

Introduction to Advanced Java

Table of Contents

<u>Core Java v/s Advance Java</u>	4
<u>Java Editions</u>	4
<u>Java Enterprise Edition</u>	5
<u>Java Full Stack Developer</u>	5
<u>What is Full Stack?</u>	5
<u>What are these full stack technologies?</u>	5
1. <u>Front End Technologies</u>	6
2. <u>Back End Development Technologies</u>	6
3. <u>Build Tools</u>	6
4. <u>Database Technologies</u>	6
5. <u>Integrated Development Environment (IDE)</u>	6
6. <u>Server</u>	6
7. <u>E.T.C</u>	7
<u>J2EE Versions</u>	7
<u>J2EE to Jakarta EE</u>	7
<u>Don't get confused</u>	7
<u>What actually is JEE?</u>	7
<u>What are Specifications?</u>	7
<u>So, is JEE a specification or a JavaEE Development Kit?</u>	7
<u>What is enterprise application?</u>	8
<u>What does these specifications include?</u>	8
<u>How to build something then?</u>	8

Core Java v/s Advance Java

- ▶ We all continuously talk about **Core Java** and **Advance java**.
- ▶ However, these bifurcations are **not done** by Oracle.
- ▶ These bifurcations are done in order to make **learning easier**.
- ▶ The official versions of Java as per **Oracle** are known as **Editions**.

Java Editions

There are 5 editions of Java:

1. Java Card - Smart Card Edition

Java Card refers to a software technology that allows Java-based applications (applets) to be run securely on **smart cards** and similar **small memory footprint devices**. Java Card is the **tiniest** of Java platforms targeted for **embedded devices**.

2. Java ME - Java Micro Edition

Java Platform, Micro Edition or Java ME is a computing platform for development and deployment of portable code for **embedded and mobile devices** (**micro-controllers, sensors, gateways**, mobile phones, personal digital assistants, **TV set-top boxes, printers**)

3. Java SE - Java Standard Edition

Java Platform, Standard Edition (Java SE) is a computing platform for development and deployment of portable code for desktop and server environments.

Core Java is a sub set of Java SE

4. Java MP - Java Micro Profile

Java MicroProfile is a baseline platform definition that optimizes Enterprise Java for a **microservices architecture** and delivers application portability across multiple MicroProfile runtimes.

5. Java EE - Java Enterprise Edition

Java Platform, Enterprise Edition (**Java EE**) is a set of specifications, extending Java SE with specifications for enterprise features such as **distributed computing** and **web services**

Advance Java is a sub set of Java EE

Java Enterprise Edition

- ▶ It is an **enterprise platform** which is mainly used to develop **web and enterprise applications**.
- ▶ It is built on the top of the Java SE platform.
- ▶ It includes topics like
 - 1. Servlets
 - 2. Java Server Pages
 - 3. Web Services
 - 4. Java Persistence APIs
 - 5. Hibernate Framework
 - 6. Spring Framework
 - Spring Core
 - Spring MVC
 - Spring JDBC
 - Spring AOP
 - Spring Security
 - Spring Boot

Java Full Stack Developer

A **java full stack web developer** is a **developer** with extensive knowledge and expertise in **full stack** tools and frameworks that work with **Java**.

What is Full Stack?

Full stack web developers are those who work with full stack technologies.

What are these full stack technologies?

Every technology that is used for web development is a part of a stack. But more importantly, three layers are considered as a stack. They are:

1. The **presentation** layer - the front-end development,
2. The **logic** layer - the back-end development, and
3. The **database** layer - working with databases.

The full stack developers can work with all of these layers, and usually tend to specialize in one or more.

In case of Java, the knowledge of following technologies makes a **Full Stack Java developer**:

1. Front End Technologies

- 1.1. HTML
- 1.2. CSS
- 1.3. JavaScript

2. Back End Development Technologies

- 2.1. Java Platform Standard Edition – OOP, JDBC
- 2.2. Java Platform Enterprise Edition – Servlets, JSP, Hibernate, Spring**

3. Build Tools

- 3.1. ANT
- 3.2. Maven**
- 3.3. Gradle

4. Database Technologies

- 4.1. SQL**
- 4.2. MySQL DB**
- 4.3. Oracle DB**
- 4.4. No SQL – Mongo DB
- 4.5. PostgreSQL

5. Integrated Development Environment (IDE)

- 5.1. Eclipse**
- 5.2. Netbeans
- 5.3. IntelliJ Idea
- 5.4. Visual Studio Code

6. Server

- 6.1. Tomcat**
- 6.2. Glassfish
- 6.3. JBoss
- 6.4. Wildfly etc

7. E.T.C

Java has been in business for quite some time as such it has lots of frameworks & libraries for specific uses, and almost every language there is, can be integrated with Java.

J2EE Versions

- ▶ J2EE 1.2 (December 12, 1999)
- ▶ J2EE 1.3 (September 24, 2001)
- ▶ J2EE 1.4 (November 11, 2003)
- ▶ Java EE 5 (May 11, 2006)
- ▶ Java EE 6 (December 10, 2009)
- ▶ Java EE 7 (May 28, 2013) but April 5, 2013 according to spec document
- ▶ Java EE 8 (August 31, 2017)
- ▶ Jakarta EE 8 (September 10, 2019) - fully compatible with Java EE 8
- ▶ Jakarta EE 9 (November 22 2020) - javax.* to jakarta.* namespace change.

J2EE to Jakarta EE

- ▶ Java EE was maintained by Oracle under the Java Community Process.
- ▶ On September 12, 2017, Oracle announced that it would submit Java EE to the Eclipse Foundation.
- ▶ The Eclipse named this top-level project as Eclipse Enterprise for Java (EE4J).
- ▶ As Oracle owns “Java” trademark, the Eclipse Foundation was forced to change the name.
- ▶ The Eclipse foundation renamed the Java EE platform to Jakarta EE.

Don't get confused

Java 2EE, J2EE, JEE, and Jakarta EE **are all different names for the same thing: a set of enterprise specifications that extend Java SE.**

What actually is JEE?

JEE consists of specifications only

What are Specifications?

Specifications mean rules or contract.

So, is JEE a specification or a JavaEE Development Kit?

JEE is a set of rules and contract laid down by Oracle/Sun of services required for any enterprise application

What is enterprise application?

An enterprise application (EA) is a large software system platform designed to operate in a corporate environment spanned across **multiple systems** and **huge geographical area**.

It includes **online shopping** and **payment processing**, **interactive product catalogs**, **computerized billing systems**, security, content management, **IT service management**, **business intelligence**, human resource management, manufacturing, process automation, enterprise resource planning etc etc etc...

What does these specifications include?

These specifications include ---

Servlet API, JSP(Java server page) API, Security, Connection pooling , EJB (Enterprise Java Bean), JNDI (Naming service -- Java naming & directory i/f), JPA (java persistence API), JMS (java messaging service), Java Mail, Java Server Faces , Java Transaction API, Webservices support (SOAP/REST) etc...

How to build something then?

JEE specifications are implemented by JEE Server vendors. These servers are known as JEE compliant Web Servers. Here is a list of popular Web Servers available

1. Apache Tomcat
2. Apache Tomee
3. Oracle / Sun --- reference implementation --- Glassfish
4. Red Hat -- JBoss (wild fly)
5. Oracle / BEA – weblogic
6. IBM -- Websphere

Client Server Overview

Table of Contents

<u>Introduction</u>	10
<u>Website – What is?</u>	10
<u>Types of Websites</u>	10
<u>Static Website</u>	10
<u>Dynamic Website / Web Application</u>	10
<u>What is the Web?</u>	10
<u>What is a Client?</u>	11
<u>Client Machine</u>	11
<u>Client Softwares</u>	11
<u>What is a Server?</u>	12
<u>Server Machine</u>	12
<u>Server Software</u>	12
<u>Hosting</u>	13
<u>HTTP Protocol</u>	13
<u>HTTP Methods</u>	14
<u>How do all these things work together?</u>	14

Introduction

In this modern digital age, **billions of bytes of data** get generated on a day-to-day basis. The data such as videos, audios, text, images are uploaded every single second onto the millions of servers' worldwide.

To store and access this huge amount of data, every person sends **request** on the **network** and awaits a **response**.

In order to understand the mechanism of request and response, let us first get an understanding of the **fundamentals of the web**.

Website – What is?

A Website is a **collection of web pages**. A web page is **designed** by front end web technologies and developed using **back-end web technologies**

Types of Websites

There are 2 types of websites:

Static Website

- ▶ A static website is a website which contains Web pages with fixed content.
- ▶ Each page is coded in HTML and displays the same contents to every visitor at any given time.
- ▶ Static sites are the most basic types of websites and are the easiest to create.
- ▶ Static websites **do not** require any web programming or database designs.

Dynamic Website / Web Application

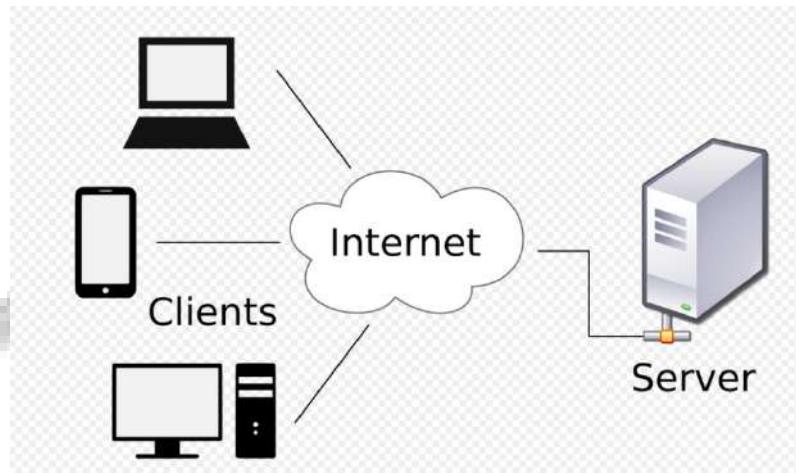
- ▶ A dynamic website contains information that changes, depending upon the viewer interaction.
- ▶ It contains client-side and server-side scripting to generate dynamic content.
- ▶ A Web application requests services over the web to generate dynamic content
- ▶ These are created using server side technologies such as Java, PHP, Python, and ASP.Net etc.

Example: Any website that presents dynamic content to the user such as Instagram, Yahoo, Google etc.

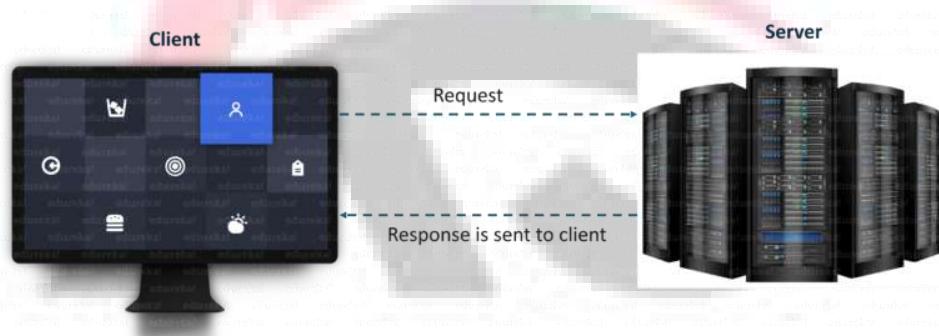
What is the Web?

Simply put, a **web or internet** is an interconnected system of computers. These computers can be categorized as:

- ▶ Clients, and
- ▶ Servers

**Figure 1. Client-Server Architecture**

The mechanism is as simple as: **A client requests, a server fulfills the requests.**

**Figure 2. Request Response Flow**

What is a Client?

A client can be anything that is used to request data from a server. There are 2 types of clients:

Client Machine

A *client machine* is a piece of *computer hardware* that accesses a service made available by a server. Some examples of client machine are:

1. Desktop/Laptop Computers
2. Mobiles
3. Smart TVs
4. Smart Watches
5. Etc.

Client Softwares

Client software is a piece of computer software that makes the request to the server and parses the response for the user. Some examples of client softwares are:

1. **Web browsers (Firefox, Chrome, Edge)** are clients that connect to web servers and retrieve web pages for display.

2. **Email clients (Thunderbird, Chipmunk)** retrieve email from mail servers.
3. **FTP client:** *FTP* stands for File Transfer Protocol. Using an *FTP client* we can upload, download, and manage files on our server.

What is a Server?

A Server can be anything that responds to the requests sent over the network by the client.

Server Machine

A *server machine* is a piece of *computer hardware* that stores in data and provides service to the client requests over the network 24X7. There are following types of server machines:

1. File Servers
2. Web Servers
3. Mail Servers
4. Virtual Servers
5. Application Servers
6. Domain Name Servers



Single Dedicated Server



Server Room (Multiple Servers)

Figure 3. Physical Servers

Server Software

All server machines must have server softwares installed on them to **handle requests and send prepared response** to the client. Some examples of server softwares are:

1. **Apache Tomcat**
2. XAMPP
3. Microsoft IIS
4. Lighttpd
5. Etc.

So, it could be said that: In a data center, the physical computer that a server program runs on is also frequently referred to as a server.

Hosting

When a **server** machine **stores** a file or **group of files** on it for **any specific time**, it is known as Hosting.

If we want our web application to be called from any client machine or any client software, we must **host the web application on a web server**.

There can be different types of hosting:

1. *Private Hosting*

The big companies (such as Facebook, Google) have their **own servers** to host their web applications.

2. *Rented Hosting*

However, small companies can take a **server on rent** from **server space provider companies**.

There are many private companies that offer different plans to host websites. Some such companies are:

- ▶ Godaddy
- ▶ Amazon
- ▶ Bigrock
- ▶ Hostgator, etc.

3. *Local Hosting*

A server installed on the local machine for development purposes is called local hosting.

When a software program is developed, the programmers need to test it frequently for correctness. In case the programmer is working directly on a remote server, it could be time and bandwidth consuming because every change will require transaction with the online server, and the server could be anywhere in the world. So, the solution is to use local-host (server on personal machine) for development. When finally the software is completely ready it can be transferred to the main server at once.

Some local servers are:

1. **Apache Tomcat**
2. **XAMPP**
3. **IIS** etc.

HTTP Protocol

- ▶ HTTP stands for Hyper Text Transfer Protocol
- ▶ It is a network **protocol (set of rules)** that clients and servers on the web use to communicate.
- ▶ It is similar to other internet protocols such as **SMTP** (Simple Mail Transfer Protocol), **FTP** (File Transfer Protocol) and **TCP/IP** (Transmission Control Protocol/Internet Protocol).

- HTTP is a **stateless protocol** i.e. it supports only **one request per connection**. This means that the protocol treats every request as a new request; it doesn't remember the client over multiple requests.
 - The above mechanism allows more users to connect to a given server over a period of time.
 - The client sends an HTTP request and the server responds with an HTTP response.
- For ex:** Web browser and Web Servers communicate using HTTP

HTTP Methods

The primary and the most commonly used HTTP methods are:

1. POST
2. GET
3. PUT
4. PATCH
5. DELETE etc.

The HTTP request can be made using any of the above of methods, but the ones which are most used are: **Get and Post**.

How do all these things work together?

1. Some user is browsing the internet on a client machine using a client software, such as Chrome.
2. The client software and client machine together are called **Client**.
3. The **Client** wants to access a **web application**
4. This **website** (web application) is **hosted** on a **remote server machine**.
5. The request for the website & response is managed by a **server software** installed on the server machine.
6. Both, server software and server machine are collectively called as **Server**
7. The website is identified by an **address** also known as a **domain name**.
8. The website can be accessed by its **domain name**.
9. The **Client** makes a request to the **Server** to access the website
10. This request is sent through the **network** using the **HTTP protocol**.
11. The **Server** gets the request, **prepares the data** and sends it back to the **Client**
12. The server also uses the **HTTP protocol** to send the response
13. The client software **parses the response** and displays the output to the end user
14. The **Server** again comes into a waiting state to handle more requests

First Advanced Java Project

Table of Contents

First Java EE Project	16
Setting up the Java Development Environment	16
Download and Install latest JDK and JRE	16
Environment Variables	16
Download and Install Eclipse IDE for Java EE Developers	16
Set Up the Workspace	17
Download and Install Tomcat	17
Configure Tomcat Server with Eclipse	17
Start Tomcat Server	17
If Block Port Error Message	18
Download and Install MySQL for Windows	18
Download MySQL Connector	18
First Web Application	18
Create a Project	18
Create an HTML File	19
Create a Servlet	19
Modify web.xml	19
Execute the Project	19
Stop the Server	19

First Java EE Project

Setting up the Java Development Environment

Download and Install latest JDK and JRE (LTS version only)

- ▶ Check if your system currently has JDK and JRE installed or not:
Open command prompt & type **java -version**. If it says “java is **not recognized as an internal or external command**”, we need to install java.
- ▶ If you have any version(s) prior to JDK 11.0.12 uninstall it/them from Control Panel
- ▶ Google for “JDK latest LTS version download”. In the displayed results, open **Oracle** specific link
- ▶ Download the latest JDK (currently its **11.0.12**) and install it. JRE will be installed along with automatically.

You can download it directly from here: <https://www.oracle.com/in/java/technologies/javase-jdk11-downloads.html>

Environment Variables

- ▶ JDK will be installed in “C:\Program Files\” directory
- ▶ Copy the path inclusive of **bin** directory: “C:\Program Files\Java\jdk-11.0.12\bin”
- ▶ Right click on “My PC”, go to “Properties” and then go to “Advanced System Settings”
- ▶ Click on “**Environment variables**” under “Advanced” tab. You will notice **2 sections** in the dialog
- ▶ In both sections, search for “**PATH**” environment variables and choose to **edit** them
- ▶ Add the path: “C:\Program Files\Java\jdk-11.0.12\bin” in both PATH variables and click OK
- ▶ **Restart** the command prompt & check again for java version

Download and Install Eclipse IDE for Java EE Developers

- ▶ Search for “Eclipse IDE latest download”
- ▶ Open the link that says eclipse.org
- ▶ Download **Eclipse IDE 2020-12** or the one that suits your systems configuration (32/64 bit)
- ▶ Run the installer.
- ▶ Choose “Eclipse IDE for **Enterprise** Java Developers”
- ▶ Select the installation folder and proceed with the installation

You can download it directly from here:

<https://www.eclipse.org/downloads/packages/release/2020-12/r>

Set Up the Workspace

- ▶ As soon as Eclipse Installation completes, launch it.
- ▶ The launcher will ask us to set up a workspace path. Give the workspace path.
- ▶ It will create a folder where all our projects will be created and stored

Download and Install Tomcat

- ▶ Google for “tomcat download”
- ▶ Click the link that displays “tomcat.apache.org”
- ▶ Choose **Tomcat 9** under downloads section
- ▶ Download Windows service installer – 32/64 bit depending on your system configuration
- ▶ You can download it directly from here: <https://tomcat.apache.org/download-90.cgi>
- ▶ Follow the usual install mechanism to install Tomcat.
- ▶ Give the path to JDK 11.0.12 when asked
- ▶ Keep the default installation folder as is.
- ▶ Uncheck “Start Tomcat” and click Finish.
- ▶ To install Tomcat on Ubuntu, follow this link: <https://tecatadmin.net/install-tomcat-9-on-ubuntu/>

Configure Tomcat Server with Eclipse

- ▶ Open Eclipse
- ▶ Go to: **Windows** ⇒ **Preferences**
- ▶ Open option: **Server** ⇒ Click on: **Runtime Environments**.
- ▶ Click on: Add ⇒ Choose: Apache Tomcat v10.0 and click next
- ▶ Browse Tomcat installation directory and click Finish
- ▶ Click “Apply and Close”. Tomcat is configured.

Start Tomcat Server

- ▶ Open Servers Tab below, or Click on Window ⇒ Show View ⇒ Other ⇒ type Servers and hit enter.
- ▶ Click the link under Servers Tab.
- ▶ Select the Server Type as Tomcat v9.0 Server, click next and then Finish.
- ▶ Tomcat will start showing in the Servers Tab with the status “Stopped, Synchronized”
- ▶ Select Tomcat and click the green play icon.
- ▶ Click “Allow Access” in “Windows Security Alert” Dialog Box if it pops up
- ▶ The status of Tomcat changes to “Started, Synchronized”
- ▶ Open a web browser and type “localhost:8080” in the address bar

- If the web page shows “**HTTP Status 404 – Not Found**”, the server has started successfully

If Block Port Error Message

- Go to Apache Tomcat installation folder =>**conf** folder
- Open **server.xml** and change port numbers to the free ones (for ex. 9494) available on the system.
- Save the file and restart the server.

Download and Install MySQL for Windows

- Search for “download mysql” on Google
- Open the link that says “<https://www.mysql.com/downloads/>”
- Under “Downloads” tab, search for “**MySQL Community (GPL) Downloads**”. Click it.
- Search for “MySQL Community Server”
- Search for “Windows (x86, 32 & 64 bit), MySQL Installer MSI” and go to download page.
- Under GA releases, search for “**Windows (x86, 32-bit), MSI Installer - (mysql-installer-community-8.0.23.0.msi)**” – Download the one with a higher size
- You can download it directly from this link: <https://dev.mysql.com/downloads/file/?id=506568>
- You will be asked to “Login” or “Sign Up”. Instead click on “**No thanks, just start my download.**”
- Go through normal installation.

Download MySQL Connector

- Google for “**MySQL Connector for Java**”
- Open the link that says: dev.mysql.com
- Download any (tar.gz or .zip file)
- You will be asked to “Login” or “Sign Up”. Instead click on “**No thanks, just start my download.**”
- Download the file and extract it

First Web Application

Create a Project

- Start Eclipse
- Create a “**Dynamic Web Project**” by clicking: File → New → Dynamic Web Project
- Give a name to the project, say **01MyFirstWebProject** here.
- Keep every entry as default and proceed with next → next options
- Keep the Context Root and Content Directory as is.
- A context root identifies a web application in a Java EE server.

- ▶ Check “Generate **web.xml** deployment descriptor” and click finish.

Create an HTML File

- ▶ Right click on **WebContent** directory, click New → HTML File, give name of the file as “**index.html**” in the dialog box, click next and then finish.
- ▶ Create a form with attributes **action** and **method** set to “**first**” and “**get**” respectively
- ▶ Create an input with attributes **type** and **name** set to “**text**” and “**user-name**” respectively
- ▶ Create a submit button

Create a Servlet

- ▶ Inside Java Resources, find the “src” directory.
- ▶ Right click on the “src” directory, click New → Servlet
- ▶ Give the name of the package (**com.first.project**) & the name of the Servlet (**FirstServlet**) & click next
- ▶ In the URL mappings section, select the URL and click **edit**. Set the new URL to “/first” & click next
- ▶ In the next dialog, **uncheck** “doPost”, keep rest of the options as they are and click finish
- ▶ In the created FirstServlet, locate the doGet() method and write the below code:

```
System.out.println("Name: " + request.getParameter("user-name"));
```

Modify web.xml

- ▶ Open **web.xml** and find the tag **<welcome-file-list></welcome-file-list>**
- ▶ Keep only the **<welcome-file>** tag that has **index.html** as its content
- ▶ **Remove** all the other **<welcome-file>** tags and save the web.xml file

Execute the Project

- ▶ Select the project and click the **green button** on the toolbar
- ▶ Select “Run on Server” in the pop-up

Result: The project will run in the Eclipse inbuilt browser with URL:

<http://localhost:8080/01MyFirstWebProject/>

- ▶ Enter “testvalue” in the text box and hit Submit

Result: The URL will change to: <http://localhost:8080/01MyFirstWebProject/first?user-name=testvalue>

Stop the Server

When you are done with executing the code, stop the server by clicking on Servers tab and clicking the **red square** button on its toolbar

Design Patterns

Table of Contents

<u>What is a Design Pattern?</u>	21
<u>Advantages of Design Patterns.</u>	21
<u>When should we use the design patterns?</u>	21
<u>MVC Design Pattern</u>	22
<u>Servlets</u>	22
<u>JSP</u>	22
<u>JDBC</u>	23
<u>DAO Design Pattern</u>	23
<u>Implementation</u>	23

What is a Design Pattern?

Design patterns are well-proved solution for solving the specific problem/task.

After multiple years of software engineering an understanding has been developed to adopt certain practices in order to achieve easily maintainable, retestable, reusable, low error prone code. It is like following a “Standard Operating Procedure” while building a software system.

Advantages of Design Patterns

1. They provide the solutions that help to define the system architecture.
2. They capture the software engineering experiences.
3. They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
- 4. Design patterns don't guarantee an absolute solution to a problem.**
5. They provide clarity to the system architecture and the possibility of building a better system

When should we use the design patterns?

We must use the design patterns during the analysis and requirement phase of SDLC (Software Development Life Cycle).

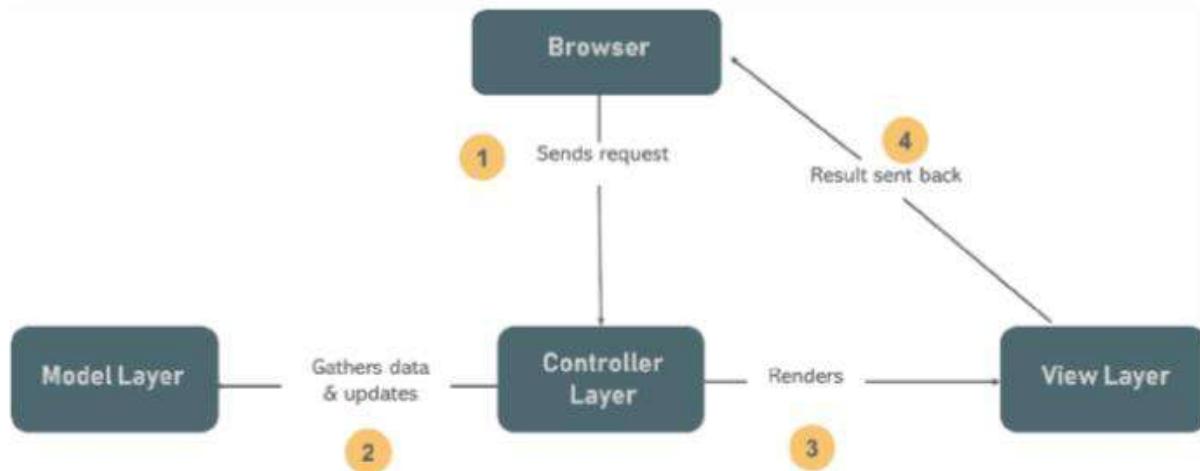
Examples: We are going to learn about:

1. MVC Design Pattern
2. DAO Design Pattern

MVC Design Pattern

MVC design Pattern is one of the most implemented software design patterns in J2EE projects. It consists of three layers:

1. The **Model Layer** - Represents the business layer of the application
2. The **View Layer** - Defines the presentation of the application
3. The **Controller Layer** - Manages the flow of the application



Now J2EE has provided us with **3 technologies** that help us implement MVC design pattern. These are:

1. Servlets
2. JSP
3. JDBC

Servlets

- ▶ Servlets are meant to process **business logic**.
- ▶ It receives the data, processes it and produces the output.
- ▶ Servlets acts as **Controller** component
- ▶ The latest version of Servlet is 5.0

JSP

- ▶ JSP stands for **Java Server Pages**
- ▶ JSP is meant for presentational logic.
- ▶ When we want to display something to the end user, we use JSP.

- ▶ JSP acts as a **View** Component.
- ▶ The latest version of JSP is 3.0

JDBC

- ▶ JDBC stands for **Java Database Connectivity**
- ▶ JDBC connects a Java application to the database.
- ▶ It helps the java application to communicate with the database and implement CRUD operations.
- ▶ It acts as a **Model** component
- ▶ The latest version of JDBC is 4.3

DAO Design Pattern

This pattern is used to separate low level data accessing API or operations from high level business services or we can say that DAO pattern creates a persistence layer for service layer to use. Following are the participants in Data Access Object Pattern.

1. **Data Access Object Interface** - This **interface** defines the standard operations to be performed on a model object(s).
2. **Data Access Object concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
3. **Model Object or Value Object** - This object is simple **POJO** containing get/set methods to store data retrieved using DAO class.

Implementation

We are going to create a Student object acting as a Model or Value **Pojo**.**StudentDao** is Data Access Object Interface.**StudentDaoImpl** is concrete class implementing Data Access Object Interface.

DaoPatternDemo, our demo class, will use **StudentDao** to demonstrate the use of Data Access Object pattern.

Java Database Connectivity

Table of Contents

<u>JDBC</u>	24
<u>What is JDBC?</u>	25
<u>What is a Driver?</u>	25
<u>Types of JDBC Drivers</u>	25
<u>Type-1 Driver</u>	26
<u>Type-2 Driver</u>	26
<u>Type-3 Driver</u>	27
<u>Type-4 Driver</u>	27
<u>Questions and Answers</u>	28
<u>Question 2: What factors decide the choice of drivers?</u>	28
<u>Answer: There are 2 factors that decide the choice of drivers: portability and performance.</u>	28
<u>Question 3: Which driver is least used?</u>	28
<u>JDBC Architecture</u>	29
<u>Operations With DB</u>	30
<u>JDBC 4.3 API</u>	30
<u>java.sql package</u>	30

What is JDBC?

- ▶ JDBC stands for Java DataBase Connectivity.
- ▶ JDBC is a **Java API** used to connect and execute queries with the database.
- ▶ JDBC API consists of a set of **classes, interfaces** and **methods** to work with databases
- ▶ JDBC can be used to interact with every type of RDBMS such as MySQL, Oracle, Apache Derby, MongoDB, PostgreSQL, Microsoft SQL Server etc.
- ▶ It is a **part of JavaSE** (Java Platform, Standard Edition).
- ▶ JDBC API uses **JDBC drivers** to connect with the database.
- ▶ The current version of JDBC is **4.3**.

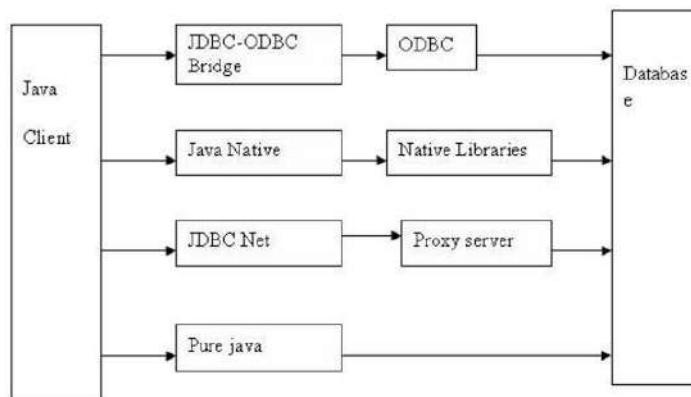
What is a Driver?

A driver is nothing but a piece of software required to connect to a database from Java program. JDBC drivers are client side adapters (**installed on the client machine**, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

Types of JDBC Drivers

There are four types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver



Pictorial representation of JDBC Drivers

Type-1 Driver

- ▶ This is the oldest JDBC driver, mostly used to connect database like MS Access from Microsoft Windows operating system.
- ▶ Type 1 JDBC driver actually translates JDBC calls into ODBC (Object Database connectivity) calls, which in turn connects to database.
- ▶ It converts the JDBC method calls into ODBC function calls.

Pros:

- ▶ Any database that provides an ODBC driver can be accessed

Cons:

- ▶ Features are limited and restricted to what ODBC driver is capable of
- ▶ Platform dependent as it uses ODBC which in turn uses **native O/S libraries**
- ▶ ODBC driver must be installed on client machine
- ▶ **Limited portability** as **ODBC driver is platform dependent** & may not be available for all platforms
- ▶ **Poor Performance** because of several layers of translation that take place before the program connects to database
- ▶ It is now **obsolete** and only used for development and testing.
- ▶ It has been removed from JDK 8 (1.8)

Type-2 Driver

- ▶ This was the second JDBC driver introduced by Java after Type 1, is hence known as type 2.
- ▶ In this driver, performance was improved by **reducing communication layer**.
- ▶ Instead of talking to ODBC driver, **JDBC driver directly talks to DB client using native API**.
- ▶ That's why it's also known as **native API or partly Java driver**
- ▶ Type 2 drivers use the client side libraries of the database.
- ▶ The driver converts JDBC method calls into native database API calls.

Pros:

- ▶ Faster than JDBC-ODBC bridge as there is no conversion like ODBC involved
- ▶ Since it required native API to connect to DB client it is also less portable and platform dependent.

Cons:

- ▶ Client side libraries needs to be installed on client machine
- ▶ Driver is platform dependent
- ▶ Not all database vendors provide client side libraries

- Performance of type 2 driver is slightly better than type 1 JDBC driver.

Type-3 Driver

- This was the third JDBC driver introduced by Java, hence known as type 3.
- Type 3 driver makes use of middle tier between the Java programs and the database.
- Middle tier is an **application server** that converts JDBC calls into vendor-specific database calls.
- It was very different than type 1 and type 2 JDBC driver in sense that **it was completely written in Java** as opposed to previous two drivers which were not written in Java.
- That's why this is also known as **all Java driver**.
- This driver uses **3 tier approach i.e. client (java program), server and database**.
- So you have a Java client talking to a Java server and Java Server talking to database.
- Java client and server talk to each other using net protocol hence this type of JDBC driver is also known as **Net protocol JDBC driver**.
- This driver **never gained popularity** because database vendor was reluctant to rewrite their existing native library which was mainly in C and C++

Pros:

- No need to install any client side libraries on client machine
- Middleware application server can provide additional functionalities
- Database independence

Cons:

- Requires middleware specific configurations and coding
- May add extra latency as it goes through middleware server

Type-4 Driver

- Type 4 drivers are also called **Pure Java Driver**.
- This is the driver you are most likely using to connect to modern database like Oracle, SQL Server, MySQL, SQLite and PostgreSQL.
- This driver is implemented in Java and directly speaks to database using its native protocol.
- It converts JDBC calls directly into vendor-specific database protocol.
- This driver includes all database calls in **one JAR file**, which makes it very easy to use.
- All you need to do to connect a database from Java program is to include JAR file of relevant JDBC driver.
- Because of **light weight**, this is also known as **thin JDBC driver**.

- Since this driver is also written in **pure Java**, it is portable across all platforms, which means you can use same JAR file to connect to MySQL even if your Java program is running on Windows, Linux or Solaris.
- Performance of this type of JDBC driver is also best among all of them because **database vendor liked this type** and all enhancements they make they also port for type 4 drivers.

Pros:

- Written completely in Java hence platform independent
- Provides better performance than Type 1 and 2 drivers as there is no protocol specific conversion is required
- Better than Type 3 drivers as it doesn't need additional middleware application servers
- Connects directly to database drivers without going through any other layer

Cons:

- Drivers are database specific

Questions and Answers

Question 1: Which driver should we use?

Answer:

- A Type 4 driver is preferred if Java application is accessing any 1 database such as Oracle, Sybase etc.
- In case multiple databases are accessed then a Type 3 driver would be preferable.
- Type 2 drivers are recommended, if Type 3 or 4 drivers are not available for the database.
- Type 1 drivers are not recommended for production deployment.

