public** Singleton getObject()

{

//if Object Reference is null then Create the object

**if** (*singleton* == **null**) {

*singleton* = **new** Singleton();

}

//if object is Already created then return the Reference

**return** *singleton*;

}

}

//Singleton Means Restrict the user to create the Multiple Object.

**SingletonMain.java**

**package** com.jspiders.designpatterns.creational.Singleton;

**public** **class** SingletonMain {

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

//// 1st Way

//// Singleton.getObject();

//// System.out.println(Singleton.getObject());

//

//2nd Way

Singleton object1 = Singleton.*getObject*();

Singleton object2 = Singleton.*getObject*();

Singleton object3 = Singleton.*getObject*();

System.***out***.println(object1);

System.***out***.println(object2);

System.***out***.println(object3);

}

}

**===================Output =================**

com.jspiders.designpatterns.creational.Singleton.Singleton@6d06d69c

com.jspiders.designpatterns.creational.Singleton.Singleton@6d06d69c

com.jspiders.designpatterns.creational.Singleton.Singleton@6d06d69c

-The feature of the Single Design Pattern is to restrict the user from creating multiple object.

**Step1:-**

-For That You Have to create object reference variable as an **private static** for the class. Hence you can directly access in the class only.

**Step2:-**

-Then You have to declare the Constructor as an **Private** to restrict the user create an Object. If you declare the constructor as an Private the user cannot use the **new** keyword with Constructor.

**Step3:-**

-Create Helper method for the class write condition such like

//if Object Reference is null then Create the object

**if** (*singleton* == **null**) {

*singleton* = **new** Singleton();

}

//if object is Already created then return the Reference

**return** *singleton*;

**Step4:-**Try to create to access the helper method from other class because we have write the logic in the helper method.

Singleton object1 = Singleton.*getObject*();

Singleton object2 = Singleton.*getObject*();

Singleton object3 = Singleton.*getObject*();

System.***out***.println(object1);

System.***out***.println(object2);

System.***out***.println(object3);

-It will return same address all the time because if **objectreference**  is **null**  then only it will create the object Once after that it will **return** same address.

-Singleton Design Pattern classified into Two Types:

**i]Lazy Instantiation**

**ii]Eager Instantiation**

**i]Lazy Instantiation**

-In Lazy Instantiation we will declare object Reference variable of A Particular class but we don’t initialized with it.

-We will initialize that object Reference variable whenever user will request for an Object of a Class.

**SingletonLazy.java**

**package** com.jspiders.designpatterns.creational.Singleton.Lazy;

**public** **class** SingletonLazy {

//Lazy means We have not initialized the Object Reference variable FIrstly

**private** **static** SingletonLazy *singletonLazy*;

**private** SingletonLazy() {

}

// Here we declare the Static method because we did not create the Object

// Constructor is Private

**static** **public** SingletonLazy getObject()

{

//if Object Reference is null then Create the object

**if** (*singletonLazy* == **null**) {

*singletonLazy* = **new** SingletonLazy();

}

//if object is Already created then return the Reference

**return** *singletonLazy*;

}

}

**SingletonLazyMain.java**

**package** com.jspiders.designpatterns.creational.Singleton.Lazy;

**public** **class** SingletonLazyMain {

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

SingletonLazy object1 = SingletonLazy.*getObject*();

SingletonLazy object2 = SingletonLazy.*getObject*();

SingletonLazy object3 = SingletonLazy.*getObject*();

System.***out***.println(object1);

System.***out***.println(object2);

System.***out***.println(object3);

}

}

**===================Output =================**

com.jspiders.designpatterns.creational.Singleton.Lazy.SingletonLazy@6d06d69c

com.jspiders.designpatterns.creational.Singleton.Lazy.SingletonLazy@6d06d69c

com.jspiders.designpatterns.creational.Singleton.Lazy.SingletonLazy@6d06d69c

-In SingletonLazy we don’t initialize the Object Reference of Class First.

-Lazy mean (Taking More Time ).

**ii]Eager Instantiation**

-In Eager Instantiation we will declare the Object Reference Variable of particular class and we initialize it so by doing this we are keeping it an Object of a class Ready and we will return the Same Object to the User whenever requested.

**SingletonEager.java**

**package** com.jspiders.designpatterns.creational.Singleton.Eager;

**public** **class** SingletonEager {

**private** **static** SingletonEager *singletonEager* = **new** SingletonEager();

**private** SingletonEager() {

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

}

**public** **static** SingletonEager getObject()

{

**return** *singletonEager*;

}

}

**SingletonLazyMain.java**

**package** com.jspiders.designpatterns.creational.Singleton.Eager;

**public** **class** SingletonEagerMain {

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

SingletonEager object1 = SingletonEager.*getObject*();

SingletonEager object2 = SingletonEager.*getObject*();

SingletonEager object3 = SingletonEager.*getObject*();

System.***out***.println(object1);

System.***out***.println(object2);

System.***out***.println(object3);

}

}

**==================Output ====================**

com.jspiders.designpatterns.creational.Singleton.Eager.SingletonEager@6d06d69c

com.jspiders.designpatterns.creational.Singleton.Eager.SingletonEager@6d06d69c

com.jspiders.designpatterns.creational.Singleton.Eager.SingletonEager@6d06d69c

-Eager Means it does not take time to initialize.

-Eager Mean At the First Time You Initialize you are Object Reference Variable.

**B]Factory Design Pattern**

-This design pattern is used to create the Object whenever user will request for an object.

-If an application contain several Classes an the Object for all the Classes need to be created.

-But there is chance that some of the created object will be left unused this leads to wastage of memory.

-This problem can be avoided using factory Design Pattern.

-With the help of this factory design pattern we can create objects during **Runtime**[Program Execution].

**Beverage.java**

**package** com.jspiders.designpatterns.creational.Factory;

**public** **interface** Beverage {

**void** order();

}

-The Beverage it is an interface.

**NormalTea.java**

**package** com.jspiders.designpatterns.creational.Factory;

**public** **class** NormalTea **implements** Beverage{

@Override

**public** **void** order() {

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

System.***out***.println("Normal Tea is Ordered.");

}

}

**GreenTea.java**

**package** com.jspiders.designpatterns.creational.Factory;

**public** **class** GreenTea **implements** Beverage{

@Override

**public** **void** order() {

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

System.***out***.println("Green Tea is Ordered.");

}

}

**BlackTea.java**

**package** com.jspiders.designpatterns.creational.Factory;

**public** **class** BlackTea **implements** Beverage {

@Override

**public** **void** order() {

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

System.***out***.println("Black Tea Is Ordered.");

}

}

**IceTea.java**

**package** com.jspiders.designpatterns.creational.Factory;

**public** **class** IceTea **implements** Beverage{

@Override

**public** **void** order() {

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

System.***out***.println("Ice Tea is Ordered.");

}

}

**FactoryDesignPattern.java**

**package** com.jspiders.designpatterns.creational.Factory;

**import** java.util.Scanner;

**public** **class** FactoryDesignPattern {

**private** **static** Beverage *beverage*;

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

**try** {

//Beverage beverage=factory();

//beverage.order();

*factory*().order();

} **catch** (Exception e) {

// **TODO**: handle exception

}

}

**public** **static** Beverage factory() {

System.***out***.println("Select the Tea Order.");

System.***out***.print("1.Normal Tea \n2.Black Tea \n3.Green Tea \n4.Ice Tea\n===>");

**try**(Scanner scanner = **new** Scanner(System.***in***)){

**int** choice = scanner.nextInt();

**switch**(choice) {

**case** 1 :

*beverage* = **new** NormalTea();

**break**;

**case** 2 :

*beverage* = **new** BlackTea();

**break**;

**case** 3 :

*beverage* = **new** GreenTea();

**break**;

**case** 4 :

*beverage* = **new** IceTea();

**break**;

**default** :

System.***out***.println("Invaid Choice");

}

}

**return** *beverage*;

}

}

**===================Output ===================**

Select the Tea Order.

1.Normal Tea

2.Black Tea

3.Green Tea

4.Ice Tea

===>1

Normal Tea is Ordered.

**C]Buider Design Pattern**

-Builder Design pattern is used to create Complex object.

**Complex Object:**

-If the object has to many Properties That object is called as Complex Object.

-It not Possible to create the Complex Object using **All Arguments Constructor** since the object of to many properties and programmer has to remember the Datatype of Each Arguments and the sequence in which the constructor accepting the arguments.

-To overcome this problem, we can make use of **Builder Design Pattern** one intermediate builder class will be there which is aware of all the properties of complex object.

-In this Builder Design Pattern we need to take the help of another class called as Builder class which has access to the constructor of complex class an which is aware of all the Properties Complex Objects.

**Contact.java**

**package** com.jspiders.designpatterns.creational.Builder;

**public** **class** Contact {

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

**private** String firstName;

**private** String lastName;

**private** **long** mobileNo;

**private** **int** age;

**private** String gender;

**private** String dob;

**private** String emailString;

**private** String address;

**private** **long** pinCode;

**private** **long** home;

**private** **long** work;

**private** **long** landline;

**public** Contact(**int** id, String firstName, String lastName, **long** mobileNo, **int** age, String gender, String dob,

String emailString, String address, **long** pinCode, **long** home, **long** work, **long** landline) {

**super**();

**this**.id = id;

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.mobileNo = mobileNo;

**this**.age = age;

**this**.gender = gender;

**this**.dob = dob;

**this**.emailString = emailString;

**this**.address = address;

**this**.pinCode = pinCode;

**this**.home = home;

**this**.work = work;

**this**.landline = landline;

}

@Override

**public** String toString() {

**return** "Contact [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", mobileNo=" + mobileNo

+ ", age=" + age + ", gender=" + gender + ", dob=" + dob + ", emailString=" + emailString + ", address="

+ address + ", pinCode=" + pinCode + ", home=" + home + ", work=" + work + ", landline=" + landline

+ "]";

}

}

**ContactBuilder.java**

**package** com.jspiders.designpatterns.creational.Builder;

**public** **class** ContactBuilder {

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

**private** String firstName;

**private** String lastName;

**private** **long** mobile;

**private** **int** age;

**private** String gender;

**private** String dobString;

**private** String email;

**private** String address;

**private** **int** pincode;

**private** **long** home;

**private** **long** work;

**private** **long** landline;

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

**return** id;

}

/\*\*

\* **@param** id the id to set

\*/

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

**this**.id = id;

**return** **this**;

}

/\*\*

\* **@param** firstName the firstName to set

\*/

**public** ContactBuilder setFirstName(String firstName) {

**this**.firstName = firstName;

**return** **this**;

}

/\*\*

\* **@param** lastName the lastName to set

\*/

**public** ContactBuilder setLastName(String lastName) {

**this**.lastName = lastName;

**return** **this**;

}

/\*\*

\* **@param** mobile the mobile to set

\*/

**public** ContactBuilder setMobile(**long** mobile) {

**this**.mobile = mobile;

**return** **this**;

}

/\*\*

\* **@param** age the age to set

\*/

**public** ContactBuilder setAge(**int** age) {

**this**.age = age;

**return** **this**;

}

**public** ContactBuilder setGender(String gender)

{

**this**.gender=gender;

**return** **this**;

}

**ContactBuilder.java**

/\*\*

\* **@param** dobString the dobString to set

\*/

**public** ContactBuilder setDobString(String dobString) {

**this**.dobString = dobString;

**return** **this**;

}

/\*\*

\* **@param** email the email to set

\*/

**public** ContactBuilder setEmail(String email) {

**this**.email = email;

**return** **this**;

}

/\*\*

\* **@param** address the address to set

\*/

**public** ContactBuilder setAddress(String address) {

**this**.address = address;

**return** **this**;

}

/\*\*

\* **@param** pincode the pincode to set

\*/

**public** ContactBuilder setPincode(**int** pincode) {

**this**.pincode = pincode;

**return** **this**;

}

/\*\*

\* **@param** home the home to set

\*/

**public** ContactBuilder setHome(**long** home) {

**this**.home = home;

**return** **this**;

}

/\*\*

\* **@param** work the work to set

\*/

**public** ContactBuilder setWork(**long** work) {

**this**.work = work;

**return** **this**;

}

/\*\*

\* **@param** landline the landline to set

\*/

**public** ContactBuilder setLandline(**long** landline) {

**this**.landline = landline;

**return** **this**;

}

**public** Contact buildContact()

{

// Contact contact = new Contact(id, firstName, lastName, mobile, age, gender, dobString, email, address, pincode, home, work, landline);

// return contact;

**return** **new** Contact(id, firstName, lastName, mobile, age, gender, dobString, email, address, pincode, home, work, landline);

}

}

**BuilderDesignPattern.java**

**package** com.jspiders.designpatterns.creational.Builder;

**public** **class** BuilderDesignPattern {

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

Contact contact = **new** ContactBuilder().setFirstName("Amol").setLastName("Pawar").buildContact();

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

}

}

**===================Output ===================**

Contact [id=0, firstName=Amol, lastName=Pawar, mobileNo=0, age=0, gender=null, dob=null, emailString=null, address=null, pinCode=0, home=0, work=0, landline=0]

**2]Structural Design Patterns**

-This Design patterns the requirements or the issues associated with the Structure of Classes an interfaces in an application.

-Widely used Structural Design Pattern is

**A]Adapter Design Pattern**

**1]ADAPTER DESIGN PATTERN**

-This design pattern is used to adapt Properties of One Entity and Behaviour of Another Entity using One Intermediate Class called as Adapter Class.

-This Adapter Class is going to inherit the Properties of One Class and Implement the Behaviour of One Interface.

**Events.java**

**package** com.jspiders.designpatterns.structural.Adapter;

**public** **interface** Events {

**void** womensDay();

**void** mensDay();

}

**Employee.java**

**package** com.jspiders.designpatterns.structural.Adapter;

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

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

**private** String name;

**private** String email;

**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;

}

}

**Employee.java**

**package** com.jspiders.designpatterns.structural.Adapter;

**public** **class** EmployeeAdapter **extends** Employee **implements** Events{

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

EmployeeAdapter employeeAdapter = **new** EmployeeAdapter();

employeeAdapter.womensDay();

employeeAdapter.mensDay();

}

@Override

**public** **void** womensDay() {

EmployeeAdapter employeeAdapter = **new** EmployeeAdapter();

employeeAdapter.setId(1);

employeeAdapter.setName("Radha");

employeeAdapter.setEmail("radha@gmail.com");

System.***out***.println("Chief guest for the women's day is " + employeeAdapter.getName());

}

@Override

**public** **void** mensDay() {

EmployeeAdapter employeeAdapter = **new** EmployeeAdapter();

employeeAdapter.setId(1);

employeeAdapter.setName("Ramesh");

employeeAdapter.setEmail("ramesh@gmail.com");

System.***out***.println("Chief guest for the men's day is " + employeeAdapter.getName());

}

}

**===================Output ===================**

Chief guest for the women's day is Radha

Chief guest for the men's day is Ramesh

**JDBC**

-JDBS Stands for **Java Database Connectivity**.

-It is used to establish communication between A Java Application and Database Application.

-It is required because java application understand Java Programming Language where as the Database Application understand only Query Language.

-As both the Applications are incompatible to understand each other ,hence JDBC needs to acts as Mediator between them.

-JDBC is capable understanding and processing Java Language as well as Query Language.

**JAVA**

**APPLICATION**

**J**

**D**

**B**

**C**

**DATABASE**

**APPLICATION**

Java Query

Java Query

-It is Only technology which can be used to connect any Java Application[Application which is created using Java Programming Language] **with any** database Application.

-JDBC can be called as **Java API**.

-**API[Application Programming Interface]**

-API Stands for Application Programming Interface.

-it is Software which can be used by one Application to communicate with another application.

-There is no other technology other than JDBC which can help us interact between Java Application And Database.

-Since Java Application cannot understand Query Language and Database Application does not understand Java Programming language.

-That means there are Incompatible to each other.

-If Java Application has to be connected, we have to make the use of JDBC API which converts Java Commands into Query Command vice-versa.

-Need for Storing the data outside the java Application.

-We cannot store any data within the application permanently but we want that data in future the data has to be store Outside the Application.

**==================== NOTE ==================**

-JDBC is package present inside Java System Library has **Java.sql** package.

**Database**

**Application**

**Java**

**Application**

**DRIVER**

**JDBC**

-JDBC is a part of Java Application so that it cannot communicate directly Database Application.

-So that JDBC will make use of a driver to communicate with database application.

**DRIVER**

-It is software which acts as bridge between Java Application and Database Application.

-It is external resource which we have to add to our Java Application.

-It is an implementing class of Driver interface .

-The qualified name of Driver Class is **com.mysql.cj.jdbc.Driver**

**=================== NOTE====================**

-Each and every Database Application will have its own dedicated Driver Software

**Prerequisites JDBC**

-If we want to connect with the database application we have to add the connector software for the respective database application to our Java Application.

-In order to Access the Classes and Interface required for JDBC Operation we need to add an External **JAR** File “**MySqlConnetor/J**”

-Follow the below the steps to add the required JAR file to a Java Project.

1.Go to Google and Search for **‘Maven Repository’**

2.Go to the First Link in the Search Result.

3.In Maven Repository Search for ‘**MySql-Connector/J**’

4.Select the Version similar to the version installed in Your System.

5.From the Selected version download the JAR file.'

6.Go to the Project In Eclipse Right Click on the Project Name.

7.In the Drop Down go to ‘**Build Path**’.

8.And Then go to ‘**Configure Build Path**’.

9.In the Next window go to the ‘**Libraries Tab**’ and click on ‘**Add External JARS Button**’.

10.Open the downloaded JARS file and then click on Apply & Close button.

-After the JAR file is added externally we can see a section ‘**Reference Libraries**’ added to the Project Structure.

**=============== NOTE ================**

-Adding JAR file externally into a Project, increase the load on the Build Path of that Project.

-This Results in Delayed Build Process.

**1.Loads the Driver Class**

**2.Open Connection**

**3.Create/Prepare Statement**

**4.Process the Result**

**5.Close Connection.**

**6 Steps of JDBC**

**1.LOADS THE DRIVER CLASS**

-For loading the Driver class we need to take help of **forName()**.

-It is a static method from predefined class Java Named ‘**Class**’.

-This method can be used to load any class with the help of its path.

-The path of the class is given as a String Argument to the Method.

**Driver Class Path**

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

**OR**

**1.Register The Driver**

Driver driver = new Driver();

DriverManager.registerDriver(driver);

**Example:-**

// 1.Loads The Driver

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

// or

// 1.Register The Driver

Driver driver = **new** Driver();

DriverManager.*registerDriver*(driver);

**2.OPEN CONNECTION**

-In order to open the connection between the Java Application and Database Application we need to take help of an overloaded method called as ‘**getConnection()**’

-It is a static method present in the class named ‘**DriverManager**’

-The connection can be established in Three different ways.

================ **GetConnection()** ==============

1.getConnection (String url);

2.getConnection (String url**,** Properties info);

3.getConnection (String url**,** String user**,** String Password);

**Example:-**

Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/j2ee?user=root&&password=root");

-To establish the connection we need to provide the Database url(**DB URL**)

**DB URL Format :**

protocol : sub-protocol :// host\_name : port\_no/db\_name?user=**username**&password=**password**

**Protocol**

-The protocol defines which technology is being used for the **url**

**Sub-Protocol**

-The sub-protocol defines the technology for which technology defined in protocol is used.

**Host\_Name**

-It defines that the database application is located in which System exactly.

**Port\_No**

-If the Database application is present in the Same System as that of the Java Application, then the Host\_Name declared as ‘**localhost**’.

-But if the database Application is some different system then the

Host\_Name as the ‘**IP Address**’ of that System.

-It defines the exact location of Database Application inside the System identified by Host\_Name.

**Database Name[DB Name]**

-It defines that which database is currently under operation.

**User**

-It defines the user name of the accounts in which the database is present.

**Password**

-It defines the password of the Accounts that is being used.

**Eaxmple**

jdbc **:** mysql **:** **//**localhost:3306**/**weja2**?**user**=**root**&&**password**=**root

**3.CREATE / PREAPARE STATEMENT**

// 3.Create / Prepare Statement

Statement statement = connection.createStatement();

String query = "insert into students values(01,'Amol','Pawar',9373428581,'24amolpawar@gmail.com')";

statement.executeUpdate(query);

**4.PROCESS THE RESULT**

// 4.Process The Result

System.***out***.println("Row Is Inerted");

**5.CLOSE THE CONNECTION**

// 5.Close The Connection

connection.close();

statement.close();

**6.DE-REGISTER THE DRIVER**

// 6.De-Register Driver

DriverManager.*deregisterDriver*(driver);

**PROGRAM FOR INSERTING ROW IN MYSQL FOR DEMO**

**Type1.java**

**package** com.jspiders.jdbc.JDBC\_STEPS;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** com.mysql.cj.jdbc.Driver;

**public** **class** Type1 {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, SQLException {

// 1.Loads The Driver

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

// or

// 1.Register The Driver

Driver driver = **new** Driver();

DriverManager.*registerDriver*(driver);

// 2.Open The Connection

Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/j2ee?user=root&&password=root");

// 3.Create / Prepare Statement

Statement statement = connection.createStatement();

String query = "insert into students values(01,'Amol','Pawar',9373428581,'24amolpawar@gmail.com')";

statement.executeUpdate(query);

// 4.Process The Result

System.***out***.println("Row Is Inserted");

// 5.Close The Connection

connection.close();

statement.close();

// 6.De-Register Driver

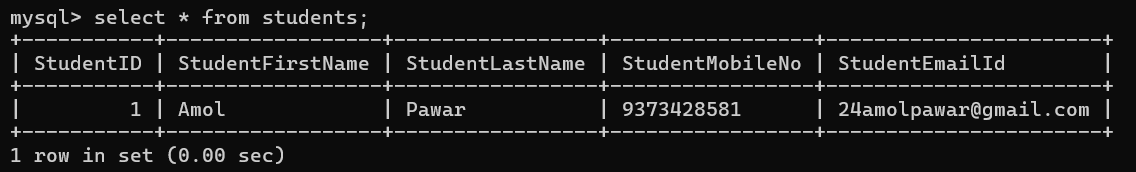
DriverManager.*deregisterDriver*(driver);

}

}

**===================OUTPUT ==================**

Row Is Inserted



**DRIVER**

-It is an implementing class of Driver interface .