Question 2: What factors decide the choice of drivers?

Answer: There are 2 factors that decide the choice of drivers: **portability** and **performance**.

Question 3: Which driver is least used?

Answer: Type 1 JDBC driver is the poorest in terms of portability and performance. **It is no longer used.**

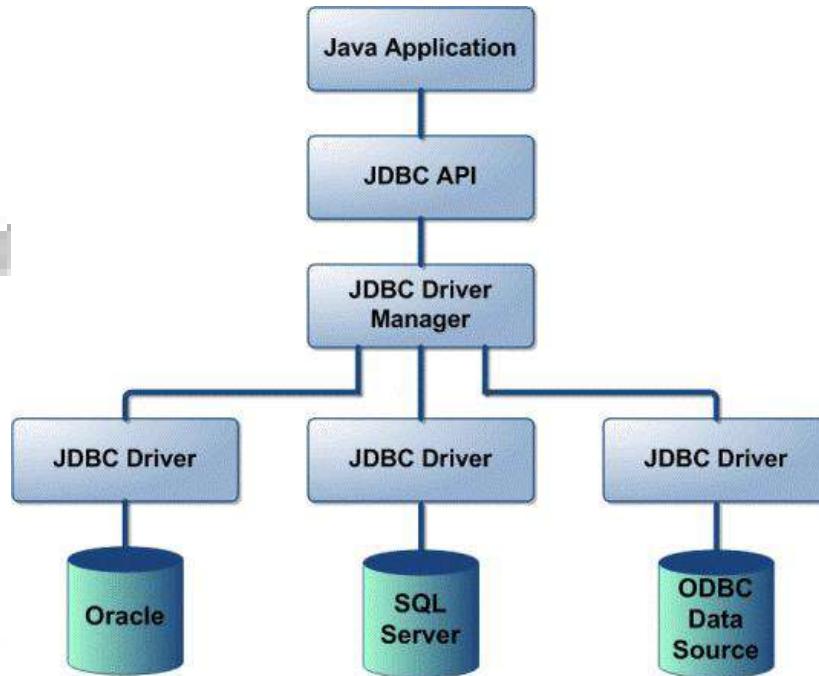
Question 4: Which is the best driver?

Answer: Type 4 JDBC driver is highly portable and gives the best performance.

Question 5: Which driver do we use to connect and transact with MySQL DB?

Answer: We use Type 4 driver

JDBC Architecture



1. **Java Application** is any application that likes to connect and transact with any database.
2. **JDBC API**
 - a. It provides the application-to-DB Connection
 - b. It provides the driver manager (`java.sql.DriverManager`)
 - c. It uses the database specific driver to connect to heterogeneous databases.
3. **Driver Manager**
 - a. The JDBC driver manager ensures that the **correct driver is used** to access each data source
 - b. The driver manager is capable of supporting **multiple concurrent drivers** connected to **multiple heterogeneous databases**.
 - c. One application can connect to different databases simultaneously.
4. **JDBC driver API** supports the JDBC Database connection.
 - a. **Database vendors** provide the **JDBC drivers**.
 - b. For example: MySQL vendor provides "`mysql-connector-java-8.0.19`" jar file that contains "`com.mysql.cj.jdbc.Driver`"
5. **Databases**
 - a. A **Java application** can connect and transact with **multiple databases** simultaneously or one at a time.
 - b. The vendors provide their specific drivers
 - c. The Driver Manager takes care of all the drivers

Operations With DB

The following are the key operations we do with a database frequently.

1. Connect to DataBase
2. Execute Queries
 - a. Create/Insert Data
 - b. Retrieve Data
 - c. Update Data
 - d. Delete Data
3. Close Connections/Resources

JDBC 4.3 API

JDBC 4.0 API is mainly divided into two package

- ▶ java.sql
- ▶ javax.sql

java.sql package

This package include classes and interface to perform almost all JDBC operation such as creating and executing SQL Queries.

Important classes and interface of java.sql package

classes/interface	Description
java.sql.BLOB	Provide support for BLOB(Binary Large Object) SQL type.
java.sql.Connection	Creates a connection with specific database
java.sql.CallableStatement	Execute stored procedures
java.sql.CLOB	Provide support for CLOB(Character Large Object) SQL type.
java.sql.Date	Provide support for Date SQL type.
java.sql.Driver	Create an instance of a driver with the DriverManager.
java.sql.DriverManager	This class manages database drivers.
java.sql.PreparedStatement	Used to create and execute parameterized query.

java.sql.ResultSet	It is an interface that provide methods to access the result row-by-row.
java.sql.Savepoint	Specify savepoint in transaction.
java.sql.SQLException	Encapsulate all JDBC related exception.
java.sql.Statement	This interface is used to execute SQL statements.

First JDBC Program

A succesfull operation/transaction with the database involves the exceution of **several small steps**. Each step has a **significance** of its own. To write efficient database code, all these steps must be correctly implemented. These steps are:

- ✓ **Step 1.** Pre-requisites
- ✓ **Step 2.** Connect to database
- ✓ **Step 3.** Create Statement
- ✓ **Step 4.** Prepare the Query
- ✓ **Step 5.** Execute the Query and Collect Data
- ✓ **Step 6.** Close Resources

Lets ponder over the above steps one by one

Step 1. Pre-requisites

Before writing a JDBC API, we need to:

1. Install any database server (here we will use **MySQL Server**).
2. Install a **GUI to operate on** the database server (here, MySQL GUI - **MySQL Workbench**)
3. While installation, set a **username** and a **password** (We set username: **root** and password: **1234**)
4. Create a schema (We start with **schema - school**)
5. Create the first table (We create **table - students**)
6. Create columns

S. No	Column Name	Column Properties
1.	_id	Int, Primary Key, Not Null, Unique, Auto Increment
2.	student_name	Varchar (45), Not Null
3.	student_class	Int (2), Not Null

4.	student_fees	Double, Not Null
----	--------------	------------------

7. Make sure that MySQL Server is running before writing the JDBC API.
8. Place the suitable connector jar file in **WEB-INF>>lib** folder. (We use **mysql-connector-java-8.0.23.jar**)

Step 2. Connect to Database

1. Import the correct packages.

```
import java.sql.*;
```

2. Load the JDBC Driver

```
Class.forName("com.mysql.cj.jdbc.Driver");
where com.mysql.cj.jdbc.Driver is the location of Driver Class
```

3. Establish the connection

```
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/<schema-name>",
"<username>, <password>");

where jdbc:mysql://localhost:3306/<schema-name> is Connection String
```

Step 3. Create a Statement

```
Statement statement = conn.createStatement();
```

Step 4. Prepare the query

```
String insertQuery = "INSERT INTO students (student_name, student_class,
student_fees) values('Abhishek Verma', 1, 5000.0);"
```

Step 5. Execute the query and collect data

```
int noOfRowsInserted = stmt.executeUpdate(insertQuery);
```

Step 6. Close the Resources

```
statement.close();
connection.close();
```

Connect to Database

The first step before we can do any database transactions is to **connect with the database**.

JDBC API to Connect with DB

4. Import the correct packages.

```
import java.sql.*;
```

5. Load the JDBC Driver

The first thing you need to do before you can open a JDBC connection to a database is to load the JDBC driver for the database. You load the JDBC driver like this:

```
Class.forName("com.mysql.cj.jdbc.Driver");  
where com.mysql.cj.jdbc.Driver is the location of Driver Class
```

- ▶ You only have to load the driver **once for the whole application**.
- ▶ You do not need to load it before every connection opened.
- ▶ Only before the first JDBC connection opened.

6. Establish the connection

```
Connection conn =  
DriverManager.getConnection("jdbc:mysql://localhost:3306/<schema-name>,  
<username>, <password>");  
where jdbc:mysql://localhost:3306/<schema-name> is Connection String
```

The following classes and interfaces are used to connect to the database:

S. No	Class/Interface Name	Function
1	Class (C)	Class is a class in which all the drivers should be registered which will be used by the Java Application
2	forName(String)	forName() is a static method in class Class which loads and register our driver dynamically (by calling DriverManager.registerDriver())
3	Connection (I)	The JDBC Connection class, java. sql. Connection , represents a database connection to a relational database. Before we can read or write data from and to a database via JDBC, we need to open a connection to the database.
4.	DriverManager (C)	The DriverManager class maintains a list of Driver classes that have

		registered themselves by calling the method Class.forName() that calls DriverManager.registerDriver() automatically
5.	getConnection (String, String, String)	This method of Java DriverManager class attempts to establish a connection to the database by using the given database url, username and password

Close the Resources

You should explicitly close *Statements*, *ResultSets*, and *Connections* when you no longer need them, unless you declare them in a *try-with-resources* statement (available in JDK 7 and after).

To close a Statement, ResultSet, or Connection object that is not declared in a **try-with-resources** statement, use its close method.

```
resultSet.close();
statement.close();
connection.close();
```

Statement

- ▶ Once a connection is obtained we can interact with the database.
- ▶ The JDBC framework provides **3 interfaces** and related methods and properties with which we can send SQL or PL/SQL commands to operate on the database.
- ▶ These interfaces are:
 1. **Statement interface**
 2. **PreparedStatement interface**

Let us look at them one by one

Statement (I)

1. Statement is an **Interface** in Java (JDBC)
2. It is present in java.sql package
3. We can obtain a JDBC Statement from a JDBC Connection
4. It is used to execute **static SQL statements** at runtime against an RDBMS.
5. It can be used to execute SQL **DDL** statements, for example data retrieval queries (**SELECT**)
6. It can be used to execute SQL **DML** statements, such as **INSERT, UPDATE and DELETE**

Syntax

```
Statement stmt = null;
try {
```

```

stmt = connection.createStatement( ) ; // conn is Connection object
. . .
}
catch (SQLException e) {
. . .
}
finally {
    if(stmt!=null)
        stmt.close();
}

```

CRUD Operations

Create Operation - INSERT

```

Statement statement = connection.createStatement();

String insertQuery = "INSERT INTO students (student_name, student_class,
student_fees) values('Kinjal', 1, 5000.0)";

int noOfRowsInserted = statement.executeUpdate(insertQuery);

```

Retrieve Operation - SELECT

```

Statement statement = connection.createStatement();

String selectQuery = "SELECT * FROM students";
ResultSet resultSet = statement.executeQuery(selectQuery);

while (resultSet.next()) {
    resultSet.getInt("_id"); // resultSet.getInt(1);
    resultSet.getString("student_name"); // resultSet.getString(2);
    resultSet.getInt("student_class"); // resultSet.getInt(3);
    resultSet.getDouble("student_fees"); // resultSet.getDouble(4);
}

```

Update - UPDATE

```

Statement statement = connection.createStatement();

String updateQuery = "UPDATE students SET student_class = 12 WHERE student_class
= 11";

int noOfRowsUpdated = statement.executeUpdate(updateQuery);

```

Delete - DELETE

```

Statement statement = connection.createStatement();

```

```
String deleteQuery = "DELETE FROM students WHERE student_class = 12";
int noOfRowsDeleted = statement.executeUpdate(deleteQuery);
```

Methods

`int executeUpdate(String sqlQuery)`

This method is used to execute a DML SQL Query (Insert, Update and Delete queries)

`ResultSet executeQuery(String sqlQuery)`

This method is used to retrieve data from the table and returns a result set

The following table provides a summary of each interface's purpose to decide on the interface to use.

Interfaces	Recommended Use
Statement	Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

It has the following steps:

Create Statement

Step 4. Execute SQL command

Execute the Query via the Statement

You do so by calling its `executeQuery()` method, passing an SQL statement as parameter.

Step 5. Collect Information

There are different types of operations that we do with the DB such as insert, update, delete and retrieve.

In every case we get different informations.

- ▶ In case of insert, update and delete operations, we get number of records inserted, updates or deleted.
- ▶ In case of retrieval, we get rows of data in a ResultSet

ResultSet

1. ResultSet is an Interface in Java (JDBC)

2. It is used to hold records that are returned as a result of a SELECT query
3. We need to travel/traverse/iterate through the ResultSet to get records

Create ResultSet

```
ResultSet resultSet = statement.executeQuery(sql);
```

Get the data from ResultSet

1. To iterate the ResultSet you use its **next()** method.
2. The next() method returns **true** if the ResultSet has a next record, and moves to the next record.
3. If there were no more records, next() returns **false**, and you can no longer extract data.

```
while(resultSet.next()) {  
    System.out.println("Id = " + resultSet.getInt(1));  
    System.out.println("Name = " + resultSet.getString(2));  
    System.out.println("Class = " + resultSet.getInt(3));  
}
```

Step 5. Close the Resources

```
resultSet.close();  
statement.close();  
connection.close();
```

Web Server Architecture

Table of Contents

<u>I2EE</u>	39
<u>Server Runtime Environment</u>	39
<u>Server</u>	39
<u>Web Server (HTTP Server)</u>	39
<u>Web Container</u>	40
<u>Tomcat Web Server</u>	40
<u>What is Tomcat?</u>	40
<u>Is Tomcat a Web Server or a Web Container?</u>	40
<u>Servlet Container</u>	40
<u>JSP Container</u>	40
<u>Schematic Diagram of Tomcat Web Server</u>	41
<u>Tomcat Architecture</u>	41
<u>Tomcat Port</u>	42

J2EE

J2EE is a set of rules and protocols given by Sun/Oracle also known as specifications

Server Runtime Environment

Server vendors provide J2EE specification's implementation. There are many such implementations in the industry such as:

1. Apache Tomcat
2. J BOSS
3. Web sphere
4. Glassfish
5. Wild Fly

Server

A piece of hardware where we put our website for everyone to access it 24 by 7 is known as a Server Machine and the operating system that manages the incoming requests, takes the request to appropriate resource and send back the prepared response is known as Server Software. Both the server machine and server software are jointly called as Server. **The Server Software is also known as Web Server**

Web Server (HTTP Server)

- A Web Server handles HTTP requests sent by a client and responds back with an HTTP response
- It provides an **environment** in which requests and responses are handled and web pages are viewed.
- A Web server is also known as an **HTTP server**

There are many web servers in the industry, for example:

- Apache HTTP Server – **Tomcat**
- Internet Information System (IIS)
- Lighttpd, Resin, Jigsaw
- XAMPP, WAMP, LAMP etc.

The most used Web Server is **Apache Tomcat**.

Web Container

- ▶ A Web Container is used to generate dynamic content on user request
- ▶ It is server side JVM that **resides** either **inside a web server** to handle requests to servlets, JSPs, and other types of files that include server-side code.
- ▶ The web container
 - ✓ Creates servlet instances
 - ✓ Loads and Unloads servlets
 - ✓ Creates and manages request and response objects
 - ✓ Performs other servlet-management tasks.
- ▶ The Web Container is also known as **Servlet Container or J2EE container**

Tomcat Web Server

What is Tomcat?

Apache Tomcat is an open-source **web server** and **servlet/jsp container** developed by the Apache Software Foundation (ASF).

Tomcat implements several Java EE specifications including Java Servlet, Java Server Pages (JSP), Java EL, and WebSocket, and provides a “pure Java” HTTP web server environment for Java code to run in

Is Tomcat a Web Server or a Web Container?

Tomcat is **both** a web server (supports HTTP protocol) and a web container (supports JSP/Servlet API, also called "**Servlet container**" at times).

The Tomcat Web Container contains 2 components:

- ✓ Servlet Container
- ✓ JSP Container

Servlet Container

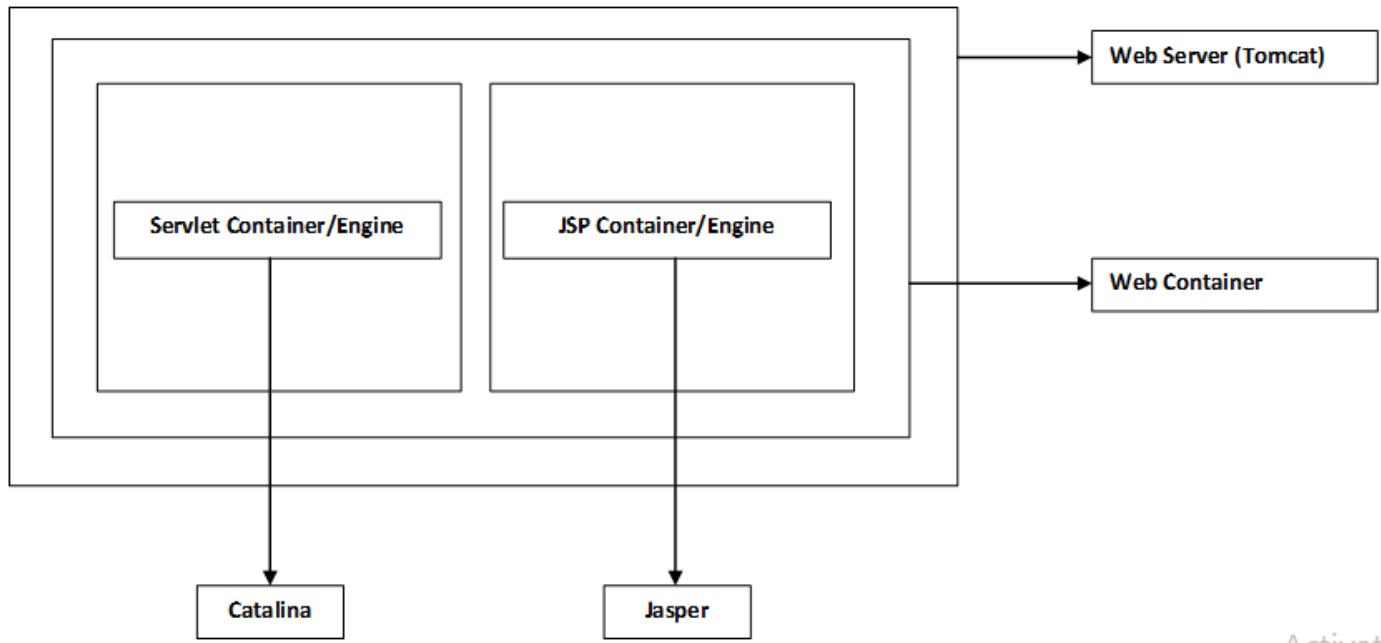
- ▶ A servlet container is essentially a part of the web container that **interacts** with the **servlets**.
- ▶ The Servlet Container is also known as **Servlet Engine**
- ▶ In Tomcat, the name of the Servlet Container Component is **Catalina**

JSP Container

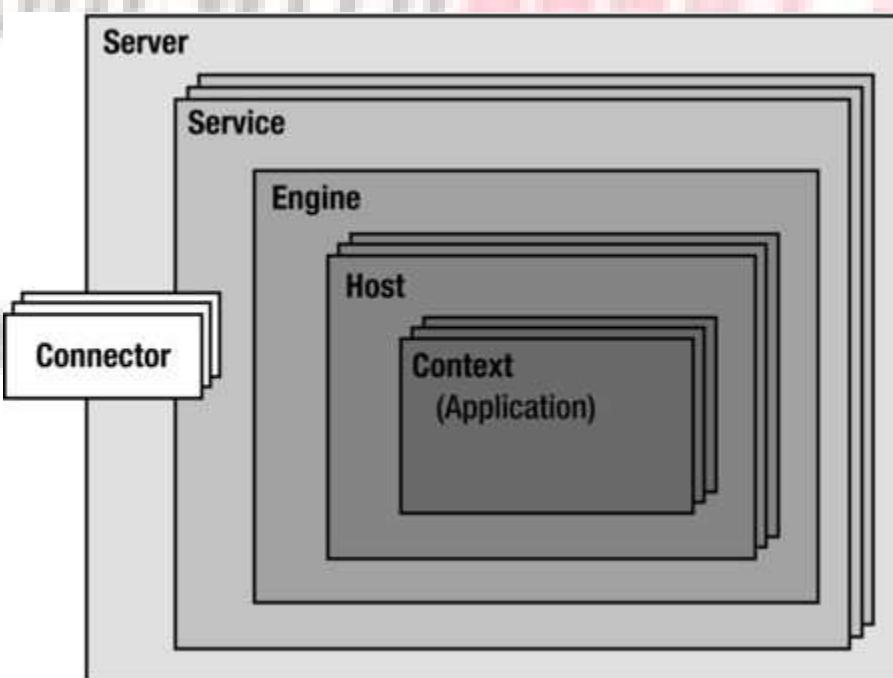
- ▶ A JSP container is a part of a web container that **interacts** with **JSP** requests.

- ▶ The JSP Container is also known as **JSP Engine**
- ▶ In Tomcat, the name of the JSP Container Component is **Jasper**

Schematic Diagram of Tomcat Web Server



Tomcat Architecture



The structure of each server installation (via these functional components) is defined in the file **server.xml**, which is located in the **/conf** subdirectory of Tomcat's installation folder.

Tomcat Port

By default, Tomcat is configured to run on port **8080**. That's why all the deployed web applications are accessible through URLs like <http://localhost:8080/yourapp>

How to Change port

To make this port change, open **server.xml** and find below entry:

```
<Connector
    port="8080"          !! Change this line
    protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
```

Servlets API & Ways to create a Servlet

Table of Contents

What is a Servlet?	2
Servlet API	2
java.servlet	3
java.servlet.http	3
How to create a Servlet?	4
Servlet API Hierarchy	4
Implementing Servlet Interface	5
Servlet Interface	5
Create a Servlet	5
Inheriting GenericServlet Class	7
GenericServlet Class	7
Creating the Servlet	7
Inheriting HttpServlet class	8
HttpServlet Class	8
HttpServlet class methods	8
Creating the Servlet	8
Serial Version UID	9
Serialization	9
Deserialization	9
SerialVersionUID	9

What is a Servlet?

Servlets are an overwhelming technology; it is not one but many things. It can be defined in many ways as below depending on the context:

- ▶ Servlet is a technology which is used to create a web application.
- ▶ Servlet is an **API** that provides many **interfaces** and **classes** to receive request and send response.
- ▶ Servlet is **an interface** that must be implemented for creating any Servlet.
- ▶ Servlet **extends the capabilities** of the servers and responds to the incoming requests.
- ▶ Servlet is a **web component** that is deployed on the server to create a dynamic web page.
- ▶ Servlets are **server-side programs** that run inside a *Java-capable* HTTP server (ex: Apache Tomcat) that handle clients' requests and return a **customized** or **dynamic response** for each request.

This dynamic response could be based on user's input such as

- ✓ Search
- ✓ Online shopping
- ✓ Online transaction
- ✓ Chat, Like, Share
- ✓ Submit a form etc.

Servlet API

The **Apache Tomcat software** is an **open source implementation** of the

1. Java Servlet
2. Java Server Pages
3. Java Expression Language
4. Java WebSocket
5. Java Annotations, and

Important Points:

- ▶ These specifications are part of the **Java EE platform**.
- ▶ The Java EE platform is the **evolution of** the Java EE platform.
- ▶ **Tomcat 10 and later** implement specifications developed as part of **Jakarta EE**.
- ▶ **Tomcat 9 and earlier** implement specifications developed as part of **Java EE**.

Servlet API

1. [javax.servlet](#)
2. [javax.servlet.annotation](#)
3. [javax.servlet.descriptor](#)
4. [javax.servlet.http](#)

We will discuss the most important packages here: **java.servlet** and **java.servlet.http**

java.servlet

It is the core package of the Servlet API, here is a list of all its important interfaces, classes and methods:

S. No	Interface/ Class	Description
1.	Servlet (I)	Defines methods that all servlets must implement.
2.	ServletConfig (I)	A servlet configuration object is used by a servlet container to pass information to a servlet during initialization.
3.	ServletContext (I)	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.
4.	ServletRequest (I)	Defines an object to provide client request information to a servlet. The servlet container creates a <code>ServletRequest</code> object and passes it as an argument to the servlet's <code>service</code> method.
5.	ServletResponse (I)	Defines an object to assist a servlet in sending a response to the client. The servlet container creates a <code>ServletResponse</code> object and passes it as an argument to the servlet's <code>service</code> method.
6.	RequestDispatcher (I)	Defines an object that receives requests from the client and sends them to any resource (such as a Servlet, HTML file, or JSP file) on the server
7.	GenericServlet (C)	Defines a generic, protocol-independent servlet.

We will be learning about every one of them in detail later in the tutorial

java.servlet.http

It contains the following classes and interfaces that hold the responsibility of handling HTTP requests and sending out HTTP responses:

S. No	Interface/Class	Description
1.	HttpServlet (C)	Provides an abstract class to be sub-classed to create an HTTP servlet suitable for a Web site.
2.	HttpServletRequest (I)	Extends the <code>ServletRequest</code> interface to provide request information for HTTP servlets
3.	HttpServletResponse (I)	Extends the <code>ServletResponse</code> interface to provide HTTP-specific functionality in sending a response.
4.	Cookie (C)	Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management.
5.	HttpSession (I)	Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

We will be learning about every one of them in detail later in the tutorial

How to create a Servlet?

As we know that a Servlet is used to exchange requests and response with a client. In order to do so, we need to create a **user defined servlet** and define request and response handling mechanism.

We can create a servlet in the following 3 ways:

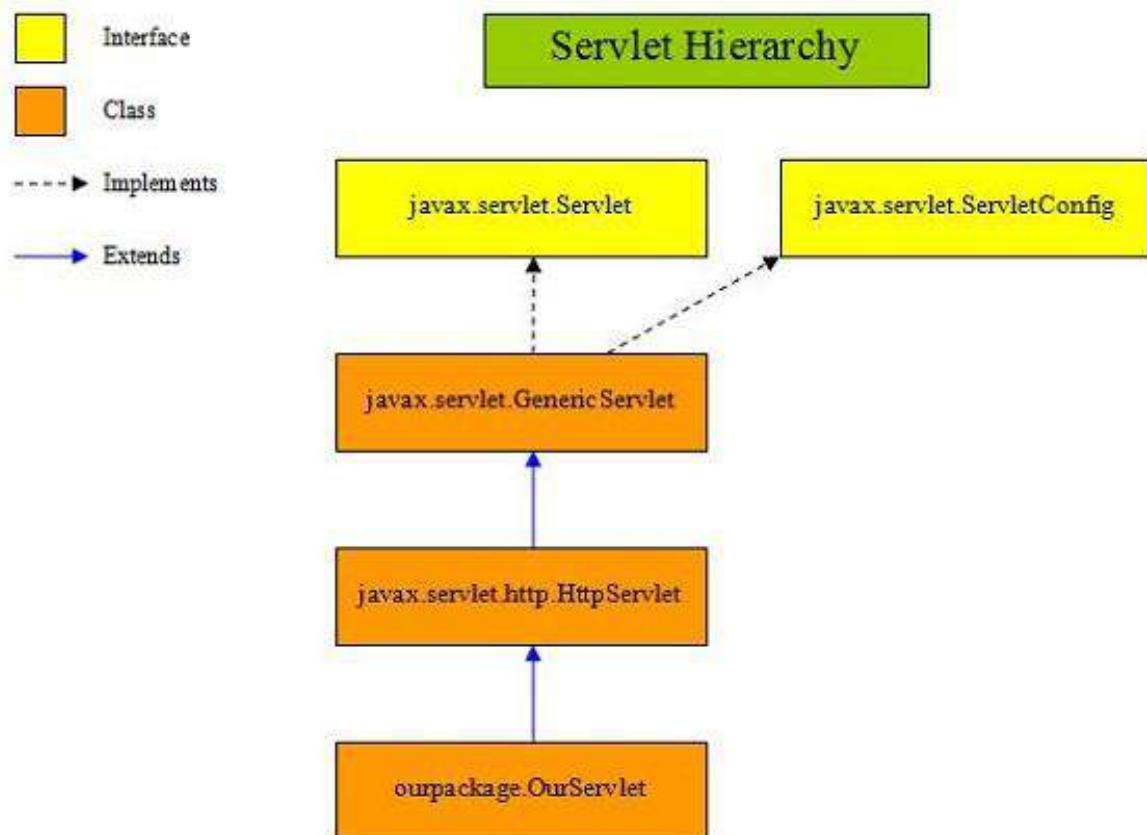
1. By implementing Servlet interface
2. By inheriting GenericServlet class
3. By inheriting HttpServlet class

Before studying each of them in brief, it is important to see the relationship between all the 3 interfaces and classes.

Servlet API Hierarchy

The below diagram depicts:

1. Servlet is an interface. It has 5 abstract methods
2. GenericServlet is a class. It **implements** Servlet Interface and defines 4 of its methods
3. HttpServlet **extends** GenericServlet class. It defines the last remaining method service()
4. Our custom servlet must **extend** HttpServlet in case we need to deal with HTTP requests/responses



Now, as we have got an abstract look at the hierarchy, let's look the 3 ways of creating a servlet one by one.

Implementing Servlet Interface

Servlet Interface

Servlet is the main interface that declares methods that all servlets must implement.

Main Points

The main points about the Servlet are:

- ✓ Servlet is an interface
- ✓ Fully qualified name: javax.servlet.Servlet
- ✓ Signature: public interface Servlet {}
- ✓ Extends: Nothing
- ✓ Implemented by: GenericServlet class

Abstract Methods in Servlet Interface

1. public abstract void **init**(ServletConfig config) throws ServletException
2. public abstract void **service**(ServletRequest req, ServletResponse res) throws ServletException, IOException
3. public abstract void **destroy**()
4. public abstract ServletConfig **getServletConfig**()
5. public abstract String **getServletInfo**()

Create a Servlet

Step 1: Define a normal class that implements java.servlet.Servlet

```
class FirstServlet implements Servlet {  
}
```

Step 2: Override all the abstract methods of Servlet interface

```
public class FirstServlet implements Servlet {  
  
    ServletConfig servletConfig;  
  
    @Override  
    public void init(ServletConfig servletConfig) throws ServletException {  
        this.servletConfig = servletConfig;  
        System.out.println("Servlet is initialized");  
    }  
  
    @Override  
    public void service(ServletRequest req, ServletResponse res) throws
```

```

ServletException, IOException {
    System.out.println("Request is being serviced...");
}

@Override
public void destroy() {
    System.out.println("Servlet is destroyed");
}

@Override
public ServletConfig getServletConfig() {
    return this.servletConfig;
}

@Override
public String getServletInfo() {
    return "FirstServlet";
}

}

```

Note: Put SOPs inside the overridden methods to track the servlet lifecycle

Step 3: Map the servlet in web.xml

```

<servlet>
    <servlet-name>first</servlet-name>
    <servlet-class>com.java.servlets.FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>first</servlet-name>
    <url-pattern>/first</url-pattern>
</servlet-mapping>

```

Step 4: Execute the project

Start the Server and run the project, type: **localhost:8080/02ServletCreationTechniques/first** to run the Servlet

Step 5: Track the Output

- ▶ init() will run once when the first request is made
- ▶ service() will run on every request
- ▶ destroy() will be called when we stop the server

Observations

- ▶ init(), service() and destroy() are lifecycle methods
- ▶ Lifecycle methods are called implicitly by the web container.
- ▶ Their sequence and mode of calling depends on the state of the servlet
 - When the 1st request is made, the web container initializes the servlet by calling init()

- Every request to the servlet triggers the service()
- To call destroy , we have to explicitly stop the server
- ▶ getServletConfig() and getServletInfo() are non-life cycle methods
- ▶ These methods are not called implicitly & have to be called explicitly

Note: You can find the above example in Project: 02ServletCreationTechniques

Inheriting GenericServlet Class

GenericServlet Class

GenericServlet defines a generic, protocol-independent servlet. It must be used to handle any type of request

Main Points

The main points about the GenericServlet are:

- ✓ GenericServlet is a class
- ✓ Fully qualified name: javax.servlet.GenericServlet
- ✓ Signature: public abstract class GenericServlet
- ✓ Extends: java.lang.Object
- ✓ Implements: Servlet, ServletConfig, Serializable
- ✓ Sub-classes: HttpServlet

Creating the Servlet

Step 1: Create a new class and extend it from GenericServlet

```
public class SecondServlet extends GenericServlet {  
}
```

Step 2: Override unimplemented service() of GenericServlet class

GenericServlet class provides implementation of 4 methods of Servlet interface, however service() still remains unimplemented.

```
public class SecondServlet extends GenericServlet {  
  
    private static final long serialVersionUID = 1L;  
  
    @Override  
    public void service(ServletRequest req, ServletResponse res) throws  
        ServletException, IOException {  
        System.out.println("Request is being serviced");  
    }  
}
```

Step 3: Execute the project & we will notice that service is triggered whenever there is a new request

Note: You can find the above example in Project: 02ServletCreationTechniques

Inheriting HttpServlet class

HttpServlet Class

- ▶ The websites communicate with the server through HTTP protocol
- ▶ If we make a web application, we extend our Servlet from HttpServlet
- ▶ We can say that HttpServlet class is used to create http protocol specific servlet.
- ▶ HttpServlet class is abstract although all its methods are concrete. This is done to force its child class to override request specific doXXX() method

Main Points

The main points about the HttpServlet are:

- ✓ HttpServlet is a class
- ✓ Fully qualified name: java.servlet.http.HttpServlet
- ✓ Signature: public abstract class HttpServlet
- ✓ Extends: java.servlet.GenericServlet
- ✓ Implements: Servlet, ServletConfig, Serializable (**all indirectly through GenericServlet**)
- ✓ Predefined Sub-classes: No

HttpServlet class methods

A subclass of HttpServlet must override at least one method, usually one of these:

1. doGet(), for HTTP GET requests
2. doPost(), for HTTP POST requests
3. doPut(), for HTTP PUT requests
4. doDelete(), for HTTP DELETE requests
5. getServletInfo(), which the servlet uses to provide information about itself
6. service(): There's no reason to override service() as it passes HTTP requests to HttpServlet handler methods

Creating the Servlet

Step 1: Create a new class and extend it from HttpServlet

```
public class ThirdServlet extends HttpServlet {  
}
```

Step 2: Override any one method that you wish to be handled

```
public class ThirdServlet extends HttpServlet {  
  
    private static final long serialVersionUID = 1L;  
  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse  
        resp) throws ServletException, IOException {  
  
        System.out.print("Request is being processed");  
    }  
}
```

{}

Step 3: Execute the project & we will notice that doGet is triggered for get request

Note: You can find the above example in Project: 02ServletCreationTechniques

Serial Version UID

GenericServlet and HttpServlet classes implement **java.io.Serializable** interface as we can see in class definitions. Whenever we create a servlet by extending these classes, we need to provide a serialVersionUID. To understand its significance, we must have familiarity with the concept of serialization and deserialization.

Serialization

Serialization is a mechanism of converting the state of an object into a byte stream.

Deserialization

Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

SerialVersionUID

The Serialization runtime associates a version number with each Serializable class called a SerialVersionUID, which is used during Deserialization to verify that sender and receiver of a serialized object have loaded classes for that object which are compatible with respect to serialization.

If there is a mismatch in UIDs of serialized and de-serialized classes, the deserialization will result in an **InvalidClassException**. A Serializable class can declare its own UID explicitly by declaring a field name.