-The qualified name of Driver Class is **com.mysql.cj.jdbc.Driver**

**url**

-It is the path of databases application with which a java Application is Connected.

Format: [Ip Address]

**API:database\_app\_name://localhost:port\_number/[database\_name]**

**API**

-It represents the technology used for the Database connection.

**Database Application Name**

-it represents the name of the Database Application with which Java Application is to be connected.

**LocalHost[Ip Address]**

-It represents the address of the location of a system[Computer] inside which the database application is present.

**Port Number**

-It represent the exact location of the database application inside the recognize system.

**[Database Name]**

-It represent the name of the Database inside the database application.

-It is an optional in **url.**

url for MySql Database Application:

jdbc : mysql : //localhost:3306/weja2?user=root&&password=root

**TYPE 1**

**CODE STANDARDLIZATION**

**Type1.java**

**package** com.jspiders.jdbc.JDBC\_STEPS;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** com.mysql.cj.jdbc.Driver;

**public** **class** Type1 {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, SQLException {

// 1.Loads The Driver

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

// or

// 1.Register The Driver

Driver driver = **new** Driver();

DriverManager.*registerDriver*(driver);

// 2.Open The Connection

Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/j2ee?user=root&&password=root");

// 3.Create / Prepare Statement

Statement statement = connection.createStatement();

String query = "insert into students values(01,'Amol','Pawar',9373428581,'24amolpawar@gmail.com')";

statement.executeUpdate(query);

// 4.Process The Result

System.***out***.println("Row Is Inserted");

// 5.Close The Connection

connection.close();

statement.close();

// 6.De-Register Driver

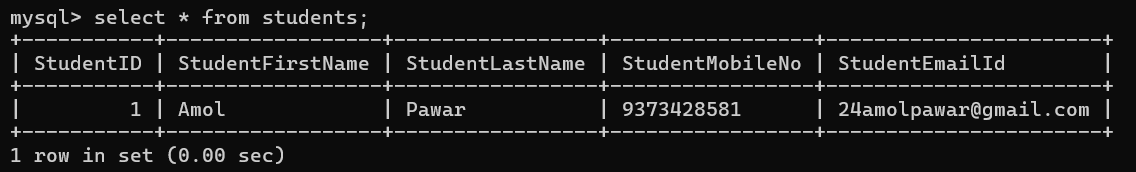
DriverManager.*deregisterDriver*(driver);

}

}

**===================OUTPUT ==================**

Row Is Inserted



**Type2.java**

**package** com.jspiders.jdbc.JDBC\_STEPS;

**import** java.sql.Connection;

**import** java.sql.Driver;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CODE STANDARDLIZATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **class** Type2 {

**private** **static** Driver *driver*;

**private** **static** Statement *statement*;

**private** **static** Connection *connection*;

**private** **static** String *query*;

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

**try** {

// 1.Register The Driver

// 2.Open The Connection

*connection* = *openConnection*();//Here it will directly return the connection reference

// 3.Create / Prepare The Statement

*statement* = *connection*.createStatement();

*query* = "INSERT into Students Values(2,'Rohit','Nalwade','9374798765','rohit@gmail.com')";

//4.Process The Result

**int** row = *statement*.executeUpdate(*query*);

System.***out***.println(row+" row(s) affected");

} **catch** (Exception e) {

e.printStackTrace();

}

**finally** {

**try** {

*closeConnection*();

} **catch** (Exception e2) {

// **TODO**: handle exception

e2.printStackTrace();

}

}

}

**private** **static** Connection openConnection() **throws** SQLException {

// 1.Register The Driver

*driver* = **new** com.mysql.cj.jdbc.Driver();

DriverManager.*registerDriver*(*driver*);

// 2.Open The Connection

**return** DriverManager.*getConnection*("jdbc:mysql://localhost:3306/j2ee","root","root");

}

**private** **static** **void** closeConnection() **throws** SQLException {

**if** (*connection* != **null**) {

*connection*.close();

}

**if** (*statement* != **null**) {

*statement*.close();

}

DriverManager.*deregisterDriver*(*driver*);

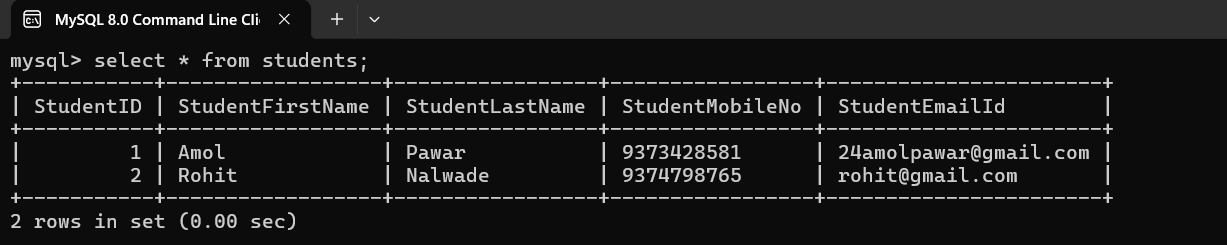
}

}

**TYPE 2**

**===================OUTPUT ==================**

**1 row(s) affected**



**Type3.java**

**package** com.jspiders.jdbc.JDBC\_STEPS;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** com.mysql.cj.jdbc.Driver;

//############################# CODE STANDARDLIZATION ##################################

//If you don't pass the database name in the connection properties then

//you can pass in the query itself.

//databaseName.TableName

//"INSERT into weja3.Student Values(...)"

**public** **class** Type3 {

**private** **static** Driver *driver*;

**private** **static** Statement *statement*;

**private** **static** Connection *connection*;

**private** **static** String *query*;

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

**try** {

// 1.Register The Driver

// 2.Open The Connection

*connection* = *openConnection*();

// 3.Create / Prepare the Statement

*statement* = *connection*.createStatement();

//If you does not provide the database then you have pass the databaseName.

//Here for accessing the database into query

*query* = "INSERT into j2ee.Students values (3,'Ganesh','Gidde',9370049100,'giddeganesh07@gmail.com')";

// 4.Process the Result

**int** row = *statement*.executeUpdate(*query*);

System.***out***.println(row+" row(s) affected");

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

**finally** {

*closeConnection*();

}

}

**private** **static** Connection openConnection() **throws** SQLException {

//1.Register The Driver

*driver* = **new** Driver();

DriverManager.*registerDriver*(*driver*);

//2.Open The Connection

**return** DriverManager.*getConnection*("jdbc:mysql://localhost:3306","root","root");

}

**private** **static** **void** closeConnection() **throws** SQLException {

**if** (*connection* != **null**) {

*connection*.close();

}

**if** (*statement* != **null**) {

*statement*.close();

}

DriverManager.*deregisterDriver*(*driver*);

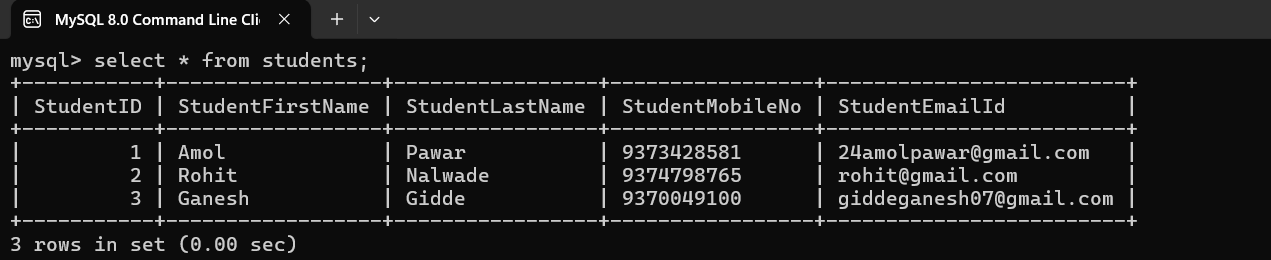
}

}

**TYPE 3**

**===================OUTPUT ==================**

**1 row(s) affected**



**getConnection()**

-it is a static method present inside the Driver Class.

-It is used to established connection for jdbc.

-It is responsible to return the Object of Connection Interface.

**1st Way Of Connection**

package com.jspiders.jdbc.select;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class SelectDemo2 {

private static Connection connection;

private static Statement statement;

private static ResultSet resultSet;

private static String driverPath = "com.mysql.cj.jdbc.Driver";

private static String dburl = "jdbc:mysql://localhost:3306/"

+ "weja2?user=root&password=root";

private static String query;

public static void main(String[] args) {

try {

//1. Load Driver class

Class.forName(driverPath);

//2. Open Connection

connection = DriverManager.getConnection(dburl);

//3. Create Statement

statement = connection.createStatement();

query = "select \* from student";

resultSet = statement.executeQuery(query);

//4. Process the result

while (resultSet.next()) {

System.out.println(resultSet.getString(1) + " || "+ resultSet.getString(2));

}

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

} finally {

//5. Close connection

try {

if (connection != null) {

connection.close();

}

if (statement != null) {

statement.close();

}

if (resultSet != null) {

resultSet.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

**2nd Way Of Connection**

**SelectDemo2.java**

package com.jspiders.jdbc.select;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class SelectDemo3 {

private static Connection connection;

private static Statement statement;

private static ResultSet resultSet;

private static String driverPath = "com.mysql.cj.jdbc.Driver";

private static String dburl = "jdbc:mysql://localhost:"

+ "3306/weja2";

private static String user = "root";

private static String password = "root";

private static String query;

public static void main(String[] args) {

try {

// 1. Load the Driver class

Class.forName(driverPath);

// 2. Open connection

connection = DriverManager.getConnection (dburl, user, password);

// 3. Create Statement

statement = connection.createStatement();

query = "select \* from student";

resultSet = statement.executeQuery(query);

// 4. Process the result

while(resultSet.next()) {

System.out.println(resultSet.getString(1) + " | "+ resultSet.getString(2) + " | "

+ resultSet.getString(3) + " | "+ resultSet.getString(4));

}

} catch (ClassNotFoundException | SQLException e)

{

}

finally

{

try {

// 5. Close connection

if (connection != null) {

connection.close();

}

if (statement != null) {

statement.close();

}

if (resultSet != null) {

resultSet.close();

}

} catch (SQLException e)

{

}

}

}

}

**3rd Way Of Connection**

-In the previous two ways of connection, the Database connection(Username&password) were exposed in the Program.

-Hence it may lead to the illegal Access of are Data.

-In order to avoid such Security issue we can use the 3rd way of Connection in which the Database Credentials are not exposed in the program.

-Instead we will declare the Database Credentials in “**.properties**” file.

-This file can be loaded in an Object of Properties class.

And then this object can be passed as an argument to the getConnection() along with the **dburl**.

-In this case the **dburl** will not contain the Username and Password.

**Note:-**The Database Credentials are declared in the properties files in the format of **key value** pair.

**Creating Properties File**

1.Create a new General Folder under the Project folder Directly.

The Name of This older will be “**resources**”.

2.Under the resources folder create a new **General File**.

The name of this file will be “**db\_info**” Aand it will be mandatory to provide the extension as “**.properties**”

3.In the properties file , we will declare the Database credentials as follows

**user=root**

**password=root**

**Note:** The username and password are Implicitly loaded from the properties Object.

package com.jspiders.jdbc.select;

import java.io.FileInputStream;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.Properties;

public class SelectDemo4 {

private static Connection connection;

private static Statement statement;

private static ResultSet resultSet;

private static FileInputStream file;

private static Properties properties = new Properties();

private static String dburl = "jdbc:mysql://localhost:"

+ "3306/weja2";

private static String driverPath = "com.mysql.cj.jdbc.Driver";

private static String filePath = "C:\\Users\\Dhananjay\\Desktop"

+ "\\J2EE Workspaces\\WEJA2\\jdbc"

+ "\\resources\\db\_info.properties";

private static String query;

public static void main(String[] args) {

try {

Class.forName(driverPath);

file = new FileInputStream(filePath);

properties.load(file);

connection = DriverManager.getConnection

(dburl, properties);

statement = connection.createStatement();

query = "select \* from student";

resultSet = statement.executeQuery(query);

while(resultSet.next()) {

System.out.println(resultSet.getString(1) + " | "

+ resultSet.getString(2) + " | "

+ resultSet.getString(3) + " | "

+ resultSet.getString(4));

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if (connection != null) {

connection.close();

}

if (statement != null) {

statement.close();

}

if (resultSet != null) {

resultSet.close();

}

if (file != null) {

file.close();

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

**execute()**

-It is Non-Static(abstract) method present inside the **Statement** interface.

-It return the **boolean** type of values.

-We can execute **All** types statements(**DDL,DML,DQL**) using execute().

-It will return **true** if we execute **DQL** statement(**Select**).

-Otherwise for all other statement it will return the **false**.

**executeUpdate()**

-It is a Non-Static(abstract) method present inside the **Statement** interface.

-it return the **Integer** type of data.

-It return the number of the Rows affected in the table after executing the **DML** commands(**Insert,Update,Delete**).

-We can execute only DML commands using executeUpdate().

**executeQuery()**

-It is Non-Static(abstract) method present inside the **Statement** interface.

-It return the **Object** of type **ResultSet**.

-executeQuery() method is used to execute the **DQL** (**Select**)statements.

**next()**

-It is Non-Static method present in ResultSet Interface.

-It is used to check if a next object / value is present or not.

-If present, then this method is returns “**true**” as the output else it will return “**false**”.

**get[DataType]()**

-From the resultSet object we can retrieve any value according to its Datatype based on its column position(ColumnIndex).

-The methods that are available for this operation are

**getString(),getInt,getLong(),getDouble(),** etc.

-This methods accept an Integer argument which is consider as the Column Index.

**load()**

-It is Non-Static method present in **Properties** class.

-It is used to load the content from the input file as properties for the project.

**getProperty()**

-It is No-Static method from the Properties class which is used to retrieve the value of a Property based on its Key.

-This method accept a String argument which is consider as the Key of Property.

**ResultSet**

-ResultSet it is an interface.

-The ResultSet interface holds entire data in the database.

**Statement**

**PreparedStatement**

Note :- We can also execute the Dynamic query using Statement interface and Static query using Prepared Statement.

|  |  |  |
| --- | --- | --- |
| SrNo | Statement | Prepared Statement |
| 1 | Statement is a Parent interface of Prepared Statement Interface | Prepared Statement is a Child interface of Statement interface |
| 2 | While creating the Statement no need of passing the query. | While Preparing the statement we need to pass the query. |
| 3 | Methods for execution will take a query as an argument. | Method for execution will not take query as an argument. |
| 4 | Statement is not secured while executing Dynamic Queries. | Prepared Statement is Secured while executing the Dynamic queries. |
| 5 | Statement is slower compare to Prepare Statement. | Prepared Statement is faster compare to Statement. |
| 6 | In static query the values required for the execution are Coded. | In Dynamic query the values required for execution can be represented as a Place Holder |
| 7 | The values are presented in the query before its gets compiled | The query can be compiled without the actual values |
| 8 | The query is compiled at time of Execution | The Query is compiled and kept ready before it is required to be executed |
| 9 | To change the values  Changing the Query is required | The Values can be changed without changing the query. |
| 10 | In static each time new values are required then the query will be compiled for each time | The query can be compiled once and can be executed multiple times with new values. |
| 11 | To handle static queries we can use statement Interface or Prepared Statement Interface | We can only use Prepared Statement Interface to Handle Dynamic Queries |

**STATIC QUERY**

-The query that hold or contain all the required values before compilation are known as static queries.

-In JDBC the static queries are compiled at the time of execution of execution itself.

-To handle static queries in jdbc, we can ,make use of the statement Interface

-the prepared statement Interface can also handle static queries

-It is because the Prepared statement Interface is a child of Statement Interface.

**DYNAMIC QUERIES**

-The queries which can be compiled without the actual values are known as Dynamic Queries.

-The Queries are compiled by the database and kept reday for the Execution.

-In the Dynamic Query we need to reserve the place for actual values so that is satisfy or follows the required syntax this is done with the help of “**Placeholder**”(”**?**”).

-A placeholder in the query represents that a values will be assigned in its position before the executing the query

-in JDBC the dynamic queries can be handled by the Prepared Statement Interface

|  |  |  |
| --- | --- | --- |
| SrNo | Static Query | Dynamic Query |
| 1 | In static query the values required for the execution are Coded. | In Dynamic query the values required for execution can be represented as a Place Holder |
| 2 | The values are presented in the query before its gets compiled | The query can be compiled without the actual values |
| 3 | The query is compiled at time of Execution | The Query is compiled and kept ready before it is required to be executed |
| 4 | To change the values  Changing the Query is required | The Values can be changed without changing the query. |
| 5 | In static each time new values are required then the query will be compiled for each time | The query can be compiled once and can be executed multiple times with new values. |
| 6 | To handle static queries we can use statement Interface or Prepared Statement Interface | We can only use Prepared Statement Interface to Handle Dynamic Queries |

Difference between Static Query And Dynamic Query

**Declaring a Stored Procedure**

-Like Methods in java we have Stored procedure in SQL.

-Stored procedure can be used to execute set of queries.

**Syntax:**

delimiter /

CREATE PROCEDURE procedure\_name()

**BEGIN**

statement 1;

statement 2;

:

:

:

:

statement n;

**END**/

delimiter ;

delimiter /

create procedure proc1()

begin

insert into student values(3,’Antony’,’antony@gmail.com’,’837378644’);

select \* from student;

end /

delimiter ;

**Calling a Stored Procedure For Execution**

**Syntax:**

**call procedure\_name();**

**Example :**

**call proc1;**

**CALLABLE INETRFACE**

-CallableStatement interface from **Java.sql** package can be used to call stored procedures in the databases.

-CallableStatement interface child interface PreparedStatement interface

**Stored Procedure**

#Program

**package** com.jspiders.jdbc.storedProcedure;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**public** **class** CallableDemo {

**private** **static** Connection *connection*;

//CallableStatement Interface is used to call the Stored Procedure

**private** **static** CallableStatement *callableStatement*;

**private** **static** ResultSet *resultSet*;

**private** **static** String *query*;

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

**try** {

*connection*=*openConnection*();

//Inside the query we have to pass the stored procedure call statement

*query*="call p1()";

*callableStatement*=*connection*.prepareCall(*query*);

*callableStatement*.execute();

*resultSet*=*callableStatement*.getResultSet();

**while**(*resultSet*.next())

{

System.***out***.println(*resultSet*.getInt(1)+" ||"

+*resultSet*.getString(2)+ " || "

+*resultSet*.getString(3)+ " || "

+*resultSet*.getLong(4)+ " || "

+*resultSet*.getString(5));

}

} **catch** (Exception e) {

*closeConnection*();

}

}

**private** **static** Connection openConnection() **throws** ClassNotFoundException, SQLException

{

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

**return** DriverManager.*getConnection*("jdbc:mysql://localhost:3306/weja3","root","root");

}

**private** **static** **void** closeConnection()

{

**if**(*connection*!=**null**)

{

**try** {

*connection*.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**if** (*resultSet*!=**null**) {

**try** {

*resultSet*.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**if** (*callableStatement*!=**null**) {

**try** {

*callableStatement*.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

-JDBC used raw data as Query Parameters.

-But as we Know Java will process data in Object format.

-Hence it becomes the responsibility of programmer to establish the relationship between the Data from Database and the Java Object Data.

-A table in Database must be considered as Class in Java ,Columns of the Table must be considered as the Properties of the Class and the Row in the table must be considered as the Object of the class.

**Student.java**

Public class Student

{

Private int sid;

Private String name;

Private String email;

Private String contact;

//getters and setters toString Method

public int getSid()

{

return sid;

}

Public void setSid(int sid)

{

this.sid=sid;

}

**SelectDemo5.java**

**Public class SelectDemo5**

{

Private static Connection connection;

Private static Statement statement;

Private static ResultSet resultSet;

Private static String dburl = “jdbc:mysql://localhost:3306/weja2”;

Private static String user = “root”;

Private static String password = “root”;

Private static String query;

Public static void main(String args[])

{

Try

{

connection=DriverManager.

getConnection(dburl,user,password);

statement = connection.createStatement();

query=”Select \* from student”;

resultSet=statement.executeQuery(query);

while(resultSet.next())

{

Student student = new Student();

student.setSid(resultSet.getInt(1));

student.setName(resultSet.getString(2));

student.setEmail(resultSet.getString(3));

student.setContact(resultSet.getLong(4));

}

}

catch(Exception e)

{

e.printStacktrace();

}

Finally

{

Try

{

if(connection != null)

{

conncetion.close();

}

If(resultSet != null)

{

resultSet.close();

}

If(statement != null)

{

statement.close();

}

}

Catch(Exception e)

{

e.printStackTrace();

}

}

**Note: The Step to load the Driver Class in JDBC operation in deprecated.**

**It means JDBC will implicitly identify and load the Driver Class.**

**=================== NOTE ===================**

- The Step to load the Driver Class in JDBC operation in deprecated.

-It means JDBC will implicitly identify and load the Driver Class.

**Drawbacks of JDBC**

1]In JDBC, being a Java developer, we need to write SQL query as well. It means to write a JDBC program, a programmer is forced to have a SQL knowledge as well.

2]For every JDBC operation, we need to define and use the database properties

3]In JDBC, there is no Implicit Relationship between the Relational Database and Java Objects.

**=================== NOTE ===================**

**-**Due to this Drawbacks, JDBC become a Deprecated Technology.

**ORM**

-ORM stands for Object Relational mapping

-ORM was developed to overcome the drawbacks of the JDBC.

-Although the ORM technology is build upon JDBC Framework itself.

-It means the ORM technology internally works on JDBC logic only.

-In ORM we are not forced to write the SQL query.

-It means while using the ORM Technology, SQL Knowledge is not mandatory.

-In ORM the Database Properties can be declared just once and used multiple times.

-As the name suggest, the Object from Java Application are implicitly mapped in the Relational Database.

-It means in Class(Entity) it implicitly mapped as a Table, Properties of the Class Are Mapped as the Column Of the Table and Objects of the Class are Mapped as the Row of the Table.

**==================== NOTE ====================**

-ORM tools will maps Java Classes as table inside the Databases and will maps Java Object to the record/row in the Tables.

-Properties of class will be map as Column of Tables

**=================== NOTE ===================**

One of the most widely used ORM technology is **Hibernate.**

-JPA stands for Java Persistent API

-JPA is used to make migration between One ORM tool to another easier hence Hibernate can also be use along with JPA and it will be known as Hibernate JPA.

-It is a specification which contain Interfaces, Abstract Classes, Abstract Methods.

-Like Hibernate we have several Hibernate Tools in the Market.

-It is the responsibility of one of these ORM tools to provide the implementation JPA.

-If we make use of JPA then the migration from one ORM tool to another ORM tool becomes easy because all the ORM tools follow a common standard set by JPA.

-Without JPA migration from one ORM tool to another ORM tool is a tedious(long/boring) procedure because all the ORM tools are incompatible with each other.

Programmer

Driver

JDBC

ORM

tools

JPA

ABASEDAT

**Hibernate Layers**

Hibernate Entity

Logic Classes

**DB**

**DAO**

**LAYERS**

**DTO**

**LAYERS**

zzz

-Hibernate Structure Consist of two layers

1]DTO

2]DAO

1]DTO

-The DTO layers consist of the entity classes required in the application.

-These entity classes are the Data that is transferred between the Java Application the Database hence the name **Data Transfer Object**.

2]DAO

-The DAO layer consists Core Hibernate logic required for the application.

-The classes in DAO layer will be responsible to perform the operation (Access) or the entity classes from the DTO layers.

Hence the name **Data Access Object**.

-Theses layers are represented as Packages in the Project.

entityManager.persist()

-This method is used to put your Java Class Object into Database in the table format.

OR  
-This method is used to insert the Java object inside the Database table.

**Steps to Create The Maven Project For Hibernate**

1.Click ctrl+N and Search for Maven.

2.Select “**Maven Project** “and Click on next.

3.Click on Next.

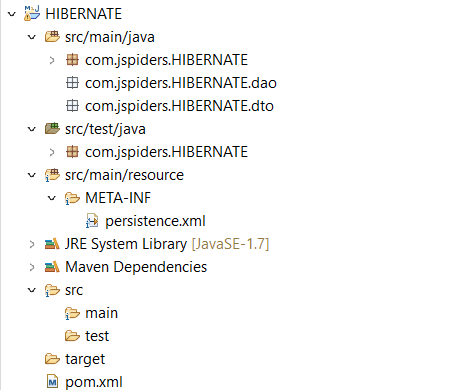
4.In the “**Select an Archetype** “window search for “**org.apcahe.maven**”, and find the “**maven-archetype-quickstrat**”of the version **1.4**  and click on next.

5.In the Next window enter the **Group Id** in the format of “**Domain.Hostname**”.

6.In the **Artifact Id** enter the Project name.

7.untick the checkbox “**run archetype generation interactively**” then click on finish.

**Project Structure For Hibernate**

****

**POM**

-POM stands for Project Object Model.

-pom.xml file is used to define the Dependencies which are required to build a project. **Dependency**(External jar Files).

-pom.xml contains information about the project and object required to build that project.

**How to Add the dependency?**

1.Go to the browser

2.Go the Maven Repository (maven.com)

3.Click on search option.

4.Type required dependency name in search box.

5.Then copy the xml code.

**>>>> DEPENDENCIES REQUIRED FOR HIBERNATE >>>>**

**1.MySql Connector/J**

**2.Hibernate Core Relocation**

**3.Hibernate Entity Manager**

**4.Project Lombok**

-We have to do some modification inside the Maven project so we can use that project structure to implement Hibernate logic.

**Step-1**:Select the Project then press ctrl+N one window will appear

**Step-2:**in filter section type folder from that select Source folder

The name of the Source folder is “**src/main/resources”**.

**Step-3:**Inside that source folder we have to create the one General folder. Select the created source folder press ctrl+N one window will appear in filter section type folder select General folder.The Name of the Folder is **META-INF**.

**Step-4:** inside META-INF folder we have to create one xml file. Select the META-INF folder then press ctrl+N then one window will appear inside filter section type xml then select the xml file.The name of the xml file should be **persistence.xml**.

**Note:** Inside created persistence.xml file we have to define the property related to JDBC and properties related to Hibernate ORM tool.

**====================== NOTE =======================**

-These dependencies are supposed to be added inside the dependencies in the Pom.xml file.

-Any required dependencies can be obtained from Maven Repository.

Version Of Dependencies

**5.6.15**

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.6.15.Final</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>5.6.15.Final</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.28</version>

<scope>provided</scope>

</dependency>

**Adding Schema Definition in persistence.xml**

**Step-1:**Go to Maven Dependencies section

**Step-2:**Find the jar file by the name “**javax.persistence-api-2.2.jar**” and open it.

**Step-3:**Find a package by the name “**javax.persistence**” and open it.

**Step-4:**Scroll down to the Bottom of the Package to find a file by the name “**persistence\_2\_1.xsd**” and open it.

**Step-5:**From the File, copy the content from line No 50 to line No 56 and paste it to the persistence.xml file.

**Step-6:**Remove the 3 dot(**…**)Present within the opening and closing persistence tag and then add a persistence unit tag(**<persistence-unit>**)

**Defining the database Properties in Persistence.xml file**

**Step1:**Inside the persistence-unit tag, add a properties tag(<properties>)

**Step2:**For defining a single property we need to declare the property tag inside the properties tag.

**Step3:**For each property, the name and value has to be defined.

**Properties Required in Persistence.xml**

**1]Driver Path**

**2]Dburl**

**3]Username**

**4]Password**

**5]Dialect**

**6]Mapping Protocol Hibernate**

**7]Display Query**

**Persistence Properties and Value**

**1]DriverPath**

<property name=”javax.persistance.jdbc.driver”

Value=”com.mysql.cj.jdbc.Driver” />

**2]Dburl**

<property name=”javax.persistance.jdbc.url”

Value=”jdbc:mysql://localhost:3306/weja2” />

**3]Username**

<property name=”javax.persistance.jdbc.user”

Value=”root” />

**4]Password**

<property name=”javax.persistance.jdbc.password”

Value=”root” />

**5]Dialect**

<property name=”hibernate.dialect”

Value=”org.hibernate.dialect.MySql8Dialect” />

**6]Mapping Protocol**

<property name=”hibernate.hbm2ddl.auto”

Value=”update” />

**7]Display Query**

<property name=”hibernate.show\_sql”

Value=”true” />

**Adding the Value for Dialect Property**

**Step1:**Go to the Maven dependencies section and open the jar file name “**hibernate-core.jar**”

**Step2:**From the jar file open the package named “**org.hibernate.dialect**” and search for the “**MySql8Dialect.Class**” file.

**Step3:**Open the Class file, copy the Qualified Name of that class and paste it in the value Attribute for “**Hibernate.property**”

**================== NOTE ===================**

-The properties declared defined in the persistence unit tag can be access by the name value set for the persistence unit.

**DAO**

-this hibernate layers contains the Core Hibernate logic

**DTO**

-This Hibernate layer contain the Entity classes.

**Note:**

Hibernate layers will be represented as packages inside the projects.

entityManager.persist()

-This method is used to put your Java Class Object into Database in the table format.

OR  
-This method is used to insert the Java object inside the Database table.

**Annotation**

@Entity

-This annotation **javax.persistence** package is used to mark class an entity.

-It is class level annotation.

-if class is annotated with this annotation hibernate ORM tool will recognize entity class and will create the respective table inside the database.

@Id

-This annotation from **javax.persistence** package is use to mark one of the properties of an entity class to be treated as a primary key inside the table(It is a Property level annotation).

@Data

-This annotation from project Lombok is used to add getters(), setters(), toString(),equlas(),hashCode() to the class

JPQL

-JPQL stands for Java Persistence Query Language.

-It is used to write programmer defined queries(Customized query).

-It is similar to Structured Query Language but it uses class name instead of Table Name and Properties of a class instead of column name.

-All the ORM tools available have their own query languages which are incompatible to each other.

For Example: Hibernate uses Hibernate Query Language(HQL)

IBatis or MyBaties uses Dynamic Query Language(DQL).

-JPQL is understandable to all the ORM tools.

-

**HIBERNATE MAPPING**

-Hibernate mapping is used to define or establish a relationship between two entities.

-We have four types of Mapping

**1]One-to-One mapping**

**2]One-to-Many Mapping**

**3]Many-to-One Mapping**

**4]Many-to-Many mapping**

**1]One-to-One mapping**

-When a record from Table 1 is related to exactly One Record from Table 2 then it is called as One-to-One mapping.

Table 1 Table2

R1 R1

R2 R2

R3 R3

**Uni-Directional Mapping**

-If the Relationship is defined inside either one of the Table is called as **Uni-Directional Mapping**.

**Bi-Directional Mapping**

-If the Relationship is defined inside Both Table is called as **Bi-Directional Mapping**.

**2]One-to-Many Mapping**

-When a record from Table 1 is related to More than One Records from Table 2 then it is called as One-to-Many mapping.

Table 1 Table2

R1 R1

R2 R2

R3 R3

**3]Many-to-One Mapping**

-When More than One Records from Table 1 are related to exactly one record form Table 2 then it is called Many-to-One Mapping.

Table 1 Table2

R1 R1

R2 R2

R3 R3

**4]Many-to-Many mapping**

-When more than one Records from Table 1 are related to more than one Records Table 2 then it is called as Many-To-Many mapping.

Table 1 Table2

R1 R1

R2 R2

R3 R3

**Uni-Directional Mapping**

-If the Relationship is defined inside either one of the Table is called as **Uni-Directional Mapping**.

**Bi-Directional Mapping**

-If the Relationship is defined inside Both the Table is called as **Bi-Directional Mapping**.

**One-To-One Unidirectional**

**AadharCardDTO.java**

#Program

**package** com.jspiders.dto;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** lombok.Data;

@Data

@Entity

**public** **class** AadharCardDTO {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**int** id;

**long** aadharNum;

}

**PersonDTO.java**

#Program

**package** com.jspiders.dto;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.OneToOne;

**import** lombok.Data;

@Data

@Entity

**public** **class** PersonDTO {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**int** id;

String name;

**long** phoneNo;

String email;

@OneToOne

**private** AadharCardDTO aadharCardDTO;

}

**PersonDAO.java**

#Program

**package** com.jspiders.dao;

**import** javax.persistence.EntityManager;

**import** javax.persistence.EntityManagerFactory;

**import** javax.persistence.EntityTransaction;

**import** javax.persistence.Persistence;

**import** com.jspiders.dto.AadharCardDTO;

**import** com.jspiders.dto.PersonDTO;

**public** **class** PersonDAO {

**private** **static** EntityManagerFactory *entityManagerFactory*;

**private** **static** EntityManager *entityManager*;

**private** **static** EntityTransaction *entityTransaction*;

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

*openConnection*();

*entityTransaction*.begin();

AadharCardDTO aadharCardDTO=**new** AadharCardDTO();

aadharCardDTO.setAadharNum(931300905827L);

*entityManager*.persist(aadharCardDTO);

PersonDTO personDTO=**new** PersonDTO();

personDTO.setAadharCardDTO(aadharCardDTO);

personDTO.setName("Amol");

personDTO.setPhoneNo(9373428581L);

personDTO.setEmail("24amolpawar@gmail.com");

*entityManager*.persist(personDTO);

*entityTransaction*.commit();

*closeConnection*();

}

**private** **static** **void** openConnection()

{

*entityManagerFactory*=Persistence.*createEntityManagerFactory*("Person");

*entityManager*=*entityManagerFactory*.createEntityManager();

*entityTransaction*=*entityManager*.getTransaction();

}

**private** **static** **void** closeConnection()

{

**if** (*entityManagerFactory*!=**null**) {

*entityManagerFactory*.close();

}

**if** (*entityManager*!=**null**) {

*entityManager*.close();

}

**if** (*entityTransaction*!=**null**) {

**if** (*entityTransaction*.isActive()) {

*entityTransaction*.rollback();

}

}

}

}

**=================== NOTE ====================**

-In Hibernate Mapping there dependency between tables so if you want insert then You have first persist the Non-Dependend Object later persist the Dependent Object.

-In case of Deletion operation it is reverse process of Insertion First you have to delete the Dependent Object then you have to delete the Non-Dependent Object neither you get the Exception called SQLEntitgrityViolation

**One-To-One Unidirectional**

**DTO PACKAGE**

**package** com.jspiders.HIBERNATE\_1.dto;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.OneToOne;

**import** lombok.Data;

@Data

@Entity

**public** **class** Person {

@Id

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

**private** String name;

**private** String email;

@OneToOne

**private** AadharCard aadharCard

}

**package** com.jspiders.HIBERNATE\_1.dto;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** lombok.Data;

@Data

@Entity

**public** **class** AadharCard {

@Id

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

**private** **long** aadharNumber;

**private** String dateOfIssueString;

}

**DAO PACKAGE**

**package** com.jspiders.HIBERNATE\_1.dao;

**import** javax.persistence.EntityManager;

**import** javax.persistence.EntityManagerFactory;

**import** javax.persistence.EntityTransaction;

**import** javax.persistence.Persistence;

**import** com.jspiders.HIBERNATE\_1.dto.AadharCard;

**import** com.jspiders.HIBERNATE\_1.dto.Person;

**public** **class** PersonDao {

**private** **static** EntityManagerFactory *entityManagerFactory*;

**private** **static** EntityManager *entityManager*;

**private** **static** EntityTransaction *entityTransaction*;

**private** **static** **void** openConnection() {

*entityManagerFactory*=Persistence.*createEntityManagerFactory*("person");

*entityManager* = *entityManagerFactory*.createEntityManager();

*entityTransaction* = *entityManager*.getTransaction();

}

**private** **static** **void** closeConnection()

{

**if** (*entityManagerFactory* != **null**) {

*entityManagerFactory*.close();

}

**if** (*entityManager* != **null**) {

*entityManager*.close();

}

**if** (*entityTransaction* != **null**) {

**if** (*entityTransaction*.isActive()) {

*entityTransaction*.rollback();

}

}

}

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

*openConnection*();

*entityTransaction*.begin();

Person person = **new** Person();

person.setId(1);

person.setName("Amol");

person.setEmail("24amolpawar@gmail.com");

AadharCard aadharCard = **new** AadharCard();

aadharCard.setId(1);

aadharCard.setAadharNumber(931300905827l);

aadharCard.setDateOfIssueString("24/04/2016");

*entityManager*.persist(aadharCard);

person.setAadharCard(aadharCard);

*entityManager*.persist(person);

*entityTransaction*.commit();

*closeConnection*();

}

}

**One-To-One Unidirectional**

**AadharCardDTO.java**

#Program

**package** com.jspiders.dto;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** lombok.Data;

@Data

@Entity

**public** **class** AadharCardDTO {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**int** id;

**long** aadharNum;

}

**PersonDTO.java**

#Program

**package** com.jspiders.dto;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.OneToOne;

**import** lombok.Data;

@Data

@Entity

**public** **class** PersonDTO {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**int** id;

String name;

**long** phoneNo;

String email;

@OneToOne

**private** AadharCardDTO aadharCardDTO;

}

**PersonDAO.java**

#Program

**package** com.jspiders.dao;

**import** javax.persistence.EntityManager;

**import** javax.persistence.EntityManagerFactory;

**import** javax.persistence.EntityTransaction;

**import** javax.persistence.Persistence;

**import** com.jspiders.dto.AadharCardDTO;

**import** com.jspiders.dto.PersonDTO;

**public** **class** PersonDAO {

**private** **static** EntityManagerFactory *entityManagerFactory*;

**private** **static** EntityManager *entityManager*;

**private** **static** EntityTransaction *entityTransaction*;

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

*openConnection*();

*entityTransaction*.begin();

AadharCardDTO aadharCardDTO=**new** AadharCardDTO();

aadharCardDTO.setAadharNum(931300905827L);

*entityManager*.persist(aadharCardDTO);

PersonDTO personDTO=**new** PersonDTO();

personDTO.setAadharCardDTO(aadharCardDTO);

personDTO.setName("Amol");

personDTO.setPhoneNo(9373428581L);

personDTO.setEmail("24amolpawar@gmail.com");

*entityManager*.persist(personDTO);

*entityTransaction*.commit();

*closeConnection*();

}

**private** **static** **void** openConnection()

{

*entityManagerFactory*=Persistence.*createEntityManagerFactory*("Person");

*entityManager*=*entityManagerFactory*.createEntityManager();

*entityTransaction*=*entityManager*.getTransaction();

}

**private** **static** **void** closeConnection()

{

**if** (*entityManagerFactory*!=**null**) {

*entityManagerFactory*.close();

}

**if** (*entityManager*!=**null**) {

*entityManager*.close();

}

**if** (*entityTransaction*!=**null**) {

**if** (*entityTransaction*.isActive()) {

*entityTransaction*.rollback();

}

}

}

}

**=================== NOTE ====================**

-In Hibernate Mapping there dependency between tables so if you want insert then You have first persist the Non-Dependend Object later persist the Dependent Object.

-In case of Deletion operation it is reverse process of Insertion First you have to delete the Dependent Object then you have to delete the Non-Dependent Object neither you get the Exception called SQLEntitgrityViolation

**One-To-Many Unidirectional**

-In case of One-to-Many Unidirectional mapping Hibernate will create one Extra Table Called as Mapping Table to establish the relationship between Two Entities.

**Many-To-One Unidirectional**

**Many-To-Many Bi-directional**

**Annotation In Hibernate**

**@One-To-One**

-it is used to define the One-To-One relationship between two entities.

-It is property level annotation.

**@One-To-Many**

-It is used to define the One-To-Many relationship between the Two Entities.

-It is Property level Annotation

**@Many-To-One**

-It is used to define Many-To-One relationship between two entities.

-It is a property level annotation.

**@Many-To-Many**

-It is used to define Many-To-Many relationship between two entities.

-It is a property level annotation.

**Data Redundancy in Bi-Directional mapping**

-In One-To-One Bi-directional mapping Hibernate will add Foreign Key in both the entity table but foreign key column is sufficient to establish the connection between two table

-So we can Normalize one of the table using the @JoinColumn Annotation and mapped by attribute.

-In case of many-to-many bidirectional mapping hibernate will create two mapping tables but one mapping table is sufficient to establish the relationship between two tables so we have to Normalize either one of the Mapping tables.

-We can make use of @JoinTable Annotation and Mapped by Attribute.

Syntax:

**@JoinTable(joinColumns = @JoinColumn(referencedColumnName = "id"),**

**inverseJoinColumns = @JoinColumn(referencedColumnName = "id"))**

**More Annotation**

@Table

-This annotation is used to give user defined to the it take one attribute called as name

@Column

-This is used to give user defined name to the column of a table it takes one attribute called as name

We can make use an attribute called as cascade along with all the mapping annotation it is used to perform cascading operation on entities.

@generatedvalue

-This is used to generate automatic value for primary key for only int type

**JPQL**

-JPQL stands for the JAVA PERSISTENCE QUERY LANGUAGE.

-It is used to write customized queries.

-Since we have different different ORM tools available in the market each ORM tool will understand one specific Query language i.e. Hibernate ORM tool will understand HQL(Hibernate Query Language).

-And iBATIS or Mybaties ORM tool understand DQL(Dynamic Query Language)

-In this case both the Query Language are Incompatible to each other so it will be difficult to migrate from One ORM tool to another ORM tool to overcome this problem JPQL is introduced.

-JPQL is understandable for all the ORM tools.

**=================== NOTE ===================**

-In JPQL instead of using Table Name and Column name we will make use of ClassName and Its Properties.

**EmployeeDao6.java**

**#Program**

Public class EmployeeDao6

{

Public static EntityManagerFactory entityManagerFactory;

Public static EntityManager entityManager;

Public static EntityTransaction entityTransaction;

Private static void openConnection()

{

entityManagerFactory =Persistence .createEntityManagerFactory(“employee”);

EntityManager = entityManagerFactory.createEntityManager();

entityTransaction = EntityManager.getTransaction();

}

Private static void closeConnection()

{

If(entityManagerFactory != null)

{

entityManagerFactory.close;

}

If(entityManager != null)

{

entityManager.close;

}

If(entityTransaction != null)

{

If(entitytransaction.isActive())

{

entityTransaction.rollback();

}

}

}

Public static void main(String args[])

{

openConnection();

entityTransaction.begin();

Query query = EntityManager.createQuery(“Select emp from EmployeeDTO emp”);

List<EmployeeDTO> employees = query.getResultList();

For(EmployeeDTO employee : employees)

{

System.out.println(employee);

}

entityTransaction.commit();

closeConnection();

}

**Create Database If Not Exist**

-This particular statement can we used to create the specified Database Automatically.

-If it does not exist

**Syntax:-**

**Jdbc:mysql://localhost:3306/weja2\_demo?createDatabaseIfNotExist=true”**

**SERVLETS**

-Servlets is the technology (only one) or java API which is used to accept web request and we can generate the web response in web-based application.

**WEB APPLICATION**

-An application which is present inside a server and which can be accessed using standard web Browser.

**WEB BROWSER**

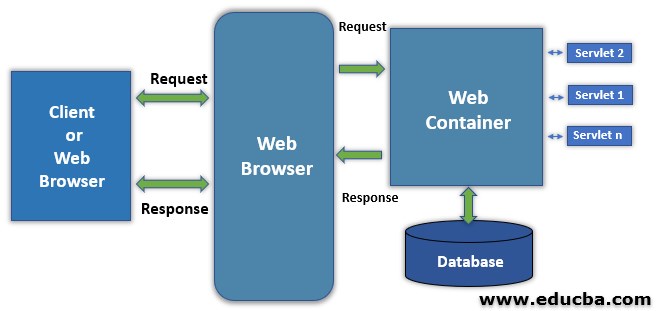
-Web-Browser is the application using which we can generate the web request and we can receive the web response.

**WEB REQUEST**

-A web request is used to access the specific web resource from the server.

**WEB RESPONSE**

-The send web request will be identified by the servlets and respective web response will be generated by the servlet.



**URL**

-URL stands for Uniform Resource Locator it is used to locate the Web-Resource present inside a web application.