It must be static, final and of type long: i.e. private static final long serialVersionUID=42L;

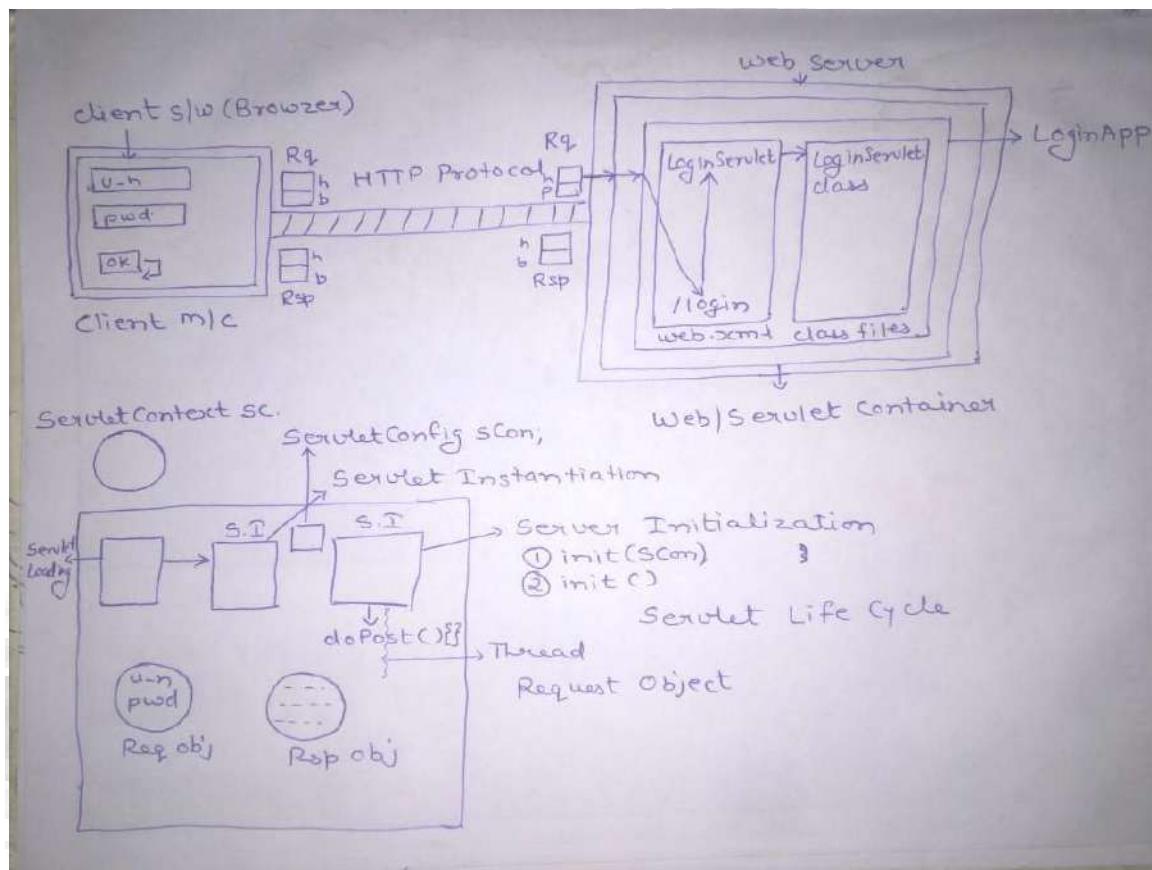
Note: You can find the above example in Project: 02ServletCreationTechniques

Servlet Work Flow and Life Cycle

Table of Contents

Servlet Workflow	2
Example Application	2
Starting the Server	2
Task 1 of Web Server	2
Task 2 of Web Server	3
Request is made	3
Scenario 1: A static page is requested	3
Scenario 2: A dynamic page is requested	3
HTTP Protocol	3
Request reaches the Web Server	4
Web Server checks the request	4
Web Server locates the resources	4
Servlet Life Cycle	5
1. Servlet Loading	5
2. Servlet Instantiation	5
3. Servlet Initialization	5
4. Thread Creation and Data Processing	6
Brief Life cycle	7

Servlet Workflow



Example Application

To understand Servlet Workflow, consider an example application. This example includes:

- ▶ A Web Application named as **LoginApplication**
- ▶ This Web Application contains an html page **login.html**, a servlet **LoginServlet** and a **web.xml** file
- ▶ This login form will have 2 fields: **user_name** and **password**.
- ▶ The form tag contains attributes: **method="post"**, **action="/login"**
- ▶ The servlet LoginServlet has URL pattern "**/login**"

Starting the Server

Task 1 of Web Server

When we start the web server the job of the web container is:

- ▶ To recognize all the web applications

- ▶ Deploy each web application on the server, and
- ▶ Prepare a separate object for each web application.
- ▶ This object is known as **ServletContext Object**.

For example, as we start **Tomcat Web Server**, the web container

- ▶ Recognizes the Login application
- ▶ Deploy it to the server and
- ▶ Prepare an object for Login Application called **ServletContext Object**.

Task 2 of Web Server

The web container also

- ▶ Recognizes the web.xml file present inside the WEB-INF folder.
- ▶ Loads the web.xml file
- ▶ Parses the file and
- ▶ Reads the content of the file.
- ▶ If any application level data is available, the container will store this data inside the ServletContext object.

Request is made

Scenario 1: A static page is requested

The Web Server is responsible to deliver static content to the client. So, as a request is made from the client browser, the requested static page is delivered through the web server to the client browser.

Scenario 2: A dynamic page is requested

Prerequisite

- 1: A page is displayed in the browser.
- 2: The page contains a login form with 2 input fields: username and password.

Step 1: The data is entered in the form fields and button is clicked.

HTTP Protocol

- ▶ The request will come to protocol (here HTTP).
- ▶ Protocol will establish a virtual socket connection between client and server on the basis of server IP address and port number.
- ▶ After preparing the socket connection, the protocol will prepare a request format.

- The request format contains 2 parts:

- Header part
 - Body part.

Header Part

The header will manage **request headers**. Request headers contain **details about the client browser** such as:

- The Locale managed by the client browser
- The Zipping formats supported by the client browser
- The Encoding mechanism supported by the client browser.

Body Part

The body part will manage **request parameters** data. The request parameters data is nothing but the data provided by the user through the form.

Request reaches the Web Server

- Whenever the request format is prepared, the protocol will carry this request format to web server.
- This request format is given to main web server.

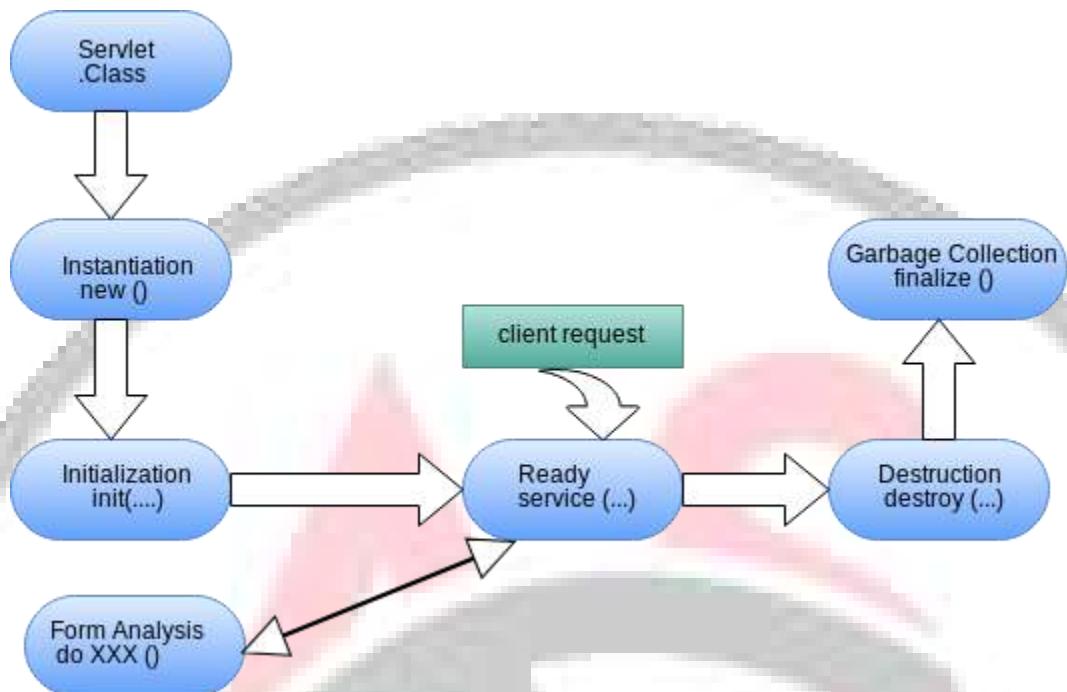
Web Server checks the request

- The web server will first check whether it is a valid request or not.
- If the request is valid, the web server will forward the request to the web container.
- The web container will identify the name of the requested application and of the requested resource.
- The information required by container is mentioned in the form tag in **action** attribute. It contains either the **name of the servlet** or the **URL pattern** of the requested servlet.

Web Server locates the resources

- The web container will go to our Login Application Folder and then to web.xml present inside WEB-INF folder.
- In web.xml, mapping details are available i.e. the mapping between the URL pattern (/login) and the servlet name (LoginServlet).
- The container identifies the LoginServlet resource from web.xml.
- Now the container searches for **.class** files in class' folder for the particular servlet.
- As the container finds the LoginServlet.class file, it will begin the **Servlet Life Cycle**.

Servlet Life Cycle



1. Servlet Loading

Container will load LoginServlet.class file's bytecode into the memory i.e. perform Servlet loading.

```
Class c = Class.forName("LoginServlet.class")
```

Before loading the servlet, the container will first check whether this particular servlet is loaded beforehand or not. If the servlet is not loaded, then only the container will perform the loading of the servlet.

2. Servlet Instantiation

After LoginServlet class loading, the container will prepare an object of LoginServlet (Servlet Instantiation).

```
Object obj = c.newInstance()
```

3. Servlet Initialization

After Servlet Instantiation, the web container will perform Servlet Initialization. Here the container will call init() on Servlet and pass ServletConfig object in it.

- ▶ init (ServletConfig servletConfig)

```
Server Initialization: init(ServletConfig s)
```

This method belong to **GenericServlet class**.

- ▶ The servlet container calls the init method exactly once after instantiating the servlet.
- ▶ The init method must complete successfully before the servlet can receive any requests.
- ▶ The servlet container will call service () after init is completed

4. Thread Creation and Data Processing

```
service(ServletRequest, ServletResponse) in GenericServlet class  
service(HttpServletRequest, HttpServletResponse) in HttpServlet class  
doXXX(HttpServletRequest, HttpServletResponse) in LoginServlet class
```

- ▶ As soon as the servlet initialization is complete, the web container will spawn a new thread and call the service () method.
- ▶ The service () method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- ▶ As the form contains get method, we must implement the doGet() in LoginServlet class.
- ▶ The container prepares HttpServletRequest and HttpServletResponse objects.
- ▶ The HttpServletRequest object will contain 3 types of data
 - Request **Headers** Data: the client browser data
 - Request **Parameters** Data: the data that we entered in form: u_n, pwd
 - Request **Attributes** Data: the dynamic data the servlet may include in request attributes

5. Response is prepared

- ▶ As container executes the doPost(), some data is created in Response object.
- ▶ As the response is created and the thread reaches the end point, it service() and doPost() ends
- ▶ As the thread is near destroy, the servlet sends the response object to the main server.
- ▶ Main server will format the response and give it to protocol and this protocol prepares the header and body part of response.
 - Header will contain type of response, length of response etc.
 - The body will contain dynamic data.

6. Response is sent back

- ▶ The protocol will carry this data to client machine.
- ▶ Browser will get response from body part and parse and display the result.
- ▶ As the information is displayed, the protocol will destroy the connection.
- ▶ As soon as the connection is destroyed the server will destroy the request and response objects.

- The container will again come in waiting state.

Brief Life cycle

- init() is called when the request is made for the 1st time
- service() is called on every request
- destroy() is called, when the server stops which in turn destroys all servlets



<CODEWITHARRAYS/>

Servlets API & Ways to create a Servlet

Table of Contents

What is a Servlet?	2
Servlet API	2
java.servlet	3
java.servlet.http	3
How to create a Servlet?	4
Servlet API Hierarchy	4
Implementing Servlet Interface	5
Servlet Interface	5
Create a Servlet	5
Inheriting GenericServlet Class	7
GenericServlet Class	7
Creating the Servlet	7
Inheriting HttpServlet class	8
HttpServlet Class	8
HttpServlet class methods	8
Creating the Servlet	8
Serial Version UID	9
Serialization	9
Deserialization	9
SerialVersionUID	9

What is a Servlet?

Servlets are an overwhelming technology; it is not one but many things. It can be defined in many ways as below depending on the context:

- ▶ Servlet is a technology which is used to create a web application.
- ▶ Servlet is an **API** that provides many **interfaces** and **classes** to receive request and send response.
- ▶ Servlet is **an interface** that must be implemented for creating any Servlet.
- ▶ Servlet **extends the capabilities** of the servers and responds to the incoming requests.
- ▶ Servlet is a **web component** that is deployed on the server to create a dynamic web page.
- ▶ Servlets are **server-side programs** that run inside a *Java-capable* HTTP server (ex: Apache Tomcat) that handle clients' requests and return a **customized** or **dynamic response** for each request.

This dynamic response could be based on user's input such as

- ✓ Search
- ✓ Online shopping
- ✓ Online transaction
- ✓ Chat, Like, Share
- ✓ Submit a form etc.

Servlet API

The **Apache Tomcat software** is an **open source implementation** of the

6. Java Servlet
7. Java Server Pages
8. Java Expression Language
9. Java WebSocket
10. Java Annotations, and

Important Points:

- ▶ These specifications are part of the **Java EE platform**.
- ▶ The Java EE platform is the **evolution of** the Java EE platform.
- ▶ **Tomcat 10 and later** implement specifications developed as part of **Jakarta EE**.
- ▶ **Tomcat 9 and earlier** implement specifications developed as part of **Java EE**.

Servlet API

5. [javax.servlet](#)
6. [javax.servlet.annotation](#)
7. [javax.servlet.descriptor](#)

8. [javax.servlet.http](#)

We will discuss the most important packages here: **java.servlet** and **java.servlet.http**

java.servlet

It is the core package of the Servlet API, here is a list of all its important interfaces, classes and methods:

S. No	Interface/ Class	Description
1.	Servlet (I)	Defines methods that all servlets must implement.
2.	ServletConfig (I)	A servlet configuration object is used by a servlet container to pass information to a servlet during initialization.
3.	ServletContext (I)	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.
4.	ServletRequest (I)	Defines an object to provide client request information to a servlet. The servlet container creates a <code>ServletRequest</code> object and passes it as an argument to the servlet's <code>service</code> method.
5.	ServletResponse (I)	Defines an object to assist a servlet in sending a response to the client. The servlet container creates a <code>ServletResponse</code> object and passes it as an argument to the servlet's <code>service</code> method.
6.	RequestDispatcher (I)	Defines an object that receives requests from the client and sends them to any resource (such as a Servlet, HTML file, or JSP file) on the server
7.	GenericServlet (C)	Defines a generic, protocol-independent servlet.

We will be learning about every one of them in detail later in the tutorial

java.servlet.http

It contains the following classes and interfaces that hold the responsibility of handling HTTP requests and sending out HTTP responses:

S. No	Interface/Class	Description
1.	HttpServlet (C)	Provides an abstract class to be sub-classed to create an HTTP servlet suitable for a Web site.
2.	HttpServletRequest (I)	Extends the <code>ServletRequest</code> interface to provide request information for HTTP servlets
3.	HttpServletResponse (I)	Extends the <code>ServletResponse</code> interface to provide HTTP-specific functionality in sending a response.
4.	Cookie (C)	Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management.
5.	HttpSession (I)	Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

We will be learning about every one of them in detail later in the tutorial

How to create a Servlet?

As we know that a Servlet is used to exchange requests and response with a client. In order to do so, we need to create a **user defined servlet** and define request and response handling mechanism.

We can create a servlet in the following 3 ways:

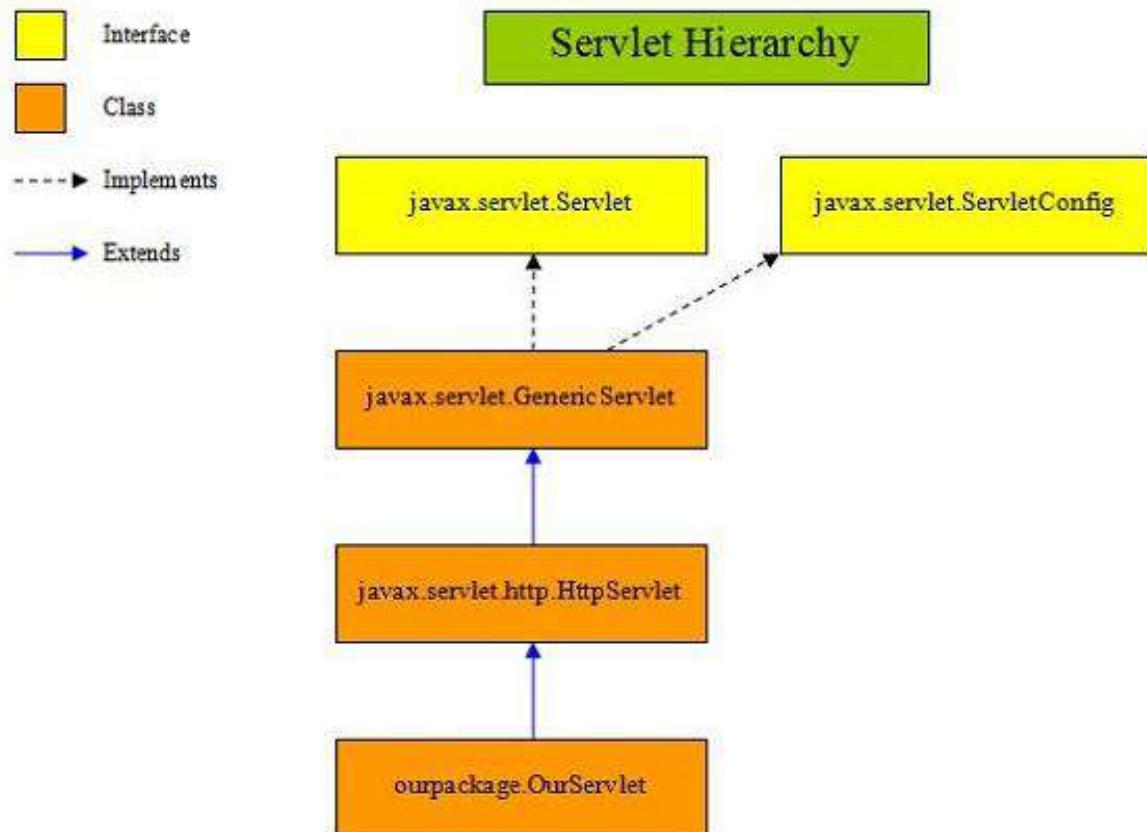
4. By implementing Servlet interface
5. By inheriting GenericServlet class
6. By inheriting HttpServlet class

Before studying each of them in brief, it is important to see the relationship between all the 3 interfaces and classes.

Servlet API Hierarchy

The below diagram depicts:

1. Servlet is an interface. It has 5 abstract methods
2. GenericServlet is a class. It **implements** Servlet Interface and defines 4 of its methods
3. HttpServlet **extends** GenericServlet class. It defines the last remaining method service()
4. Our custom servlet must **extend** HttpServlet in case we need to deal with HTTP requests/responses



Now, as we have got an abstract look at the hierarchy, let's look the 3 ways of creating a servlet one by one.

Implementing Servlet Interface

Servlet Interface

Servlet is the main interface that declares methods that all servlets must implement.

Main Points

The main points about the Servlet are:

- ✓ Servlet is an interface
- ✓ Fully qualified name: javax.servlet.Servlet
- ✓ Signature: public interface Servlet {}
- ✓ Extends: Nothing
- ✓ Implemented by: GenericServlet class

Abstract Methods in Servlet Interface

6. public abstract void **init**(ServletConfig config) throws ServletException
7. public abstract void **service**(ServletRequest req, ServletResponse res) throws ServletException, IOException
8. public abstract void **destroy**()
9. public abstract ServletConfig **getServletConfig**()
10. public abstract String **getServletInfo**()

Create a Servlet

Step 1: Define a normal class that implements java.servlet.Servlet

```
class FirstServlet implements Servlet {  
}
```

Step 2: Override all the abstract methods of Servlet interface

```
public class FirstServlet implements Servlet {  
  
    ServletConfig servletConfig;  
  
    @Override  
    public void init(ServletConfig servletConfig) throws ServletException {  
        this.servletConfig = servletConfig;  
        System.out.println("Servlet is initialized");  
    }  
  
    @Override
```

```

public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException {
    System.out.println("Request is being serviced...");
}

@Override
public void destroy() {
    System.out.println("Servlet is destroyed");
}

@Override
public ServletConfig getServletConfig() {
    return this.servletConfig;
}

@Override
public String getServletInfo() {
    return "FirstServlet";
}

}

```

Note: Put SOPs inside the overridden methods to track the servlet lifecycle

Step 3: Map the servlet in web.xml

```

<servlet>
    <servlet-name>first</servlet-name>
    <servlet-class>com.java.servlets.FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>first</servlet-name>
    <url-pattern>/first</url-pattern>
</servlet-mapping>

```

Step 4: Execute the project

Start the Server and run the project, type: **localhost:8080/02ServletCreationTechniques/first** to run the Servlet

Step 5: Track the Output

- ▶ init() will run once when the first request is made
- ▶ service() will run on every request
- ▶ destroy () will be called when we stop the server

Observations

- ▶ init (), service() and destroy() are lifecycle methods
- ▶ Lifecycle methods are called implicitly by the web container.
- ▶ Their sequence and mode of calling depends on the state of the servlet

- When the 1st request is made, the web container initializes the servlet by calling init()
 - Every request to the servlet triggers the service()
 - To call destroy, we have to explicitly stop the server
- getServletConfig() and getServletInfo() are non-life cycle methods
- These methods are not called implicitly & have to be called explicitly

Note: You can find the above example in Project: 02ServletCreationTechniques

Inheriting GenericServlet Class

GenericServlet Class

GenericServlet defines a generic, protocol-independent servlet. It must be used to handle any type of request

Main Points

The main points about the GenericServlet are:

- ✓ GenericServlet is a class
- ✓ Fully qualified name: javax.servlet.GenericServlet
- ✓ Signature: public abstract class GenericServlet
- ✓ Extends: java.lang.Object
- ✓ Implements: Servlet, ServletConfig, Serializable
- ✓ Sub-classes: HttpServlet

Creating the Servlet

Step 1: Create a new class and extend it from GenericServlet

```
public class SecondServlet extends GenericServlet {  
}
```

Step 2: Override unimplemented service() of GenericServlet class

GenericServlet class provides implementation of 4 methods of Servlet interface, however service() still remains unimplemented.

```
public class SecondServlet extends GenericServlet {  
  
    private static final long serialVersionUID = 1L;  
  
    @Override  
    public void service(ServletRequest req, ServletResponse res) throws  
        ServletException, IOException {  
        System.out.println("Request is being serviced");  
    }  
}
```

Step 3: Execute the project & we will notice that service is triggered whenever there is a new request

Note: You can find the above example in Project: 02ServletCreationTechniques

Inheriting HttpServlet class

HttpServlet Class

- ▶ The websites communicate with the server through HTTP protocol
- ▶ If we make a web application, we extend our Servlet from HttpServlet
- ▶ We can say that HttpServlet class is used to create http protocol specific servlet.
- ▶ HttpServlet class is abstract although all its methods are concrete. This is done to force its child class to override request specific doXXX() method

Main Points

The main points about the HttpServlet are:

- ✓ HttpServlet is a class
- ✓ Fully qualified name: java.servlet.http.HttpServlet
- ✓ Signature: public abstract class HttpServlet
- ✓ Extends: java.servlet.GenericServlet
- ✓ Implements: Servlet, ServletConfig, Serializable (**all indirectly through GenericServlet**)
- ✓ Predefined Sub-classes: No

HttpServlet class methods

A subclass of HttpServlet must override at least one method, usually one of these:

7. doGet(), for HTTP GET requests
8. doPost(), for HTTP POST requests
9. doPut(), for HTTP PUT requests
10. doDelete(), for HTTP DELETE requests
11. getServletInfo(), which the servlet uses to provide information about itself
12. service(): There's no reason to override service() as it passes HTTP requests to HttpServlet handler methods

Creating the Servlet

Step 1: Create a new class and extend it from HttpServlet

```
public class ThirdServlet extends HttpServlet {  
}
```

Step 2: Override any one method that you wish to be handled

```
public class ThirdServlet extends HttpServlet {  
  
    private static final long serialVersionUID = 1L;  
  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse  
    resp) throws ServletException, IOException {  
  
        System.out.print("Request is being processed");  
    }  
}
```

```
}
```

Step 3: Execute the project & we will notice that doGet is triggered for get request

Note: You can find the above example in Project: 02ServletCreationTechniques

Serial Version UID

GenericServlet and HttpServlet classes implement **java.io.Serializable** interface as we can see in class definitions. Whenever we create a servlet by extending these classes, we need to provide a serialVersionUID. To understand its significance, we must have familiarity with the concept of serialization and deserialization.

Serialization

Serialization is a mechanism of converting the state of an object into a byte stream.

Deserialization

Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

SerialVersionUID

The Serialization runtime associates a version number with each Serializable class called a SerialVersionUID, which is used during Deserialization to verify that sender and receiver of a serialized object have loaded classes for that object which are compatible with respect to serialization.

If there is a mismatch in UIDs of serialized and de-serialized classes, the deserialization will result in an **InvalidClassException**. A Serializable class can declare its own UID explicitly by declaring a field name.

It must be static, final and of type long: i.e. private static final long serialVersionUID=42L;

Note: You can find the above example in Project: 02ServletCreationTechniques

Handling Form Data & Generating Dynamic Response

Table of Contents

Generating Dynamic Response	2
PrintWriter	2
Handling Data	2
Link click	2
Requesting a Static Page	2
Requesting a Dynamic Page	2
Requesting a Dynamic Page with parameters	3
Requesting a Servlet	3
Requesting a Servlet with data	3
Form Submission	3
Text Input	4
Note: Find the practical implementation in the project: 04RequestResponseWithServlets	6
File Input	6

Generating Dynamic Response

PrintWriter

- ▶ Java PrintWriter class is the implementation of **Writer** class.
- ▶ It is used to print the **formatted** representation of objects to the **text-output stream**.
- ▶ It is present in `java.io` package.
- ▶ It enables us to write formatted data to an underlying Writer.
- ▶ For instance, writing int, long and other primitive data **formatted as text**, rather than as their byte values.
- ▶ It creates a character based output streams
- ▶ It is useful where we have to mix text and numbers.
- ▶ The PrintWriter class has all the same methods as the **PrintStream**
- ▶ Being a Writer subclass the PrintWriter is intended to **write text**.

Handling Data

The major source of request from a client includes:

1. Link click
2. Form submission

Link click

The following code shows the creation of a link:

Requesting a Static Page

Code Snippet

```
<a href="home.html">Home</a>
```

When “Home” link is clicked, a **get** request will be generated for the static page “home.html”

Requesting a Dynamic Page

Code Snippet

```
<a href="home.jsp">Home</a>
```

When “Home” link is clicked, a **get** request will be generated for the dynamic page “home.jsp”

Requesting a Dynamic Page with parameters

Code Snippet

```
<a href="home.jsp?greeting=good-morning&name=kaustubh">Home</a>
```

When “Home” link is clicked, the following data is sent to home.jsp in a get request:

- “greeting” parameter containing the value “good-morning”
- “name” parameter containing the value “kaustubh”

Requesting a Servlet

Code Snippet

```
<a href="register">Register</a>
```

When “Register” is clicked, a **get** request will be generated for the Servlet mapped by “/register”

Requesting a Servlet with data

Code Snippet

```
<%  
    String name="kaustubh";  
    int age = 25;  
%>  
<a href="register?name=<%=name%>&age=<%=age%>">Register</a>
```

When “Register” link is clicked, the following data is sent to the servlet with mapping “register” and following data:

- “name” parameter containing the value “kaustubh”
- “age” parameter containing the value “25”

Form Submission

A form contains many elements that take user input. These user inputs can be:

- Text: name, password, mobile no, email id, gender, hobbies, address etc.
- Date & Time: date of birth, time of an event, timestamp of a picture
- Files: Images, PDF file, MS document file, audios and videos etc.

Let us take a look at the different user input elements and how they must be fetched in a servlet

Text Input

1. Input type text

```
<input type="text" name="full-name"/>

String fullName = request.getParameter("full-name");
```

2. Input type password

```
<input type="password" name="user-password"/>

String userPassword = request.getParameter("user-password ");
```

3. Input type number

```
<input type="number" name="mobile-number"/>

String mobileNumber = request.getParameter("mobile-number");
```

4. Input type email

```
<input type="email" name="email-id"/>

String emailId = request.getParameter("email-id");
```

5. Input type radio

```
<input type="radio" name="gender" value="male" />Male

<input type="radio" name="gender" value="female" />Female
```

```
String gender = request.getParameter("gender");
```

6. Text Area

```
<textarea name="address" rows="8" cols="40"></textarea>
```

```
String address = request.getParameter("address");
```

7. Dropdown Select

```
<select name="courses">
    <option value="java">java</option>
    <option value="python">python</option>
    <option value="javascript">javascript</option>
</select>
```

```
String courses = request.getParameter("courses"); // In case of 1 selection for a dropdown
```

```
String[] courses = request.getParameterValues("courses"); // In case of multiple selections
```

8. Input type checkbox

```
<input type="checkbox" name="hobby" value="drawing" /> Drawing
<input type="checkbox" name="hobby" value="singing" />Singing
<input type="checkbox" name="hobby" value="dancing" />Dancing
<input type="checkbox" name="hobby" value="cooking" />Cooking
```

```
String[] hobbies = request.getParameterValues("hobby");
```

Note: Find the practical implementation in the project: **04RequestResponseWithServlets**

File Input

Before Java EE 6, applications usually have to use an external library like **Apache's Common File Upload** to handle file upload functionality. Fortunately, developers do no longer have to depend on any external library, since Java EE 6 provides **built-in file upload API**.

File Upload API in Servlet 3.0

The Servlet 3.0 API provides some new APIs for working with upload data:

1. **Annotation @MultipartConfig:** A servlet can be annotated with this annotation in order to handle multipart/form-data requests which contain file upload data. The MultipartConfig annotation has the following options:
 - ▶ fileSizeThreshold: file's size that is greater than this threshold will be directly written to disk, instead of saving in memory.
 - ▶ location: directory where file will be stored via Part.write() method.
 - ▶ maxFileSize: maximum size for a single upload file.
 - ▶ maxRequestSize: maximum size for a request.
 - ▶ All sizes are measured in bytes.
2. **Interface Part:** represents a part in a multipart/form-data request. This interface defines some methods for working with upload data (to name a few):
 - ▶ getInputStream(): returns an InputStream object which can be used to read content of the part.
 - ▶ getSize(): returns the size of upload data, in bytes.
 - ▶ write(String filename): this is the convenience method to save upload data to file on disk. The file is created relative to the location specified in the MultipartConfig annotation.
3. **New methods** introduced in HttpServletRequest interface:
 - ▶ getParts(): returns a collection of Part objects
 - ▶ getPart(String name): retrieves an individual Part object with a given name.

These new APIs make our life easier, really!

The Code

The code to save upload file is as follows:

```
for (Part part : request.getParts()) {  
    String fileName = extractFileName(part);  
    part.write(fileName);  
}
```

The above code simply iterates over all parts of the request, and save each part to disk using the write() method.

The file name is extracted in the following method:

```
private String extractFileName(Part part) {  
    String contentDisp = part.getHeader("content-disposition");  
    String[] items = contentDisp.split(";");  
    for (String s : items) {  
        if (s.trim().startsWith("filename")) {  
            return s.substring(s.indexOf("=") + 2, s.length() - 1);  
        }  
    }  
    return "";  
}
```

Because file name of the upload file is included in **content-disposition** header like this:

form-data; name="dataFile"; filename="PHOTO.JPG"

So the extractFileName() method gets PHOTO.JPG out of the string.

Header content-disposition

In a multipart/form-data body, the HTTP Content-Disposition general header is a header that must be used on **each subpart** of a multipart body to give information about the field it applies to. The subpart is delimited by the **boundary** defined in the **Content-Type** header.

Example:

```
Content-Disposition: form-data; name="fieldName"  
Content-Disposition: form-data; name="file"; filename="filename.jpg"  
3  
4  
5  
6  
.  
.  
.
```

Java Server Pages (JSP)

Table of Contents

<u>What is JSP?</u>	10
<u>Difference between JSP and HTML</u>	10
<u>Advantages of JSP over Servlets</u>	10
1. <u>Extension to Servlet</u>	10
2. <u>Giving flexibility to Designing</u>	11
3. <u>Easy to maintain</u>	11
<u>Lifecycle of JSP Page</u>	11
<u>Translation</u>	12
<u>Compilation</u>	12
<u>Class Loading</u>	12
<u>Instantiation</u>	12
<u>Initialization</u>	13
<u>Thread Creation & Request Processing</u>	13
<u>Destroy is called</u>	13
<u>Important Points</u>	13
<u>JSP Architecture</u>	13
<u>JSP Architecture Flow</u>	14
<u>First JSP program in Eclipse</u>	14
<u>Directory structure of JSP</u>	15
<u>The JSP API</u>	15

<u>Package javax.servlet.jsp</u>	15
<u>Interfaces</u>	Error! Bookmark not defined.
<u>Classes</u>	15
<u>JSP Page (I)</u>	16
<u> Methods of JspPage interface</u>	16
<u>HttpJspPage (I)</u>	16
<u> Method of HttpJspPage interface</u>	16



What is JSP?

- ▶ JSP stands for Java Server Pages
- ▶ JSP is a java technology (just like Servlets) used to create web applications.
- ▶ **JSP can be said as an extension to Servlet**
- ▶ A JSP page consists of HTML tags and JSP tags.
- ▶ Using JSP we can separate view with controller

Difference between JSP and HTML

The differences between JSP and HTML are given below:

S. No.	HTML	JSP
1.	HTML is a Client side technology	JSP is a Server side technology .
2.	HTML generates static web pages .	JSP generates dynamic web pages .
3.	HTML is given by W3C (World Wide Web Consortium).	JSP is given by Sun Micro System .
4.	An HTML page can have only HTML tags and content	A JSP page can have both - HTML as well as java code.
5.	Need HTML Interpreter to execute this code.	Need JSP container to execute JSP code.
6.	It does not allow us to place custom tag or third party tag.	It allows us to place custom tag or third party tag.
7.	The file with all HTML content has .HTML or .htm extension	The file with even 1 JSP statement must have .jsp extension

Advantages of JSP over Servlets

"Servlet is HTML inside Java", while **"JSP is Java inside HTML"**.

There are many advantages of JSP over the Servlet. They are as follows:

1. Extension to Servlet

- ▶ JSP technology is the extension to Servlet technology.
- ▶ We can use all the features of the Servlet in JSP, plus
- ▶ JSP provides additional functionalities (such as implicit objects, directives, third party tag libraries, expression language) of its own that makes development faster and easier

2. Giving flexibility to Designing

- ▶ Both JSP and Servlet can be used to design a web page
- ▶ Servlet used `out.println()` to generate static web content at runtime.
- ▶ Designing a web page with “`out.println()`” is **exponentially cumbersome**
- ▶ JSP **eases designing** by embedding dynamic java code inside static HTML code

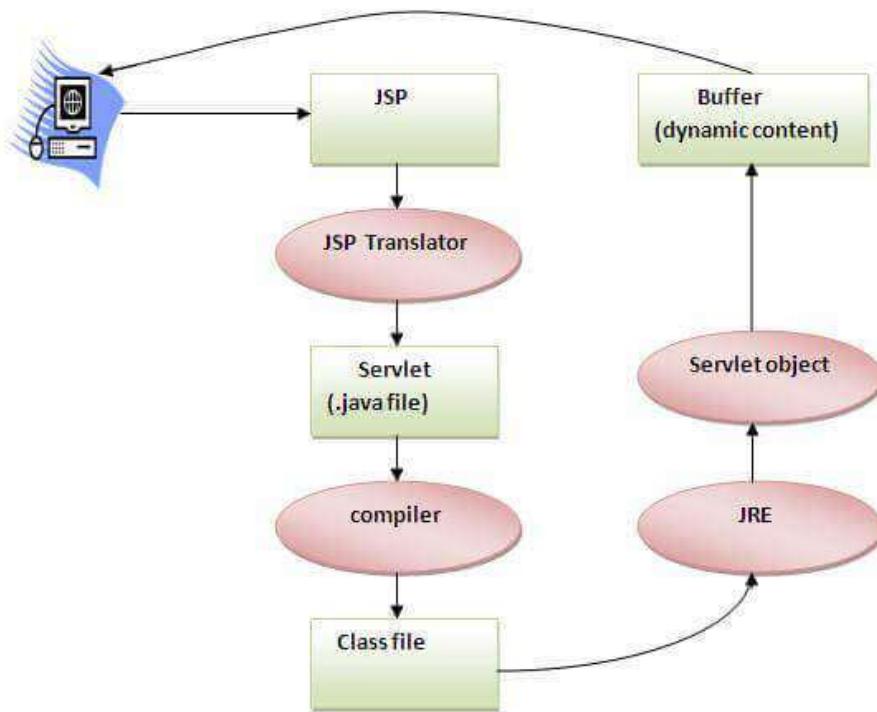
3. Easy to maintain

- ▶ JSP can be easily managed because we can easily separate our business logic with view logic.
- ▶ In Servlet technology, we mix our business logic with the view logic.

Lifecycle of JSP Page

The JSP pages follow these phases:

1. Translation of JSP Page
2. Compilation of JSP Page
3. Class loading (the class-loader loads class file)
4. Instantiation
5. Initialization
6. Request processing
7. Response Generation
8. Destroy



Translation

1. JSP page is translated into Servlet by the help of **JSP engine**, i.e. (XXX.jsp to XXX.java)
2. The JSP engine is a **part of the web server**

From here Servlet Life Cycle begins as discussed earlier in **Servlet Life Cycle**. Here they are for your quick revision:

Compilation

The translated Servlet page is then compiled by the **JSP Engine** into class file. (XXX.java to XXX.class)

You can find the .class files of .JSP's at the path:

[<eclipse-workspace> \ .metadata \ .plugins \ org.eclipse.wst.server.core \ tmp0 \ work \ Catalina \ localhost \ <project-context> \ org \ apache \ jsp]

Class Loading

The Servlet class is then loaded into the memory

Instantiation

Object of the Generated Servlet is instantiated

Initialization

The container invokes `_jspInit()` method

Thread Creation & Request Processing

The container invokes `_jspService()` method, we get request and response objects to get data and output the response

Destroy is called

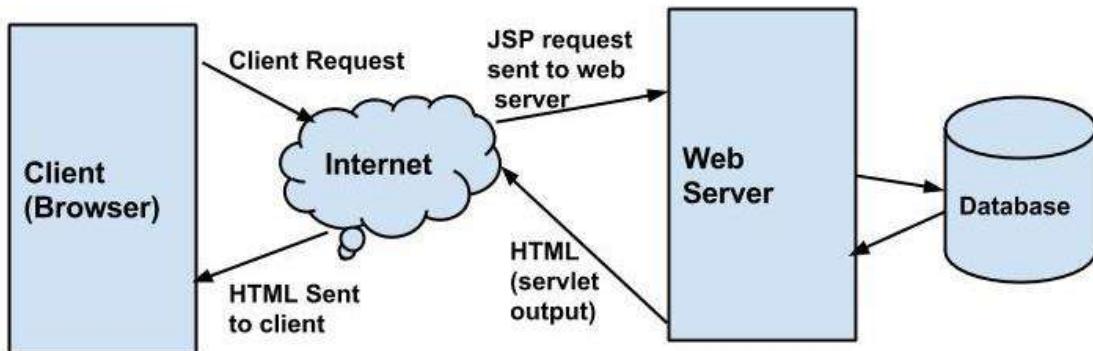
The container invokes `_jspDestroy()` method when Server is closed

Important Points

- ▶ The methods `_jspInit()`, `_jspDestroy()` and `_jspService()` of the translated Servlet corresponds to `init()`, `destroy()` and `service()` (`doGet()`, `doPost()`) of a regular servlet.
- ▶ Similar to servlet's `service()`, `_jspService()` takes two parameters, request and response, encapsulating the HTTP request and response messages.
- ▶ Similar to **PrintWriter** object, a JSP translated Servlet provides **JspWriter** object to write the response message over the network to the client.
- ▶ The HTML statements in the JSP script are written out as part of the response via `out.write(...)`, as it-is without modification.
- ▶ The Java code in the JSP scripts are translated according to their type of tag used.
 - The code written inside JSP **Declaration tag** is placed in translated Servlet class as it **data members** and **member methods**
 - The code written inside JSP **Scriptlet** is placed inside the `_jspService()` method of the translated servlet as "it is". Scriptlets form the program logic.
 - The code written inside JSP **Expression** is placed inside an "`out.print (...)`" inside `_jspService()`

JSP Architecture

Java Server Pages are part of a **3-tier architecture**. A server (generally referred to as application or web server) hosts and supports the Java Server Pages. This server will act as a mediator between the client browser and a database. The following diagram shows the **JSP architecture**.



JSP Architecture Flow

1. The user requests a JSP page from the client browser
2. The browser sends an HTTP request to the web server.
3. The web server recognizes that the HTTP request is for a JSP page and forwards it to a **JSP engine**.
4. The JSP engine **translates the JSP page** into an equivalent Servlet source code.
5. The JSP engine **compiles** the servlet into an executable class and forwards the original request to a **servlet engine**.
6. A part of the web server called the **servlet engine** loads the Servlet class and executes it.
7. During execution, the servlet produces an output in HTML format.
8. The output is then passed on to the web server by the servlet engine inside an HTTP response.
9. The web server forwards the HTTP response to the user's browser in terms of static HTML content.
10. Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

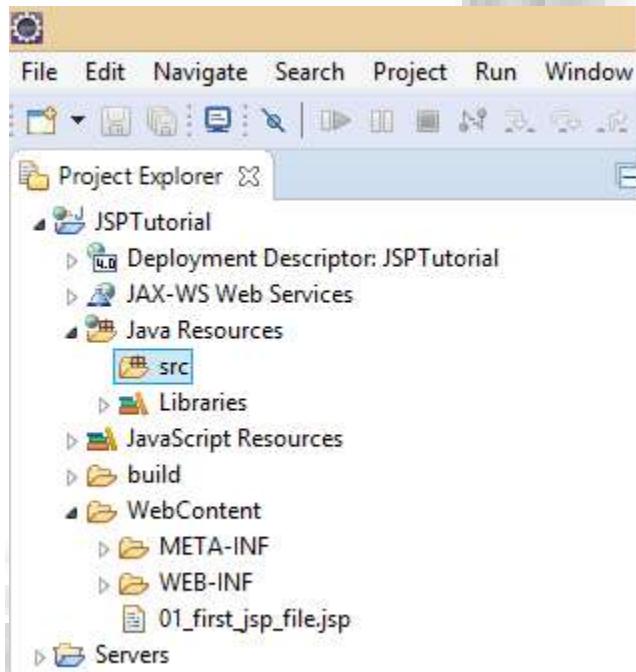
First JSP program in Eclipse

- Step 1. Start Eclipse
- Step 2. Create a new “Dynamic Web Project”
- Step 3. Create a JSP page inside the Web Content folder in the new project.
- Step 4. Write some JSP code in the <body></body> tag.
- Step 5. Run the file on server.
- Step 6. See the output on the browser.
- Step 7. Locate the translated servlet and class files at below path:

[<workspace-

path>\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\<project-name>\org\apache\jsp]

Directory structure of JSP



The JSP API

The JSP API consists of following package:

1. javax.servlet.jsp

Package javax.servlet.jsp

1. JspPage Interface
2. HttpJspPage class

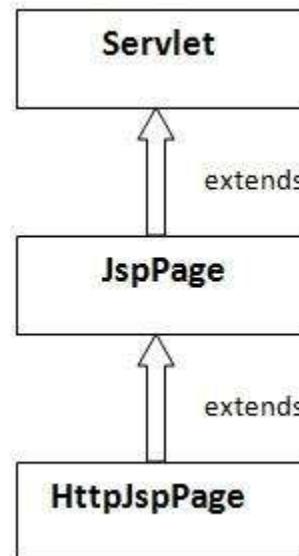
Classes

1. JspWriter
2. PageContext
3. JspFactory
4. JspEngineInfo
5. JspException
6. JspError

7. HttpJspBase

JSP Page (I)

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.



Methods of JspPage interface

1. **public void jspInit()**: It is invoked only once during the life cycle of the JSP when JSP page is requested for the first time. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy()**: It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some cleanup operations.

HttpJspPage (I)

The **HttpJspPage** interface provides one life cycle method of JSP. It extends the **JspPage** interface.

Method of HttpJspPage interface

1. **public void _jspService()**: It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.



JSP Scripting Elements

Table of Contents

JSP Scripting Elements	19
----------------------------------------	-------	----

<u>JSP Declaration Tag</u>	19
<u>Syntax</u>	19
<u>Example</u>	19
<u>JSP Scriptlet Tag</u>	20
<u>Syntax</u>	20
<u>Example</u>	20
<u>JSP Expression Tag</u>	22
<u>Syntax</u>	22
<u>Example</u>	22
<u>Similarities and Differences</u>	22

JSP Scripting Elements

In JSP, java code is written inside the JSP page using the scripting element (also called scripting tags).

These scripting elements provide the ability to insert java code inside a JSP page.

There are three types of scripting elements:

- ▶ Declaration tag
- ▶ Scriptlet tag
- ▶ Expression tag

JSP Declaration Tag

- ▶ The JSP declaration tag is used to declare **fields** and **methods**
- ▶ The implicit objects are **not** available in a declaration tag
- ▶ The content of declaration tag is member of generated Servlet. For ex:
 - The variables are converted to **instance variables**, and
 - The methods are converted to **member functions**

Syntax

```
<%!
    field1 declaration;
    .
    .
    .
    fieldN declarationl;
    method1 declaration;
    .
    .
    .
    methodN declaration;
%>
```

Example

```
<html>
<body>
<%!
```

```
int age = 50;  
public int returnAge(){  
    return age;  
}  
%>  
<%= "Value of the variable is : " + returnAge() %>  
</body>  
</html>
```

JSP Scriptlet Tag

- ▶ A Scriptlet tag is used to execute java source code in JSP
- ▶ We **can** declare variables in a Scriptlet tag
- ▶ We **cannot** declare and define methods in a Scriptlet tag
- ▶ The content of the Scriptlet tag goes inside **_jspService()** method of the converted Servlet
- ▶ All the implicit objects can be used inside a Scriptlet tag

Syntax

```
<%  
    java source code statement 1;  
    java source code statement 2;  
    .  
    .  
    java source code statement N;  
%>
```

Example

```
<html>  
<body>  
<%  
    String name = request.getParameter("name");  
  
    out.print("Welcome : " + name);  
%>  
</body>  
</html>
```



JSP Expression Tag

- ▶ The code placed within JSP expression tag is written to the output stream of the response.
- ▶ We use **out.print()** to write data, but in expression tag we can replace it with (=)
- ▶ It is mainly used to print the values of variable or method.
- ▶ **Only 1 statement** can be written in an expression tag
- ▶ We do not end the statement in an expression tag with semi-colon (;)
- ▶ We can't declare or define a variable/method inside expression tag.
- ▶ The content of the expression tag goes inside **_jspService()** method of the converted Servlet
- ▶ All the implicit objects are available inside expression tag

Syntax

```
<%= statement %>
```

Example

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Similarities and Differences

S. No	Property	Declaration Tag	Scriptlet Tag	Expression Tag
1.	Declare Variable	Yes	Yes	No
2.	Declare Method	Yes	No	No
3.	Servlet class members	Yes	No	No
4.	Inside _jspService()	No	Yes	Yes
5.	Implicit Objects	No	Yes	Yes
6.	Statement Terminator	Yes	Yes	No

Implicit Objects

Table of Contents

<u>JSP Implicit Objects</u>	24
<u>1. out</u>	24
<u>2. request</u>	24
<u>3. response</u>	25
<u>4. config</u>	25
<u>5. application</u>	25
<u>6. session</u>	25
<u>7. pageContext</u>	25
<u>8. page</u>	26
<u>Difference between page and pageContext</u>	26
<u>9. exception</u>	27

JSP Implicit Objects

In order to give programmer flexibility, JSP provides **some implicit objects**.

- ▶ These objects can be directly used in **Scriptlet & Expression** tag that goes in the `_jspService` method.
- ▶ They are created by the **container** automatically, and they are readily available for use.

JSP provides **9 implicit objects** for our disposal:

1. `out`
2. `request`
3. `response`
4. `config`
5. `application`
6. `session`
7. `pageContext`
8. `page`
9. `exception`

1. `out`

- ▶ JSP implicit object `out` is an instance of `javax.servlet.jsp.JspWriter`
- ▶ It is used to write the data to the buffer and send output to the client in response
- ▶ `Out` object allows us to access the servlet's output stream
- ▶ It is one of the most used JSP implicit object, and
- ▶ That is why we have JSP Expression to easily invoke `out.print()` method.
- ▶ Ex: `<%= new Date()%>` converts to `out.print(new Date());`

2. `request`

- ▶ JSP implicit object `request` is an instance of `javax.servlet.http.HttpServletRequest`
- ▶ It is one of the argument of `_jspService()` method.
- ▶ It is created by container for every request.
- ▶ We can use `request` object to get the request parameters, cookies, request attributes, session, header information and other details about client request.
- ▶ Ex: `<%= request.getHeader("User-Agent") %>`

3. response

- ▶ JSP implicit object response is an instance of `javax.servlet.http.HttpServletResponse`
- ▶ It is one of the arguments of `_jspService()` method.
- ▶ Response object will be created by the container for each request.
- ▶ We can use response object to set content type, character encoding, header information in response, adding cookies to response and redirecting the request to other resource
- ▶ Ex: `<% response.addCookie(new Cookie("Test","Value")); %>`

4. config

- ▶ JSP implicit object config is an instance of `javax.servlet.ServletConfig`
- ▶ It is created by the container for each JSP page
- ▶ Ex: `<%= config.getServletContext() %>`

5. application

- ▶ JSP implicit object application is instance of `javax.servlet.ServletContext`
- ▶ It is created by container **one per application**, when the application gets deployed.
- ▶ It is used to get the context information and attributes in JSP.
- ▶ It contains a set of methods which are used to interact with the servlet container
- ▶ Ex: `<%= application.getAttribute(name) %>`

6. session

- ▶ JSP session implicit object is instance of `javax.servlet.http.HttpSession`
- ▶ Whenever we request a JSP page, container automatically creates a session for the JSP in the service method.
- ▶ Ex: `<%=session.getId() %>`

7. pageContext

- ▶ JSP pageContext implicit object is an instance of `javax.servlet.jsp.PageContext`
- ▶ pageContext object also hold reference to other implicit object.
- ▶ It can be used to set, get or remove attributes from any of the following scopes –
 - JSP Page – Scope: PAGE_SCOPE
 - HTTP Request – Scope: REQUEST_SCOPE
 - HTTP Session – Scope: SESSION_SCOPE

- Application Level – Scope: APPLICATION_SCOPE
- ▶ It has 4 main methods
 - **Object findAttribute (String attributeName):** This method searches for the specified attribute in all four levels in the following order – Page, Request, Session and Application. It returns NULL when no attribute found at any of the level.
 - **Object getAttribute (String attributeName, int scope):** It looks for an attribute in the specified scope. This method is similar to findAttribute method; the only difference is that findAttribute looks in all the four levels in a sequential order while getAttribute looks in a specified scope.
 - **void removeAttribute(String attributeName, int scope):** This method is used to remove an attribute from a given scope
 - **void setAttribute(String attributeName, Object attributeValue, int scope):** It writes an attribute in a given scope.

- ▶ Ex:

```
In 1st page:  
<% pageContext.setAttribute("Test", "Test Value"); %>  
  
In 2nd Page  
<%= pageContext.getAttribute("Test") %>
```

8. page

- ▶ JSP implicit object **page** is an instance of `java.lang.Object` class
- ▶ It can be thought of as an object that represents the entire current JSP page
- ▶ Or it can be thought of as a reference to the current Servlet instance (Translated from JSP).
- ▶ This object is assigned to the reference of auto generated servlet class.
- ▶ This object is an actual reference to the instance of the page. (`Object page = this;`)
- ▶ Ex: `<%= page.getClass().getName() %>`

Difference between page and pageContext

- ▶ The **page** object represents the generated servlet instance itself and is used as a scope with in one jsp.
- ▶ The **pageContext** is used to initialize all implicit objects for example :- **page** attributes, access to the request, response and session objects, as well as the JspWriter referenced by out.

9. exception

- ▶ JSP exception implicit object is instance of **java.lang.Throwable** class
- ▶ It is used to provide exception details in **JSP error pages**.
- ▶ We can't use this object in normal JSP pages and it's available only in JSP error pages



Implicit Objects

Table of Contents

<u>JSP Implicit Objects</u>	29
<u>1. out</u>	29
<u>2. request</u>	29
<u>3. response</u>	30
<u>4. config</u>	30
<u>5. application</u>	30
<u>6. session</u>	30
<u>7. pageContext</u>	30
<u>8. page</u>	31
<u>Difference between page and pageContext</u>	32
<u>9. exception</u>	32

JSP Implicit Objects

In order to give programmer flexibility, JSP provides **some implicit objects**.

- ▶ These implicit objects are created during the **translation phase** of JSP to the servlet.
- ▶ These objects can be directly used in **Scriptlet & Expression** tag that goes in the `_jspService` method.
- ▶ They are created by the **container** automatically, and they are readily available for use.

JSP provides **9 implicit objects** for our disposal:

10. `out`
11. `request`
12. `response`
13. `config`
14. `application`
15. `session`
16. `pageContext`
17. `page`
18. `exception`

1. `out`

- ▶ JSP implicit object `out` is an instance of `javax.servlet.jsp.JspWriter`
- ▶ It is used to write the data to the buffer and send output to the client in response
- ▶ `Out` object allows us to access the servlet's output stream
- ▶ It is one of the most used JSP implicit object, and
- ▶ That is why we have JSP Expression to easily invoke `out.print()` method.
- ▶ Ex: `<%= new Date()%>` converts to `out.print(new Date());`

2. `request`

- ▶ JSP implicit object `request` is an instance of `javax.servlet.http.HttpServletRequest`
- ▶ It is one of the argument of `_jspService()` method.
- ▶ It is created by container for every request.
- ▶ We can use `request` object to get the request parameters, cookies, request attributes, session, header information and other details about client request.

- ▶ Ex: `<%= request.getParameter("username") %>`

3. response

- ▶ JSP implicit object response is an instance of `javax.servlet.http.HttpServletResponse`
- ▶ It is one of the arguments of `_jspService()` method.
- ▶ Response object will be created by the container for each request.
- ▶ We can use response object to set content type, character encoding, header information in response, adding cookies to response and redirecting the request to other resource
- ▶ Ex: `<% response.addCookie(new Cookie("Test","Value")); %>`

4. config

- ▶ JSP implicit object config is an instance of `javax.servlet.ServletConfig`
- ▶ It is created by the container for each JSP page
- ▶ It is used to get the JSP initialization parameters configured in deployment descriptor `web.xml`.
- ▶ Ex: `<%= config.getInitParameter("Course") %>`

5. application

- ▶ JSP implicit object application is instance of `javax.servlet.ServletContext`
- ▶ It is created by container **one per application**, when the application gets deployed.
- ▶ It is used to get the context information and attributes in JSP.
- ▶ We can use it to get the RequestDispatcher object in JSP to forward the request to another resource or to include the response from another resource in the JSP
- ▶ It contains a set of methods which are used to interact with the servlet container
- ▶ Ex: `<%= application.getInitParameter("User") %>`

6. session

- ▶ JSP session implicit object is instance of `javax.servlet.http.HttpSession`
- ▶ Whenever we request a JSP page, container automatically creates a session for the JSP in the service method.
- ▶ Ex: `<%=session.getId() %>`

7. pageContext

- ▶ JSP pageContext implicit object is an instance of `javax.servlet.jsp.PageContext`

- ▶ pageContext object also hold reference to other implicit object.
- ▶ It can be used to set, get or remove attributes from any of the following scopes –
 - JSP Page – Scope: PAGE_SCOPE
 - HTTP Request – Scope: REQUEST_SCOPE
 - HTTP Session – Scope: SESSION_SCOPE
 - Application Level – Scope: APPLICATION_SCOPE
- ▶ It has 4 main methods
 - **Object findAttribute (String attributeName):** This method searches for the specified attribute in all four levels in the following order – Page, Request, Session and Application. It returns NULL when no attribute found at any of the level.
 - **Object getAttribute (String attributeName, int scope):** It looks for an attribute in the specified scope. This method is similar to findAttribute method; the only difference is that findAttribute looks in all the four levels in a sequential order while getAttribute looks in a specified scope.
 - **void removeAttribute(String attributeName, int scope):** This method is used to remove an attribute from a given scope
 - **void setAttribute(String attributeName, Object attributeValue, int scope):** It writes an attribute in a given scope.

- ▶ Ex:

```
In 1st page:
```

```
<% pageContext.setAttribute("Test", "Test Value"); %>
```

```
In 2nd Page
```

```
<%= pageContext.getAttribute("Test") %>
```

8. page

- ▶ JSP implicit object **page** is an instance of **java.lang.Object** class
- ▶ It can be thought of as an object that represents the entire current JSP page
- ▶ Or it can be thought of as a reference to the current Servlet instance (Translated from JSP).
- ▶ This object is assigned to the reference of auto generated servlet class.
- ▶ This object is an actual reference to the instance of the page. (**Object page = this;**)
- ▶ Ex: `<%= page.getClass().getName() %>`

Difference between page and pageContext

- ▶ The **page** object represents the generated servlet instance itself and is used as a scope with in one jsp.
- ▶ The **pageContext** is used to initialize all implicit objects for example :- **page** attributes, access to the request, response and session objects, as well as the JspWriter referenced by out.

9. exception

- ▶ JSP exception implicit object is instance of **java.lang.Throwable** class
- ▶ It is used to provide exception details in **JSP error pages**.
- ▶ We can't use this object in normal JSP pages and it's available only in JSP error pages

JSP Scopes

Table of Contents

<u>JSP Scopes</u>	34
<u>Types of Scopes in JSP</u>	34
<u>Page Scope</u>	34
<u>Request Scope</u>	34
<u>Session Scope</u>	34
<u>Application Scope</u>	35
<u>Parameters and Attributes</u>	35
<u>Parameters</u>	35
<u>Servlet API methods to access parameters</u>	35
<u>Attributes</u>	36
<u>Servlet API methods to manipulate attributes</u>	36
<u>Differences between Parameters and Attributes</u>	36

JSP Scopes

- ▶ Every program relies heavily on **variables**.
- ▶ In Java till now we have learnt about **instance**, **static** and **local** variables.
- ▶ We also know that each variable has a **scope** that decides its accessibility.
- ▶ Similarly in JSP too we need variables to carry and manipulate **data**.
- ▶ JSP provides the **mechanism of scope** for its created objects.
- ▶ The availability of a JSP object for use from a particular place of the application is defined as the scope of that JSP object.
- ▶ Every object created in a JSP page will have a scope

Types of Scopes in JSP

Object scope in JSP is segregated into four parts and they are:

1. Page Scope
2. Request Scope
3. Session Scope
4. Application Scope

Page Scope

- ▶ Page scope means, the JSP object can be accessed only from within the same page where it was created.
- ▶ JSP implicit objects **out**, **response**, **config**, **page**, **pageContext** and **exception** have 'page' scope.

Request Scope

- ▶ A JSP object created using the 'request' scope can be accessed from all the pages that serve that request.
- ▶ More than one page can serve a single request.
- ▶ The JSP object will be bound to the request object.
- ▶ Implicit object **request** has the 'request' scope.

Session Scope

- ▶ Session scope means, the JSP object is accessible from pages that belong to the same session from where it was created.

- ▶ The JSP object that is created using the session scope is bound to the session object.
- ▶ Implicit object session has the 'session' scope.

Application Scope

- ▶ A JSP object created using the 'application' scope can be accessed from any pages across the application.
- ▶ The JSP object is bound to the application object.
- ▶ Implicit object application has the 'application' scope.

Note: We will practice these scopes while learning about **pageContext** implicit object

Parameters and Attributes

Parameters

- ▶ Parameters may come into our application
 - From a client request, or
 - May be configured through deployment descriptor (web.xml) elements or
 - Their corresponding annotations.
- ▶ When we submit a form, form values are sent as request parameters to a web application.
 - In case of a **GET** request, these parameters are exposed in the URL as name value pairs and
 - In case of **POST**, parameters are sent within the body of the request.
- ▶ Servlet init parameters and context init parameters are set through the deployment descriptor (web.xml) or their corresponding annotations.
- ▶ All parameters except application context are **read-only** from the application code.
- ▶ We have methods in the Servlet API to retrieve various parameters.

Servlet API methods to access parameters

S. No	Object	Method
1	ServletRequest	String[] getParameterValues (String paramName)
2	ServletRequest	String getParameter (String parmName)
3	ServletRequest	Enumeration<String> getParameterNames ()
4	ServletRequest	Map <String, String[]>getParameterMap ()
5	ServletConfig	Enumeration<String> getInitParameterNames ()
6	ServletConfig	String getInitParameter (String paramName)

7	ServletContext	Enumeration<String> getInitParameterNames ()
8	ServletContext	String getInitParameter (String paramName)

Attributes

- ▶ **Attributes** are objects that are attached to various scopes and can be modified, retrieved or removed.
- ▶ They can be read, created, updated and deleted by the web container as well as our application code.
- ▶ When an object is added to an attribute in any scope, it is called **binding** as the object is bound into a scoped attribute with a given name.
- ▶ We have methods in the Servlet API to **add, modify, retrieve and remove** attributes.

Servlet API methods to manipulate attributes

S. No	Method
1	public void setAttribute(String name, Object value)
2	public Object getAttribute(String name)
3	public Enumeration<String> getAttributeNames()
4	public void removeAttribute(String name)

Differences between Parameters and Attributes

1. Parameters are **read-only**; attributes are **read/write** objects.
2. Parameters are **String objects**, attributes can be objects of **any type**.

JSP Page Directives

Table of Contents

<u>JSP Directives</u>	38
<u>What are JSP Directives?</u>	38
<u>Syntax of a Directive:</u>	38
<u>Syntax 1: Directive with a single attribute</u>	38
<u>Syntax 2: Directive with multiple attributes</u>	38
<u>Types of Directives</u>	38
<u>Page Directive</u>	38
<u>Syntax of Page directive</u>	38
<u>Attributes of Page Directive</u>	39
1. <u>Attribute: language</u>	39
2. <u>Attribute: contentType</u>	39
3. <u>Attribute: pageEncoding</u>	39
3. <u>Attribute: info</u>	40
4. <u>Attribute: import</u>	40
5. <u>Attribute: errorPage</u>	40
6. <u>Attribute: isErrorPage</u>	41
7. <u>Attribute: isELIgnored</u>	41
8. <u>Attribute: session</u>	41
<u>Include Directive</u>	42
<u>Syntax</u>	42
<u>Example</u>	42

JSP Directives

What are JSP Directives?

- ▶ JSP directives are the elements of a JSP source code that guide the web container on how to translate the JSP page into its respective servlet.
- ▶ They provide global information about an entire JSP page

Syntax of a Directive:

- ▶ In JSP, directives are described in `<%@ ... %>` tags
- ▶ Directives can have many space separated attributes as key-value pairs.

Syntax 1: Directive with a single attribute

```
<%@ directive attribute="" %>
```

Syntax 2: Directive with multiple attributes

```
<%@ directive attribute1 = " " attribute2 = " " attribute3 = " " %>
```

Types of Directives

There are three types of directives:

1. Page directive
2. Include directive
3. Taglib directive

Each one of them is described in detail below with examples.

Page Directive

- ▶ It provides attributes that get applied to entire JSP page.
- ▶ It defines page dependent attributes, such as imported classes, scripting language, error page etc.
- ▶ It is used to provide instructions to the container for current JSP page.

Syntax of Page directive

```
<%@page attribute1 = "" attribute2 = ""%>
```

Attributes of Page Directive

Following is the list of attributes associated with page directive:

1. language
2. contentType
3. pageEncoding
4. info
5. import
6. errorPage
7. isErrorPage
8. isELIgnored
9. session

More details about each attribute are given below

1. Attribute: language

It specifies the scripting language (underlying language) being used in the page.

Syntax:

```
<%@ page language="value" %>
```

Here in our case value will be **java** as it is the underlying programming language. The code in the tags would be compiled using java compiler.

2. Attribute: contentType

You can use contentType to set

1. The character encoding (charset) of the page source (during translation)
2. The character encoding (charset) of the response (during runtime)

Syntax:

```
<%@ page contentType="charset=character_set" %>
```

- ▶ The default value is "text/html; charset=ISO-8859-1"

3. Attribute: pageEncoding

- ▶ You can use pageEncoding to **set** the **character encoding** of the **page source**.
- ▶ Its main purpose is to specify a different page source character encoding than that of the response.
- ▶ The default pageEncoding is specified as "ISO-8859-1".

Syntax:

```
<%@ page pageEncoding = "character_set" %>
```

3. Attribute: info

It provides a description to a JSP page which can be accessed by getServletInfo() method.

Syntax:

```
<%@ page info="Directive Training JSP" %>
```

This attribute is used to set the servlet description.

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1
"pageEncoding="value" info="Directive Training JSP" %>
```

4. Attribute: import

- ▶ This attribute is most used attribute in page directive attributes.
- ▶ It is used to tell the container to import other java classes, interfaces, enums, etc. in the code
- ▶ It is similar to import statements in java.

Syntax:

```
<%@ page import="value1, value2, value3" %>
```

Here value indicates the classes which have to be imported.

All classes of the following **four** packages are imported by default into JSP pages:

1. java.lang
2. javax.servlet
3. javax.servlet.http
4. javax.servlet.jsp

No import statement is required in JSP files for classes in the above packages.

Example:

```
<%@ page import="java.util.Random" %>
<%@ page import="java.util.Random, java.util.ArrayList, java.io.*" %>
```

5. Attribute: errorPage

This attribute is used to set the error page for the JSP page if JSP throws an exception and then it redirects to the exception page.

Syntax:

```
<%@ page errorPage = "value" %>
```

Here value represents the error JSP page value

Example:

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

In the above code, to handle exceptions we have errorHandler.jsp

6. Attribute: isErrorPage

- ▶ It indicates that JSP Page that has an errorPage will be checked in another JSP page
- ▶ Any JSP file declared with "isErrorPage" attribute is then capable to receive exceptions from other JSP pages which have error pages.
- ▶ Exceptions are available to these pages only.
- ▶ The default value is false

Syntax:

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

Here in this case session attribute can be set to true or false

7. Attribute: isELIgnored

- ▶ EL stands for **Expression Language**
- ▶ IsELIgnored is a flag attribute where we have to decide whether to ignore EL tags or not.
- ▶ Its datatype is java enum, and the default value is false hence EL is **enabled by default**.
- ▶ We can activate Expression language in the page by setting it to false
- ▶ We can de-activate Expression language in the page by setting it to true

Syntax:

```
<%@ page isELIgnored = "true/false" %>
```

Here, true/false represents the value of EL whether it should be ignored or not.

8. Attribute: session

- ▶ Every time a JSP is requested, JSP creates an HttpSession object to maintain state

- ▶ This session data is accessible in the JSP as the implicit **session** object.
- ▶ In JSPs, sessions are enabled by default. By default `<%@ page session="true" %>`
- ▶ Sometimes when we don't need a session to be created in JSP, we can set this attribute to **false**.
- ▶ When it is set to false, it indicates to the compiler to **not create** the session by default.
- ▶ Disabling the session in some pages will **improve the performance** of your JSP container.
- ▶ Session object **uses** the server **resources**.
- ▶ Each session object uses up a small amount of system resources as it is stored on the server side.
- ▶ This also **increases** the **traffic** as the session ID is sent from server to client.
- ▶ Client also sends the same session ID along with each request.
- ▶ If some of the JSP pages on your web site are getting millions of hits from internet browser and there is no need to identify the user, it's **better to disable** the session in that JSP page.
- ▶ No session object will be created in translated servlet

Syntax:

```
<%@ page session = "true/false" %>
```

Here in this case session attribute can be set to true or false

Note: Learn more about JSP tuning [here](#) and extends keyword [here](#)

Include Directive

- ▶ JSP "include directive" is used to include one file to the another file
- ▶ This included file can be HTML, JSP, text files, etc.
- ▶ It is also useful in creating templates with the user views
- ▶ It helps to break the pages into header, footer and sidebar actions.
- ▶ It includes file during translation phase

Syntax

```
<%@ include file = "<filename>" %>
```

Here <filename> represents the name of the file to be included

Example

File to be included: header.jsp

```
<h1>This is header</h1>
```

File where header.jsp is to be included: index.jsp

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<%@include file="header.jsp" %>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

JSP Taglib Directive & JSTL

Table of Contents

<u>JSP Taglib Directive</u>	44
<u>Prefix</u>	44
<u>URI</u>	44
<u>JSTL</u>	45
<u>Advantage of JSTL</u>	45
<u>JSTL Tag Libraries</u>	45
<u>Configure JSTL jar file</u>	45
<u>JSTL Core Tags</u>	47
<u>JSTL Functions</u>	48
<u>JSTL Formatting</u>	50

JSP Taglib Directive

The Taglib directive is used to define the tag libraries that the current JSP page uses. A JSP page might include several tag libraries. One such popular tag library is **JSTL**, i.e. JSP Standard Tag Library

Syntax:

```
<%@ taglib prefix = "prefixOfTag" uri = "uriOfTagLibrary" %>
```

Example:

Defining prefix as c when using core part of JSTL

```
<%@ taglib prefix = "c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Using JSTL in code

```
<c:set var = "salary" value = "${2000}" />
<p>My salary is: <c:out value = "${salary}" /></p>
```

Prefix

- ▶ The prefix is used to distinguish the custom tag from other library custom tags used in a JSP page.
- ▶ Prefix is prepended to the custom tag name. Every custom tag must have a prefix.
- ▶ In the example See how prefix: “**c**” is pre-pended to the custom tag name.
- ▶ If there were 2 libraries used in the same page, then this prefix helps the programmer to distinguish between the different tag libraries.
- ▶ Hence for each library used, the prefix must be unique

URI

- ▶ Every tag library has a **Tag Library Descriptor (TLD)**.
- ▶ It is an **XML document** that contains information about the library as a whole and about each tag contained in the library.
- ▶ The extension of a TLD file is **.tld**
- ▶ When we want to use any tag library (here JSTL) in our JSP page, we have to provide the URI of this tag library in our “**uri**” attribute.
- ▶ This URI is a **unique** identifier defined in the JSTL Tag Library Descriptor (TLD)
- ▶ Following is a list of Tag Libraries and their URI:

1. Core Tags - URI → <http://java.sun.com/jsp/jstl/core>
2. Formatting Tags - URI → <http://java.sun.com/jsp/jstl/fmt>
3. SQL Tags - URI → <http://java.sun.com/jsp/jstl/sql>
4. XML Tags - URI → <http://java.sun.com/jsp/jstl/xml>
5. JSTL Functions - URI → <http://java.sun.com/jsp/jstl/functions>

JSTL

- ▶ JSTL stands for JSP Standard Tag Library
- ▶ The JSP Standard Tag Library (JSTL) is a collection of predefined tags to simplify the JSP development

Advantage of JSTL

1. **Fast Development:** JSTL provides many tags that simplify the JSP.
2. **Code Reusability:** We can use the JSTL tags on various pages.
3. **No need to use Scriptlet tag:** It avoids the use of Scriptlet tag.