-Using this URL we can access the any particular Web-Resource using any device or using any System.

Format

Protocol://host\_name or domain\_name:port\_number/name\_of\_resources?Query\_String#fragment\_id

**Protocol**

-The protocol indicate whether the request is protocol dependent or protocol independent and also it indicates which protocol has been used to access the specific web-resource.

**HostName or DomainName**

-It indicates the location of the server in which the requested web resource is present.

**PortNumber**

-The port number indicates the exact location of the application inside an identified server

**Name Of Resource**

-It indicates the location of a requested web resource inside an identified the application

**Query String**

-In order to send some Data from client side application to the Server side application through the URL then it is declared as Query String.

-It will be in the form of Key Value pair.

-Multiple Key value Pairs will be separated using **&**(Ambercent) symbol.

**Fragment Id**

-It is the id of the page which is currently being displayed.

**Web-Resource**

-It is resource present inside a web application there are two types of the web resources.

1]Static Web-resource

2]Dynamic Web-Resource

1]Static Web-resource

The resource which generates static or fixed response is called as Static web-resource

2]Dynamic Web-Resource

-The resource which generates Dynamic Response (changing Response) is called as Dynamic Web Resource

**Static Web Application**

The application which contain only static web resources is called as Static Web Application.

**Dynamic Web Application**

-If the Application contain dynamic web resource then it is called as Dynamic Web Application

**Note**

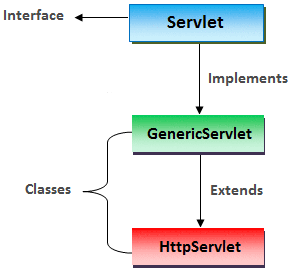
-Static web Resources can be develop using the HTML.

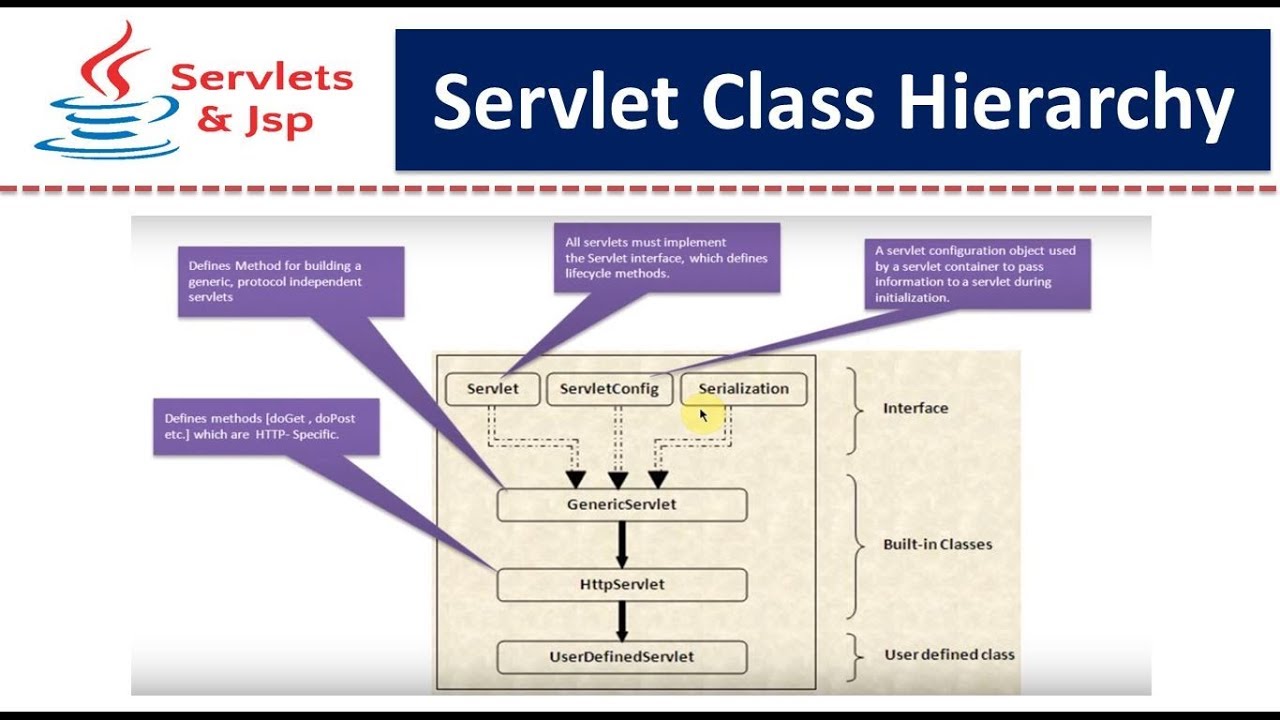
-Dynamic web Resources can be develop using Servlets and JSP

**SERVLETS IN JAVA**

-It is a special class in java

**SERVLET HIERARCHY**

****

****

-there are two ways creating the servlets in java

**1]By Extending Generic Servlets Abstract Class**

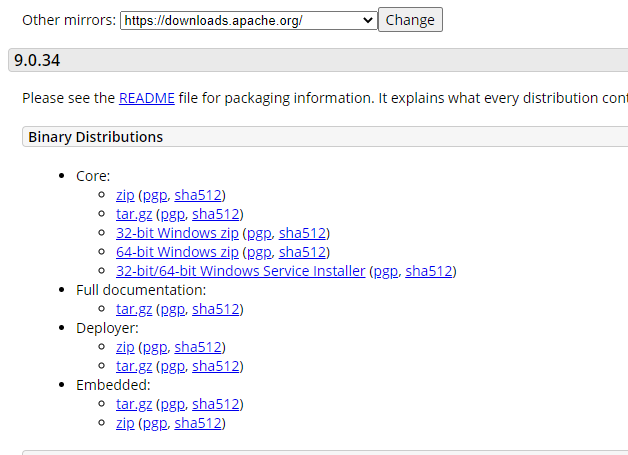
-If we create by extending the Generic Servlet Abstract Class then that particular servlet will become Protocol Independent.

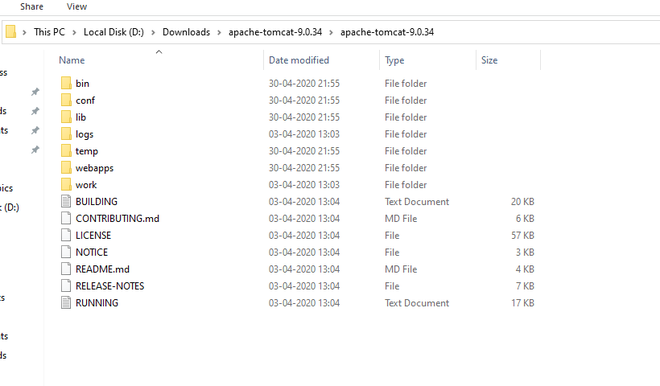
**2]By Extending Servlets Class**

-We can create a servlet class in java by Extending the HTTP servlet class.

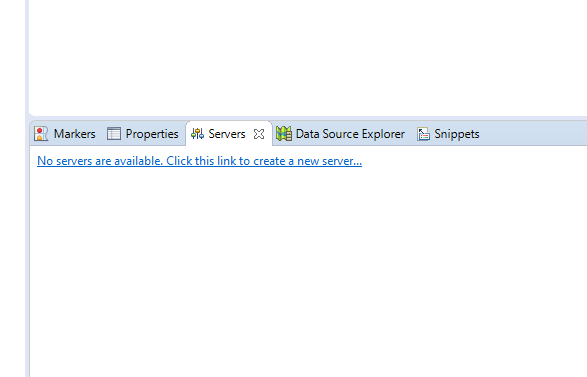
-The Servlet class which extends HTTP servlet class will become Protocol Dependent.

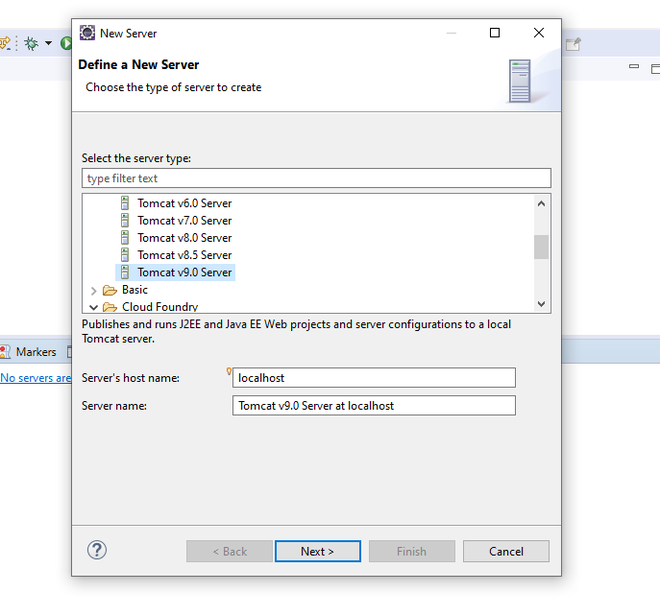
**Steps to install Apache Tomcat Server**

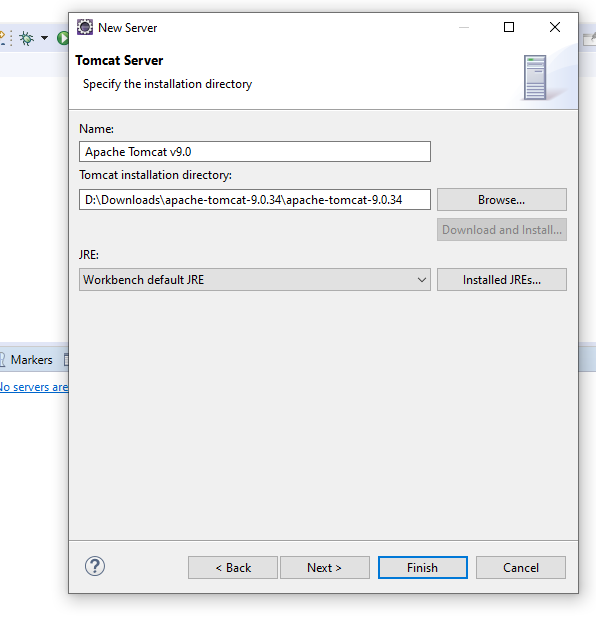
**Step 1:** Download the latest version ( 9.0.34 ) of apache tomcat server from[this link](https://tomcat.apache.org/download-90.cgi) according to your platform. 

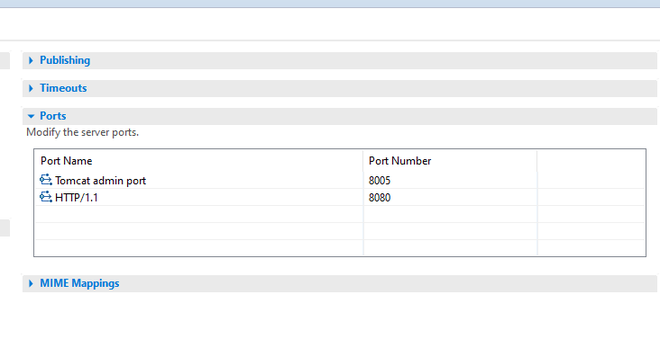
**Step 2:** Extract the files from zip folder. The structure of the folder should look like this: 

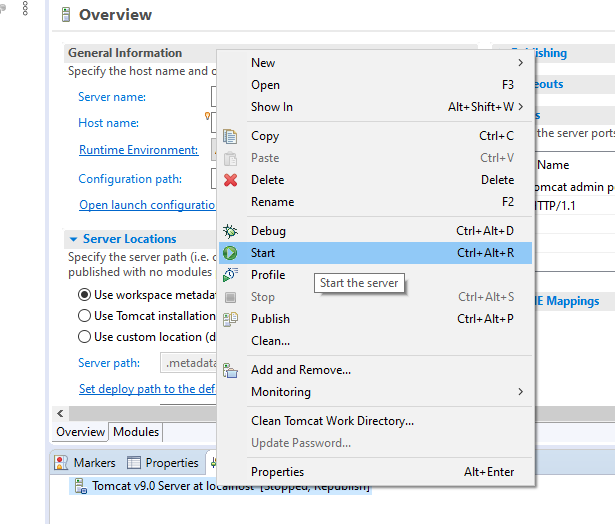
**Step 3:** Open Eclipse Java EE (Enterprise edition ) environment. Click on *Servers* tab at bottom. Click on *No servers are available. Click this to create server.*

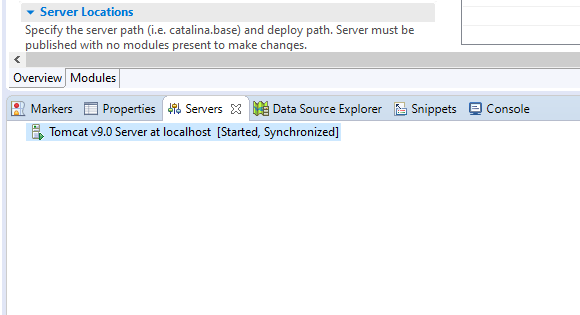


**Step 4:** A dialog box will appear. Select *Tomcat 9.0 server*folder. Click *Next.* 

**Step 5:** Browse to *Apache Tomcat 9.0* folder select it. Click *Finish.* 

**Step 6:** You should see *Tomcat v9.0 Server at localhost [Stopped, Republish ]* under *Servers* tab. Double click on it and verify the HTTP ports. By default HTTP/1.1 port is 8080. If there is any application running on default port 8080 then change it to any other port. 

**Step 7:** Now right click on Server and click *Start.* 

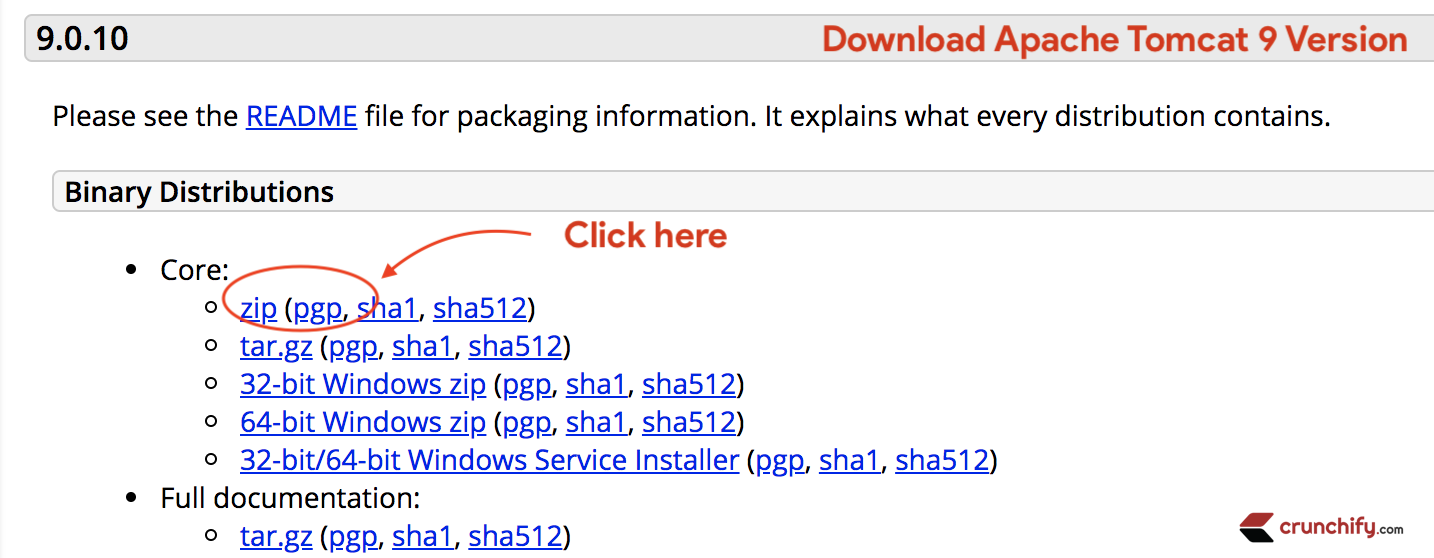
**Step 8:** The server will be started and shows the following *Tomcat v9.0 Server at localhost [ Started, Synchronized ].* 

OR

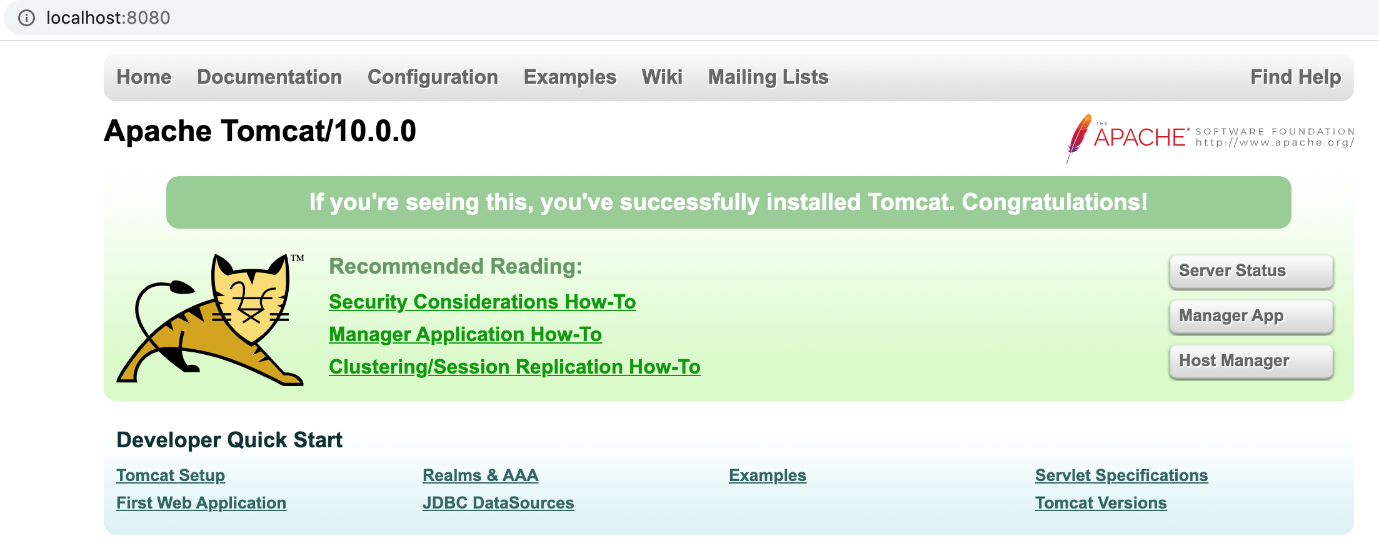
The following server is running at [port 8080 on localhost](https://www.geeksforgeeks.org/what-is-local-host/) and we can visit the webpage using the URL *http://localhost:8080/*

**Step-1**

Download Apache Tomcat from [this link](https://tomcat.apache.org/download-90.cgi). I’m using version 9.0.10.