JSTL Tag Libraries

JSTL mainly provides five types of tags:

1. Core Tags

The JSTL core tags provide variable support, URL management, flow control, etc. The URL for the core tag is **http://java.sun.com/jsp/jstl/core**. The prefix of core tag is **c**.

2. Function Tags

The functions tags provide support for string manipulation and string length. The URL for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**.

3. Formatting Tags

The Formatting tags provide support for message formatting, number and date formatting, etc.

The URL for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**.

4. XML Tags

The XML tags provide flow control, transformation, etc. The URL for the XML tags is **http://java.sun.com/jsp/jstl/xml** and prefix is **x**.

5. SQL Tags

The JSTL SQL tags provide SQL support. The URL for the SQL tags is

http://java.sun.com/jsp/jstl/sql and prefix is **sql**.

Configure JSTL jar file

1. Go to: <http://tomcat.apache.org/download-taglibs.cgi>
2. Download:
 - a. Impl: [taglibs-standard-impl-1.2.5.jar](#)
 - b. Spec: [taglibs-standard-spec-1.2.5.jar](#)
3. Put the .jar files in **WEB-INF/lib** folder

4. Add these to build Path: Right click on .jar file and add it to build path



JSTL Core Tags

The core tags in JSTL defines core functionalities in java such as variable assignment, conditional statements, loop statements, printing statements etc.

Include the library at the top of the jsp pages with the syntax:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Below is the list **JSTL core tags** with their explanations

S. No.	Tags	Description
1.	<c:out>	It is used for displaying the content on client after escaping XML and HTML markup tags. Main attributes are default and escapeXML.
2.	<c:set>	This tag is useful for setting up a variable value in a specified scope. It basically evaluates an expression and sets the result in given variable.
3.	<c:remove>	It is used for removing an attribute from a specified scope or from all scopes (page, request, session and application). By default removes from all.
4.	<c:if>	This JSTL core tag is used for testing conditions. There are two other optional attributes for this tag, which are var and scope, test is mandatory.
5.	<c:choose>	It's like switch statement in Java.
6.	<c:when>	It's like case statement in Java.
7.	<c:otherwise>	It works like default attribute in switch-case statements.
8.	<c:catch>	This tag is used in exception handling. In this post we have discussed exception handling using <c:catch> core tag.
9.	<c:import>	This JSTL core tag is used for importing the content from another file/page to the current JSP page. Attributes – var, URL and scope.
10.	<c:forEach>	This tag in JSTL is used for executing the same set of statements for a finite number of times.
11.	<c:forTokens>	It is used for iteration but it only works with delimiter.
12.	<c:param>	This JSTL tag is mostly used with <c:url> and <c:redirect> tags. It adds parameter and their values to the output of these tags

13.	<c:url>	It is used for url formatting or url encoding. It converts a relative url into a application context's url. Optional attributes var, context and scope.
14.	<c:redirect>	It is used for redirecting the current page to another URL, provide the relative address in the URL attribute of this tag and the page will be redirected to the url.

JSTL Functions

The function tags in JSTL defines the functions in java that are used to manipulate Strings.

Include the library at the top of the jsp pages with the syntax:

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

Below is the list **JSTL functional tags** with their explanations

S. No.	Tags	Description
1.	fn:contains()	This function checks whether the given string is present in the input as sub-string. It does a case sensitive check.
2.	fn:containsIgnoreCase()	It does a case insensitive check to see whether the provided string is a sub-string of input.
3.	fn:indexOf()	It is used for finding out the start position of a string in the provided string. Function returns -1 when string is not found in the input.
4.	fn:escapeXML()	It is used for HTML/XML character escaping which means it treats html/xml tags as a string. Similar to the escapeXml attribute of <c:out> tag.
5.	fn:join()	It concatenates the strings with a given separator and returns the output string.
6.	fn:split()	It splits a given string into an array of substrings.
7.	fn:length()	It is used for computing the length of a string or to find out the number of elements in a collection. It returns the length of the object.
8.	fn:startsWith()	It checks the specified string is a prefix of given string.
9.	fn:endsWith()	It is used for checking the suffix of a string. It checks whether the given string ends with a particular string.
10.	fn:substring()	This JSTL function is used for getting a substring from the provided string.
11.	fn:substringAfter()	It is used for getting a substring which is present in the input string before a specified string.
12.	fn:substringBefore()	It gets a substring from input which comes after a specified string.
13.	fn:trim()	It removes spaces from beginning and end of a string and function.

14.	fn:toUpperCase()	It is just opposite of fn:toLowerCase() function. It converts input string to a uppercase string.
15.	fn:toLowerCase()	This function is used for converting an input string to a lower case string.
16.	fn:replace()	It searches for a string in the input and replace it with the provided string. It does case sensitive processing.



JSTL Formatting

The formatting tags provide support for message formatting, number and date formatting etc.

Include the library at the top of the jsp pages with the syntax:

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Below is the list **JSTL functional tags** with their explanations

S. No.	Tags	Description
1.	fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number.
2.	fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting.
3.	fmt:formatNumber	It is used to format the numerical value with specific format or precision.
4.	fmt:parseDate	It parses the string representation of a time and date.
5.	fmt:bundle	It is used for creating the ResourceBundle objects which will be used by their tag body.
6.	fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
7.	fmt:setBundle	It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
8.	fmt:message	It display an internationalized message.
9.	fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.

JSP Taglib Directive & JSTL

Table of Contents

<u>JSP Taglib Directive</u>	44
<u>Prefix</u>	44
<u>URI</u>	44
<u>JSTL</u>	45
<u>Advantage of JSTL</u>	45
<u>JSTL Tag Libraries</u>	45
<u>Configure JSTL jar file</u>	45
<u>JSTL Core Tags</u>	47
<u>JSTL Functions</u>	48
<u>JSTL Formatting</u>	50

JSP Taglib Directive

The Taglib directive is used to define the tag libraries that the current JSP page uses. A JSP page might include several tag libraries. One such popular tag library is **JSTL**, i.e. JSP Standard Tag Library

Syntax:

```
<%@ taglib prefix = "prefixOfTag" uri = "uriOfTagLibrary" %>
```

Example:

Defining prefix as c when using core part of JSTL

```
<%@ taglib prefix = "c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Using JSTL in code

```
<c:set var = "salary" value = "${2000}" />
<p>My salary is: <c:out value = "${salary}" /></p>
```

Prefix

- ▶ The prefix is used to distinguish the custom tag from other library custom tags used in a JSP page.
- ▶ Prefix is prepended to the custom tag name. Every custom tag must have a prefix.
- ▶ In the example See how prefix: “**c**” is pre-pended to the custom tag name.
- ▶ If there were 2 libraries used in the same page, then this prefix helps the programmer to distinguish between the different tag libraries.
- ▶ Hence for each library used, the prefix must be unique

URI

- ▶ Every tag library has a **Tag Library Descriptor (TLD)**.
- ▶ It is an **XML document** that contains information about the library as a whole and about each tag contained in the library.
- ▶ The extension of a TLD file is **.tld**
- ▶ When we want to use any tag library (here JSTL) in our JSP page, we have to provide the URI of this tag library in our “**uri**” attribute.
- ▶ This URI is a **unique** identifier defined in the JSTL Tag Library Descriptor (TLD)
- ▶ Following is a list of Tag Libraries and their URI:

1. Core Tags - URI → <http://java.sun.com/jsp/jstl/core>
2. Formatting Tags - URI → <http://java.sun.com/jsp/jstl/fmt>
3. SQL Tags - URI → <http://java.sun.com/jsp/jstl/sql>
4. XML Tags - URI → <http://java.sun.com/jsp/jstl/xml>
5. JSTL Functions - URI → <http://java.sun.com/jsp/jstl/functions>

JSTL

- ▶ JSTL stands for JSP Standard Tag Library
- ▶ The JSP Standard Tag Library (JSTL) is a collection of predefined tags to simplify the JSP development

Advantage of JSTL

4. **Fast Development:** JSTL provides many tags that simplify the JSP.
5. **Code Reusability:** We can use the JSTL tags on various pages.
6. **No need to use Scriptlet tag:** It avoids the use of Scriptlet tag.

JSTL Tag Libraries

JSTL mainly provides five types of tags:

6. Core Tags
The JSTL core tags provide variable support, URL management, flow control, etc. The URL for the core tag is <http://java.sun.com/jsp/jstl/core>. The prefix of core tag is **c**.
7. Function Tags
The functions tags provide support for string manipulation and string length. The URL for the functions tags is <http://java.sun.com/jsp/jstl/functions> and prefix is **fn**.
8. Formatting Tags
The Formatting tags provide support for message formatting, number and date formatting, etc. The URL for the Formatting tags is <http://java.sun.com/jsp/jstl/fmt> and prefix is **fmt**.
9. XML Tags
The XML tags provide flow control, transformation, etc. The URL for the XML tags is <http://java.sun.com/jsp/jstl/xml> and prefix is **x**.
10. SQL Tags
The JSTL SQL tags provide SQL support. The URL for the SQL tags is <http://java.sun.com/jsp/jstl/sql> and prefix is **sql**.

Configure JSTL jar file

5. Go to: <http://tomcat.apache.org/download-taglibs.cgi>
6. Download:
 - c. Impl: [taglibs-standard-impl-1.2.5.jar](#)
 - d. Spec: [taglibs-standard-spec-1.2.5.jar](#)
7. Put the .jar files in **WEB-INF/lib** folder

8. Add these to build Path: Right click on .jar file and add it to build path



JSTL Core Tags

The core tags in JSTL defines core functionalities in java such as variable assignment, conditional statements, loop statements, printing statements etc.

Include the library at the top of the jsp pages with the syntax:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Below is the list **JSTL core tags** with their explanations

S. No.	Tags	Description
1.	<c:out>	It is used for displaying the content on client after escaping XML and HTML markup tags. Main attributes are default and escapeXML.
2.	<c:set>	This tag is useful for setting up a variable value in a specified scope. It basically evaluates an expression and sets the result in given variable.
3.	<c:remove>	It is used for removing an attribute from a specified scope or from all scopes (page, request, session and application). By default removes from all.
4.	<c:if>	This JSTL core tag is used for testing conditions. There are two other optional attributes for this tag, which are var and scope, test is mandatory.
5.	<c:choose>	It's like switch statement in Java.
6.	<c:when>	It's like case statement in Java.
7.	<c:otherwise>	It works like default attribute in switch-case statements.
8.	<c:catch>	This tag is used in exception handling. In this post we have discussed exception handling using <c:catch> core tag.
9.	<c:import>	This JSTL core tag is used for importing the content from another file/page to the current JSP page. Attributes – var, URL and scope.
10.	<c:forEach>	This tag in JSTL is used for executing the same set of statements for a finite number of times.
11.	<c:forTokens>	It is used for iteration but it only works with delimiter.
12.	<c:param>	This JSTL tag is mostly used with <c:url> and <c:redirect> tags. It adds parameter and their values to the output of these tags

13.	<c:url>	It is used for url formatting or url encoding. It converts a relative url into a application context's url. Optional attributes var, context and scope.
14.	<c:redirect>	It is used for redirecting the current page to another URL, provide the relative address in the URL attribute of this tag and the page will be redirected to the url.

JSTL Functions

The function tags in JSTL defines the functions in java that are used to manipulate Strings.

Include the library at the top of the jsp pages with the syntax:

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

Below is the list **JSTL functional tags** with their explanations

S. No.	Tags	Description
1.	fn:contains()	This function checks whether the given string is present in the input as sub-string. It does a case sensitive check.
2.	fn:containsIgnoreCase()	It does a case insensitive check to see whether the provided string is a sub-string of input.
3.	fn:indexOf()	It is used for finding out the start position of a string in the provided string. Function returns -1 when string is not found in the input.
4.	fn:escapeXML()	It is used for HTML/XML character escaping which means it treats html/xml tags as a string. Similar to the escapeXml attribute of <c:out> tag.
5.	fn:join()	It concatenates the strings with a given separator and returns the output string.
6.	fn:split()	It splits a given string into an array of substrings.
7.	fn:length()	It is used for computing the length of a string or to find out the number of elements in a collection. It returns the length of the object.
8.	fn:startsWith()	It checks the specified string is a prefix of given string.
9.	fn:endsWith()	It is used for checking the suffix of a string. It checks whether the given string ends with a particular string.
10.	fn:substring()	This JSTL function is used for getting a substring from the provided string.
11.	fn:substringAfter()	It is used for getting a substring which is present in the input string before a specified string.
12.	fn:substringBefore()	It gets a substring from input which comes after a specified string.
13.	fn:trim()	It removes spaces from beginning and end of a string and function.

14.	fn:toUpperCase()	It is just opposite of fn:toLowerCase() function. It converts input string to a uppercase string.
15.	fn:toLowerCase()	This function is used for converting an input string to a lower case string.
16.	fn:replace()	It searches for a string in the input and replace it with the provided string. It does case sensitive processing.



JSTL Formatting

The formatting tags provide support for message formatting, number and date formatting etc.

Include the library at the top of the jsp pages with the syntax:

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Below is the list **JSTL functional tags** with their explanations

S. No.	Tags	Description
1.	fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number.
2.	fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting.
3.	fmt:formatNumber	It is used to format the numerical value with specific format or precision.
4.	fmt:parseDate	It parses the string representation of a time and date.
5.	fmt:bundle	It is used for creating the ResourceBundle objects which will be used by their tag body.
6.	fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
7.	fmt:setBundle	It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
8.	fmt:message	It display an internationalized message.
9.	fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.

Session Management in Java

Table of Contents

<u>Session Management in Java</u>	2
<u>Why Session Management?</u>	2
<u>What is a Session?</u>	2
<u>Session Management Mechanism</u>	2
<u>Session Management Techniques</u>	2
<u>Cookies</u>	2
<u>Important points</u>	3
<u> Signature</u>	3
<u>Create a Cookie</u>	3
<u>Set the Cookie Expiration Date</u>	3
<u>Advantages of cookies</u>	3
<u> But</u>	4
<u>Disadvantages of cookies</u>	4
<u> Technical Disadvantages</u>	5
<u>HttpSession</u>	5
<u>How HttpSession works?</u>	5
<u> Signature</u>	6
<u>Important Points</u>	6
<u>Preventing Caching of JSP Pages</u>	6
<u> Solution</u>	7

Session Management in Java

Why Session Management?

HTTP protocol and Web Servers are **stateless**, what it means is that for web server every request is a new request to process and they can't identify if it's coming from client that has been sending request previously.

But sometimes in web applications, we should know who the client is and process the request accordingly. For example, a shopping cart application should know who is sending the request to add an item and in which cart the item has to be added or who is sending checkout request so that it can charge the amount to correct client

What is a Session?

- ▶ **Session** is a conversational state between client & server where both are aware about each other
- ▶ A session indicates a **period of time** during which a **single user uses or visits a website**.
- ▶ A session starts when the user requests a page for the first time.
- ▶ During a session, the user can view as many pages as he wants, this browsing build his history
- ▶ The session ends when
 - The user hasn't requested any pages for a given amount of time (timeout).
 - The user has logged out of the website
- ▶ The session timeout varies, depending on server configuration – typically from 15 to 30 minutes.

Session Management Mechanism

Since HTTP and Web Server both are stateless, the only way to **maintain a session** is when some **unique information** about the session (**session id**) is passed between server and client in every request and response.

Session Management Techniques

There are several ways through which we can provide unique identifier in request and response:

1. Cookies
2. Session Management API

Let us go through each one of them one by one

Cookies

Simply put, a cookie is a small piece of data stored on the client-side which servers use when communicating with clients.

They're used to identify a client when sending a subsequent request. They can also be used for passing some data from one servlet to another.

- ▶ Cookies are small pieces of information that are sent by web server in response header.
- ▶ It gets stored in the browser cookies.
- ▶ When client make further request, it adds the cookie to the request header
- ▶ We can utilize cookies to keep track of the session.
- ▶ If the client disables the cookies, then it won't work.

Important points

- ▶ Cookie is a class
- ▶ Cookie is defined in javax.servlet.http package
- ▶ It implements Cloneable and Serializable interface
- ▶ Contains only 1 constructor: Cookie (String key, String value)

Signature

```
public class Cookie  
extends Object  
implements Cloneable, Serializable
```

Create a Cookie

To send a cookie to the client, we need to create one and add it to the response:

```
Cookie myCookie = new Cookie("userEmail", "kaustubhchoudhary@yahoo.com");  
response.addCookie(myCookie);
```

Set the Cookie Expiration Date

We can set the max age which defines how many seconds a given cookie should be valid for:

```
myCookie.setMaxAge(60*60);
```

We set a max age to one hour. After this time, the cookie cannot be used by a client (browser) when sending a request and it also should be removed from the browser cache.

Advantages of cookies

1. The cookies are simple to use & implement.
2. They do not require any server resources.

3. They are stored on the user's computer, so no extra burden on the server & they can lighten the load on the server's memory.
4. They are light in size, so they occupy less memory and you do not need to send back the data to the server.
5. You can configure the cookies to expire when the browser session ends (session cookies) or
6. They can exist for a specified length of time on the client's computer (persistent cookies) and
7. One of the most advantages of the cookies is their persistence , When the cookie is set on the client's browser, it can persist for days, months or years, this makes it easy to save user preferences & visit information.
8. The cookies are stored on the client's hard disk, so if the server crashes, the cookies are still available.
9. The cookies do not only remember which websites you have been to, they remember the information about forms, and they can fill out the address forms quick & efficient.
10. Most online shopping websites allow the cookies for the address & email information but they make you fill out your credit card information each time.
11. Many companies collect the data from the cookies to run the marketing campaigns aimed at a very specific market segment including the product group, geo-location, search term & the demographics.
12. You can manage your cookies easily , if you know how, Most browsers make it easy for you to clear your browsing history , Just go to the tools , clear the history and select the cookies, the cookies are stored on your hard drive in the text file under cookie.txt , You can view or edit & delete them .
13. The cookies make browsing the Internet faster & easier, the cookies allow the website to know who you are, they can tailor your browsing experience based on the previous visits, certain websites customize site information based on your location (city) & you do not have to enter the same information every time you visit the site.

But

- ▶ Although the cookies make browsing the Internet a bit easier, they are seen by many as an invasion of privacy, since most websites will not allow their site to be accessed unless cookies are enabled, so the **browsers are set to accept the cookies by default**.
- ▶ So, the cookies are being stored "invisibly" on your hard drive every time you browse the internet, since your IP address is collected, your browsing history and online activities become public knowledge.
- ▶ The browsers such as Mozilla Firefox and Internet Explorer have the options to clear the cache and delete the cookies either manually or automatically when you exit the browser.

Disadvantages of cookies

- ▶ The cookies are not secure as they are stored in a clear text & no sensitive information should be stored in cookies, they may pose to a possible security risk because anyone can open & tamper with the cookies.
- ▶ You can manually encrypt & decrypt the cookies, but it requires extra coding, you can affect the application performance because of the time that is required for encryption & decryption.

- ▶ The user has the option of disabling the cookies on his computer from the browser's setting in response to the security or the privacy worries which will cause the problem for the web applications that require them and the cookies will not work if the security level is set to high in the browser.
- ▶ The cookies can't store complex information as they are limited to simple string information , Many limitations exist on the size of the cookie text , The individual cookie can contain a very limited amount of information (not more than 4 kb)
- ▶ A lot of security holes have been found in different browsers, Some of these holes are very dangerous that they allow malicious webmasters to gain access to the users' email, different passwords & credit card information.

Technical Disadvantages

- ▶ We cannot store space separated text in cookies
- ▶ We cannot use cookies in different browsers

HttpSession

JEE provides us with HttpSession API to manage session across different clients and servers in a standard way. This way we solve the disadvantages presented by above methods, some of which are:

- a. Most of the times we don't want to only track the session; we have to store some data into the session that we can use in future requests. This will require a lot of effort if we try to implement this.
- b. All the above methods are not complete in themselves; all of them won't work in a particular scenario.

So we need a solution that can utilize these methods of session tracking to provide session management in all cases.

How HttpSession works?

1. On client's **first request**, the Web Container generates a **unique session** and gives it back to the client with response. This is a temporary session created by web container.
 2. The client sends back the session id with each request making it easier for the web container to identify where the request is coming from.
 3. The Web Container uses this session, finds the matching session and associates the session with the request
- ▶ HttpSession object is used to store entire session with a specific client.
 - ▶ We can store, retrieve and remove attribute from HttpSession object.
 - ▶ Any servlet can have access to HttpSession object throughout the getSession() method of the HttpServletRequest object. If we try to print HttpSession Object, it will give the output as:
`org.apache.catalina.session.StandardSessionFacade@1bb6a9ba`

- [StandardSessionFacade](#) is the class that implements HttpSession Interface
- It is present in: org.apache.catalina.session package

Signature

```
public class StandardSessionFacade  
extends java.lang.Object  
  
implements HttpSession
```

Important Points

- [HttpSession](#) is an interface
- It is defined in **javax.servlet.http** package
- It is implemented by StandardSessionFacade class in org.apache.catalina.session package
- An object of HttpSession can be used to perform two tasks:
 - ✓ Bind objects
 - ✓ View & manipulate session information, such as session identifier, creation/last accessed time.
- The main methods in HttpSession interface are:
 - ✓ public long getCreationTime();
 - ✓ public long getLastAccessedTime();
 - ✓ public ServletContext getServletContext();
 - ✓ public void setMaxInactiveInterval(int interval);
 - ✓ public int getMaxInactiveInterval();
 - ✓ public Object getAttribute(String name);
 - ✓ public Enumeration<String> getAttributeNames();
 - ✓ public void setAttribute(String name, Object value);
 - ✓ public void removeAttribute(String name);
 - ✓ public void invalidate();
 - ✓ public boolean isNew();

Preventing Caching of JSP Pages

- A browser can cache web pages so that it doesn't have to get them from the server every time the user asks for them.
- Proxy servers can also cache pages that are frequently requested by all users going through the proxy.

- ▶ Caching helps cut down the network traffic and server load, and provides the user with faster responses.
- ▶ But caching can also cause problems in a web application in which you really want the user to see the latest version of a dynamically generated page.

Solution

Both browsers and proxy servers can be told not to cache a page by setting response headers.

We can use a Scriptlet tag like this in your JSP pages to set these headers:

```
<%
    response.addHeader("Pragma", "no-cache");
    response.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
%>
```

Servlet Chaining

Table of Contents

<u>Servlet Chaining</u>	2
<u>Ways to send data from Servlet</u>	2
<u>Request Dispatcher</u>	2
<u>When to use RequestDispatcher interface?</u>	2
<u>How to use RequestDispatcher object</u>	3
<u>Step1. Get RequestDispatcher object</u>	3
<u>Step2. Call forward() /include() method from RequestDispatcher object</u>	4
<u>Practical Scenario: include()</u>	4
<u>Data Flow Diagram</u>	5
<u>Practical Scenarios: forward()</u>	5
<u>Data Flow Diagram</u>	5
<u>Similarities between include() and forward()</u>	5
<u>Differences between include() and forward()</u>	6
<u>HttpServletResponse sendRedirect()</u>	7
<u>Features</u>	7
<u>When to use sendRedirect() method?</u>	7
<u>Difference between forward() and sendRedirect()</u>	7

Servlet Chaining

Taking a request from a browser window and processing that request by using multiple servlets as a chain is called servlet chaining. In servlet chaining, we perform communication between servlet programs to process the request given by a client. Servlet Chaining is also known as **Servlet Collaboration**

Ways to send data from Servlet

It is important to know how to call a resource (an html page, a JSP page or a servlet) from a Servlet

There are 2 ways to perform servlet collaboration:

1. By RequestDispatcher interface
 - a. include() method
 - b. forward() method
2. By HttpServletResponse interface
 - a. sendRedirect() method

Request Dispatcher

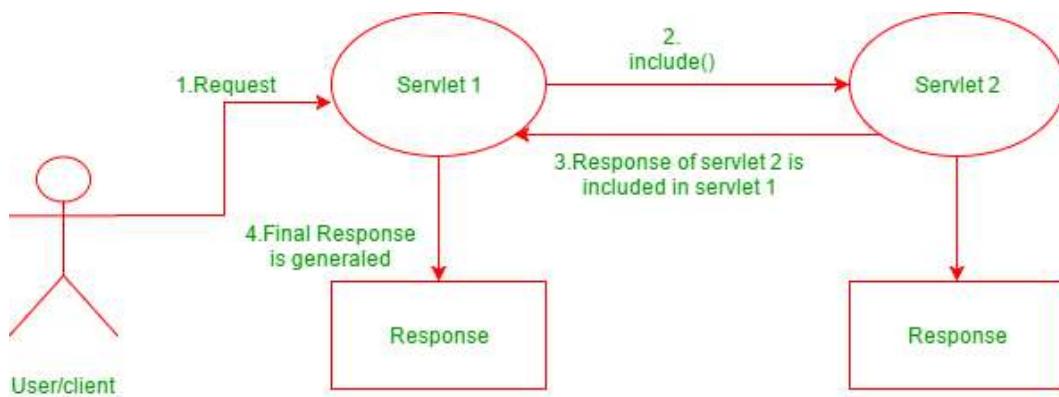
- ▶ RequestDispatcher is an interface
- ▶ It is a part of Java Servlet API
- ▶ It is found in **javax.servlet** package
- ▶ Its signature is: public interface RequestDispatcher
- ▶ It has two unimplemented methods:
 1. public void forward (ServletRequest request, ServletResponse response) throws ServletException, IOException
 2. public void include (ServletRequest request, ServletResponse response) throws ServletException, IOException

When to use RequestDispatcher interface?

- ▶ When a client's request needs to be processed by one or more resources, we must use RequestDispatcher interface.
- ▶ The RequestDispatcher interface provides the option of dispatching the client's request to another web resource, which could be an HTML page, another servlet, JSP etc.
- ▶ We can't forward a request to a URL which is external to your webapp, i.e. we can't dispatch a request to an external website/webpage

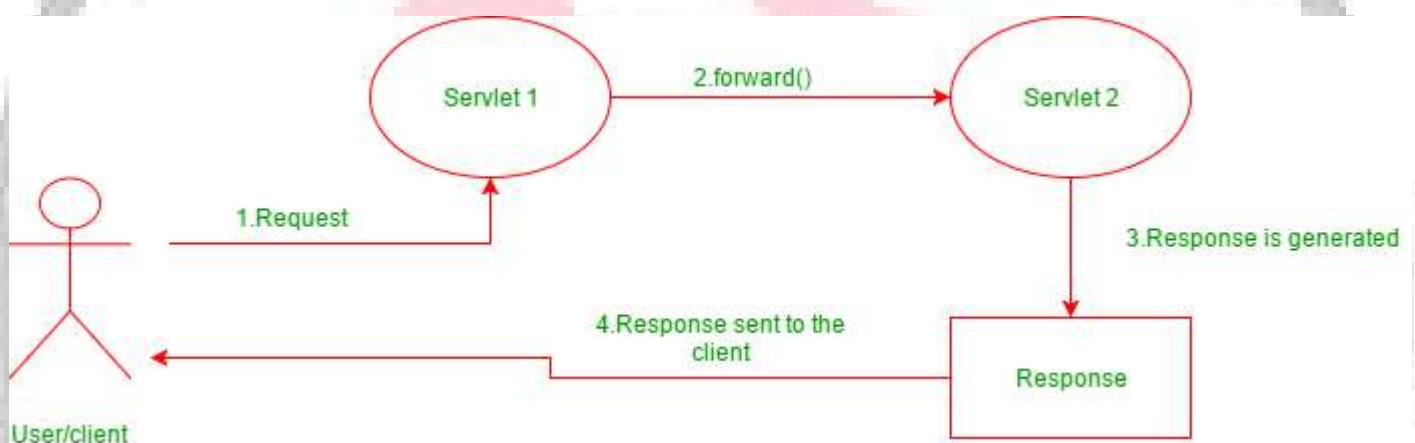
It provides the following two methods:

1. The **include (request, response)** method is used to include the contents of the called resource into the calling one.



When `include()` is called, the control still remains with the calling resource. It simply includes the processed output of the calling resource into the called one.

2. The **forward (request, response)** method is used to transfer the client request to another resource (HTML file, servlet, JSP etc).



When this method is called, the control is transferred to the next resource called

How to use RequestDispatcher object

Step1. Get RequestDispatcher object

- As we know that `RequestDispatcher` is an interface and **we can't instantiate an interface**.
- Therefore, we use `getRequestDispatcher()`, this method returns the instance of the class that implements `RequestDispatcher` interface.

Signature of the method

```
public RequestDispatcher getRequestDispatcher(java.lang.String path)
```

The `getRequestDispatcher("")` method is defined by the implementation classes of both: `ServletContext` and `ServletRequest` interfaces. Let us see how they are used and what is the difference?

Syntax 1: Calling getRequestDispatcher() on HttpServletRequest object

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

Important Points:

1. HttpServletRequest object is provided as a parameter in our Servlet, we use it to call the method.
2. The pathname specified may be relative, although it cannot extend outside the current servlet context.
3. If the path begins with a "/" it is interpreted as relative to the current context root.
4. This method returns **null** if the servlet container cannot return a RequestDispatcher.
5. getRequestDispatcher() is declared in ServletRequest Interface and defined in ServletRequestWrapper class

Syntax 2: Calling getRequestDispatcher() on ServletContext object

```
ServletContext servletContext = getServletContext();
RequestDispatcher rd = servletContext.getRequestDispatcher("/login");
```

Important Points

1. Get an instance of ServletContext object in our Servlet and then use it to call the method
2. The pathname **must begin with a "/"** and is interpreted as relative to the current context root.
3. Use getContext() to obtain a RequestDispatcher for resources in foreign contexts.
4. This method returns **null** if the ServletContext cannot return a RequestDispatcher.

Step2. Call forward() / include() method from RequestDispatcher object

Syntax

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

or

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

Practical Scenario: include()

Example:

Electrical Billing: Consider the problem statement

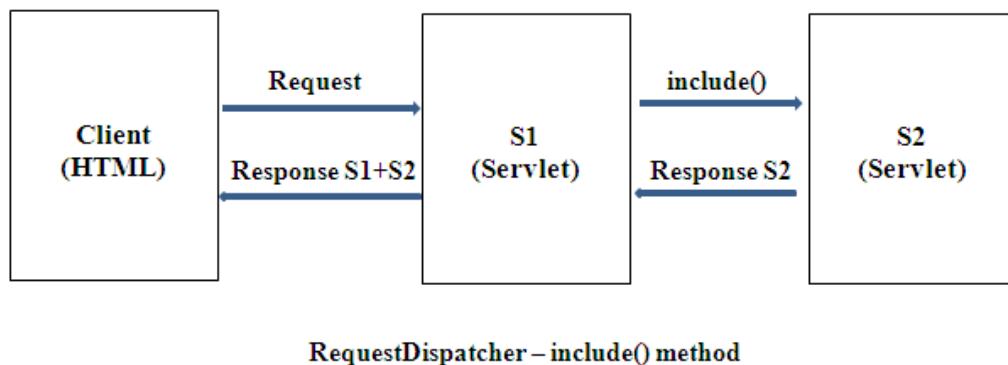
1. The bill collector takes the latest meter reading and sends it to a servlet (along previous reading), say **S1** for billing.
2. The **S1** servlet has the calculating factor **rate per unit** but it does not have any code to estimate the bill amount.
3. S1 knows that Servlet **S2** can calculate the bill
4. **S1** passes the client data to **S2**.
5. **S2** will estimate, prepares the bill and return to **S1**.

6. **S1** in turn, sends the response to client.

7. Now **S1** has three responsibilities –

- to send actual client data (of two readings, previous and latest) as it is to S2
- to send rate per unit to S2
- to receive the response of S2, add its own response, if any, like last date of bill payable etc. to client.

Data Flow Diagram



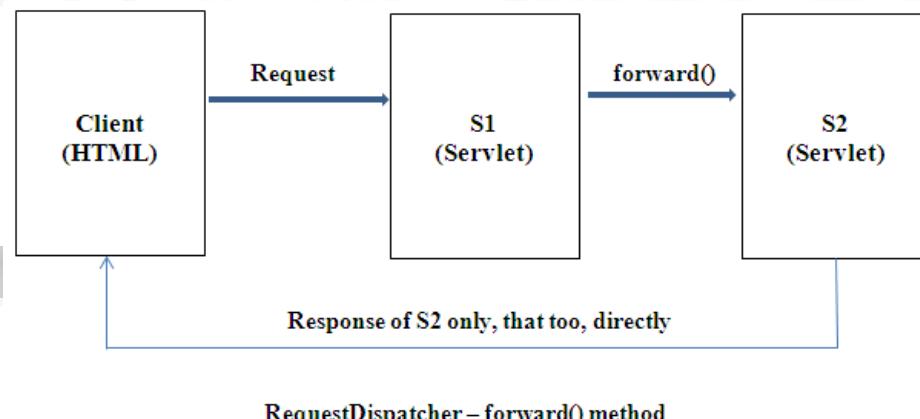
Practical Scenarios: forward()

Example:

Product: Consider the problem statement

1. Client sends two numbers to a Servlet **S1** to know their product.
2. **S1** forwards the client request to **S2** Servlet using forward() method.
3. **S2** Servlet gets the request data, calculates the product and send the response to client
4. The response of **S1** is not included in the final response

Data Flow Diagram



Similarities between include() and forward()

- 1) Both include() and forward() methods are part of RequestDispatcher interface of Servlet API
- 2) Both methods accept objects of ServletRequest and ServletResponse interface.

- 3) Both include() and forward() can interact with static and dynamic resource e.g. another Servlet, JSP or HTML files.
- 4) Both cannot be used to dispatch a request to an external webpage/website

Differences between include() and forward()

Property	include() method	forward() method
What can be done?	Includes another file in our current file.	Will forward the client request to the forwarding page.
Merge of response	Response of S1 and S2 are merged and sent to client (as if a single response). This way, the Programmer can achieve "server side includes".	No merge of response. Only S2 response will go to the client.
Retaining execution control	Shifted temporarily from S1 to S2. It works like a general simple method call.	Shifted permanently from S1 to S2.
Control coming back	Execution control comes back to S1 after executing S2 for further processing of S1 after include() statement from where the execution control shifted.	Once shifted, the control never returns to S1. It is permanent shifting.
Response placement	Response of S2 is placed in S1.	Response of S2 is not placed in S1
Client receives	Response of S1 and S2 is received by client.	Only response of S2 is received by client.
Control returned	After executing S2, control returns to S1.	After executing S2, control returns to client.
Extra activity	Once control is returned to S1 from S2, any activity can be done on the server like calling another servlet with another RequestDispatcher object.	Once control returns to client, no activity can be done on S1 or S2.
Usage	Used by Programmer when the output of both servlets S1 and S2 is required.	Used only when S2 response is required.
Speed of delivery to client	Comparatively slower.	Faster.
Access	S2 has access to the request and response objects of S1, but limitations exist. S2 cannot set headers and also cannot call any other method like setCookie etc. affecting the response headers. That is, S2 cannot attempt to change the HTTP headers or response status code etc. and performing any activities like this is simply ignored.	Here also S2 cannot alter as response is delivered on S1 URL.
Usage	Often useful just to include another web resource response like banner content or copyright information etc.	Used when further processing responsibility and replying to user is given to another Web resource.
out.println	Output of S1's out.println() statements	Output of S1's out.println() statements

	go to client.	never go to client.
Client response	Client receives the response from the same servlet which he requested.	Client actually receives the response from a different servlet (not known to client).
Treatment of processing	The processing S2 can be treated as part of S2 processing.	The processing of S2 can be treated completely as a different entity. It is used to show a different resource on the server by S1.
Used when	Used when static information is to be included.	Used when dynamic information is to be included. It can be used where a Servlet can play the role of a controller to process the client input and deciding what response page is to be returned.

HttpServletResponse sendRedirect()

Features

1. HttpServletResponse is an interface
2. It is present in **javax.servlet.http** package
3. Signature: public interface HttpServletResponse
4. Syntax of the method is: **void sendRedirect (String location)**
5. **String location** is the specified redirect location URL.

When to use sendRedirect() method?

- We must use sendRedirect when a programmer would like to redirect the client request to another Web site on a different server.
- The client request cannot be fulfilled by the current Servlet and the programmer may like to send the client request to another resource.

Difference between forward() and sendRedirect()

S. No.	Property	forward()	sendRedirect()
1.	Defined in	Defined in RequestDispatcher	Defined in HttpServletResponse
2.	Signature	void forward (ServletRequest request, ServletResponse response)	void sendRedirect (String URL)
3.	Client awareness	Client is not aware that it is getting response from a different Servlet as the URL will not change in client's browser.	Client can know easily as the URL (from where it is getting response) changes in the client browser's prompt.
4.	Execution control	Execution control changes to another Servlet on the same server without client being informed that altogether a different Servlet is going to process his request.	Control changes to client
5.	Where is what?	It is the same request that is processed, not a new one. Forward is done on server side without client's knowledge.	Browser issues a new request on the URL that is redirected (sent as parameter) by the server and client can easily aware of.

6.	Where happens	Everything happens on server side within the Web container and client is not involved.	sendRedirect() causes the Web container to return to the client's browser. Client in turn can redirect to different servers or domains.
7.	Speed	Faster as forward runs on server-side entirely and no extra network trip (to client) is required.	Due to round trip between browser-server-browser (running on client as well as on server side), it is slower.
8.	Content	forward() sends the same request to another resource of the same Web application.	Calls another page with a different request URL but not on the same request.
9.	Usage	Used when processing is done by another Servlet	Used when wanted to redirect the client request to another Web site (completely out of current context) or to redirect the errors to another resource
10.	Reusability	forward() reuses the current request object	Redirects create a new request object; consequently loses the original request with all its parameters and attributes.
11.	Transfer of parameters	Original request and response objects transfer data coming from client along with additional information set with setAttribute() method (if any) to another resource request and response objects.	Redirect action sends header back to the client. Browser uses the URL contained in the header to call a new resource. As client initiates a new request, the original request and response objects are lost and fresh ones are to be created.
12.	Transfer control	Internally, the Servlet container transfers control of client request to another Servlet (or JSP).	This method sends the HTTP response to client browser to allow the client to send another request with a different URL. Usage of this method is equivalent to opening a new browser window and typing the URL.
13.	What is sent?	Server sends the response (information required) to the client.	With this method, server sends a URL to the client.
14.	Visual difference	Client cannot see the address of new resource that honors the client request in the address bar of the browser.	Client can see the new redirected address in address bar.
15.	Examples	Calling another resource to process the data like validation of Login data.	Calling advertisements on the Web page or payment gateways.
16.	Task separation	With this method, the responsibility of handling the client request can be distributed between many Servlets (or JSPs).	Used to transfer control altogether to a different domain. Also used to write separation of tasks.
17.	Back and Forward buttons	As everything happens on server with forward, nothing is stored on browser history. So, Back and Forward buttons will not work.	As client makes new request and updated in browser history, back and forward buttons work.
18.	URL	Use only relative URLs with forward().	Use absolute URLs.
19.	MVC to hide	Useful in MVC design pattern to hide JSP/Servlet from direct access.	Once redirected to client, server loses control.

20.	Which one to prefer?	If you would like to forward the client request to a new resource on the same server for further process, prefer forward() where data of the original resource can be passed to the new resource.	If you would like to transfer the control to a new server or domain where client treats as a new task, prefer sendRedirect(). If the data of the original resource (which client requested) is needed in the new resource, store them in Session object and reuse.
-----	----------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



Introduction to Maven

Table of Contents

<u>Why do we need Maven?</u>	11
<u>Executing a Java Project</u>	11
<u>So, how did we do it?</u>	11
<u>For single java file applications</u>	11
<u>For multiple java files applications</u>	11
<u>For multiple types of files Java Project</u>	11
<u>Build System</u>	12
<u>Dependencies</u>	12
<u>The problem without Maven</u>	13
<u>Why Study Maven?</u>	13
<u>What is Maven?</u>	13
<u>Maven Features</u>	13
<u>Installing Maven</u>	14
<u>Maven Overview - Core Concepts</u>	14
<u>Project Object Model (pom.xml)</u>	14
<u>Project Identifiers</u>	16
<u>Dependencies</u>	16
<u>Repositories</u>	16

Why do we need Maven?

To understand Maven, let us first make ourselves familiar with some terminologies of a Java Project

Executing a Java Project

We have developed many java applications until now such as:

- ✓ Simple Java Application containing a single .java file
- ✓ A bit more complex Java Program containing multiple .java files
- ✓ Java Project that contains not only .java files but also html, css, js, jpg, jpeg, mp4, xml and jar files

Whatever be the size of the application/project, we have to execute the application.

So, how did we do it?

For single java file applications

When we were developing single .java files application, we did it by

- ✓ Compiling the source code : `javac MyFirstProgram.java`
- ✓ Executing the generated .class file: `java MyFirstProgram 12 45`
- ✓ And then we get the result

For multiple java files applications

- ✓ We have to compile all the .java file that the main program needed
- ✓ We can only then run the .class file that contained main method

For multiple types of files Java Project

If we do it manually

This task will be exhausting and virtually impossible as it will include

- ✓ Compiling all the .java files that does not depend upon any other .java file
- ✓ After the above step we will have to compile .java files that depend upon other smaller classes
- ✓ Converting the other resources and attaching them to the java application
- ✓ Including the third party .jar files such as connector.jar, servlet-api.jar, jstl.jar, hibernate.jar in the project
- ✓ Attach xml files and resources with the executable code

If we used IDE (Eclipse)

- ✓ Click on the green arrow and the IDE will take care of compiling all the .java files into .class files, attaching the resources, parsing xml files, generate compressed .jar, .war files etc.
- ✓ In the same flow after compilation, the IDE will execute the code

Build System

So, as you can see from the above discussion that writing code and executing code requires a lot of effort. This process of bringing all the resources together to a state where we can successfully execute our Java Application is known as **Building the Project**.

And the system that we follow to build the project is known as a Build System.

In other words we can say that:

Software Build is an activity to translate the human-readable source code into the efficient executable program.

There are a lot of build systems, softwares, plugins available in the market, famous amongst which are:

1. Jenkins
2. Apache Ant
3. Gradle
4. Maven
5. Some IDEs use their custom build process
6. Some IDEs use third party plugins to build their projects

So, we can say that building a software project typically includes one or more of these activities:

- ✓ Generating source code (if auto-generated code is used in the project).
- ✓ Generating documentation from the source code.
- ✓ Compiling source code.
- ✓ Packaging compiled code into JAR files or ZIP files.
- ✓ Installing the packaged code on a server, in a repository or somewhere else.

Dependencies

Suppose we write a simple java application which contains two classes: Box and BoxDemo. The main method of BoxDemo class contains the code to instantiate Box class. Now in order to execute the code we will require .class files of both classes: Box.class and BoxDemo.class. The programs will not execute if Box.class is not available.

That is what we call a dependency. Our program depends upon Box.class to execute.

Now take this knowledge into our JEE projects.

- ✓ We use Tomcat server that provides servlet-api.jar package that contains .class files for implementing Servlet logic
- ✓ We use connector.jar files to connect our Java application to database by using interfaces such as Connection, PreparedStatement, ResultSet etc

All these .jar files that we use are known as dependencies that have to be included in our Project

The problem without Maven

The good thing is that our IDE takes care of building the whole Java Project by taking proper care of all the dependencies in just a click. Still we need to search, paste and include the jars in our project manually. To automate this redundant task, Maven is used

Why Study Maven?

In the modern world, we need faster, cost effective and efficient development cycles. So, to achieve this there had been continuous efforts in the field of software development to

- ✓ Ease the life of developers,
- ✓ Reduce the time of development,
- ✓ To reduce the cost of development
- ✓ Reduce the bugs

Maven is the result of one such effort

We know that a Java Application needs a lot of dependencies, so it becomes hard for the java programmer to maintain them at a certain point of time. Maven simplifies this process

What is Maven?

Building a software project typically consists of such tasks as downloading dependencies, putting additional jars on a class-path, compiling source code into binary code, running tests, packaging compiled code into deployable artifacts such as JAR, WAR, and ZIP files, and deploying these artifacts to an application server or repository.

Apache Maven automates these tasks, minimizing the risk of humans making errors while building the software manually and separating the work of compiling and packaging our code from that of code construction.

Maven Features

The key features of Maven are:

- ✓ **Simple project setup that follows best practices:** Maven tries to avoid as much configuration as possible, by supplying project templates (named *archetypes*)
- ✓ **Dependency management:** It includes automatic updating, downloading and validating the compatibility, as well as reporting the dependency closures (known also as transitive dependencies)
- ✓ **Isolation between project dependencies and plugins:** with Maven, project dependencies are retrieved from the *dependency repositories* while any plugin's dependencies are retrieved from the *plugin repositories*, resulting in fewer conflicts when plugins start to download additional dependencies

- ✓ **Central repository system:** project dependencies can be loaded from the local file system or public repositories, such as Maven Central

Installing Maven

To install Maven on your own system (computer), go to the Maven download page and follow the instructions there. In summary, what you need to do is:

1. Download Maven from: <https://maven.apache.org/download.cgi>
2. Set the JAVA_HOME environment variable to point to a valid Java SDK
3. Example: (JAVA_HOME = C:\Program Files\Java\jdk-11.0.2\).
4. Download and unzip Maven.
5. Set the M2_HOME environment variable to point to the directory you unzipped
Example: (**M2_HOME** = C:\apache-maven-3.6.3)
6. Set the MAVEN_HOME environment variable to point to the directory you unzipped
Example: (**MAVEN_HOME** = C:\apache-maven-3.6.3)
7. Set the M2 environment variable to point to:
Example: (**M2** = M2_HOME/bin)
8. Add M2 to the PATH environment variable (%M2% on Windows)
9. Open a command prompt and type '**mvn -version**' (without quotes) and press enter.
10. After typing in the mvn -version command you should be able to see Maven execute, and the version number of Maven written out to the command prompt

Maven Overview - Core Concepts

Project Object Model (pom.xml)

Maven is centered on the concept of POM files (Project Object Model). A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc. The POM contains references to all of these resources. The POM file should be located in the root directory of the project it belongs to.

Let's look at the basic structure of a typical *POM* file:

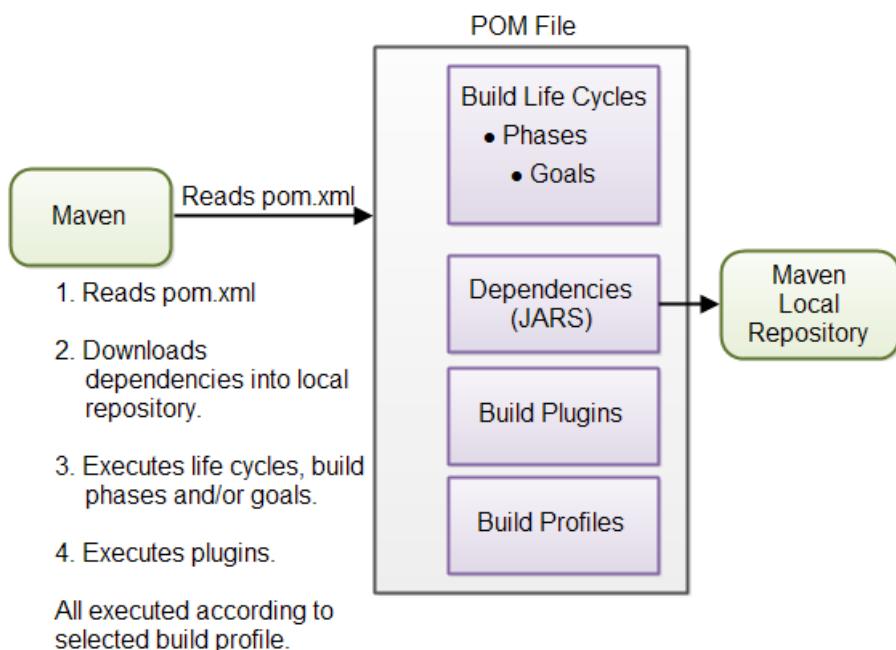
```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.baeldung</groupId>
  <artifactId>org.baeldung</artifactId>
  <packaging>jar</packaging>
```

```

<version>1.0-SNAPSHOT</version>
<name>org.baeldung</name>
<url>http://maven.apache.org</url>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      //...
    </plugin>
  </plugins>
</build>
</project>

```

Here is a diagram illustrating how Maven uses the POM file, and what the POM file primarily contains:



Project Identifiers

Maven uses a set of identifiers, also called coordinates, to uniquely identify a project and specify how the project artifact should be packaged:

- ✓ *groupId* – a unique base name of the company or group that created the project
- ✓ *artifactId* – a unique name of the project
- ✓ *version* – a version of the project
- ✓ *packaging* – a packaging method (e.g. WAR/JAR/ZIP)

The first three of these (*groupId:artifactId:version*) combine to form the unique identifier and are the mechanism by which you specify which versions of external libraries (e.g. JARs) your project will use.

Dependencies

These external libraries that a project uses are called dependencies. The dependency management feature in Maven ensures automatic download of those libraries from a central repository, so you don't have to store them locally.

This is a key feature of Maven and provides the following benefits:

- ✓ uses less storage by significantly reducing the number of downloads off remote repositories
- ✓ makes checking out a project quicker

In order to declare a dependency on an external library, you need to provide the *groupId*, *artifactId*, and the *version* of the library. Let's take a look at an example:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.3.5.RELEASE</version>
</dependency>
```

Repositories

A repository in Maven is used to hold build artifacts and dependencies of varying types. The default local repository is located in the *.m2/repository* folder under the home directory of the user.

If an artifact or a plug-in is available in the local repository, Maven uses it. Otherwise, it is downloaded from a central repository and stored in the local repository. The default central repository is Maven Central.

Some libraries, such as JBoss server, are not available at the central repository but are available at an alternate repository. For those libraries, you need to provide the URL to the alternate repository inside *pom.xml* file:

```
<repositories>
  <repository>
    <id>JBoss repository</id>
    <url>http://repository.jboss.org/nexus/content/groups/public/</url>
  </repository>
</repositories>
```

Note: Please note that you can use multiple repositories in your projects

Introduction to Hibernate

Table of Contents

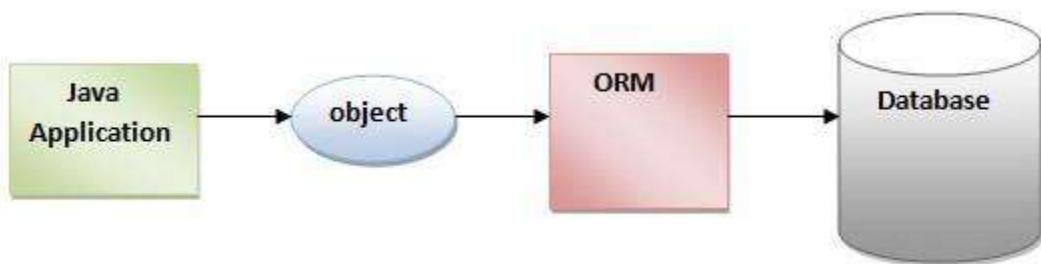
<u>What is Hibernate?</u>	19
<u>What is ORM?</u>	19
<u>What is JPA?</u>	19
<u>Hibernate Origins</u>	19
<u>Hibernate Version History</u>	19
<u>Features of Hibernate</u>	20
1. <u>Open Source</u>	20
2. <u>Lightweight</u>	20
3. <u>Non-invasive</u>	20
4. <u>Fast Performance</u>	20
5. <u>Database Independent Query</u>	20
6. <u>Automatic Table Creation</u>	21
7. <u>Simplifies Complex Join</u>	21
8. <u>Provides Query Statistics and Database Status</u>	21
<u>Why Hibernate?</u>	21
<u>Why use Hibernate over JDBC?</u>	21
<u>Advantages of Hibernate over JDBC</u>	21

What is Hibernate?

- ▶ Hibernate is a **Java framework**
- ▶ It simplifies the development of Java applications to interact with the database.
- ▶ Hibernate is an **ORM** (Object Relational Mapping) tool.
- ▶ Hibernate implements the specifications of JPA (**Java Persistence API**) for data persistence.
- ▶ Latest JPA specification is **2.2**.
- ▶ Other implementations of JPA are **iBatis** and **Toplink**
- ▶ Hibernate can be used to develop all types of application be it web, enterprise or desktop application

What is ORM?

An ORM tool simplifies data creation, data manipulation and data access. It is a programming technique that **maps an object to the data stored in the database**.



Note: The ORM tool **internally uses** the **JDBC API** to interact with the database.

What is JPA?

Java Persistence API (JPA) is a Java **specification** that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

Hibernate Origins

- ▶ Hibernate was developed by **Gavin King** in **2001** at **Cirrus Technologies**
- ▶ In early 2003, the Hibernate development team began Hibernate 2 releases
- ▶ **JBoss Inc.** (now part of **Red Hat**) later hired the lead Hibernate developers for further development.

Hibernate Version History

- ▶ Hibernate Core 3.0 was released in 2005
- ▶ Hibernate Core 4.0 was released in 2011

- Hibernate **Core 5.0** was released in 2018

Features of Hibernate

1. Open Source

- Hibernate is **open source**.
- It is distributed under the [GNU Lesser General Public License 2.1](#)
- Hence, it is free to download and use

2. Lightweight

A framework is called lightweight when it comes to size and transparency, the term lightweight is sometimes applied to a program, protocol, device, or anything that is **relatively simpler or faster** or **that has fewer parts** than something else.

Hibernate is lightweight because:

- Hibernate is intended for **one and the only task i.e.** of saving object data to our database.
 - Because Hibernate is focused on just one thing, it is **relatively simple and efficient**.
 - Hibernate is implemented with a set of **simple POJOs** and POJOs are **simple classes**.
 - **Logic** in the POJOs is directly **executed** in the **same thread** of control as the web layer.
- Hence Hibernate is simple and lightweight

3. Non-invasive

Hibernate is a **non-invasive** framework, means it won't force the programmers to extend/implement any class/interface.

4. Fast Performance

Hibernate implements various mechanisms to achieve fast performance, such as:

- ✓ **Caching** – mechanism to save number of queries to the database
- ✓ **Lazy Loading** – mechanism to make query database when the data is actually needed

5. Database Independent Query

Hibernate Query Language (**HQL**) is an **object-oriented query language**, similar to SQL, but instead of operating on tables and columns, HQL **works with persistent objects** and their properties.

- **HQL** (Hibernate Query Language) is the **object-oriented version** of SQL.
- It generates **database independent queries**. So there is no need to write database specific queries.

- ▶ HQL queries are **translated** by Hibernate **into conventional SQL queries**, which in turns perform action on database.
- ▶ Before Hibernate, if database was changed for the project, we had to change the SQL query as well to suit to changed database. It lead to the maintenance problem.

6. Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

7. Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

8. Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status

Why Hibernate?

- ▶ Hibernate was developed as an alternative to using **EJB2-style entity beans**.
- ▶ The original goal was to offer **better persistence capabilities** than those offered by EJB2
- ▶ It **simplified the complexities** and supplemented certain missing features.

Why use Hibernate over JDBC?

JDBC has its place, but Hibernate comes ready with an arsenal of **helpful tools and capabilities** that make connecting and transacting with the database a **much easier** prospect.

Advantages of Hibernate over JDBC

S. No.	JDBC	Hibernate
1.	In case of JDBC we need to learn SQL (Structured Query Language) along with Java to write database code.	Hibernate is set of objects, so we don't need to learn SQL language. Having only Java knowledge is sufficient.
2.	JDBC is database dependent i.e. one needs to write different codes for different database.	Hibernate is database independent and same code can work for many different databases such as MySQL, Oracle, SQL Server etc. with minor changes
3.	JDBC enables developers to create queries and update data to a relational database using the	Hibernate uses HQL which is similar to SQL but understands object-oriented concepts like

	Structured Query Language (SQL).	inheritance, association etc.
4.	JDBC code needs to be written in a try catch block as it throws checked exception (SQLException).	Whereas Hibernate manages the exceptions itself by marking them as unchecked .
5.	In JDBC, the developer needs to write code to map the object model's data representation to the schema of the relational model.	Hibernate maps the object model's data to the schema of the database itself with the help of annotations.
6.	Creating associations between relations is quite hard in JDBC.	Associations like one-to-one, one-to-many, many-to-one, and many-to-many can be acquired easily with the help of annotations.



CODEWITHARRAYS

First Hibernate Application

Table of Contents

<u>First Hibernate Application</u>	24
<u>Setup Development Environment</u>	24
<u>Create Application</u>	24
<u>Step 1: Create a Maven Project</u>	24
<u>Step 2: Add Dependencies</u>	24
<u>Step 3: Create a POJO class</u>	24
<u>Step 4: Create Hibernate Configuration File</u>	25
<u>Step 5: Create an Execution Class</u>	26
<u>Step 6: Execute the code</u>	27

First Hibernate Application

Setup Development Environment

Download and Install

1. Java Development Kit (Current version: **11.0.2**)
2. Eclipse IDE (Current version: Eclipse IDE **2020-12 R**)
3. MySQL Server and MySQL Workbench (Current version: **8.0.23**)
4. Apache Maven (Current version: **3.6.3**)

Create Application

Step 1: Create a Maven Project

- ▶ Choose **Internal Catalogue**
- ▶ Choose **Quicktype archetype**
- ▶ Add other project details such as
 1. group id: com.hibernate.demo,
 2. artifact id: FirstHibernateProject and
 3. version: 0.0.1-SNAPSHOT (Do not change – Keep it as it is)
 4. package: com.hibernate.demo

Step 2: Add Dependencies

In order to use Java Persistent API, Hibernate Framework and Database API we have to include the certain dependencies in our project.

- ▶ Search for **Hibernate Maven** and **MySQL Connector Maven** on the internet
- ▶ Among the search results, open the link that specifies: <https://mvnrepository.com/>.
- ▶ Copy paste the dependencies in **pom.xml** file inside **<dependencies>** tag
- ▶ The project will **automatically** update its Maven Dependencies
- ▶ If there is no automatic update, right click on the project, choose **Maven>Update** manually

Step 3: Create a POJO class

Hibernate Framework maps a class-object with RDBMS. In order to do so, we must create a POJO class.

- ▶ Create a POJO class (say **Student**) in **src/main/java** folder in a package (say **com.hibernate.pojo**)
- ▶ Create 3 fields: **private int id**, **private String name** and **private int section**.

- ▶ Annotate class Student with **@Entity** and id with **@Id**.
- ▶ Be sure to check the above annotations belong to **javax.persistence** package.

Step 4: Create Hibernate Configuration File

Hibernate will take our object and use it to transact with database, as such we need to provide some **connection details** as well as some **hibernate specific configurations**

We will use **XML** to do the configurations

- ▶ Step 1: Make an xml file in src/main/java folder
- ▶ Step 2: Its name should be **hibernate.cfg.xml**
- ▶ Step 3: Write **Hibernate DTD 3.0** (from <http://hibernate.org/dtd/>) as its first tag

```
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

- ▶ DTD stands for Document Type Definition

Note:

1. Please keep in mind that DTD versions do NOT match the Hibernate ORM version.
 2. Just use the **highest** DTD version that is **lower than or equal to** your Hibernate ORM version.
- ▶ Step 4: Put the parent tags

```
<hibernate-configuration>
  <session-factory>
    </session-factory>
</hibernate-configuration>
```

- ▶ Step 5: Write down the following properties inside **<session-factory>** tag

```
<property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
<property
  name="connection.url">jdbc:mysql://localhost:3306/hibernate_demo</property>
<property name="connection.username">root</property>
<property name="connection.password">1234</property>
<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
<property name="hbm2ddl.auto">update</property>
<property name="show_sql">true</property>
<property name="format_sql">true</property>
```

► Step 6: Tell Hibernate about Entity classes

Copy the fully qualified name of the Entity class (Student class in our example) and mention it in **<mapping>** tag inside **<session-factory>** tag after all the **<property>** tags

```
<mapping class="com.hibernate.pojo.Student"/>
```

The configuration file is complete

Step 5: Create an Execution Class

Write the following code

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class App {
    public static void main(String[] args) {
        // Load Configuration file
        Configuration configuration = new Configuration();
        configuration.configure();

        // Create a session factory
        SessionFactory sessionFactory = configuration.buildSessionFactory();

        // Create a session
        Session session = sessionFactory.openSession();

        // Start Transaction
        Transaction transaction = session.beginTransaction();

        // Create Student object to save
        Student student = new Student(101, "Kaustubh", 5);

        // Save the student object
        session.save(student);
    }
}
```

```
// Commit the transaction  
transaction.commit();  
  
// Close Resources  
session.close();  
sessionFactory.close();  
}  
}
```

Step 6: Execute the code

- ▶ Make sure the MySQL Server is running with the config data mentioned in config file
- ▶ Make sure there is a schema: hibernate_demo in place
- ▶ Run the application as a Java Application

Note: We will learn about the above code in later classes

Introduction to Hibernate

Table of Contents

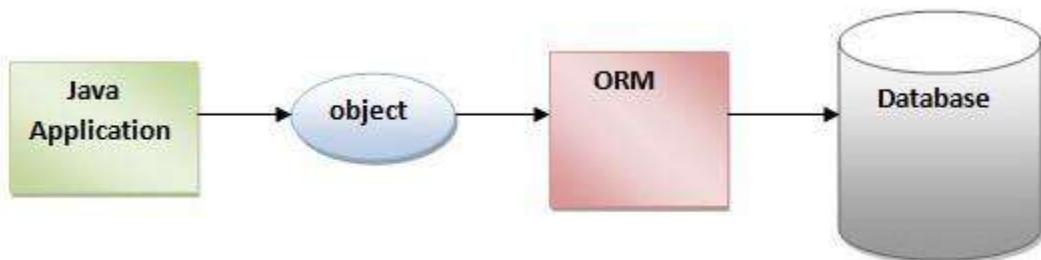
<u>What is Hibernate?</u>	19
<u>What is ORM?</u>	19
<u>What is JPA?</u>	19
<u>Hibernate Origins</u>	19
<u>Hibernate Version History</u>	19
<u>Features of Hibernate</u>	20
1. <u>Open Source</u>	20
2. <u>Lightweight</u>	20
3. <u>Non-invasive</u>	20
4. <u>Fast Performance</u>	20
5. <u>Database Independent Query</u>	20
6. <u>Automatic Table Creation</u>	21
7. <u>Simplifies Complex Join</u>	21
8. <u>Provides Query Statistics and Database Status</u>	21
<u>Why Hibernate?</u>	21
<u>Why use Hibernate over JDBC?</u>	21
<u>Advantages of Hibernate over JDBC</u>	21

What is Hibernate?

- ▶ Hibernate is a **Java framework**
- ▶ It simplifies the development of Java applications to **interact with the database**.
- ▶ Hibernate is an **ORM** (Object Relational Mapping) tool.
- ▶ Hibernate implements the specifications of **JPA (Java Persistence API)** for data persistence.
- ▶ Latest JPA specification is **2.2**.
- ▶ Other implementations of JPA are **iBatis** and **Toplink**
- ▶ Hibernate can be used to develop all types of application be it web, enterprise or desktop application

What is ORM?

An ORM tool simplifies data creation, data manipulation and data access. It is a programming technique that **maps an object to the data stored in the database**.



Note: The ORM tool **internally uses** the **JDBC API** to interact with the database.

What is JPA?

Java Persistence API (JPA) is a Java **specification** that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

Hibernate Origins

- ▶ Hibernate was developed by **Gavin King** in **2001** at **Cirrus Technologies**
- ▶ In early 2003, the Hibernate development team began Hibernate 2 releases
- ▶ **JBoss Inc.** (now part of **Red Hat**) later hired the lead Hibernate developers for further development.

Hibernate Version History

- ▶ Hibernate Core 3.0 was released in 2005
- ▶ Hibernate Core 4.0 was released in 2011

- Hibernate **Core 5.0** was released in 2018

Features of Hibernate

9. Open Source

- Hibernate is **open source**.
- It is distributed under the [GNU Lesser General Public License 2.1](#)
- Hence, it is free to download and use

10. Lightweight

A framework is called lightweight when it comes to size and transparency, the term lightweight is sometimes applied to a program, protocol, device, or anything that is **relatively simpler or faster** or **that has fewer parts** than something else.

Hibernate is lightweight because:

- Hibernate is intended for **one and the only task i.e.** of saving object data to our database.
 - Because Hibernate is focused on just one thing, it is **relatively simple and efficient**.
 - Hibernate is implemented with a set of **simple POJOs** and POJOs are **simple classes**.
 - **Logic** in the POJOs is directly **executed** in the **same thread** of control as the web layer.
- Hence Hibernate is simple and lightweight

11. Non-invasive

Hibernate is a **non-invasive** framework, means it won't force the programmers to extend/implement any class/interface.

12. Fast Performance

Hibernate implements various mechanisms to achieve fast performance, such as:

- ✓ **Caching** – mechanism to save number of queries to the database
- ✓ **Lazy Loading** – mechanism to make query database when the data is actually needed

13. Database Independent Query

Hibernate Query Language (**HQL**) is an **object-oriented query language**, similar to SQL, but instead of operating on tables and columns, HQL **works with persistent objects** and their properties.

- **HQL** (Hibernate Query Language) is the **object-oriented version** of SQL.
- It generates **database independent queries**. So there is no need to write database specific queries.

- ▶ HQL queries are **translated** by Hibernate **into conventional SQL queries**, which in turns perform action on database.
- ▶ Before Hibernate, if database was changed for the project, we had to change the SQL query as well to suit to changed database. It led to the maintenance problem.

14. Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

15. Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

16. Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status

Why Hibernate?

- ▶ Hibernate was developed as an alternative to using **EJB2-style entity beans**.
- ▶ The original goal was to offer **better persistence capabilities** than those offered by EJB2
- ▶ It **simplified the complexities** and supplemented certain missing features.

Why use Hibernate over JDBC?

JDBC has its place, but Hibernate comes ready with an arsenal of **helpful tools and capabilities** that make connecting and transacting with the database a **much easier** prospect.

Advantages of Hibernate over JDBC

S. No.	JDBC	Hibernate
1.	In case of JDBC we need to learn SQL (Structured Query Language) along with Java to write database code.	Hibernate is set of objects, so we don't need to learn SQL language. Having only Java knowledge is sufficient.
2.	JDBC is database dependent i.e. one needs to write different codes for different database.	Hibernate is database independent and same code can work for many different databases such as MySQL, Oracle, SQL Server etc. with minor changes
3.	JDBC enables developers to create queries and update data to a relational database using the	Hibernate uses HQL which is similar to SQL but understands object-oriented concepts like

	Structured Query Language (SQL).	inheritance, association etc.
4.	JDBC code needs to be written in a try catch block as it throws checked exception (SQLException).	Whereas Hibernate manages the exceptions itself by marking them as unchecked .
5.	In JDBC, the developer needs to write code to map the object model's data representation to the schema of the relational model.	Hibernate maps the object model's data to the schema of the database itself with the help of annotations.
6.	Creating associations between relations is quite hard in JDBC.	Associations like one-to-one, one-to-many, many-to-one, and many-to-many can be acquired easily with the help of annotations.