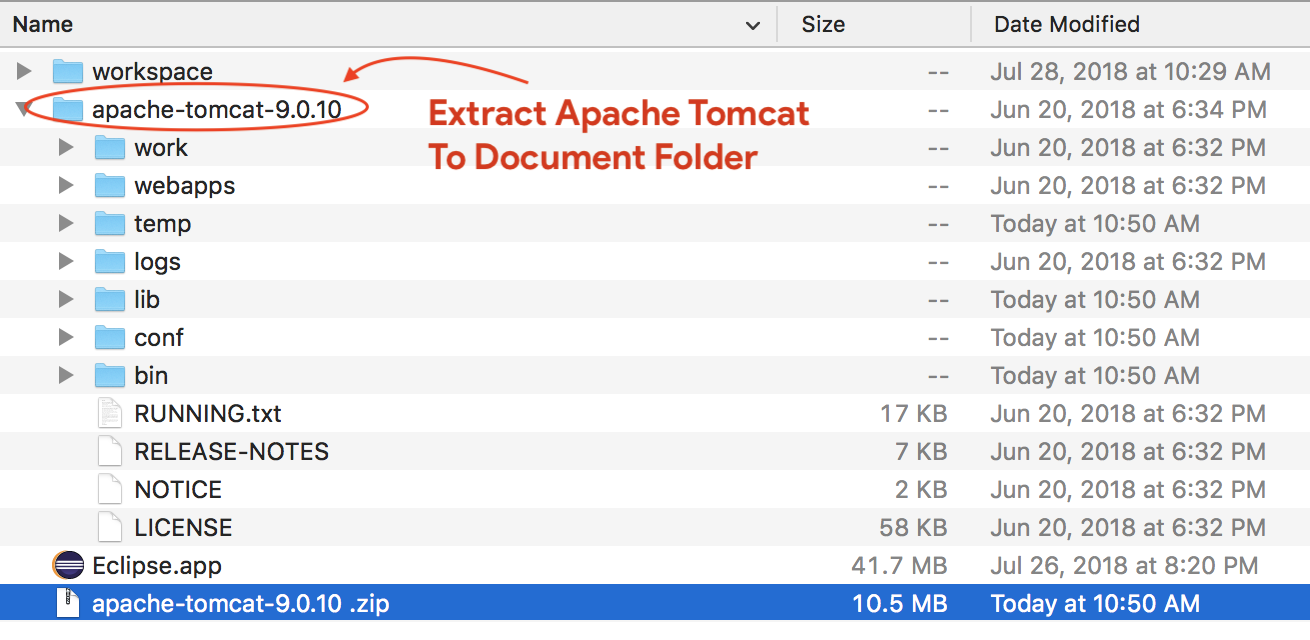
[](https://crunchify.com/wp-content/uploads/2015/01/Download-Apache-Tomcat-9-Version-Crunchify-Tips.png)

*Same tutorial works with Tomcat 10.0 too 🙂 .*

[](https://crunchify.com/wp-content/uploads/2020/12/Apache-Tomcat-10-running-fine-on-Mac.png)

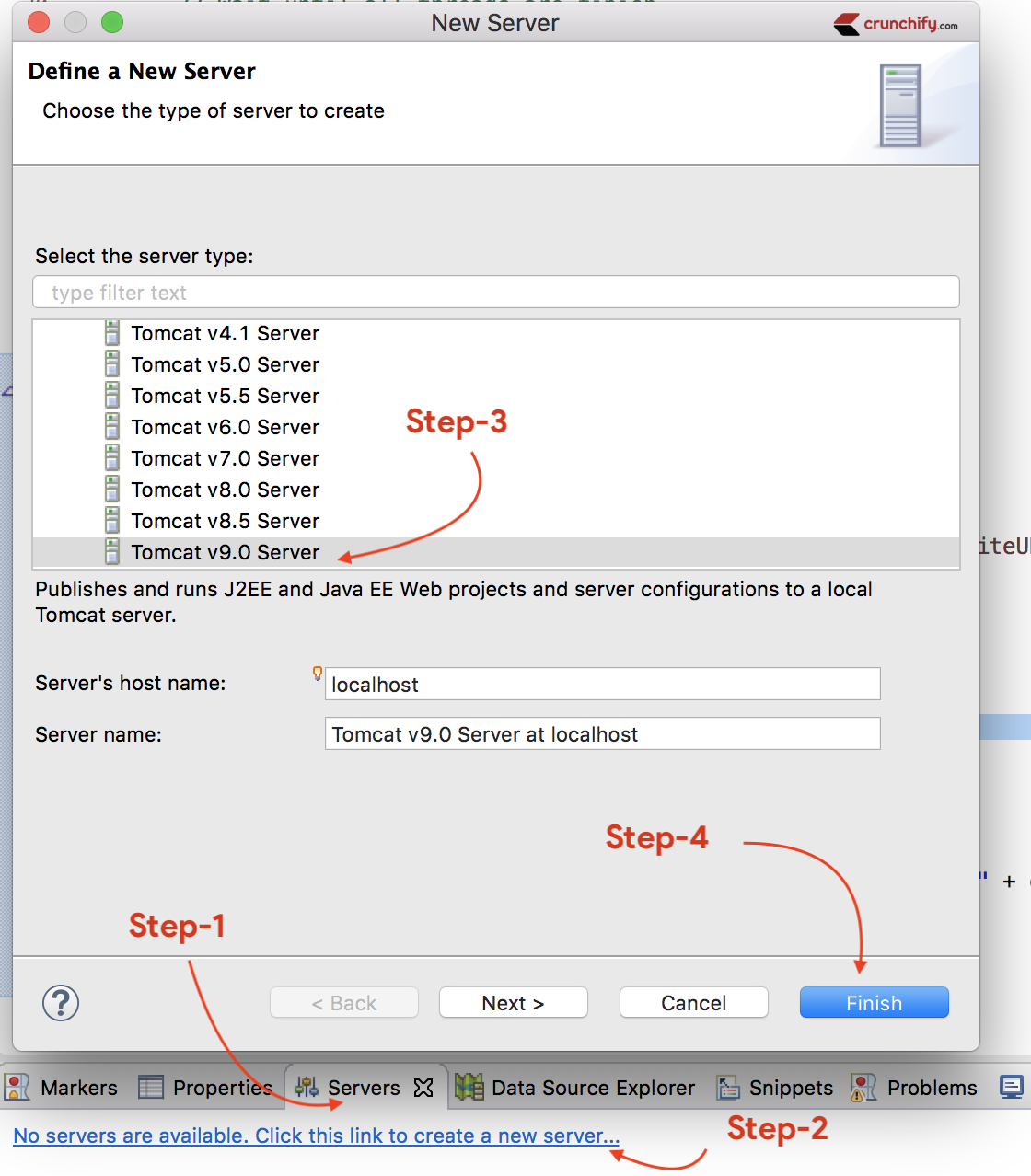
**Step-2**

Extract it to Document folder.

[](https://crunchify.com/wp-content/uploads/2015/01/Extract-Apache-Tomcat-To-Document-Folder.png)

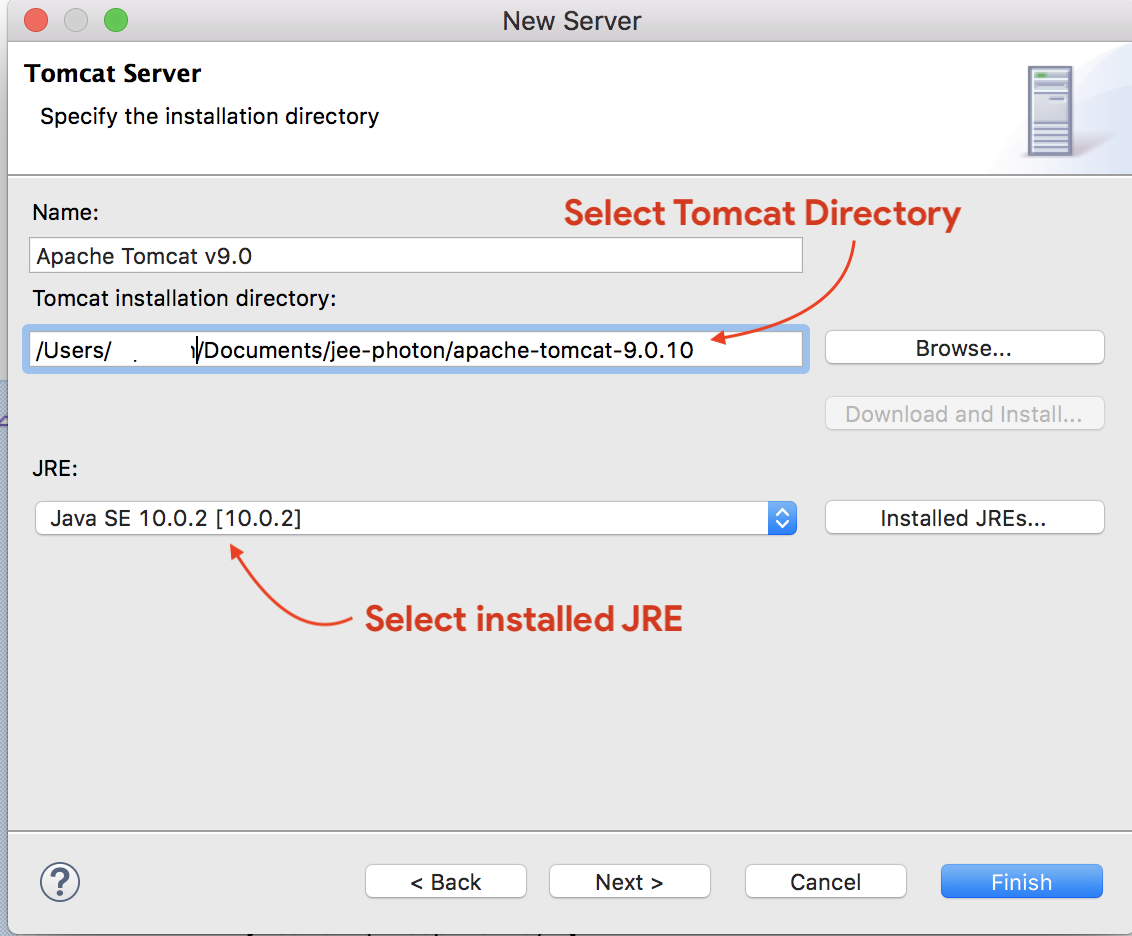
**Step-3**

* Open [Eclipse Environment](https://crunchify.com/create-and-deploy-simple-web-service-and-web-service-client-in-eclipse/)
* Click on Servers Tab
* Click on No servers are available. Click this link to create a new server...
* Click Tomcat v9.0 Server and Next

[](https://crunchify.com/wp-content/uploads/2015/01/Choose-Eclipse-Server-Tab-Add-Server-Select-Apache-Tomcat.png)

**Step-4**

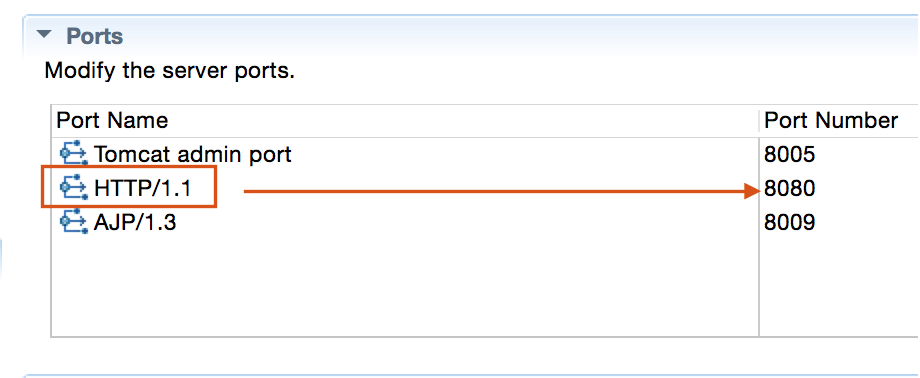
Select Apache installation Directory and click Finish.

[](https://crunchify.com/wp-content/uploads/2015/01/Add-new-Apache-Tomcat-Server-in-Eclipse.png)

**Step-5**

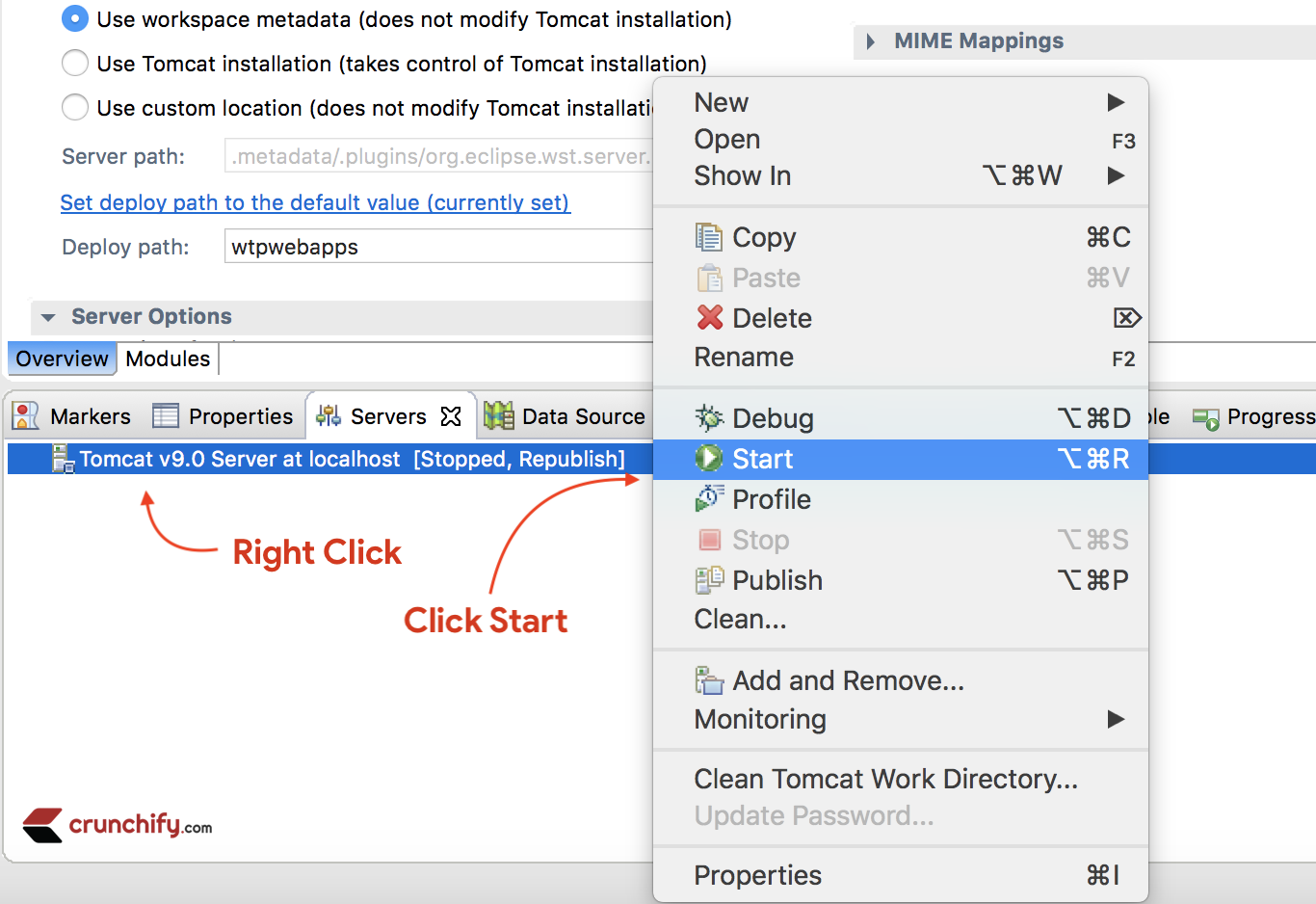
You should see Tomcat v9.0 Server at localhost [Stopped, Republish] under Servers tab. Double click on it verify [HTTP ports](https://crunchify.com/java-asynchronous-httpclient-overview-and-tutorial-sendasync/) information.

By default HTTP port is 8080.

[](https://crunchify.com/wp-content/uploads/2015/01/Apache-Tomcat-8-HTTP-Port-8080.png)

**Step-6**

Now right click on Server and click Start.

[](https://crunchify.com/wp-content/uploads/2015/01/Start-Apache-Tomcat-in-Eclipse-Crunchify-Tips.png)

**Console output:**

Aug 04, 2018 11:12:57 AM org.apache.catalina.startup.VersionLoggerListener log

INFO: Server version: Apache Tomcat/9.0.10

INFO: OS Name: Mac OS X

INFO: OS Version: 10.13.6

INFO: Architecture: x86\_64

INFO: Java Home: /Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Home

INFO: JVM Version: 10.0.2+13

INFO: JVM Vendor: "Oracle Corporation"

INFO: CATALINA\_BASE: /Users/appshah/Documents/jee-photon/workspace/c/.metadata/.plugins/org.eclipse.wst.server.core/tmp0

INFO: CATALINA\_HOME: /Users/appshah/Documents/jee-photon/apache-tomcat-9.0.10

INFO: Command line argument: -Dcatalina.base=/Users/appshah/Documents/jee-photon/workspace/c/.metadata/.plugins/org.eclipse.wst.server.core/tmp0

INFO: Command line argument: -Dcatalina.home=/Users/appshah/Documents/jee-photon/apache-tomcat-9.0.10

INFO: Command line argument: -Dwtp.deploy=/Users/appshah/Documents/jee-photon/workspace/c/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps

INFO: Command line argument: -Dfile.encoding=UTF-8

INFO: The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: **[**/Users/appshah/Library/Java/Extensions:/Library/Java/Extensions:/Network/Library/Java/Extensions:/System/Library/Java/Extensions:/usr/lib/java:.**]**

INFO: Initializing ProtocolHandler **[**"http-nio-8080"**]**

INFO: Using a shared selector **for** servlet write/read

INFO: Initializing ProtocolHandler **[**"ajp-nio-8009"**]**

INFO: Using a shared selector **for** servlet write/read

INFO: Initialization processed in 1946 ms

INFO: Starting service **[**Catalina**]**

INFO: Starting Servlet Engine: Apache Tomcat/9.0.10

INFO: At least one JAR was scanned **for** TLDs yet contained no TLDs. Enable debug logging **for** this logger **for** a complete list of JARs that were scanned but no TLDs were found in them. Skipping unneeded JARs during scanning can improve startup time and JSP compilation time.

WARNING: Creation of SecureRandom instance **for** session ID generation using **[**SHA1PRNG**]** took **[**262**]** milliseconds.

INFO: Starting ProtocolHandler **[**"http-nio-8080"**]**

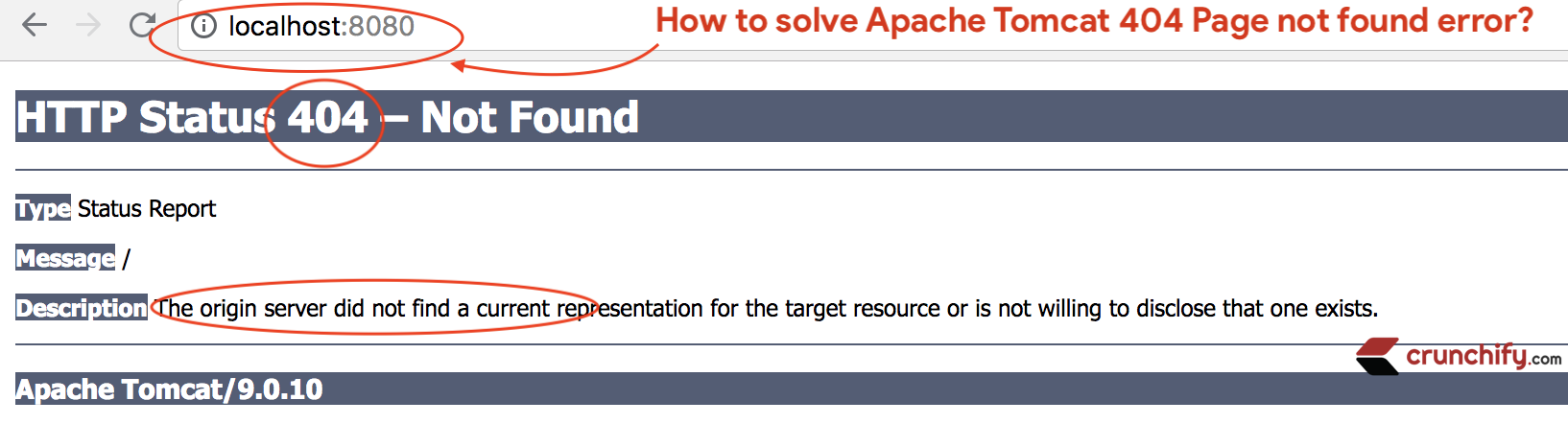
INFO: Starting ProtocolHandler **[**"ajp-nio-8009"**]**

INFO: Server startup in 1655 ms

It should be up and running on port 8080 and you could visit default page using URL: http://localhost:8080/

# Tomcat starts but Home Page does NOT open on browser with URL http://localhost:8080

Updated on Dec 20, 2020by App  [42](https://crunchify.com/tomcat-starts-but-home-page-does-not-open-on-browser-with-url-http-localhost8080/#comments)

[](https://crunchify.com/wp-content/uploads/2018/08/how-to-solve-Apache-Tomcat-404-Page-not-found-error.png)

Today I was running [Apache Tomcat](https://crunchify.com/error-starting-apache-tomcat-server-the-jre-could-not-be-found-edit-the-server-and-change-the-jre-location/) from [Eclipse](https://crunchify.com/ideal-eclipse-ini-file-setup-for-your-eclipse-environment-best-practice-for-macos-and-windows/) and while accessing URL http://localhost:8080 found HTTP Status 404 – Not Found error.

The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.

### Do you have any of below questions?

* Tomcat starts but doesn’t display webpage
* Can’t connect to Tomcat even though it’s running
* How to Solve Common [Tomcat Problems](https://crunchify.com/mavenmvn-clean-install-update-project-and-project-clean-options-in-eclipse-ide-to-fix-any-dependency-issue/)
* Can’t connect to localhost via browser. Can ping localhost
* How to open tomcat [home page](https://crunchify.com/how-to-insert-ads-on-home-page-after-2nd-and-5th-post-in-genesis-framework/) in browser
* localhost 8080 not working for tomcat

For all above types of issues,  you are at right place.

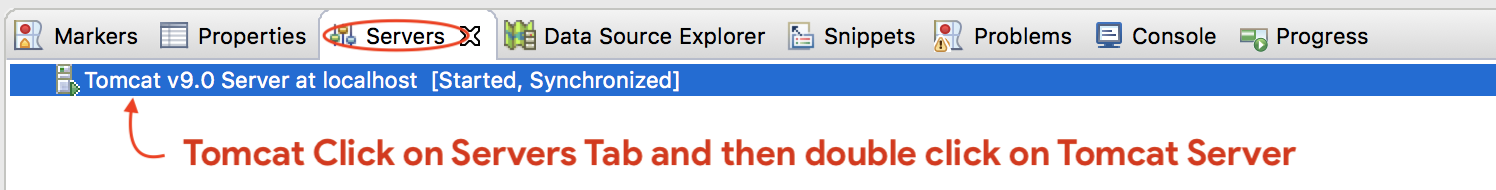
I’ve setup [Apache Tomcat by following detailed steps](https://crunchify.com/step-by-step-guide-to-setup-and-install-apache-tomcat-server-in-eclipse-development-environment-ide/) using in-depth tutorial.

Steps worked perfectly fine but as I didn’t have any projects added to tomcat webapps folder it threw 404 error for me.

If you also face 404 Page not found error then try following below steps:

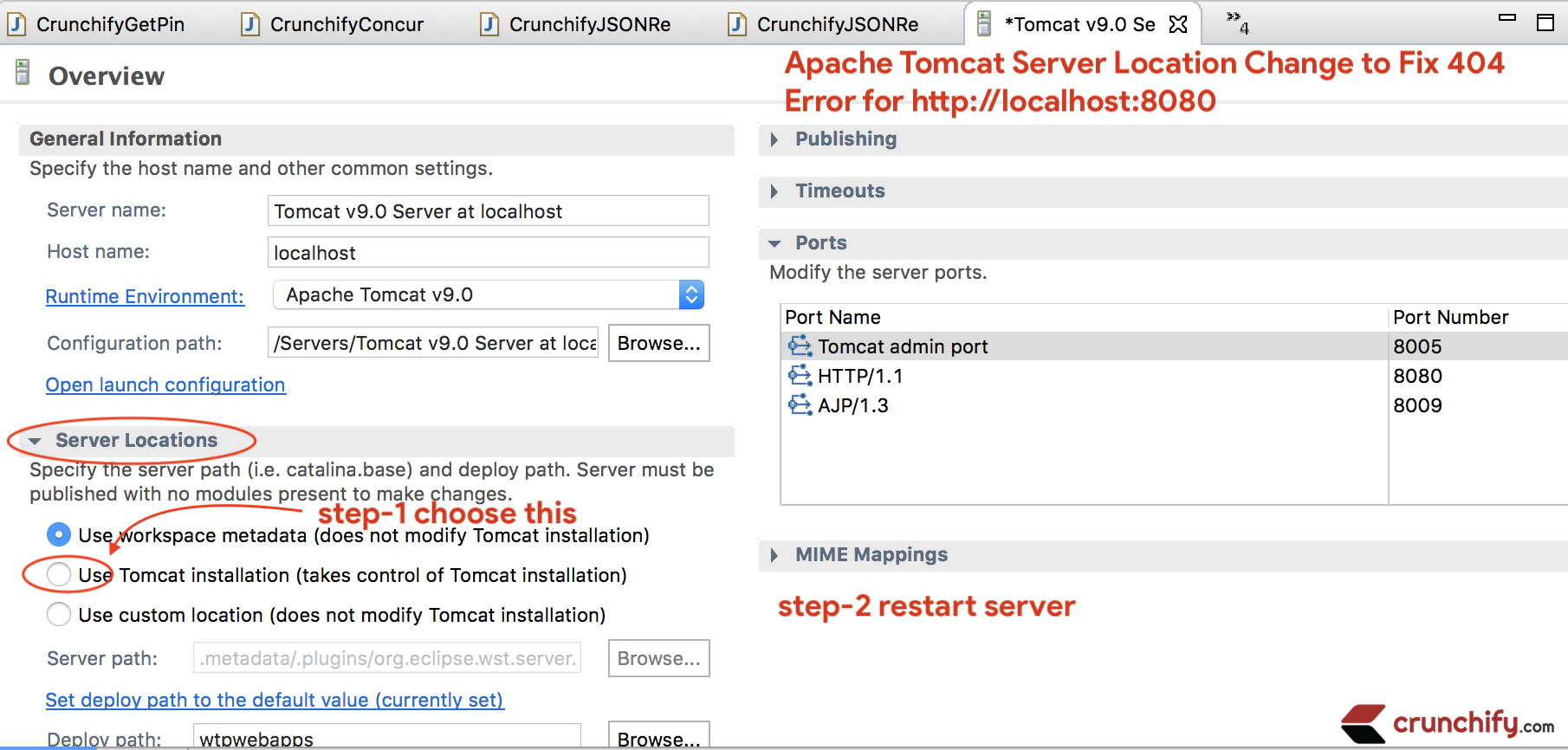
### Step-1

* Go to [Eclipse](https://crunchify.com/mavenmvn-clean-install-update-project-and-project-clean-options-in-eclipse-ide-to-fix-any-dependency-issue/) IDE
* Click on Servers Tab
* Double click on Tomcat v9.0 Server at localhost

[](https://crunchify.com/wp-content/uploads/2018/08/Tomcat-Click-on-Servers-Tab-and-then-double-click-on-Tomcat-Server.png)

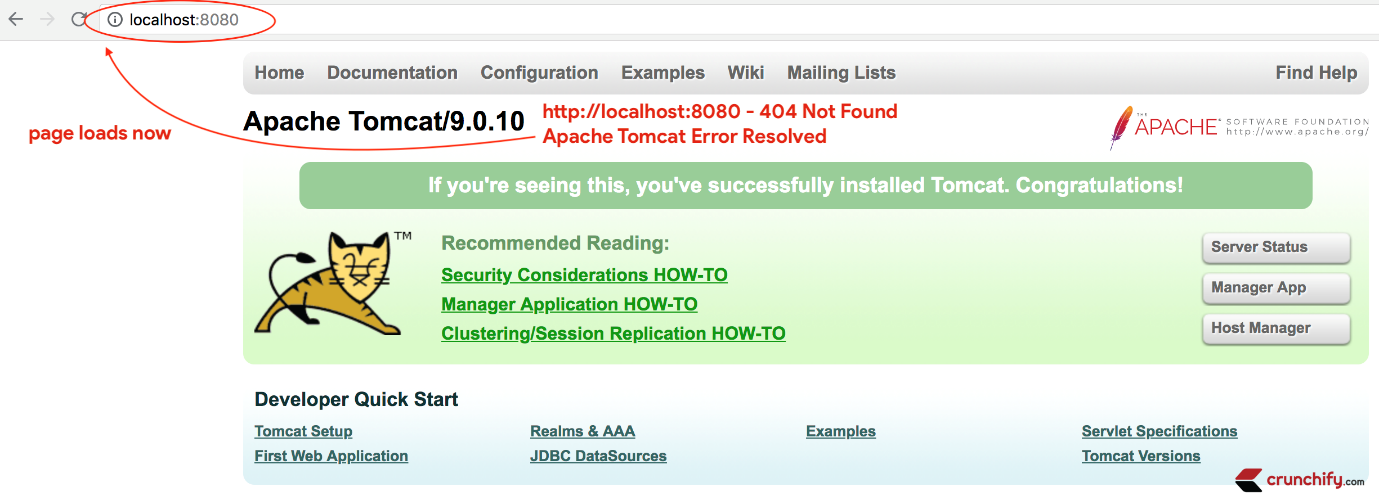
### Step-2

* New Apache [Tomcat configuration](https://crunchify.com/how-to-increase-tomcat-heapsize-jvm-heap-in-eclipse-ede/) page will open
* Go to Server Location section
* Select Use Tomcat installation (takes control of Tomcat installation)

[](https://crunchify.com/wp-content/uploads/2018/08/Apache-Tomcat-Server-Location-Change-to-Fix-404-Error.png)

### Step-3

* Save configuration
* Restart Server by right clicking on tomcat server and click Restart
* Visit http://localhost:8080 again and now you should see working tomcat page

[](https://crunchify.com/wp-content/uploads/2018/08/Localhost-8080-404-not-found-apache-tomcat-error-resolved.png)

I hope this tutorial works well for you. Happy coding and keep visiting.

**Steps To Create The Dynamic Web Project**

1.Press ctrl + n one window will appear as select wizard

2.The the Filter section dynamic web project

3.Then select dynamic web Project option and click on enter

4.One window will appear as dynamic web project there provide a

Project Name.

5.Then click on Next.

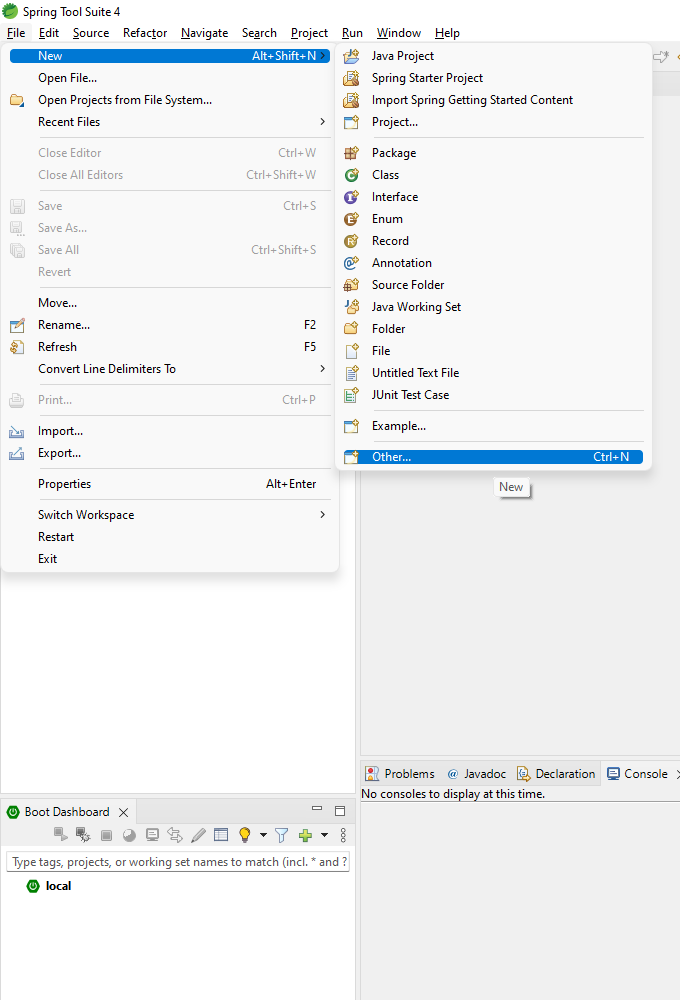
6.Again click on Next

7.In web Module window tick the Checkbox to generate web.xml deployment descriptor

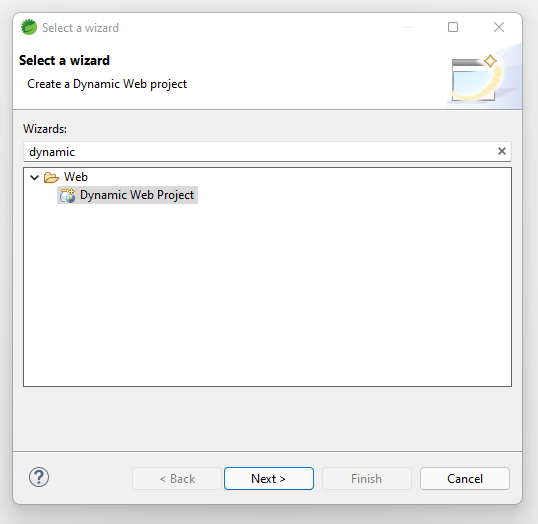
8.Then Click on Finish

**Implementation:**

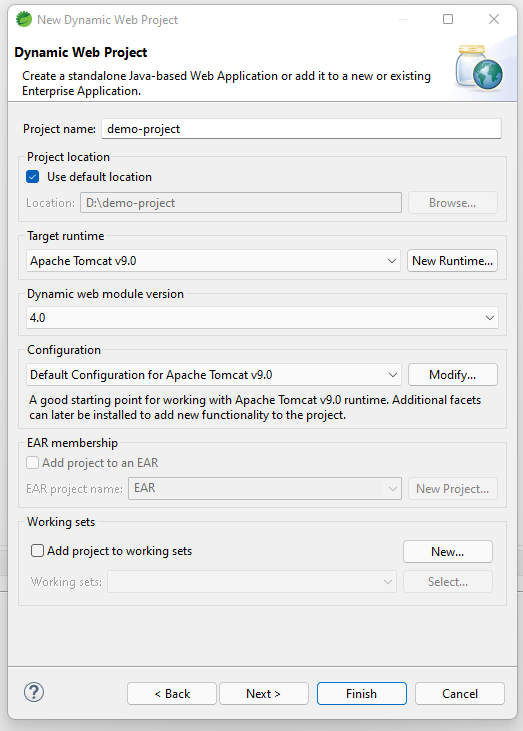
**Step 1:** Open your Eclipse/Spring Tool Suite IDE then go to the **File > New > Other** as shown in the below image.



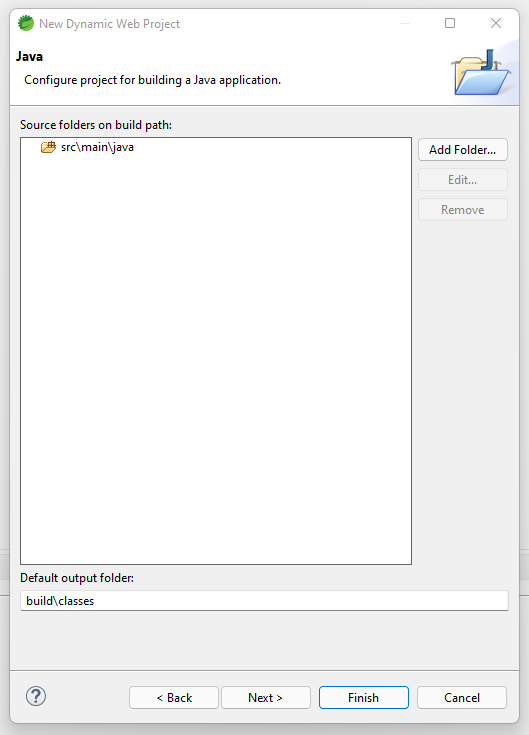
**Step 2:** Now in the select wizard search for the **Dynamic Web Project** as shown in the below image. And click on the **Next**button.



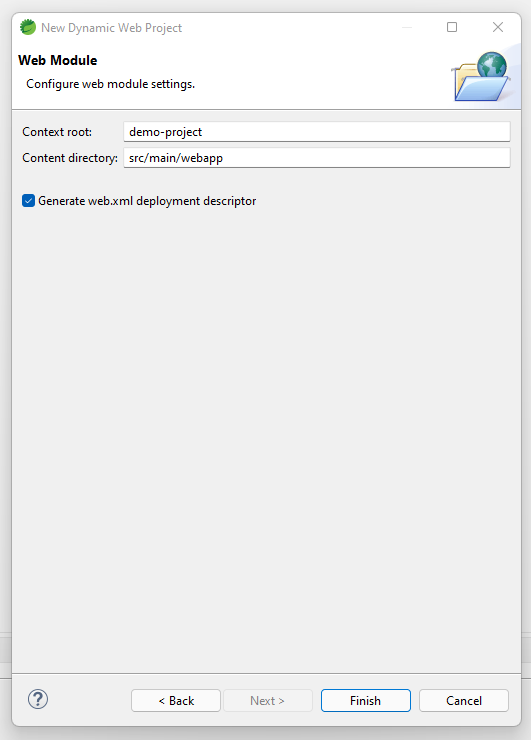
**Step 3:** In the next screen you have to provide your**project name** as per your choice and don’t touch anything else for now. Click on the **Next**button.



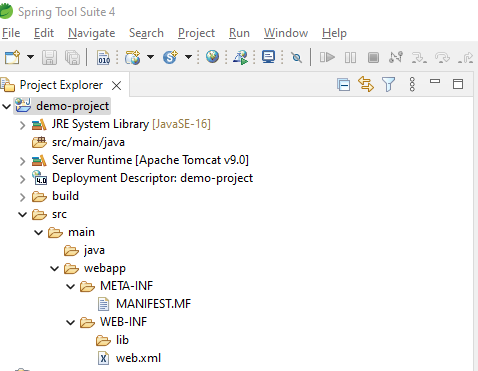
**Step 4:** In the next screen just click on the **Next**button.



**Step 5:**In the last screen just **check out the Generate web.xml deployment descriptor box**because we need it during the development of a Spring MVC project. Now click on the **Finish** button and you are done.



Now your Dynamic Web Project is ready and below is the file structure.



**WEB.XML**

-It is an Entry Point for the Dynamic Web Project

-It is also called as Deployment Descriptor.

-Web.xml can be used to configure the User Created Servlet.

**WEB-SERVER**

-It is the Software it is capable of accept the web request and generating the web response.

**SERVLET CONTAINER**

-it is the component of web server which contain the servlet.

-It is responsible for the managing the servlet Lifecyle.

-Servlet container will create the Object for the Servelt Request and Servlet Response.

**2nd Way Creating Servlet In Java**

**@webserver annotation**

-This annotation is used to provide theurl pattern for the user defined servlet.

-If we make you use of this annotation then configuring the user created servlet inside the web.xml is not required.

**Implicit Mapping Methods**

**doGet()**

-Whenever the request is send to the particular server a doGet() will be called automatically

-the doGet() is used to provide the execution logic for that particular servlet.

**doPost()**

-doPost() will be not be called automatically but we can make use of this method to provide the execution logic for the servlet.

**Note:**

-The html pages has to be created under webapp folder the input name given inside the HTML document will be considered as Parameter inside the Servlet.

**Difference between doGet() and doPost()**

|  |  |  |
| --- | --- | --- |
| SrNo | doGet() | doPost() |
| 1 | The doGet() method will called implicitly by the request | The doPost() has to be called explicitly |
| 2 | doGet() method is used to when some data has to be retrieved from the s | doPost() is used when some data has to be sent to the Server. |
| 3 | Data sent through URL by using doGet() method will be exposed will to the url | Data sent through the URL by using the doPost() method will be hidden in the URL |
|  |  |  |

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

JSP

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

**Implicit Object**

-The object which are present in the JSP file for use by default are called as implicit object.

Example:-

request,repsonce,out

**JSP LIFECYCLE**

-Every JSP file will be translated to a corresponding a servlet class

-After translation the Lifecycle of the JSP will remains same as that of Servlet Lifecycle.

**1.Translation Phase**

-Before beginning its Lifecycle, a JSP file is internally translated into a Servlet class file this is how we can say that the JSP technology is based on Servlet technology.

-After the translation the JSP file can accept web request and can generate web response just like a Servlet file.

**2.Servlet Class Will be Loaded**

**3.Object will Created for That Particular Class**

**4.JSP init() method will be invokes only once**

**====🡺1 time**

**5.JSP service() method will be invokes.**

**===🡺n times**

**6.JSP destroy() method will be invokes.**

**==🡺1 times**

SPRING FRAMEWORK

-Spring is a opensource and light weight framework.

-It is used to develop the java enterprise edition.

-Spring framework is dedicated for java which help java developers to developed enterprise application in more efficient and productive way.

**Spring Modules:**

**1.Spring Core**

**2.Spring MVC**

**3.Spring REST**

**4.Spring Boot**

**1.Spring Core**

-Spring core module deals with the main features of spring framework.

**2.Spring MVC**

-Spring MVC module deal with the development of monolithic application.

**3.Spring REST**

-Spring REST module deal with the implementation of Restful API’s.

**4.Spring Boot**

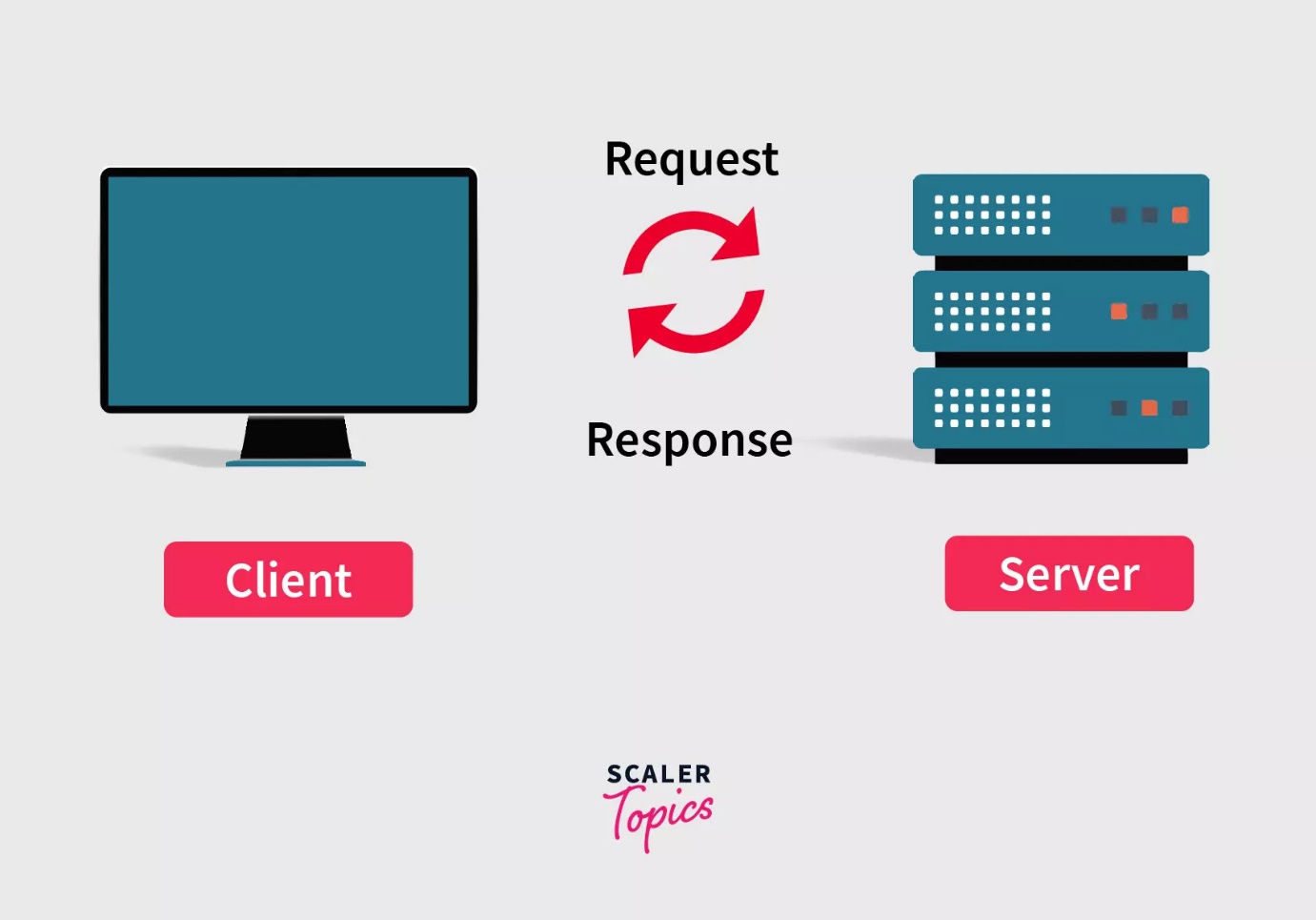
-Spring boot modules deals with the development of web service or micro service.

**1.Spring Core**

-Features of Spring Framework

**1.IOC**

**2.DI**

****

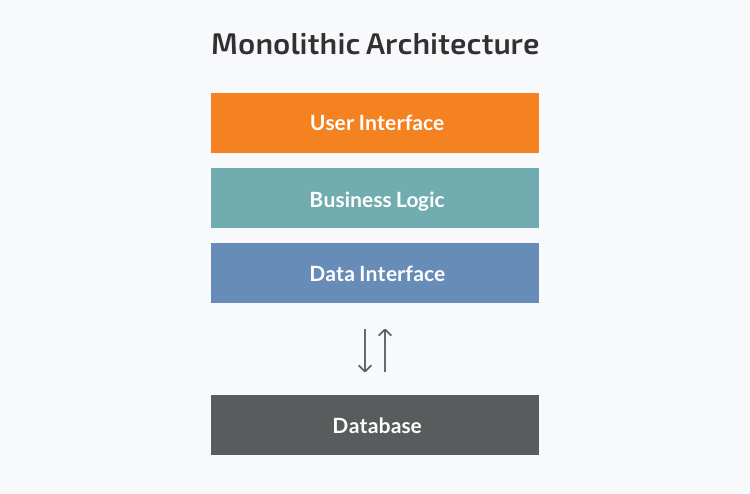
-Spring can be used to develop the monolithic applications as well as Server-Side Application.

-An application which contain the both the Front-End Logic and The Back End Logic is a called as Monolithic application.

-In monolithic application the Front-End logic and Back-end logic are developed and deployed as a Single application.

-Although the Front-End and Back-End logic can be develop independently as the client side application and Server Side application respectively

-In this case both the applications are deployed independently as two different applications.



**Spring Core**

-Spring Core module represent the features of the Spring Framework

-There are two key features of the Spring Framework

**1]IOC**

**2]DI**

**1]IOC**

-IOC stands for Inversion Of Control

-In spring Framework, the programmer is no longer n control or the responsible for the Object creation and Object management.

-Instead the spring Framework will take the help of component known as “**Spring Container**” or “**IOC Container**”.

-This component is responsible to create the object and manage their Lifecycle in Spring Framework.

-The object created in the Spring Framework are Light-Weight and hence are known as **Beans**.

-In order to create this Beans, The Spring Container or IOC Container will take the help of “**Bean Factory**”.

-Even if the Spring Framework is in control of Bean Creation, the programmer is still responsible to provide the values for the **Bean Properties**.

-The process of the Providing the Property values for a Bean is called as **Bean Configuration**.

**Bean Configuration**

-The For the Bean properties can be provided using two ways

**1]XML Based Configuration**

**2]Annotation Based Configuration**

**2]DI(Dependency Injection)**

-DI stands for Dependency Injection

-The class which depends upon another class for its own Bean creation is known as a depended class.

-The on which it depends upon is called as the dependency class.

-As in the Spring Framework, the programmer is not responsible to create any objects(Beans),hence the Spring Framework itself has to be create a bean of the dependency class and inject it into the bean of the depended class.

-This Feature of the Spring Framework is known as “**Dependency Injection**”.

**XML CONFIGURATION**

-To configure a bean, we can make use of an XML tag ‘**Bean**’ in which we can provide the values for the Bean Properties using the “**<property>**” tag “**<value>**”.

-For this we need to create an XML file to provide the configuration

**Dependencies required for Spring Core**

1.Spring core

2.Spring Context

3.Project Lombok(Optional)

NOTE :

-The XML file use for the Bean configuration purpose has to be created directly under the Source Folder(src/main/java)

-To provide the Bean configuration we first need to include the Schema Definition which provides us the Beans tags inside which multiple Bean tags can be define.

NOTE

-The bean configuration can be done with the help of the All argumented constructor as well.

-In that case we need to use the “**<constructor-arg>**” tag in place of the property tag.

-To get the all argumented constructor for the class we can use “**@AllArgsConstructor**” which is present in the Project Lombok package

-Similarly we can get a Non-Parameterized constructor for the class with the help of “**@NoArgsConstructor**” which is present in the Project Lombok Package.

**StudentConfig.xml**

<bean class=”com.jspiders.springcore.bean.StudentBean” name=”Student2”>

<Constructor-arg name=”id”>

<Value>2<value>

</constructor-arg>

<constructor-arg name=”name” value=”sujay”/>

<constructor-arg name=”email” value[suajay@gmail.com](mailto:suajay@gmail.com)/>

</bean>

**StudentBean2**

{

Int id;

String name;

String email;

}

**StudentMain.java**

Public class StudentMain

{

Public static void main(String args[])

{

ApplicationContext context = new ClassPathXmlApplicationContext(“StudentConfig.xml”);

StudentBean bean = context.getBean(“StudentBean2”);

System.out.println(student2);

((ClassPathXmlApplicationContext)context).close();

}

}

**getBean()**

-It is a Non-Static overloaded method present in ApplicationContext Interface

-This method is used to obtain or achieve the bean of the required class.

-This method can we used in two ways that is by passing by class as the argument or by passing the String which is considered as Bean Name as the argument

-If a class is given as the argument to this method, then the method will return the Bean for that specific class itself.

-But if the bean name is given as an Argument(String as an Argument) then the Method will return the Object of SuperMost class(Object).

-In this case we need to use the required class as the Cast Operator in order to Downcast the Object return by the Method to the required Class Type.

**ONE-TO-ONE**

Package com.jspiders.sprincore.bean;

@Data

**PersonBean.java**

{

Int id;

String name;

AadharBean aadhar;

}

Package com.jspiders.sprincore.bean;

@Data

@AllArgsConstructor

@NoArgsConstructor

**AadharBean.java**

{

Int id

long aadharNo;

}

**PersonAadharConfig.xml**

<bean class=”com.jspiders.springcore.bean.PersonBean”

name=”Person1”>

<property name=”id” value=”1”></property >

<property name=”name” value=”sachin”></property >

<property name=”aadhar” ref=”aadhar1”></property >

</bean>

<bean class=”com.jspiders.springcore.bean.AadharBean”

name=”aadhar1”>

<constructor-arg name=”id” value=”1”/>

<Constructor-arg name=”aadharNo” value=”12345567466”></constructor-arg>

</bean>

**PersonMain.java**

Public class PersonMain

{

Public static void main(String args[])

{

ApplicationContext context = new ClassPathXmlApplicationContext(“PersonAadharConfig.xml”)

PersonBean person1 = context.getBean(PersonBean.class);

System.out.println(person1);

((ClassPathXmlApplicationContext)context).close();

}

}

**One -To-Many**

**CompanyBean.java**

@Data

Public class Company

{

Int id;

String name;

List <EmployeesBean> employees;

}

**EmployeeBean.java**

@Data

Public class EmployeeBean

{

Int id;

String name;

String address;

}

**CompanyEmployee.config.xml**

<bean class=”com.jspiders.springcore.bean.CompanyBean”

name=”Company1”>

<property name=”id” value=”1”></property >

<property name=”name” value=”Tcs”></property >

<property name=”employees”>

<list>

<bean class=”com.jspiders.springcore.bean.

EmployeeBean” name=”employee1”>

<property name=”id” value=”1”/>

<property name=”name” value=”Aman”/>

<property name=”address” value=”Ravet”

</bean>

<bean class=”com.jspiders.springcore.bean.

EmployeeBean” name=”employee2”>

<property name=”id” value=”2”/>

<property name=”name” value=”Suman”/>

<property name=”address” value=”Hinjewadi”

</bean>

<list>

</property >

</bean>

**CompanyMain.java**

Public class CompanyMain

{

Public static void main(String args[])

{

ApplicationContext context = new ClassPathXmlApplicationContext(“CompanyEmployeeConfig.xml”);

CompanyBean company = context.getBean(CompanyBean.class);

System.out.println(company1);

((ClassPathXmlApplicationContext)context).close();

}

}

**Annotation Configuration**

-In spring Framework in order to provide the Bean Configuration

Through the annotation we can make use of the following annotations

**1]@Bean**

**2]@Value**

**3]@Component**

**4]@ComponnentScan**

**5]@Autowired**

**1]@Bean**

-This annotation is used to mark the logic (method) as the required bean configuration.

-This annotation will create the bean based on the value returned by the logic(method).

-This annotation can be included in a class.

**=================== NOTE ==================**

-In annotation configuration the object of application context interface has to be initialized with the help of class “AnnotationConfigApplicationContext”

-The constructor of this class accepts a class type as an argument in which @Bean will be present

@Data

**StudentBean.java**

{

Int id;

String name;

String email;

}

**StudentConfig.java**

Public class StudentConfig

{

@Bean

Public StudentBean getStudent()

{

StudentBean student1 = new StudentBean();

student1.setId(1);

student1.setName(“Arjun”);

student1.setEmail.(“[arjun@gmail.com](mailto:arjun@gmail.com)”);

return student1;

}

}

**StudentMain.java**

Public class StudentMain

{

Public static void main(String args[])

{

ApplicationContext context = new AnnotationConfigApplicationContext(StudentConfig.class);

StudentBean student1 = context.getBean(StudentBean.class)

System.out.println(student1);

((AnnoatationConfigApplicationContext)context).close();

}

==============================================

@Data

@AllArgsConstructor

@NoArgsConstructor

**StudentBean.java**

{

Int id;

String name;

String email;

}

**StudentConfig.java**

Public class StudentConfig

{

@Bean

Public StudentBean getStudent()

{

StudentBean student1 = new StudentBean();

student1.setId(1);

student1.setName(“Arjun”);

student1.setEmail.(“[arjun@gmail.com](mailto:arjun@gmail.com)”);

return student1;

}

@Bean(“student2”)

Public StudentBean getStudent()

{

return StudentBean(2,”Karan”,[karan@gmail.com](mailto:karan@gmail.com));

}

}

**StudentMain.java**

Public class StudentMain

{

Public static void main(String args[])

{

ApplicationContext context = new AnnotationConfigApplicationContext(StudentConfig.class);

StudentBean student2 = (StudentBean) context.getBean(“student2”)

System.out.println(student2);

((AnnoatationConfigApplicationContext)context).close();

}

**2]@Value**

-This annotation is used to provide the Setter level configuration and initialize the value of a property at declaration itself.

@Data

@AllArgsConstructor

@NoArgsContructor

**EmployeeBean.java**

Public class EmployeeBean

{

@Value(“1”)

Private int id;

@Value(“Amit”)

Private String name;

@Value(“amit@gamil.com)

Private String email;

@Value (“Wakad”)

Private String address;

}

**EmployeeConfig.java**

Public class EmployeeConfig

{

@Bean

Public class EmployeeBean getEmployee()

{

return new EmployeeBean();

}

}

**EmployeeMain.java**

Public class EmployeeMain

{

Public static void main(String args[])

{

ApplicationContext context = new AnnotationConfigApplicationContext(Employee.class);

EmployeeBean employee1 = context.getBean(EmployeeBean.class);

System.out.println(employee1);

((AnnoatationConfigApplicationContext)context).close();

}

}

**===================== NOTE ====================**

-The Spring container gives priority to the setter configuration over the constructor configuration.

-If a bean is configured using the All Argumented constructor but id the Bean properties are annotated with **@Value** then the spring container will override the constructor values with a Setter value

**@Component**

-This annotation is used to mark a class to be eligible for the process of Bean creation

-It means if class is mark as **@Component**, then it is automatically considered by the Spring Container for Bean container.

**@ComponentScan**

-This annotation is used to mark a class to be able to scans the Classes or the Packages for a @Component annotation.

**@Autowired**

-This annotation is used to mark a property or a constructor for dependency injection.

-If this annotation is given to a property, then Setter Injection will be performed.

-If this annotation is given to a constructor (Argumented Constructor required) then the Constructor Injection will be performed.

**#Program**

**AadharBean.java**

@Data

@Component

Public class AadharBean

{

@Value(“1”)

Private int id;

@Value(“1122 3344 5566”)

Private long aadhaarNo;

}

**PersonBean.java**

@Data

Public class PersonBean

{

@Value(“1”)

Private int id;

@Value(“Aman”)

Private String name;

@Autowired

Private AadharBean aadhar;

}

**PersonConfig.java**

@ComponentScan(basePackages = “com.jspiders.springcore)

Public class PersonConfig

{

@Bean

Public PersonBean getPerson()

{

Return new PersonBean();

}

}

**PersonMain.java**

Public class PersonMain

{

Public static void main(String args[])

{

ApplicationContext context = new AnnoatationConfigApplicationContext(Person.class);

PersonBean person1 = context.getBean(PersonBean.class);

System.out.println(Person1);

(AnnoatationConfigApplicationContext)Context).close();

}

}

Difference between Setter Injection and Constructor Injection

|  |  |  |
| --- | --- | --- |
| SrNo | Setter Injection | Constructor Injection |
| 1 | Setter Injection is Prioritized by the Spring Container, Over the Constructor Injection | The Constructor Injection  Is not Prioritized if the Setter Injection is exists. |
| 2 | The **@Autowired** annotation is given to the Properties. | The **@Autowired** annotation is give to the Constructor (Parameterized Constructor) |
| 3 | Initializing a Partial Properties of the Bean is allowed in case of Setter Injection. | Initializing partial arguments of the constructor is not allowed |
| 4 | If a Property is not initialized then depending on its Data Type, The default value is considered | If an argument is not provided then the Constructor cannot execute. |

**MVC**

-MVC stands for Model View Controller

-MVC is an Architectural Design Pattern

-It provides the architectural support for Monolithic Applications.

-In MVC architecture there are three main components.

**1]Model**

**2]View**

**3]Controller**

**1]Model**

-The model holds the Backend Logic for the application which is responsible to perform the required business logic and communicate with the database

-After execution, the model returns the Model data

**2]View**

-The View holds the Front-End logic for the application.

-It contains all the required View-Pages(JSP or HTML) which can be given as the web-response.

-The Web-Responses are given as the view pages that contain the model data(Whenever required).

**3]Controller**

-The Controller is responsible to accept the web request analyze it and decides which logic from the model has to be executed.

-After receiving the model data it will identify the View Page that is to be given as the Web-Response along with the Model data if required.

-We can say that the controller acts as Mediator between the Model (Back-End Logic) and View(Front-End logic).

**View Model**

**Controller**

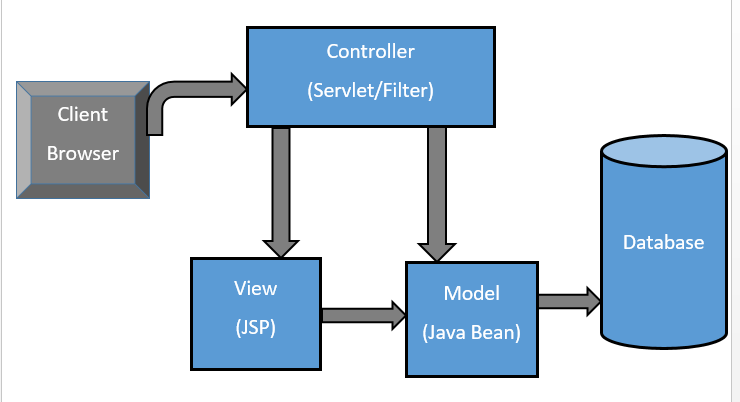
**Back End**

**Front End**

**MVC ARCHITECTURE**

-The MVC architecture manage the flow of data within the application .

-It also determines how the request will be handled in the application and how will be the response will provided.



**Bowers**

**View**

**Response**

**View**

**Resolver**

**Web Request**

**Controller**

**Dispatchers Servlet**

**Handling**

**Mapping**

**Model Data**

**Execute Business Logic**

**MODEL**

**DB**

**Repository**

**POJO**

**Service**



**1]Dispatcher Server**

-The Dispatcher Servlet component is configured in Web.xml

-Even if an Entry point for all the request coming to for application is Web.xml, it will be now forwarded to the dispatcher Servlet.

-It means, all the incoming web request will be received by the Dispatcher Servlet.

-The Dispatcher Servlet take the help of **Handler Mapping Component** in order to identify which controller is responsible to handle that specific request.

-The Dispatcher Servlet is also containing the configuration for the **View Resolver Component** which helps it to identify and to locate the specific View Pages that are supposed to be given as Web Response.

**2]Controller Layer**

-The controller layer is responsible to analyse the incoming web request based on the URL pattern and Mapping Method

-Once the request is analysed, the controller decides which logic from the service layer has to be executed.

-After the logic from the Service layer is done executing controller received the model data from it.

-Now the Controller identifies or defines that which view page has to be given as the response.

-Hence the controller has to sent the model data to the identified View page, located with the help of Dispatcher Servlet which internally refer to the View Resolver.

-The identified View page given as Web Response along with the model data.

**==================== NOTE =====================**

-In Spring MVC model is classified into three layers

1]Service

2]Repository

3]POJO

**3]Service**

-The Service layer is responsible to perform the validation on the incoming or the Outgoing Data

-It is also responsible to decide the which logic from the Repository layer should be executed and them return the Mode data back to the controller.

**4]Repository**

-The Repository layer is responsible to perform all the executional logic of the entity class Object from the POJO layer

-We may say that the Repository layer will include the **Hibernate Logic**

**5]POJO**

-POJO stands for **Plane Old Java Object.**

-The POJO layer holds the all the entity classes required for the application which will not contain any Executional logic.

**==================== NOTE =====================**

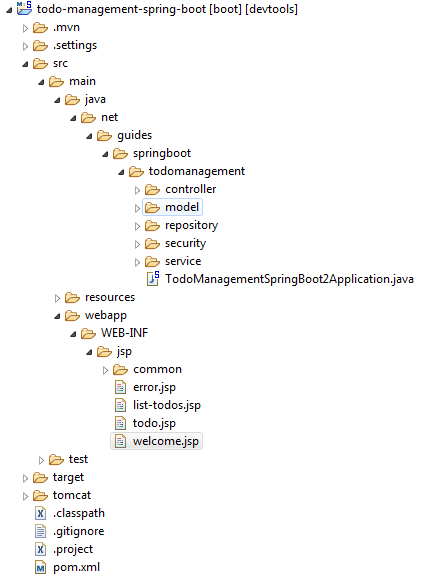
-The Controller, Service, Repository, POJO layer are defined as Packages in the Project and the classes under this packages must be annotated as **@Controller, @Service @Repository** and **@Entity** respectively

**==================== NOTE =====================**

-**@Controller, @Service @Repository** and **@Entity** are the combination of **@Target @Retention @ Documented @ Component** Annotation

**==================== NOTE =====================**

-To create the MVC application We need create a Project “**Maven Webapp 1.4**”



## Spring MVC Project Structure

In a typical Spring MVC web application, the web application packaging structure may look:

── pom.xml

└── src

├── main

│ ├── java

│ │ └── com

│ │ └── companyname

│ │ └── projectname

│ │ ├── domain

│ │ │ └── MyDomain.java

│ │ ├── repository

│ │ │ └── MyDomainRepository.java

│ │ ├── service

│ │ │ ├── MyDomainService.java

│ │ │ └── internal

│ │ │ └── MyDomainServiceImpl.java

│ │ └── web

│ │ └── MyDomainController.java

│ ├── resources

│ │ ├── META-INF

│ │ │ └── spring

│ │ │ ├── applicationContext.xml

│ │ │ └── database.properties

│ │ ├── logback-access.xml

│ │ └── logback.xml

│ └── webapp

│ ├── WEB-INF

│ │ ├── classes

│ │ ├── i18n

│ │ ├── layouts

│ │ ├── spring

│ │ │ └── webmvc-config.xml

│ │ ├── views

│ │ │ ├── myDomain

│ │ │ │ ├── create.jsp

│ │ │ │ ├── list.jsp

│ │ │ │ ├── show.jsp

│ │ │ │ └── update.jsp

│ │ │ ├── dataAccessFailure.jsp

│ │ │ ├── index.jsp

│ │ │ ├── resourceNotFound.jsp

│ │ │ ├── uncaughtException.jsp

│ │ │ └── views.xml

│ │ └── web.xml

│ ├── images

│ └── styles

├── site

│ ├── apt

│ ├── fml

│ ├── site.xml

│ └── xdoc

└── test

├── java

│ └── com

│ └── companyname

│ └── projectname

│ └── service

│ └── MyDomainServiceTests.java

└── resources

├── com

│ └── companyname

│ └── projectname

│ └── service

│ └── MyDomainServiceTests-context.xml

└── logback-test.xml

**Production**

* **src/main/java** – Java Source code packages and classes
* **src/main/resources** – NON-Java Resources, such as property files and Spring configuration
* **src/main/webapp** - deployment descriptor(web.xml), WEB-INF, JSP views under WEB-INF folder, static resources.

**Test**

* **src/test/java** – Test Source code packages and classes
* **src/test/resources** – NON-Java Resources, such as property files and Spring configuration

### Spring MVC - placing your JSP files in a directory under the 'WEB-INF' directory

As a best practice, we strongly encourage placing your JSP files in a directory under the 'WEB-INF' directory so there can be no direct access by clients.

Example:

│ └── webapp

│ ├── WEB-INF

│ │ ├── classes

│ │ ├── i18n

│ │ ├── layouts

│ │ ├── spring

│ │ │ └── webmvc-config.xml

│ │ ├── views

│ │ │ ├── myDomain

│ │ │ │ ├── create.jsp

│ │ │ │ ├── list.jsp

│ │ │ │ ├── show.jsp

│ │ │ │ └── update.jsp

│ │ │ ├── dataAccessFailure.jsp

│ │ │ ├── index.jsp

│ │ │ ├── resourceNotFound.jsp

│ │ │ ├── uncaughtException.jsp

│ │ │ └── views.xml

│ │ └── web.xml

**Configuration for MVC**

-We need to create the layers of the model as Packages along with the Controller Layer.

-The dependency is required for the MVC must be added int the **POM.xml**

**========== Dependencies For Spring MVC =========**

**1]MySQL Connector**

**2]Hibernate Core**

**3]Hibernate Entity Manager**

**4]Spring MVC**

**5]Spring ORM**

**6]Servlet API**

**7]Project Lombok**

1]Provide the annotation to the respective layers

2]We need to configure the Dispatcher Servlet in **Web.Xml**

<Servlet>

<Servlet-name>dispatcher</Servlet-name>

<Servlet-class>org.springframework.web.servlet.DispactcherServlet</Servlet-class><Servlet>

<Servlet>

<Servlet-mapping>

<Servlet-name>dispatcher</Servlet-name>

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

</Servlet-mapping>

3]To obtain the class path of the Dispatcher Servlet we can make use of “ **Open Type**” window which can be access through a shortcut “**ctrl+shift+t**”

4]The “**/**” forward slash url pattern indicate that all the incoming request will be mapped to the Dispatcher Servlet

5]To provide the Configuration Handler Mapping and View Resolver Component, we need to create the “**dispatcher-servlet.xml**” in the same location as that of **web.xml** (inside **WEB-INF** folder).

<!-- HandlerMapping configuration -->

<context:component-scan

base-package="com.jspiders.springmvc" />

<!-- ViewResolver configuration -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/" />

<property name="suffix" value=".jsp" />

</bean>

User.java

Package com.jspiders.springcorexml.bean;

Import Lombok.data;

@Data

Public class user {

Private int id;

Private String email;

Private String password;

}

User\_config.xml

<beans>

<bean class=”com.jspiders.springcorexml.bean.user”>

<property name=”id” value=”1”/>

<property name=”email” value=”24amolpawar@gmail.com”/>

<property name=”password” value=”Amol@123”/>

</bean>

</beans>

UserMain.java

Public class UserMain{

Public static void main(String[] args)

{

ApplicationContext applicationContext = new ClassPathXmlApplicationContext(“user\_config.xml”);

User user = applicationContext.getBean(User.class);

System.out.println(user);

((ClassPathXmlApplicationContext) ApplicationContext).close();

}

}

-The above way of getting a bean using class name has one this disadvantage that is if a class has multiple objects then there is a ambiguity or confusion for the Spring container to return anyone of the objects of that particular class.

-To overcome this problem we can make use of another way of getting a bean as follows.

@Data

Student.java

{

Private int id;

Private String name;

Private String email;

Private int age;

}

Student\_Config.xml

<beans>

<bean class=”com.jspiders.springcorexml.bean.user” name=”student1”>

<property name=”id” value=”1”/>

<property name=”name” value=”Amol”/>

<property name=”email” value=”24amolpawar@gmail.com”/>

<property name=”age” value=”24”/>

</bean>

<bean class=”com.jspiders.springcorexml.bean.user” name=”student2”>

<property name=”id” value=”2”/>

<property name=”name” value=”Ganesh”/>

<property name=”email” value=”ganeshgidde@gmail.com”/>

<property name=”age” value=”24”/>

</bean>

</beans>

StudentMain.java

{

Public static void main(String[] args)

{

ApplicationContext applicationContext = new ClassPathXmlApplicationContext(“student\_config.xml”);

Student student1 =(Student) applicationContext.getBean(student1);

System.out.println(Student1);

Student student2 =(Student) applicationContext.getBean(student2);

System.out.println(Student2);

((ClassPathXmlApplicationContext) ApplicationContext).close();

}

}

-In the above program Spring makes use of setters methods to initialize the properties of an object.

-Spring can initialize properties of an object by making use of all arguments constructor as follows.

Mobile.Java

@AllArgsConstructor

Public class Mobile

{

Private int id;

Private String name;

Private double price;

@Override

Public String toString(){

Return “Mobile [id=”+id +”,name=”+name+”,price”+price+”]”;

}

}

Mobile\_Config.xml

<beans>

<bean class=”com.jspiders.springcorexml.bean.mobile” name=”mobile”>

<constructor-arg name=”id” value=”1”/>

<constructor-arg name=”name” value=”Nord 3”/>

<constructor-arg name=”price” value=”35000”/>

</bean>

<beans>

MobileMain.java

{

Public static void main(Srting[] args)

{

ApplicationContext applicationContext = new ClassPathXmlApplicationContext(“mobile\_config.xml”);

Mobile mobile = (mobile) applicationContext.getBean(“mobile”);

System.out.println(mobile);

((ClassPathXmlApplicationContext)applicationContext).close();

}

**Aadharcard.java**

@Data

Public class AadharCard{

Private int id;

Private int aadharNumber;

}

**Person.java**

@Data

Public class Person{

Private int id;

Private String name;

Private String email;

Private AadharCard aadharCard;

}

**Person\_Config.xml**

<beans>

<bean class=”com.jspiders.springcorexml.bean.AadharCard” name=aadharCard”>

<property name=”id” value=”1”/>

<property name=”aadharNumber” value=”9373428581”/>

</bean>

<bean class=”com.jspiders.springcorexml.bean.pERSON” name=aadharCard”>

<property name=”id” value=”1”>

<property name=”name” value=”Amol”/>

<property name=”email” value=”24amolpawar@gmail.com”/>

<property name=”aadharCard” ref=” aadharCard”/>

</bean>

</beans>

**PersonMain.java**

Public class PersonMain{

Public static void main(String[] args)

{

ApplicationContext applicationContext = new ClassPathXmlApplicationContext(“Person\_config.xml”);

Person person = (Person) applicationContext.getBean(“person”);

System.out.println(person);

((ClassPathXmlApplicationContext) applicationContext).close();

}

}

**Employee.java**

@Data

Public class Employee{

Private int id;

Private String name;

Private String email;

}

**Companay.java**

@Data

Public class Company

{

Private int id;

Private String name;

Private String location;

Private List<Employee> employees;

}

**Company\_Config.xml**

<beans>

<bean class=”com.spiders.springcorexml.bean.company” name=”company”id value=”1”>

<property name=”name” value=”Tata”>

<property name=”location” value=”Pune”>

<property name=”employess” >

<list>

<bean class=”com.jspiders.springcorexml.bean.Employee” name=”employee1”>

<property name=”id” value=”1”>

<property name=”name” value=”Ramesh”>

<property name=”email” value=”ramesh@gmail.com”>

</bean>

<bean class=”com.jspiders.springcorexml.bean.Employee” name=”employee2”>

<property name=”id” value=”2”>

<property name=”name” value=”suresh”>

<property name=”email” value=”suresh@gmail.com”>

</bean>

</list>

</property>

</bean>

</beans>

**CompanyMain.java**

Public class CompanyMain{

Public static void main(String[] args)

{

ApplicationContext applicationContext = new ClassPathXmlApplicationContext(“Company\_config.xml”);

Company company = (Company) applicationContext.getBean(“company”);

System.out.println(company);

((ClassPathXmlApplicationContext)applicationContext).close();

}

}

**Team.java**

@Data

Public class Team{

Private ling id;

Private String name;

}

**Team.java**

@Data

Public class Team{

Private long id;

Private String name;

Private Team team;

}

**Player\_Config.xml**

<beans>

<bean class=”com.jspidres.springcorexml.bean.Team” name=”team”>

<property name=”id” value=”1”/>

<property name=”name” value=”ABC”/>

</bean>

<bean class “com.jpsiders.springcorexml.bean.player” name=”player1”>

<property name=”id” value=”1”/>

<property name=”name” value=”Ramesh”/>

<property name=”team” ref=”team”></property>

</bean>

<bean class “com.jpsiders.springcorexml.bean.player” name=”player2”>

<property name=”id” value=”2”/>

<property name=”name” value=”Suresh”/>

<property name=”team” ref=”team”></property>

</bean>

</beans>

**PlayerMain.java**

Public class PlayerMain{

Public static void main(String args[])

{

ApplicationContext applicationContext = new ClassPathXmlApplicationContext(“player\_config.xml”);

Player player1 = (Player) applicationContext.getBean(“player1”);

System.out.println(player1);

Player player2 = (Player) applicationContext.getBean(“player2”);

System.out.println(player2);

((ClassPathXmlApplicationContext) appliactionContext).close();

}

}

**Product.java**

@Data

Public class Product{

Private long id;

Private String name;

Private double price;

}

**Customer.java**

@Data

Public class Customer{

Private long id;

Private String name;

Private long contact;

Private List<Product> products;

}

**Customer\_Config.xml**

<bean class=”com.jspiders.springcorexml.bean.Customer” name=”customer1”>

<property name=”id” value=”1”/>

<property name=”name” value=”Ramesh”/>

<property name=”contact” value=”9373428581”/>

<property name=”products”>

<list>

<bean class=”com.jspiders.springcore.xml.bean.Product” name=”Product1”>

<property name=”id” value=”1”/>

<property name=”name” value=”Nord3”/>

<property name=”price” value=”35000.00”/>

</bean>

<bean class=”com.jspiders.springcore.xml.bean.Product” name=”Product2”>

<property name=”id” value=”2”/>

<property name=”name” value=”Ideapad3”/>

<property name=”price” value=”65000.00”/>

</bean>

</list>

</property>

</bean>

<bean class=”com.jspiders.springcorexml.bean.Customer” name=”customer2”>

<property name=”id” value=”2”/>

<property name=”name” value=”Suresh”/>

<property name=”contact” value=”9373427581”/>

<property name=”products”>

<list>

<bean class=”com.jspiders.springcore.xml.bean.Product” name=”Product1”>

<property name=”id” value=”1”/>

<property name=”name” value=”Nord3”/>

<property name=”price” value=”35000.00”/>

</bean>

<bean class=”com.jspiders.springcore.xml.bean.Product” name=”Product2”>

<property name=”id” value=”2”/>

<property name=”name” value=”Ideapad3”/>

<property name=”price” value=”65000.00”/>

</bean>

</list>

</property>

</bean>

**CustomerMain.java**

Public class CutomerMain{

Public static void main(String args[])

{

ApplicationContext applicationContext= new ClassPathXmlAppliactionContext(“customer\_confg.xml”);

Customer customer1=(Customer) appliactionContext.getBean(“customer1”);

System.out.println(customer1);

Customer customer2=(Customer) appliactionContext.getBean(“customer2”);

System.out.println(customer2);

((ClassPathXmlApplicationContext)appliactionContext).close();

}

}

}

**Configuration Using Annotation**

-

**Student.java**

@Data

Public class Student{

@value(“1”)

Private long id;

@Value(“Ramesh”)

Private String name;

@Value(“[ramesh@gmail.com](mailto:ramesh@gmail.com)”)

Private String email;

@Value(“24”)

Private int age;

}

**StudentConfig.java**

public class StudentConfig{

@Bean(name=”student”)

Public Student getStudent(){

Return new Student();

}

}

**StudentMain.java**

Public class StudentMain{

Public static void main(String args[])

{

ApplicationContext applicationContext = new AnnoatationConfigApplicationContext(StudentConfig.class);

Student student = (Student) applicationContext.getBean(“student”);

System.out.println(student);

((AnnotationConfigApplicationContext)appliactionContext).close();

}

}

Note:

**@Bean**

-This annotation is used to achieve IOC features of Spring Framework.

-This annotation is a method level annotation which defines that a method return an object.

-Spring Framework will refer this annotation an invokes the method for Bean creation.

**@Value**

-This annotation is used to initialize properties of an object.

-This is a property level annotation

**User.java**

@Component

@Data

Public class User{

@value(“1”)

Private long id;

@Value(“Ramesh”)

Private String userName;

@Value(“[ramesh@gamil.com](mailto:ramesh@gamil.com)”)

Private String email;

@Value(“Ramesh@123”)

Private String password;

}

**Usermain.java**

Public class UserMain{

Public static void main(String args[])

{

ApplicationContext applicationContext = new AnnoatationConfigAppliactionContext(“com.jspiders.springcoreannoatation.bean”);

User user = applicationContext.getBean(User.class);

System.out.println(user);

((AnnotationConfigApplicationContext) applicationContext).close();

}

}

Note: @Value annotation can be used as method level annotation if we make use of setter methods to initialize the properties of an object.

-The Same annotation can be used with the Formal arguments of All Arguments Constructor of a class as follows:

**Mobile.java**

@Component

Public class Mobile{

Private long id;

Private String name;

Private double price;

@Value(“1”)

Public void setId(long id){

this.id=id;

}

@Value(“Nord 3”)

Public void setName(String name){

this.name=name;

}

@Value(“34000”)

Public void setPrice(double price){

this.price=price;

}

@Override

Public String toString(){

return “Mobile [id=”+id+”, name” + name + ” , price ” + price + ” ] ”;

}

}

**MobileMain.java**

Public class MobileMain{

Public static void main(String args[])

{

ApplicationContext applicationContext= new AnnotationConfigApplicationContext(“com.jspiders.springcoreannotation.bean”);

Mobile mobile = (Mobile) applicationContext.getBean(“mobile”);

System.out.println(mobile);

((AnnotationConfigApplicationContext)applicationContext).close();

}

}

**Car.java**

@Component

Public class Car{

Private long id;

Private String name;

Private double price;

Car(@Value(“1” long id, @Value(“Baleno”) String name, @Value(“ 800000“ double price){

super();

this.id=id;

this.name=name;

this.price=price;

}

}

@Override

@Override

Public String toString(){

return “Mobile [id=”+id+”, name” + name + ” , price ” + price + ” ] ”;

}

**CarMain.java**

Public class CarMain{

Public static void main(String args[])

{

ApplicationContext applicationContext = AnnotationConfigApplicationContext(“com.jspiders.springcore.annotation.bean”);

Car car = (Car) applicationContext.getBean(“car’);

System.out.println(car);

((AnnotationConfigApplicationContext)applicationContext).close();

}

}

**AadharCard.java**

@Component

@Data

Public class AadharCard{

@Value(“1”)

Private long id;

@Value(“931300905827”)

Private long aadharNumber;

}

**Person.java**

@Component(value = “PersonBean”)

@Data

Public class Person{

@value(“1”)

Private long id;

@Value(“Ramesh”)

Private String name;

@Value(“9373428581”)

Private long contact;

@Autowired

Private AadharCard aadharCard;

}

}

**PersonMain.java**

Public class PersonMain{

Public static void main(String args[])

{

ApplicationContext applicationContext = new AnnotationConfigApplicationContext(“com.jspiders.springcoreannotation”);

Person person = (Person) applicationContext.getBean(“personBean”);

System.out.println(person);

((AnnotationConfigApplicationContext)applicationContext).close();

}

}

**@Component**

-This annotation is used to achieve IOC. If a class is annotated with @Component annotation then spring will create an object for the class.

**@AutoWired**

-This annotation is used to achieve the dependency injection.

-If any property of a class(Object Reference Variable) is annotated with @autoWired annotation then spring will initialize that property(Object reference Variable Automatically).

**@Qualifier**

-This annotation is used to overcome the ambiguity problem during the dependency injection.

-To overcome the same problem we can make use of one more annotation **@Primary**.

-If a class contains an reference variable of an Interface as a property which has more than one implementing classes in this case there is ambiguity for spring by initializing the reference variable of an interface.

-Since that interface has more than one implementing classes.

**Vehicle.java**

@Component

Public class Vehicle{

Private long id;

Private String name;

Private Wheel wheel;

@Value(“1”)

Public void setId(long id){

this.id=id;

}

@Value(“Aura”)

Public void setName(String name){

this.name=name;

}

@Autowired

Public void setWheel (Wheel wheel){

this.wheel=wheel;

}

}

**VehicleMain**

package com.jspiders.springcoreannotation.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.jspiders.springcoreannotation.bean.Vehicle;

public class VehicleMain {

public static void main(String[] args) {

ApplicationContext applicationContext = new AnnotationConfigApplicationContext(

"com.jspiders.springcoreannotation");

Vehicle vehicle = applicationContext.getBean(Vehicle.class);

System.out.println(vehicle);

((AnnotationConfigApplicationContext) applicationContext).close();

}

}

**Forest**

package com.jspiders.springcoreannotation.bean;

import org.springframework.beans.factory.annotation.Autowired;

//import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component

@Data

public class Forest {

@Autowired

// @Qualifier("cat")

Animal animal;

}

**ForestMain**

package com.jspiders.springcoreannotation.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.jspiders.springcoreannotation.bean.Animal;

import com.jspiders.springcoreannotation.bean.Forest;

public class ForestMain {

public static void main(String[] args) {

ApplicationContext applicationContext = new AnnotationConfigApplicationContext(

"com.jspiders.springcoreannotation");

Forest forest = applicationContext.getBean(Forest.class);

Animal animal = forest.getAnimal();

animal.sound();

((AnnotationConfigApplicationContext) applicationContext).close();

}

}

public interface Animal {

void sound();

}

package com.jspiders.springcoreannotation.bean;

import org.springframework.stereotype.Component;

**Animal.java**

@Component

public class Cat implements Animal {

@Override

public void sound() {

System.out.println("Meow Meow..");

}

}

**Animal.java**

package com.jspiders.springcoreannotation.bean;

import org.springframework.context.annotation.Primary;

import org.springframework.stereotype.Component;

@Component

@Primary

public class Dog implements Animal {

@Override

public void sound() {

System.out.println("Bow Bow..");

}

}

**Car.java**

package com.jspiders.springcoreannotation.bean;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Component

public class Car {

private long id;

private String name;

private double price;

public Car(@Value("1") long id, @Value("Baleno") String name,

@Value("800000") double price) {

super();

this.id = id;

this.name = name;

this.price = price;

}

@Override

public String toString() {

return "Car [id=" + id + ", name=" + name + ", price=" + price + "]";

}

}

**Wheel.java**

package com.jspiders.springcoreannotation.bean;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Component

public class Wheel {

private long id;

private int size;

@Value("1")

public void setId(long id) {

this.id = id;

}

@Value("18")

public void setSize(int size) {

this.size = size;

}

@Override

public String toString() {

return "Wheel [id=" + id + ", size=" + size + "]";

}

}

**CarMain.java**

package com.jspiders.springcoreannotation.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.jspiders.springcoreannotation.bean.Car;

public class CarMain {

public static void main(String[] args) {

ApplicationContext applicationContext =

new AnnotationConfigApplicationContext(

"com.jspiders.springcoreannotation.bean");

Car car = (Car) applicationContext.getBean("car");

System.out.println(car);

((AnnotationConfigApplicationContext) applicationContext).close();

}

}

**Voter.java**

package com.jspiders.springcoreannotation.bean;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Component

public class Voter {

private long id;

private String name;

private VotingCard votingCard;

public Voter(@Value("1") long id, @Value("Ramesh") String name, @Autowired VotingCard votingCard) {

super();

this.id = id;

this.name = name;

this.votingCard = votingCard;

}

@Override

public String toString() {

return "Voter [id=" + id + ", name=" + name + ", votingCard=" + votingCard + "]";

}

}

**VotingCard.java**

package com.jspiders.springcoreannotation.bean;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Component

public class VotingCard {

private long id;

private String number;

public VotingCard(@Value("1") long id, @Value("ABC123") String number) {

super();

this.id = id;

this.number = number;

}

@Override

public String toString() {

return "VotingCard [id=" + id + ", number=" + number + "]";

}

}

**VoterMain.java**

package com.jspiders.springcoreannotation.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.jspiders.springcoreannotation.bean.Voter;

public class VoterMain {

public static void main(String[] args) {

ApplicationContext applicationContext = new AnnotationConfigApplicationContext(

"com.jspiders.springcoreannotation");

Voter voter = applicationContext.getBean(Voter.class);

System.out.println(voter);

((AnnotationConfigApplicationContext) applicationContext).close();

}

}

**Mobile.java**

package com.jspiders.springcoreannotation.bean;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Component

public class Mobile {

private long id;

private String name;

private double price;

@Value("1")

public void setId(long id) {

this.id = id;

}

@Value("Nord 3")

public void setName(String name) {

this.name = name;

}

@Value("34000")

public void setPrice(double price) {

this.price = price;

}

@Override

public String toString() {

return "Mobile [id=" + id + ", name=" + name + ", price=" + price + "]";

}

}

**MobileMain.java**

package com.jspiders.springcoreannotation.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.jspiders.springcoreannotation.bean.Mobile;

public class MobileMain {

public static void main(String[] args) {

ApplicationContext applicationContext =

new AnnotationConfigApplicationContext(

"com.jspiders.springcoreannotation.bean");

Mobile mobile = (Mobile) applicationContext.getBean("mobile");

System.out.println(mobile);

((AnnotationConfigApplicationContext) applicationContext).close();

}

}

**Product.java**

@Component

@Data

Public class Product{

@Value(“1”)

Private long id;

@Value(“laptop”)

Private String name;

@Value(“45000”)

Private double price;

}

**ProductConfig.java**

**@ComponentScan(basePackage=”com.jspiders.springcoreannotation”**

Public class ProductConfig{

}

**ProductMain.java**

Public class ProductMain{

Public static void main(String args[])

{

ApplicationContext applicationContext = new AnnotationConfigApplicationontext(productConfig.class);

Product product = applicationContext.getBean(Product.class);

System.out.println(product);

((AnnotationConfigApplicationContext) applicationContext).close();

System.out.println(product);

}

}

**SPRING MVC**

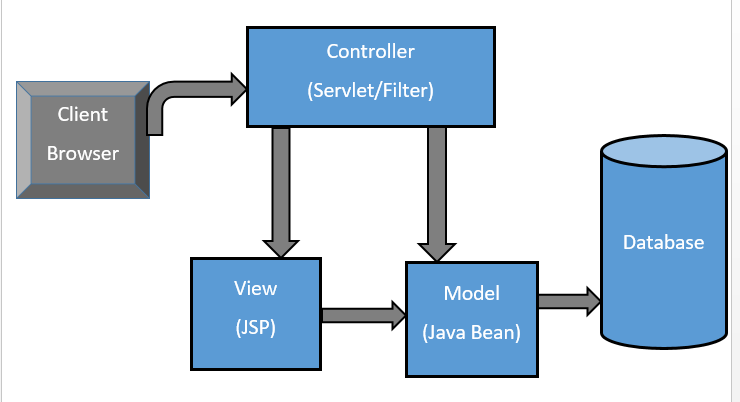
-MVC stands for Model View Controller

-This is one architectural design pattern which can be used to develop the Monolithic applications.

**Monolithic Applications**

-If an application contains Presentation layer, Business Logic Layer and Database Layer together then it is called as Monolithic Applications.

**MVC ARCHITECTURE**



**Controller**

-Controller is responsible to control the Model and View of an application.

-It is also responsible to accept the Request from the clien and give back response.

**View**

-This part of an appliaction refers to user interface which can be buid using **.html** or **.jsp** files.

**Model**

-This part of an applicatin contains actual business logic of an application and the data process this part of an application is called as Model Data.

**Bowers**

**View**

**Response**

**View**

**Resolver**

**Web Request**

**Controller**

**Dispatchers Servlet**

request

**Handling**

**Mapping**

response

**Model Data**

**Data**

**Execute Business Logic**

**MODEL**

**DB**

**Repository**

**POJO**

**Service**

**Steps to Configure Handler Mapping and View Resolver**

1.Inside Web-Inf folder create the XML file with the name “Dispatcher-Servlet”.

2.Make use of format given for Bean Creation Configuration.

3.To configure Handler Mappings make us of a tag called as

**<context:component-scan></context:component-scan>**

Which takes one attribute called as **base-package** and their provide the base-package name

4.To configure view resolver make use of Bean tag <bean> which take one attribute called as class to provide the value for class attribute press **ctrl+shift+t**, search for internal resource view resolver then select “**InternalResourceViewResolver**” and click open the implementation of the class will get open.

5.Now select the class name right click on it and copy the Qualified name and paste it inside the class attribute.Now make use of two property tags to provide the value for properties called as **Prefix** and **suffix**.

6.Make use of value attribute to provide the value as **/WEB-INF/VIEWS/**  and **.jsp** for properties prefix and suffix respectively.