CODEWITHARRAYS

First Hibernate Application

Table of Contents

<u>First Hibernate Application</u>	24
<u>Setup Development Environment</u>	24
<u>Create Application</u>	24
<u>Step 1: Create a Maven Project</u>	24
<u>Step 2: Add Dependencies</u>	24
<u>Step 3: Create a POJO class</u>	24
<u>Step 4: Create Hibernate Configuration File</u>	25
<u>Step 5: Create an Execution Class</u>	26
<u>Step 6: Execute the code</u>	27

First Hibernate Application

Setup Development Environment

Download and Install the latest version of

5. Java Development Kit (Current version: **16.0.1**)
6. Eclipse IDE (Current version: Eclipse IDE **2021-06 R**)
7. MySQL Server and MySQL Workbench (Current version: **8.0.24**)
8. Apache Maven (Current version: **3.8.1**)

Create Application

Step 1: Create a Maven Project

- ▶ Choose **Internal Catalogue**
- ▶ Choose **Quicktype** archetype
- ▶ Add other project details such as
 5. group id: com.hibernate.demo,
 6. artifact id: FirstHibernateProject and
 7. version: 0.0.1-SNAPSHOT (Do not change – Keep it as it is)
 8. package: com.hibernate.demo

Step 2: Add Dependencies

In order to use Java Persistent API, Hibernate Framework and Database API we have to include the certain dependencies in our project.

- ▶ Search for **Hibernate** Maven and **MySQL Connector** Maven on the internet
- ▶ Among the search results, open the link that specifies: <https://mvnrepository.com/>.
- ▶ Copy paste the dependencies in **pom.xml** file inside **<dependencies>** tag
- ▶ The project will **automatically** update its Maven Dependencies
- ▶ If there is no automatic update, right click on the project, choose **Maven>Update** manually

Note: It may happen while updating Maven dependencies that our project might show a different java compiler version. In order to correct this, specify the following inside properties tag in pom.xml:

```
<properties>  
<maven.compiler.source>1.11</maven.compiler.source>
```

```
<maven.compiler.target>1.11</maven.compiler.target>

<properties>
```

Step 3: Create a POJO class

Hibernate Framework maps a class-object with RDBMS. In order to do so, we must create a POJO class.

- ▶ Create a POJO class (say **Student**) in **src/main/java** folder in a package (say **com.hibernate.pojo**)
- ▶ Create 3 fields: **private int id**, **private String name** and **private int section**.
- ▶ Annotate class Student with **@Entity** and id with **@Id**.
- ▶ Be sure to check the above annotations belong to **javax.persistence** package.

Step 4: Create Hibernate Configuration File

Hibernate will take our object and use it to transact with database, as such we need to provide some **connection details** as well as some **hibernate specific configurations**

We will use **XML** to do the configurations

- ▶ Step 1: Make an xml file in **src/main/java** folder
- ▶ Step 2: Its name should be **hibernate.cfg.xml**
- ▶ Step 3: Write **Hibernate DTD 3.0** (from <http://hibernate.org/dtd/>) as its first tag

```
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

Note:

3. Please keep in mind that DTD versions do NOT match the Hibernate ORM version.
 4. Just use the **highest** DTD version that is **lower than or equal to** your Hibernate ORM version.
- ▶ Step 4: Put the parent tags

```
<hibernate-configuration>
  <session-factory>
    </session-factory>
</hibernate-configuration>
```

- ▶ Step 5: Write down the following properties inside **<session-factory>** tag

```
<property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
```

```
<property  
name="connection.url">jdbc:mysql://localhost:3306/hibernate_demo</property>  
<property name="connection.username">root</property>  
<property name="connection.password">1234</property>  
<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>  
<property name="hbm2ddl.auto">update</property>  
<property name="show_sql">true</property>  
<property name="format_sql">true</property>
```

► Step 6: Tell Hibernate about Entity classes

Copy the fully qualified name of the Entity class (Student class in our example) and mention it in **<mapping>** tag inside **<session-factory>** tag after all the **<property>** tags

```
<mapping class="com.hibernate.pojo.Student"/>
```

The configuration file is complete

Step 5: Create an Execution Class

Write the following code

```
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
public class App {  
    public static void main(String[] args) {  
        // Load Configuration file  
        Configuration configuration = new Configuration();  
        configuration.configure();  
  
        // Create a session factory  
        SessionFactory sessionFactory = configuration.buildSessionFactory();  
  
        // Create a session  
        Session session = sessionFactory.openSession();
```

```
// Start Transaction  
Transaction transaction = session.beginTransaction();  
  
// Create Student object to save  
Student student = new Student(101, "Kaustubh", 5);  
  
// Save the student object  
session.save(student);  
  
// Commit the transaction  
transaction.commit();  
  
// Close Resources  
session.close();  
sessionFactory.close();  
}  
}
```

Step 6: Execute the code

- ▶ Make sure the MySQL Server is running with the config data mentioned in config file
- ▶ Make sure there is a schema: hibernate_demo in place
- ▶ Run the application as a Java Application

Note: We will learn about the above code in later classes

Hibernate Architecture

Table of Contents

<u>Hibernate Architecture</u>	40
<u>High Level Architecture</u>	40
<u>Block Diagram</u>	41
<u>Core Objects of Hibernate API</u>	41
<u>Internal APIs used by Hibernate</u>	41
<u>Elements of Hibernate Architecture</u>	42
<u>Configuration</u>	42
<u>Significance</u>	43
<u>SessionFactory</u>	43
<u>Immutability and Thread Safety</u>	44
<u>Significance</u>	44
<u>Session</u>	44
<u>Short Lived</u>	44
<u>Light Weight</u>	45
<u>Thread Safety</u>	45
<u>IDB Connection v/s Session</u>	45
<u>Transaction</u>	45
<u>Methods of Transaction interface</u>	46
<u>Query</u>	46

[Criteria](#).....47



Hibernate Architecture

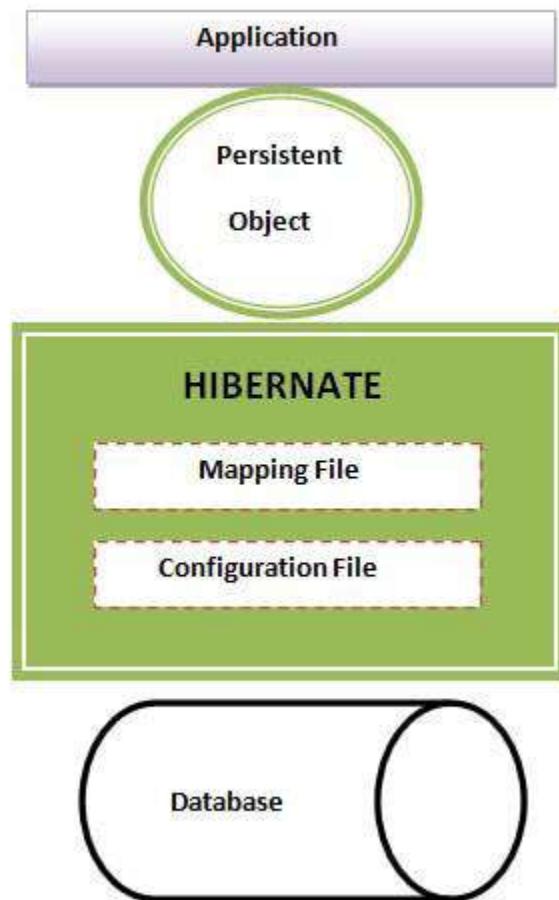
The Hibernate **architecture** includes many objects such as **persistent object, session factory, transaction factory, connection factory, session, transaction** etc.

The Hibernate architecture is categorized in four layers.

1. Java application layer
2. Hibernate framework layer
3. Backend API layer
4. Database layer

Let's see the diagram of hibernate architecture:

High Level Architecture



Java Application Layer: The layer where we write Java Code

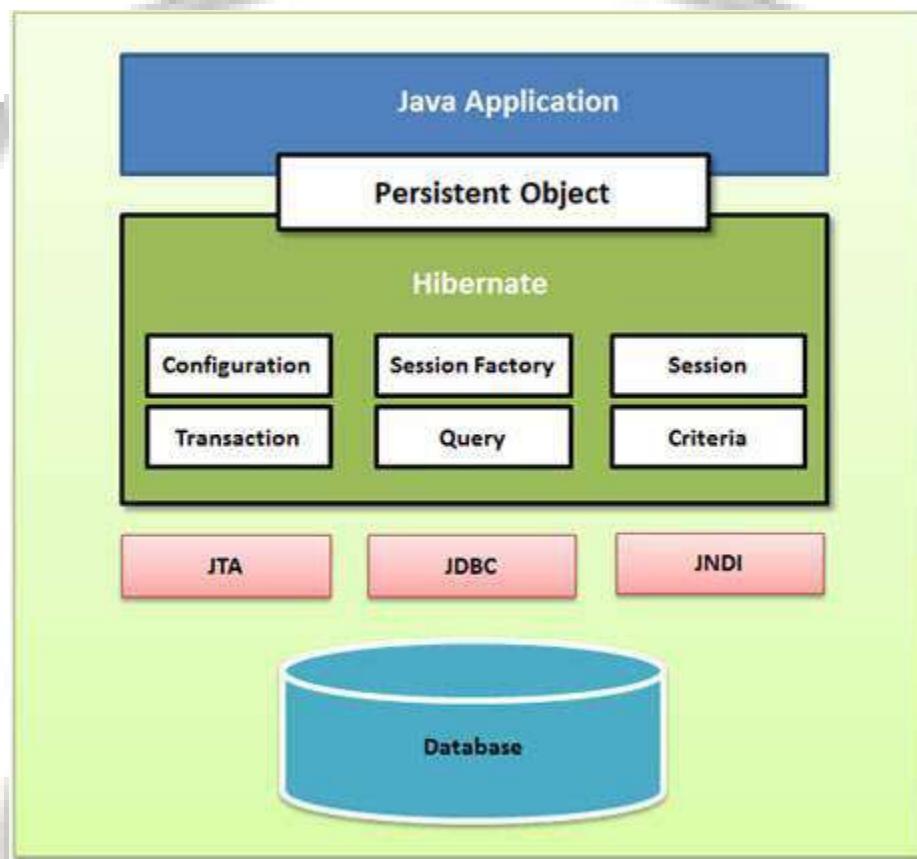
Hibernate Framework Layer: It consists the Persistent Object that is mapped with the database
pg. 40

Backend API Layer: It consists of the mapping file and configuration file

Database layer: The targeted database

Block Diagram

The following is the block diagram of Hibernate



Core Objects of Hibernate API

The following are the core objects of the Hibernate API:

1. Configuration
2. Session Factory
3. Session
4. Transaction
5. Query
6. Criteria

Internal APIs used by Hibernate

1. **JDBC** (Java Database Connectivity): JDBC is the standard API that Java applications use to interact with a database. Hibernate uses JDBC internally to insert a Java object into database

2. **JTA** (Java Transaction API): It allows applications to perform distributed transactions, that is, transactions that access and update data on two or more networked computer resources.
3. **JNDI** (Java Naming and Directory Interface)

Elements of Hibernate Architecture

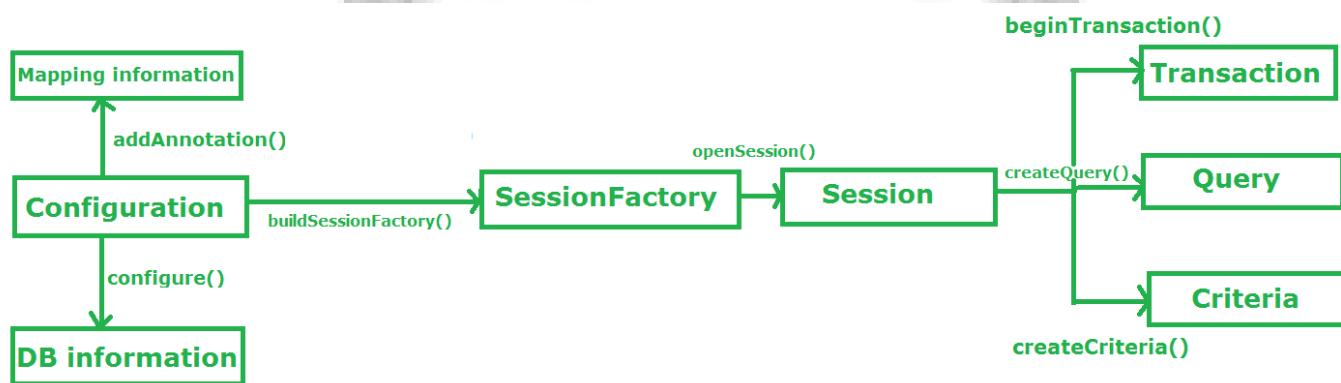


Fig: Hibernate Architecture

While creating our first hibernate application, we made use of the various classes and interfaces. Let's look at them one by one briefly.

Configuration

- ▶ Configuration is a class
- ▶ It is present in **org.hibernate.cfg** package.
- ▶ The below statement activate Hibernate Framework

```
Configuration configuration = new Configuration();
```

- ▶ The below statement **reads** both, the configuration file (hibernate.cfg.xml) & mapping files (Student entity)

```
configuration.configure();
```

- ▶ The **configure()** checks for the configuration file “**hibernate.cfg.xml**” at “**src/main/java**” location, if it can't find one it will throw a **ConfigurationException: Could not locate cfg.xml resource**
- ▶ In case our configuration file has a **different name** or is at a **different location**, we can use the **overloaded versions** of **configure()**

- It has got 5 overloaded methods:

S. No	Method Signature (All return Configuration)	Task – Loads configurations in memory
1.	configure()	Locates hibernate.cfg.xml in java folder
2.	configure(String resource)	Locates the resource from specified path
3.	configure(URL url)	Locates the resource from specified URL
4.	configure(File configFile)	Locates the resource from specified File
5.	configure(org.w3c.dom.Document document)	Deprecated

- If the resource is valid then config() **creates a meta-data** in **memory** and returns the meta-data to **configuration** object to **represent the config file**.
- Configuration class invokes **buildSessionFactory** to provide meta data to SessionFactory object

Significance

- It is the first object we create in a Hibernate Application
- It is created only once during application initialization.
- The configuration object provides 2 key components:
 1. **Database Connection:** This is handled through one or more configuration files supported by Hibernate: These files are **hibernate.properties** or **hibernate.cfg.xml**
 2. **Class mapping Setup:** This component creates the connection between the **Java Classes** and **database tables**

SessionFactory

- SessionFactory is an **Interface**
- It is present in **org.hibernate** package
- To get an object of SessionFactory, we call **buildSessionFactory()** on Configuration object.
- This method creates a SessionFactory using the **properties** and **mappings** in the configuration file.
- buildSessionFactory() takes JDBC information from the configuration object & **creates a JDBC Connection**.

```
SessionFactory sessionFactory = configuration.buildSessionFactory();
```

- The SessionFactory is **immutable** and **thread-safe** in nature.
- So changes made to this Configuration after building the SessionFactory will not affect it.
- SessionFactory holds **second level cache** of data.

- ▶ Throws exception on invalid **configuration** or invalid **mapping information**
- ▶ The SessionFactory interface provides **openSession()** method to get the object of Session.

Immutability and Thread Safety

Most problems with concurrency occur due to sharing of objects with mutable state. Once the object is immutable, its internal state is settled on creation and cannot be changed. So many threads can access it concurrently and request for sessions

However, Session is a **non-thread safe object**; you cannot share it between threads.

Significance

- ▶ It **configures Hibernate** for the application
- ▶ We will need **one** SessionFactory object **per database** using a **separate configuration file**
- ▶ So, SessionFactory must be maintained in the application as a **Singleton Object**
- ▶ If our application **connects to more than 1 database**, we can have **multiple** config files and so, **multiple and related** Configuration and SessionFactory objects

Session

- ▶ Session is an **interface**
- ▶ It is present in **org.hibernate** package.
- ▶ Session object is created when SessionFactory object calls **openSession()** method
- ▶ It **opens the Session** with Database software through Hibernate Framework.
- ▶ The session object provides an interface between the application and data stored in the database.
- ▶ It is a **short-lived** object and wraps the **JDBC connection**.
- ▶ It holds a **first-level cache** of data.
- ▶ The Session interface is used to perform **CRUD** operations through insert, update, delete methods
- ▶ It also provides **factory methods** for Transaction, Query and Criteria.
- ▶ It is a **light-weight** object and it is **not thread-safe**.

```
Session session = sessionFactory.openSession();
```

Short Lived

- ▶ Having **long running transactions** is a **bad** thing as it may cause **concurrency problems**
- ▶ The session objects **should not be kept open for a long time** because they are not usually thread safe and they should be created and destroyed as needed.

- The main function of the Session is to offer, create, update, read, and delete operations for instances of mapped entity classes.

Light Weight

- The Session object is **lightweight** and designed to be instantiated each time an interaction is needed with the database.
- Persistent objects are saved and retrieved through a Session object.

Thread Safety

- Session is not Thread Safe
- Session represents a single threaded unit of work.
- The Hibernate session is a complex object that is highly stateful (it caches objects; synchronize its internal representation with the DB, etc.).
- This is just a warning that if you share a session across different threads, 2 threads could be calling methods at the same time in a way that messes up the internal state of the session and cause bugs.
- Hibernate has no way to "detect" that 2 threads are accessing it and may not throw exceptions. It's just not designed for it.
- **Program running on a single thread is simple:** everything runs sequentially, so behaviors are very predictable.

JDBC Connection v/s Session

JDBC Connection

A JDBC connection is the physical communication channel between the Database Server and the application: the TCP socket, the named pipe, the shared memory region etc.

Session

Session is a state of information exchange. A **session** encapsulates **user interaction with the database**, from the moment user was **authenticated** until the moment the user **disconnects**.

It could be thought of as an HTTP Session.

Note: A Connection may have multiple sessions

Transaction

- It is an **interface**.
- It is present in **org.hibernate** package

- ▶ The transaction object specifies an **atomic** unit of work.
- ▶ It is optional (used only when database **modifying** query is executed)
- ▶ The Transaction **interface** provides methods for transaction management.
- ▶ It is used during the queries that changes the content of the database
- ▶ It uses **commit()** to make permanent changes to database

```
Transaction transaction = session.beginTransaction();

// Create Update Delete Operation

transaction.commit();
```

Methods of Transaction interface

S. No	Method	Description
1.	void begin()	starts a new transaction.
2.	void commit()	ends the unit of work unless we are in FlushMode.NEVER.
3.	void rollback()	forces this transaction to rollback.
4.	void setTimeout(int seconds)	It sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5.	boolean isAlive()	Checks if the transaction is still alive.
6.	void registerSynchronization(Synchronization s)	Registers a user synchronization callback for this transaction.
7.	boolean wasCommitted()	boolean wasCommitted()
8.	boolean wasRolledBack()	boolean wasRolledBack()

Query

- ▶ Query is an **interface**
- ▶ It is present in **org.hibernate** package.
- ▶ A Query instance is obtained by calling **session.createQuery()**.
- ▶ This interface exposes some extra functionality beyond that provided by **Session.iterate()** and **Session.find()**
- ▶ A particular page of the result set may be selected by calling **setMaxResults()**, **setFirstResult()**.
- ▶ Named query parameters may be used.
- ▶ Query query = session.createQuery();

Criteria

- ▶ Criteria is an **interface**
- ▶ It is present in **org.hibernate** package
- ▶ Criteria is a simplified API for retrieving entities by composing Criterion objects.
- ▶ The Session is a factory for Criteria.
- ▶ Criterion instances are usually obtained via the factory methods on Restrictions.
- ▶ Criteria criteria=session.createCriteria()

Note: In order to learn for above classes and interfaces, [you should refer to JBoss docs.](#)

Hibernate Annotations

Table of Contents

Hibernate Annotations	49
All Hibernate Annotations	49
Key Annotations	50
@Entity	50
Syntax	50
Attributes	Error! Bookmark not defined.
@Table	50
Syntax	50
Attributes	50
@Column	50
Syntax	50
Attributes	51
@Id	51
Syntax	51
Attributes: No attributes	51
@GeneratedValue	51
Syntax	51
Attributes	51
@Transient	52
@Temporal	52
Note	52
@Lob	52
@Embeddable	52

Hibernate Annotations

When we start learning and using Hibernate and JPA, the number of annotations might be **overwhelming**. But as long as we rely on the defaults, we can implement our persistence layer using only a small subset of them.

All Hibernate Annotations

@AccessType	@Generated	@NotFound
@Any	@GeneratedValue	@OneToOne
@AnyMetaDef	@GeneratorType	@OneToMany
@AnyMetaDefs	@GenericGenerator	@OrderBy
@AttributeAccessor	@GenericGenerators	@ParamDef
@BatchSize	@Id	@Parameter
@Cache	@Immutable	@Sort
@Cascade	@Index	@SortComparator
@Check	@IndexColumn	@SortNatural
@CollectionId	@JoinColumn	@Source
@CollectionType	@JoinColumnOrFormula	@SQLDelete
@ColumnDefault	@JoinColumnsOrFormulas	@SQLDeleteAll
@Column	@ListIndexBase	@SqlFragmentAlias
@Columns	@Loader	@SQLInsert
@ColumnTransformer	@Lob	@SQLUpdate
@ColumnTransformers	@ManyToMany	@Subselect
@CreationTimestamp	@ManyToOne	@Synchronize
@DiscriminatorFormula	@ManyToMany	@Table
@DiscriminatorOptions	@MapKeyType	@Tables
@DynamicInsert	@MetaValue	@Target
@DynamicUpdate	@NamedNativeQueries	@Temporal
@Entity	@NamedQueries	@Transient
@Embed	@NamedQuery	@Tuplizer
@Embeddable	@Nationalized	
@Fetch	@NaturalId	

@FetchProfile	@NaturalIdCache	
---------------	-----------------	--

Key Annotations

@Entity

- ▶ The JPA specification **requires** the @Entity annotation.
- ▶ It identifies a class as an entity class.
- ▶ It is used to mark any class as an **entity**. With this we tell Hibernate that a **table has to be made** with this Entity.

Syntax

```
@Entity
public class Author { ... }
```

@Table

By default, each entity class maps a database table with the **same name** in the default schema of your database. We can customize this mapping using the *name* attribute of the *@Table* annotation.

Syntax

```
@Entity
@Table(name = "AUTHORS")
public class Author { ... }
```

Key Attribute

1. Name: It enables us to change the name of the database table which our entity maps.

@Column

The @Column annotation:

1. Is an optional annotation
2. Enables us to customize the mapping between the entity **attribute** and the database **column**.
3. For example, we can set
 - a. A different column name
 - b. Nullable property
 - c. Length property

Syntax

```
@Entity
public class Book {

    @Column(name = "title", length="50", nullable="false")
```

```
private String title;  
...  
}
```

Key Attributes

1. **Name:** The name of the column. Defaults to the property or field name. (Optional)
2. **Nullable:** Whether the database column is nullable.(Optional)
3. **length:** The column length. (Applies only if a string-valued column is used.)(Optional)

@Id

JPA and Hibernate requires us to specify **at least one primary key** attribute for each entity. We can do that by annotating an attribute with the @Id annotation.

Syntax

```
@Entity  
public class Author {  
  
    @Id  
    private Long id;  
  
    ...  
}
```

Attributes: No attributes

@GeneratedValue

Hibernate will automatically generate values for that using an internal sequence. Therefore we don't have to set them manually

Syntax

```
@Entity  
public class Address {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column()  
    private int addressId;  
  
}
```

Attributes

1. **strategy:** specifies the key generation strategy

@Transient

It may happen sometime that we want a data member but we don't want its column to be created, in this case we use **@Transient** on the instance variable. This indicates to hibernate **not to save these fields**.

@Temporal

- ▶ If we use classes Date or Calendar as your Entity property then it prints a lot of information.
- ▶ Sometime all of this information is not necessary when we want to save it to database.
- ▶ **@Temporal** solves this problem and tells hibernate the format in which the date needs to be saved
 - **@Temporal(TemporalType.TIMESTAMP)**: To store timestamp to your database
 - **@Temporal(TemporalType.DATE)**: To store date
 - **@Temporal(TemporalType.TIME)**: To store time

Note:

- ▶ Since **Hibernate 5** you don't need and **should not use @Temporal** in new code.
- ▶ It was for annotating Date and Calendar fields which are now **deprecated**.
- ▶ Instead use classes from `java.time`, the modern Java Date and time API
- ▶ They have their expected precision built in, and you don't use that annotation with them

@Lob

This annotation specifies that a column be created to hold a large object such as an image or a document

@Embeddable

It tells hibernate to embed an object in another object

Note: We will learn about key annotations topic wise as the course progresses

Entity Life Cycle

Table of Contents

Hibernate – Entity Life Cycle	54
Entity Lifecycle States	54
Transient	54
Persistent	55
Detached	55
Removed	Error! Bookmark not defined.
Conclusion	56
What is CRUD?	57
Instance State:	Error! Bookmark not defined.
Create	Error! Bookmark not defined.
Retrieve	Error! Bookmark not defined.
Update	Error! Bookmark not defined.

Hibernate - Entity Life Cycle

- ▶ We know that Hibernate works with Plain Java objects (**POJO**).
- ▶ In raw form (without hibernate specific annotations), hibernate won't be able to identify these java classes.
- ▶ But when these POJOs are properly annotated with required annotations they are said to be **mapped with hibernate**
- ▶ Only then hibernate will be able to identify them and work with them e.g. store in the database, update them, etc.

Hibernate Session

- ▶ The Session interface is the main tool used to communicate with Hibernate.
- ▶ It provides an API enabling us to **create, read, update, and delete** persistent objects.
- ▶ The *session* has a simple lifecycle. We open it, perform some operations, and then close it.
- ▶ When we operate on the objects during the *session*, they get attached to that *session*.
- ▶ The changes we make are detected and saved upon closing.
- ▶ After closing, Hibernate breaks the connections between the objects and the session.

Entity Lifecycle States

In the context of Hibernate's *Session*, instance of a class that is **mapped to Hibernate** can be in any one of the following three possible persistence states, also known as **hibernate entity lifecycle states**

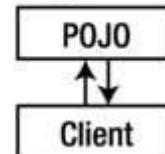
1. Transient
2. Persistent
3. Detached

Transient

- ▶ An object we haven't attached to any **session** is in the transient state.
- ▶ Transient entities exist in heap memory as normal Java objects.
- ▶ Hibernate does not manage transient entities or persist changes done on them.

```
Session session = openSession();
UserEntity userEntity = new UserEntity("Kaustubh");
```

Transient Object



To persist the changes to a transient entity, we would have to ask the hibernate session to save the transient object to the database, at which point Hibernate assigns the object an identifier and marks the object as being in persistent state.

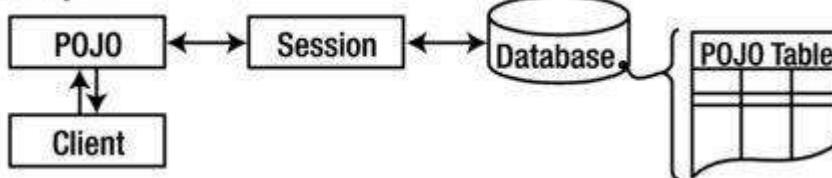
Persistent

- ▶ An object that we've associated with a **session** is in the persistent state.
- ▶ We either **saved it or read it from** a persistence context, so it **represents some row** in the database.
- ▶ Persistent entities exist in the database, and Hibernate manages the persistence for persistent objects.

```

Session session = openSession();
UserEntity userEntity = new UserEntity("John");
session.persist(userEntity);
  
```

Persistent Object



If fields or properties change on a persistent object, Hibernate will keep the database representation up to date when the application marks the changes as to be committed.

Detached

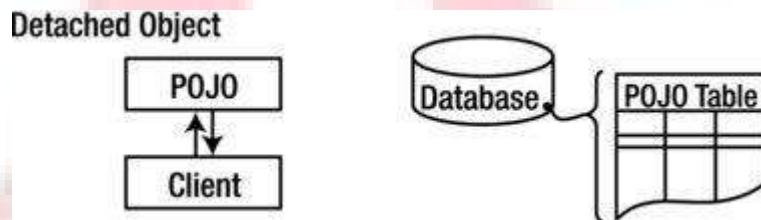
- ▶ A detached entity can be created by closing the session that it was associated with, or by evicting it from the session with a call to the session's evict() method.

- When we close the *session*, all objects inside it become detached.

```
session.persist(userEntity);

session.evict(userEntity);
OR
session.close();
```

- Although they still represent rows in the database, they're no longer managed by any session
- That is, changes to the entity will not be reflected in the database, and vice-versa.
- This temporary separation of the entity and the database is shown in the image below.



- In order to persist changes made to a detached object, the application must re-attach it to a valid Hibernate session.
- A detached instance can be associated with a new Hibernate session when your application calls one of the load(), refresh(), merge(), update(), or save() methods on the new session with a reference to the detached object.
- After the method call, the detached entity would be a persistent entity managed by the new Hibernate session.

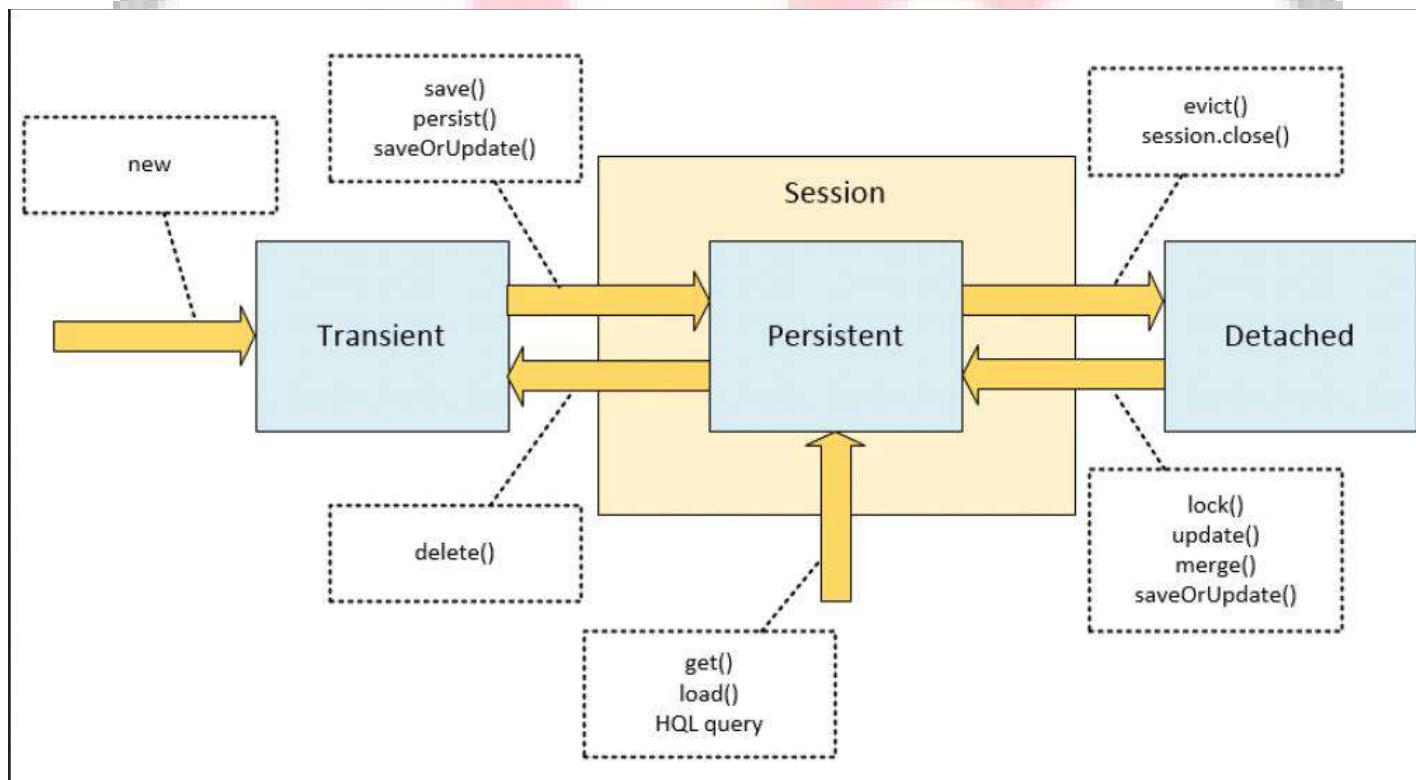
Conclusion

1. Newly created POJO object will be in the transient state. Transient entity doesn't represent any row of the database i.e. not associated with any session object. It's plain simple java object.
2. Persistent entity represents one row of the database and always associated with some unique hibernate session. Changes to persistent objects are tracked by hibernate and are saved into database when commit call happen.

3. Detached entities are those who were once persistent in the past, and now they are no longer persistent. To persist changes done in detached objects, you must re-attach them to hibernate session.

CRUD with Hibernate

Here is a simplified state diagram with comments on *Session* methods that make the state transitions happen.



Caching Mechanism in Hibernate

Table of Contents

Cache Mechanism in Hibernate	59
How Caching works?	59
Types of Cache	59
First Level Cache	59
Lifecycle	61
Accessibility	61
Working	61
Manipulation	61
Second Level Cache	61
Lifecycle	61
How second level cache works	62

Cache Mechanism in Hibernate

1. Caching is a facility provided by Hibernate:
 - a. To help users to get fast running web application,
 - b. To help framework itself to reduce number of queries made to database in a single transaction.
2. Caching is a mechanism to enhance the performance of an Application
3. Caching is implemented by Hibernate to reduce the hits in the database

How Caching works?

When our application queries the database for an object using its primary key for the first time, hibernate executes the query, fetches data, prepares an object and saves it in cache.

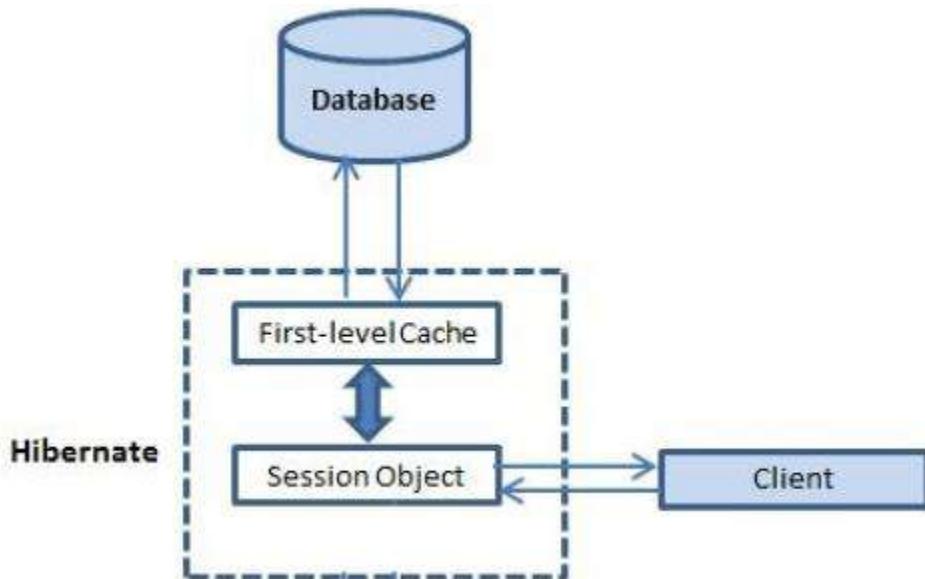
Now, if we fire a 2nd query demanding the same object, hibernate instead of querying the database fetches the stored object in the cache and returns it to the application, thereby saving 1 extra query

Types of Cache

There are 2 types of caching in Hibernate:

1. First Level cache (also known as **L1 cache**)
2. Second Level cache (also known as **L2 cache**)

First Level Cache





Lifecycle

The scope of L1 cache objects is of **session**.

1. First level cache is associated with “session” object.
2. The first level cache comes into existence as soon as a session object is created from session factory.
3. This L1 cache is not available to other session objects in the application but only to its own
4. The first level cache associated with its session object is available only till its session’s object is live.
5. Once session is closed, cached objects are gone forever.

Accessibility

1. First level cache is enabled by default and we cannot disable it, even forcefully.
2. We don’t need to do anything special to get this functionality working.

Working

1. Hibernate stores an entity in a session’s first level cache when:
 - a. An object is inserted in database for the 1st time
 - b. An object is updated in database
 - c. An object is retrieved from database for the 1st time
2. If we query same object again with **same session object**, it will be loaded from cache and no SQL query will be executed.

Manipulation

1. The loaded entity can be removed from session using **evict (Object object)** method.

Now if we try to retrieve the same object, hibernate will make a call to database

2. The whole session cache can be removed using **clear ()** method.

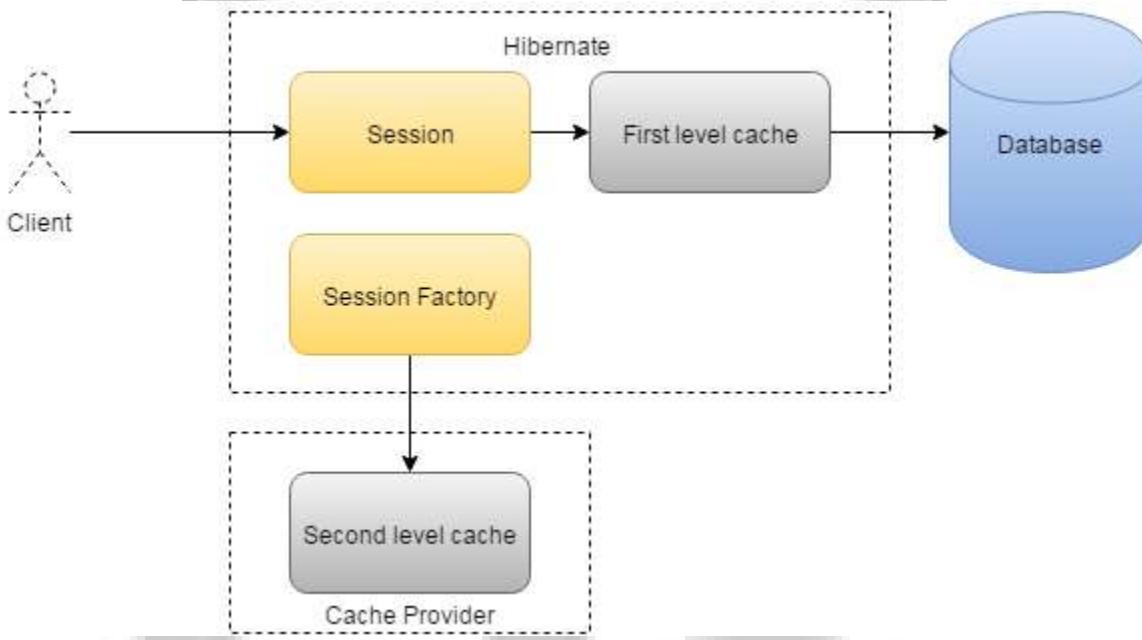
It will remove all the entities stored in cache.

Second Level Cache

Lifecycle

1. Second Level cache or Level 2 cache is associated with Session factory
2. It is created in session factory scope
3. It is available to be used globally in session factory scope.
4. It is available to be used in all sessions which are created using that particular session factory.

5. It also means that once session factory is closed, all cache associated with it die and cache manager also closed down.
6. It also means that if you have two instances of session factory (**normally no application does that**), you will have two cache managers in your application and while accessing cache stored in physical store, you might get unpredictable results.



How second level cache works

1. Whenever hibernate session try to load an entity, the very first place it look for cached copy of entity in first level cache (associated with particular hibernate session).
2. If cached copy of entity is present in first level cache, it is returned as result of load method.
3. If there is no cached entity in first level cache, then second level cache is looked up for cached entity.
4. If second level cache has cached entity, it is returned as result of load method. But, before returning the entity, it is stored in first level cache also so that next invocation to load method for entity will return the entity from first level cache itself, and there will not be need to go to second level cache again.
5. If entity is not found in first level cache and second level cache also, then database query is executed and entity is stored in both cache levels, before returning as response of load() method.
6. Second level cache validate itself for modified entities, if modification has been done through hibernate session APIs.

7. However, Second level cache cannot validate itself for modified entities, if modification has been done in the background without hibernate session APIs.

Accessibility

1. We have to manually enable second level cache
2. We will have to include some jar and do some configurations to use it
3. There are many cache providers. We will use:
 - a. EH cache
 - b. OS Cache
 - c. Hibernate cache

How To?

Step 1: Include some dependencies in pom.xml

Step 2: Make the following changes in configuration file

Step 3: By default our entity is not cacheable; we have make our entity cacheable explicitly:

```
@Entity  
@Cacheable  
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)  
class Department {  
  
}
```

Get v/s Load

Table of Contents

<u>Fetch Data</u>	65
<u>Get V/s Load</u>	65

<CODEWITHARRAYS/>

Fetch Data

Hibernate Session class provides two methods to access object: get () and load ()

Both looked quite similar to each other but there are subtle differences between both of them which can affect the performance of the application.

Get V/s Load

Property	Get()	Load()
Behavior	Get method of hibernate session returns null if object is not found in cache as well as on database	Load method throws ObjectNotFoundException if object is not found on cache as well as on data base but never returns null
Database Hit	get() involves database hit if an object doesn't exist in Session Cache and returns a fully initialized object which may involve several database calls	load method can return proxy in place and only initialize the object or hit the database if any method other than getId() is called on persistent or entity object. This lazy initialization can save a couple of database round-trip which results in better performance.
Proxy	Get method never returns a proxy, it either returns null or fully initialized Object	load() method may return proxy, which is the object with ID but without initializing other properties, which is lazily initialize
Performance	In comparison with load, get() is low on performance	load() is better in performance
Usage	Use if you are not sure that object exists in data base or not	Use if you are sure that object exists

Spring Introduction

Table of Contents

<u>Framework</u>	68
<u>Why Framework?</u>	68
<u>What is a Software Framework?</u>	68
<u>Advantages of using a software framework</u>	68
<u>Java Frameworks</u>	68
<u>Spring</u>	69
<u>What is Spring?</u>	69
<u>Why to learn Spring Framework?</u>	69
<u>Features of Spring Framework</u>	70
<u>Lightweight</u>	70
<u>Dependency Injection (DI)</u>	70
<u>Inversion of Control (IOC)</u>	70
<u>So, what Spring does?</u>	71
<u>What makes Spring Framework special?</u>	71
<u>Spring is Open Source</u>	71
<u>Framework of Frameworks</u>	72
<u>Spring has evolved over time</u>	72
<u>Easy Development & Testing Cycles</u>	72
<u>History of Spring Framework</u>	72
<u>Spring Framework Architecture</u>	73
<u>Spring Core Container</u>	73
<u>Spring Data Integration and Data Access</u>	74
<u>Spring Web</u>	75
<u>Web Services and Rest API</u>	75
<u>Aspect-Oriented Programming (AOP)</u>	75
<u>Instrumentation</u>	75
<u>Messaging</u>	75
<u>Test</u>	76
<u>What are Spring Framework, Spring JDBC, Spring MVC and Spring Boot?</u>	76

<u>Spring Framework</u>	76
<u>Spring JDBC</u>	76
<u>Spring MVC</u>	76
<u>Spring Boot</u>	76
<u>Prerequisites</u>	76



Framework

Why Framework?

Developing software is a complex process. It contains a lot of tasks such as – architecture, coding, designing, testing, deployment, maintenance, scaling etc. For only the coding part, programmers had to take a lot of care in syntax, declarations, garbage collection, statements, error handling, space complexity, time complexity, development cycles and more.

It could be really nice if some readymade library or system was there to simplify these tasks. This is where frameworks come into picture. These frameworks help developers to minimize these headaches.

What is a Software Framework?

Software frameworks make life easier for developers by allowing them to take control of the entire software development process, or most of it, from a single platform.

Advantages of using a software framework

1. Assists in establishing better programming practices and fitting use of design patterns
2. Code is more secure
3. Duplicate and redundant code can be avoided
4. Helps consistent development of code with fewer bugs
5. Makes it easier to work on sophisticated technologies
6. Several code segments and functionalities are pre-built and pre-tested. This makes applications more reliable
7. Testing and debugging the code is a lot easier and can be done even by developers who do not own the code
8. The time required to develop an application is reduced significantly
9. With the reduction in time and labor, softwares are cost efficient

Java Frameworks

Java is one of the oldest languages with a huge community of developers. There are many frameworks of Java that are currently being used for various tasks, such as:

1. Spring
2. Apache Struts
3. Grails
4. Hibernate
5. JSF (Java Server Faces)
6. GWT (Google Web Toolkit)

7. Blade
8. Play
9. Vaadin
10. DropWizard
11. And many more...

Spring

What is Spring?

Spring is a **Java framework** used to develop Enterprise Applications.

Or, in other words:

Spring is a Framework used to develop Enterprise applications using the best practices of J2EE.

So, it is clear from above facts that:

1. Spring is a **Software Development framework**
2. Spring is a **Java based** software development framework
3. Spring is used to develop **Enterprise grade applications**

More on Spring

Spring is **not just one framework** – it is a framework of frameworks.

What it means is:

1. It consists of 20+ modules – all targeted to solve some particular problem of an enterprise grade application
2. It can work well with other Java Frameworks as well with easy integration techniques.

What is Spring?

- ▶ Spring is a Dependency Injection Framework used to make Java Applications loosely coupled
- ▶ Spring is a powerful, lightweight, open source application development framework used for developing Java Enterprise Edition (JEE) Applications.

Why should we learn Spring Framework?

- ▶ Spring is a big framework
- ▶ It is the most popular application development framework for enterprise Java.
- ▶ Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code.

Features of Spring Framework

Lightweight

The Spring Framework is very lightweight with respect to its size and functionality.

1. This is due to its POJO implementation, which doesn't force it to inherit any class or implement any interfaces.
2. We don't need all of Spring to use, only part of it. For example, we can use Spring JDBC without Spring MVC.
3. It follows modular approach: Spring provides various modules for different purposes; we can just use certain according to our required module.

Dependency Injection (DI)

What is DI?

Dependency Injection is a design pattern. It is not just specific to Java, it is used in planning software architecture and therefore it can be applied in any language

1. Suppose there are 2 classes: class A and class B.
2. Now class B is a member of class A, i.e. class A has "has-a" relationship with class B.
3. Class A will be known as dependent class and,
4. Class B will be known as a dependency
5. When we inject (supply/assign) a value to B's reference in class A, it is known as injecting a dependency.

Let us see an example: [DIExample](#)

Why worry about DI?

Tight Coupling

As we saw in the example, a change in class B forced us to change the coding of class A. This interdependency of classes is known as coupling and when one change forces us to change other classes' code, it is known as tight coupling. In tight coupling, we have to change the code in both the classes and recompile the source

Loose Coupling

When we build large applications, scalability and maintainability are important aspects to keep in mind. This is where design patterns come into picture. These design patterns help us to develop an application that is easy to maintain and scale. With the help of Spring Framework, we can develop loosely coupled applications where the change in one class does not affect the other class. In loose coupling, we only need to recompile the changed code

Inversion of Control (IOC)

Spring Framework takes control of creating objects and injecting dependencies itself dynamically on run time, i.e. it uses the Dependency Injection design pattern to loosely couple

the application code. This is known as Inversion of Control (control is shifted from programmer to Spring Framework).

Spring/IOC/DI Container

Spring Framework provides us with Spring/IOC/DI container:

- ▶ To create and manage the life cycle of Java Beans
- ▶ To manage configuration of application objects.
- ▶ To achieve Dependency Injection
- ▶ Spring Container is also known as **IOC Container or DI container**
- ▶ With the help of either of the following ways, we can specify dependencies in a project:
 - Configuration through XML
 - Configuration through Java Class
 - Configuration through Annotations

So, what Spring does?

Suppose we have following layers & classes in our JEE application:

Layer	Classes
UI Layer	HTML/JSP Page
Business Layer	ProductController class
Service Layer	ProductService class
Data Access Layer	ProductDao, Product Model class
DataBase	Data

- ▶ All the above dependencies will be managed by Spring
- ▶ Spring will inject the object of:
 - ✓ Product object in ProductDao class
 - ✓ ProductDao object in ProductService class
 - ✓ ProductService object in ProductController class
- ▶ We will have to just maintain config data in xml or annotation

What makes Spring Framework special?

Spring is Open Source

- ▶ Spring is freely available. We can learn it, develop in it with absolutely no cost up front
- ▶ Strong community to support the developers
- ▶ Continuous updates
- ▶ Wide adaptation in the industry

Framework of Frameworks

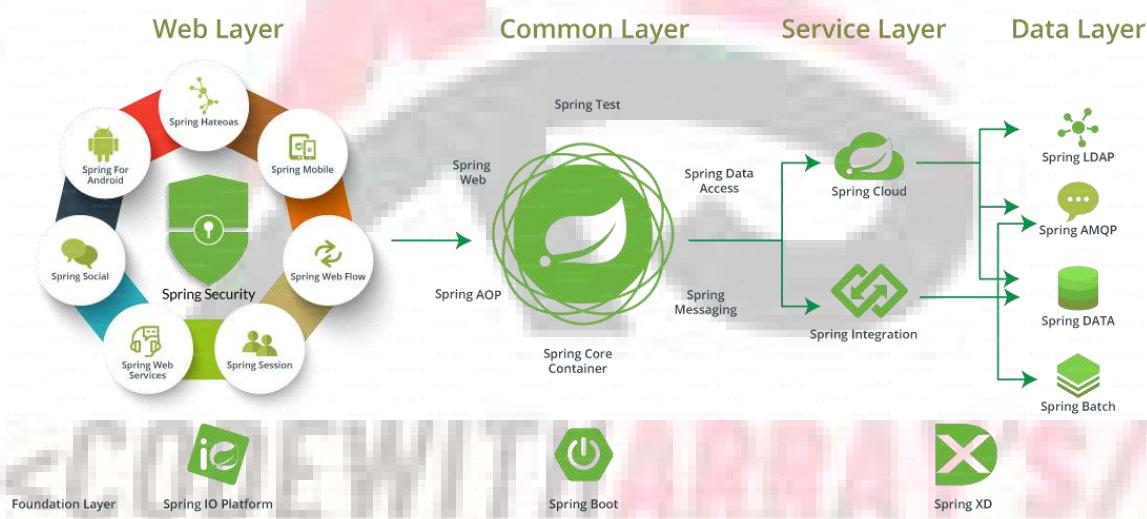
Spring is not just another java framework. It is a framework of frameworks. It provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc.

Spring has evolved over time

Since its origin till date, Spring has spread its popularity across various domains. Spring Framework now is the foundation for various other Spring Projects that have come up in the offerings in the last two to three years.

Easy Development & Testing Cycles

Spring Framework follows best practices for coding and testing thereby making the development of Java EE application easy.



History of Spring Framework

Spring was initially developed by Rod Johnson in 2002

1. The first version of the Spring framework was written by **Rod Johnson** along with a book – *“Expert One-on-One J2EE Design and Development”* in October 2002.
2. The framework was first released in **June 2003** under the **Apache license version 2.0**.
3. The first milestone release of **Spring framework (1.0)** was released in March **2004**.
4. Spring 2.0, which came in 2006, simplified the XML config files.
5. Spring 2.5, which came in 2007, introduced annotation configurations.
6. Spring 3.2, which came in 2012, introduced Java configuration, had support for Java 7, Hibernate 4, Servlet 3.0, and also required a minimum of Java 1.5.
7. Spring 4.0, which came in 2014, had support for Java 8.
8. Spring Boot also was introduced in 2014.

9. Spring 5.0 came out in 2017. Spring Boot 2.x has support for Spring 5.

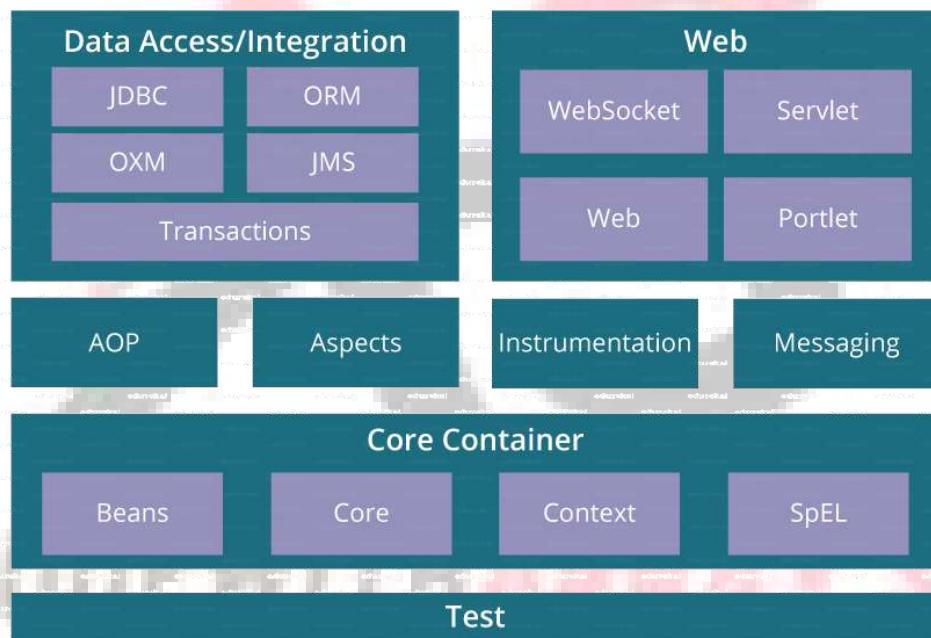
Official Website for Spring Framework: <https://spring.io/>

Spring Framework Architecture

Spring Framework architecture is an arranged layered architecture that consists of different modules. All the modules have their own functionalities that are utilized to build an application.

There are around **20 modules** that are generalized into **Core Container**, **Data Access/Integration**, **Web**, **AOP** (Aspect Oriented Programming), **Instrumentation**, and **Test**.

Here, the developer is free to choose the required module. Its modular architecture enables integration with other frameworks without much hassle.



Spring Core Container

This container has the following **four modules**:

Spring Core

This module is the core of the Spring Framework. It provides an implementation for features like IOC (Inversion of Control) and Dependency Injection with a singleton design pattern. The main concepts of Spring core are DI, IOC, Property Injection & Constructor Injection

Spring Bean

This module provides an implementation for the factory design pattern through BeanFactory. It is an important module and is used to inject objects in other class

Spring Context

Context module inherits features from bean module and adds to it: internationalization, event propagation, resource loading, and transparent creation of context.

This module is built on the solid base provided by the Core and the Bean modules and is a medium to access any object defined and configured.

Spring Expression Languages (SpEL)

This module is an extension to expression language supported by Java server pages. It provides a powerful expression language for querying and manipulating an object graph, at runtime.

Spring Data Integration and Data Access

It consists of the following five modules:

Spring JDBC

This module provides JDBC abstraction layer which eliminates the need of repetitive/hIDEOUS and unnecessary exception handling overhead JDBC code.

Spring ORM

ORM stands for Object Relational Mapping. This module provides consistency/portability to our code regardless of data access technologies based on object oriented mapping concept.

It provides integration of any other ORM tool in our spring application such as integration of hibernate to our code

Spring OXM

OXM stands for Object XML Mappers. It is used to convert the objects into XML format and vice versa. The Spring OXM provides a uniform API to access any of these OXM frameworks. It provides an abstraction layer to integrate with object xml mapping tools such as Castor, Xstream etc.

Spring JMS

JMS stands for Java Messaging Service. This module contains features for producing and consuming messages among various clients/java applications.

Transaction

This module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs. All the enterprise level transaction implementation concepts can be implemented in Spring by using this module.

Spring Web

Web layer includes the following modules:

Web

This module uses servlet listeners and a web-oriented application context to provide basic web-oriented features to help us create web projects. We can use MVC, Rest APIs, Socket, Portlets, and Http Client etc. in the applications

Web-Servlet

This module contains Model-View-Controller (MVC) based implementation for web applications. It provides all other features of MVC, including UI tags and data validations.

Web-Socket

This module provides support for WebSocket based and two-way communication between the client and the server in web applications.

Web-Portlet

This module is also known as the Spring-MVC-Portlet module. It provides the support for Spring-based Portlets and mirrors the functionality of a Web-Servlet module.

Web Services and Rest API

Aspect-Oriented Programming (AOP)

AOP language is a powerful tool that allows developers to add enterprise functionality to the application such as transaction, security etc. It allows us to write less code and separate the code logic. AOP uses cross-cutting concerns, defines method interceptors, point cuts, code decoupling.

When we want to do something before or after function calls, this is known as method interceptor.

Instrumentation

This module provides class instrumentation support and class loader implementations that are used in certain application servers.

Messaging

It serves as a foundation to build a message based application. It uses annotations to do so. It could be used whenever we need messaging functionality in applications

Test

This module supports the testing of Spring components with JUnit or TestNG. It provides consistent loading of Spring **ApplicationContexts** and caching of those contexts. It also provides mock objects that we can use to unit test our code in isolation.

What are Spring Framework, Spring JDBC, Spring MVC and Spring Boot?

Spring Framework

Spring Framework is a collection of modules. Spring JDBC and Spring MVC are modules of Spring Framework.

Spring JDBC

Spring JDBC Framework takes care of all the low-level details starting from opening the connection, preparing and executing the SQL statement, processing exceptions, handling transactions, and finally closing the connection.

Spring MVC

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

Spring Boot

Over the past few years, due to added functionalities, the Spring framework has become increasingly **complex**. It requires going through a lengthy procedure in order to start a new Spring project. To avoid starting from scratch and save time, Spring Boot has been introduced. This uses the Spring framework as a foundation.

While the Spring framework focuses on providing flexibility to you, **Spring Boot aims to shorten the code length** and provide you with the easiest way to develop a web application. With annotation configuration and default codes, Spring Boot shortens the time involved in developing an application. It helps create a stand-alone application with less or almost zero-configuration.

Prerequisites

To learn Spring Framework, one must have knowledge of following technologies:

- Core Java - OOP concepts: class, objects, constructor, method, overloading, overriding
- JDBC: Data integration layer/Access layer for Spring JDBC

- Hibernate to learn Spring ORM: Spring Hibernate Integration
- Servlet and JSP: Spring Web MVC
- Important Web and Database related terms: HTML, CSS, JS, Bootstrap, MySQL/PostGres/any database => working knowledge



Spring Core

Table of Contents

<u>Dependency Injection</u>	79
<u>Spring IOC container</u>	79
<u>Bean</u>	79
<u>Spring Configuration Metadata</u>	80
<u>Types of Configurations</u>	81
<u>Types of Dependency Injections</u>	82
<u>What to Inject?</u>	82

www.codewitharrays.in

Dependency Injection

The main concept of spring core is dependency injection (DI) and inversion of control (IOC). DI is done with the help of IOC container

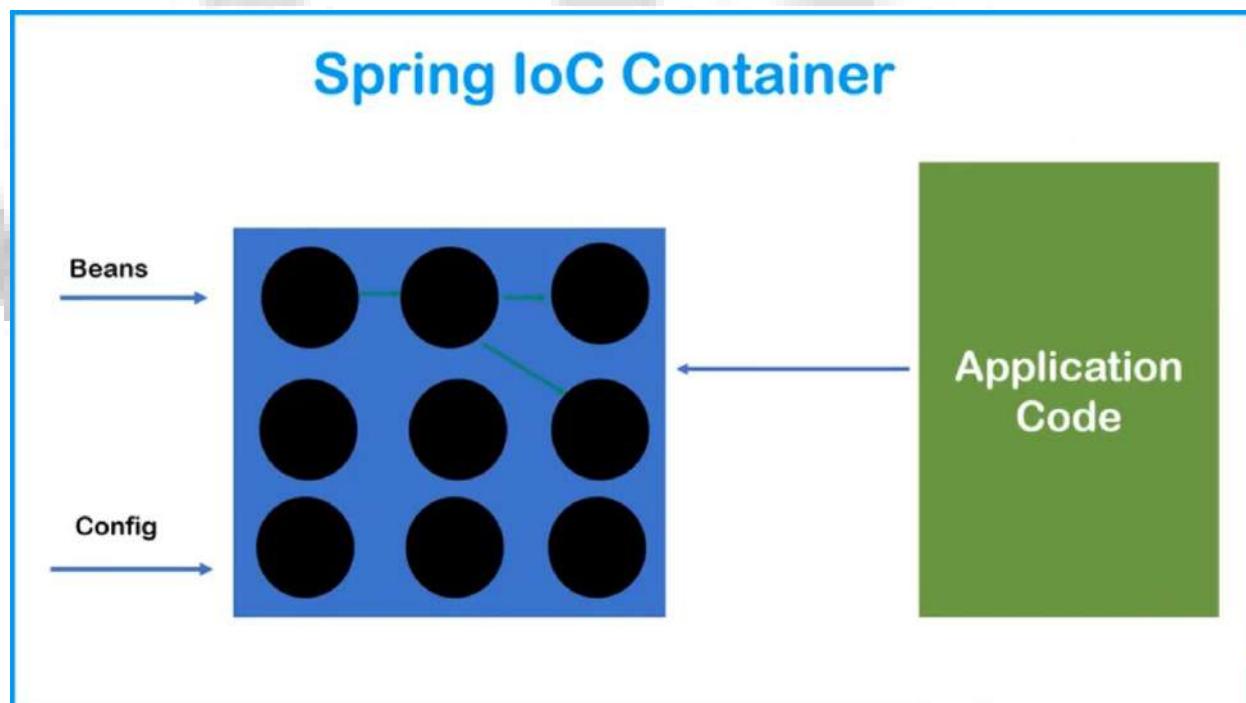
Spring IOC container

One of the main features of the Spring framework is the IoC (Inversion of Control) **container**. The Spring IoC container is responsible for managing the objects of an application. It uses dependency injection to achieve inversion of control.

We can say that Spring IOC container is a program/container in Spring framework whose main tasks are:

- ▶ Objects creation
- ▶ Holding objects in memory
- ▶ Inject an object in another object, i.e. DI
- ▶ Maintain Object lifecycle: creation to destruction

We need to tell IOC about dependencies and this we do with the help of beans & configurations. As soon as the dependencies are done, these are made available to the application code on demand



IOC API

The interfaces BeanFactory and ApplicationContext represent the Spring IoC container.

BeanFactory Interface

- ▶ BeanFactory is the root interface for accessing the Spring container.
- ▶ It provides basic functionalities for managing beans.

ApplicationContext Interface

- ▶ ApplicationContext is the sub-interface of the BeanFactory.
- ▶ Hence, it offers all the functionalities of BeanFactory.
- ▶ ApplicationContext provides more enterprise-specific functionalities.
- ▶ The important features of ApplicationContext are:
 - Resolving messages,
 - Supporting internationalization,
 - Publishing events, and
 - Application-layer specific contexts.

This is why we use ApplicationContext as the default Spring container.

Bean

Before we dive deeper into the *ApplicationContext* container, it's important to know about Spring beans.

- ▶ The objects that form the backbone of a Spring application and that are managed by the Spring IOC container are called **beans**.
- ▶ A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IOC container.
- ▶ These beans are created with the configuration metadata that is supplied to the container, such as:
 - Fully qualified name of the class
 - Name of the object that can be used as a reference
 - Property setters
 - Constructor setters
- ▶ The Spring configuration XML file has `<beans></beans>` tag as the root tag. All the other beans must be nested inside the root tag inside individual `<bean></bean>` tag

So, should we configure all the objects of our application as Spring beans? Well, as a best practice, we should not.

As per Spring documentation, in general, we should define beans for:

1. Service layer objects,
2. Data access objects (DAOs),

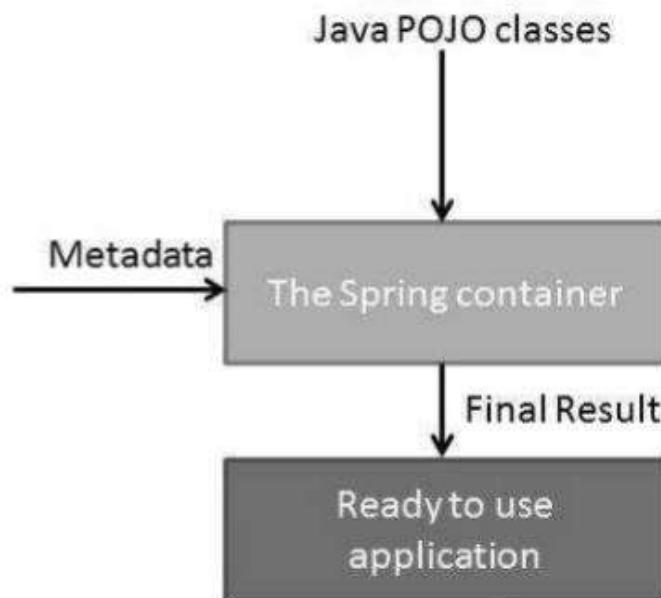
3. Presentation objects,
4. Infrastructure objects such as Hibernate *SessionFactories*,
5. JMS Queues, and so forth.

Spring Configuration Metadata

The application needs to provide bean configuration to the IoC container (ApplicationContext Container) about what beans/classes are to be injected in other classes. Therefore, a Spring bean configuration consists of one or more bean definitions.

Bean definition contains the information called as **configuration metadata**, which is needed for the container to know the following –

- ▶ How to create a bean
- ▶ Bean's lifecycle details
- ▶ Bean's dependencies



Also, Spring supports different ways of configuring beans.

Types of Configurations

Following are the three important methods to provide configuration metadata to the Spring Container –

- ▶ XML based configuration file.
- ▶ Annotation-based configuration
- ▶ Java-based configuration

Types of Dependency Injections

There are 2 types of dependency injections:

1. Property Injection (also known as Setter Injection)
2. Constructor Injection

What to Inject?

1. Primitive Data Types: Setter Injection/Constructor Injection
2. Reference Type: Setter Injection/Constructor Injection
3. Collections: Setter Injection/Constructor Injection

There are two basic types of containers in Spring – the Bean Factory and the ApplicationContext. The former provides basic functionalities, which are introduced here; the latter is a superset of the former and is most widely used

The application code can use these beans

ApplicationContext Interface

- ▶ ApplicationContext represents IOC. It extends BeanFactory
- ▶ ApplicationContext is an interface in the org.springframework.context package and it has several implementations
- ▶ We can fetch the objects from an IOC by the above interface by forming an object of its implementing class

Implementing Classes

1. **ClasspathXMLApplicationContext**
 - ▶ It searches XML configuration from Java class path
 - ▶ ClassPathXmlApplicationContext can load an XML configuration from a class-path and manage its beans
2. **AnnotationConfigApplicationContext**
 - ▶ It searches the beans on which we have used annotation.
3. **FileSystemXMLApplicationContext**
 - ▶ It searches config from a file.

We have different XML configurations for Development, Staging, QA, Production

Depending upon XML to initialize

=====

Ways of Injecting dependencies | Types of dependencies handled by IOC Container

=====

DI:

```
class Student {  
    int id;  
    String name;  
    Address address; // Dependency  
}
```

```
class Address {  
    String street;  
    String city;  
    String state;  
    String country;  
}
```

Spring Framework (IOC) will create an Address object and set the values of variables at runtime. It will also create the Student objects and its variables value at runtime.

Then we will ask the students and address's objects from IOC container.

IOC-creates the dependencies and inject automatically

OOC can do DI in 2 ways:

1. Using setter/property Injection
2. Using Constructor Injection

1. Using Setter/Property Injection

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

```
Address address; // Dependency

setId(id){}
setName(name){}
setAddress(address){}
}

class Address {
    String street;
    String city;
    String state;
    String country;
    setStreet(street){}
    setCity(city){}
    setState(state){}
    setCountry(country){}
}

}
```

IOC container will use these setter methods at runtime to set values after creating the objects automatically.

We can then ask IOC to provide us the objects

2. Using Constructor Injection

```
class Student {
    int id;
    String name;
    Address address; // Dependency
    Student(id, name, address){}
}
```

```
class Address {  
    String street;  
    String city;  
    String state;  
    String country;  
    Address(street, city, state, country){}  
}
```

Here the IOC will use the constructor to set values to the created objects

We can provide all this information by

1. XML Configuration
2. Java Configuration
3. Annotation Based Configuration

XML Configuration file

If we want to use DI feature with the help of IOC, We have to provide the details of the class in the configuration file only then IOC will maintain its lifecycle. It is an XML file where we declare beans and their dependencies.

Beans

The classes whose information is passed to an IOC are known as beans. These beans have to follow some rules:

```
<beans>  
    <bean>  
    </bean>  
</beans>
```

We need to specify the type of data that needs to be injected through IOC:

Data Types:

1. Primitive Data Types

byte, short, char, int, long, boolean, float, double

2. Collection Types

List, Set, Map and Properties: The IOC will be able to inject any List/... object in other class

3. Reference Type

Other class object

Topic 00: New Maven Project | Adding Spring Dependencies | Create Config File | Setter Injection

Softwares:

1. Eclipse/Netbeans/IntelliJ
2. Tomcat Server
3. MySQL for DB
4. SQL Yog / workbench / phpmyadmin

Steps to create a project:

1. Create Maven Project: quickstart archetype
 2. Adding dependencies=>spring core, spring context
 3. Creating beans=> Java POJO
 4. Creating configuration file=> config.xml
 5. Setting Injection
 6. Main class: which can pull the object and use
-

Topic 01: Property Injection - primitive

Property injection using p Schema and using value as attribute

Project: 01Injection

Package: property.injection.primitive

Topic 02: Property Injection - Reference | Injecting Reference Type

Project: 01Injection

Package: property.injection.reference

Topic 03: Property Injection Collections | Injecting Collection Types[List , Set , Map , Properties]

Project: 01Injection

Package: property.injection.collections

Topic 04: Standalone Collections

Spring Standalone Collections[List,Map,Properties] | Util Schema in Spring

Project: 01Injection

Package: standalone.collections

While creating a list/set in a bean, these collections are dependent, i.e they can't be used again.

Second, we dont specify the type of list/set that is injected

To overcome all this, we use standalone collections

Topic 05: Constructor Injection

Project: 01Injection

Package: constructor.injection

Topic 06: Constructor Ambiguity | Ambiguity Problem and its Solution with Constructor Injection

Project: 01Injection

Packages: constructor.ambiguity.one, constructor.ambiguity.two

Case 1:

If you observe the above output we didn't get expected result. i.e. second constructor (float,String) argument was not called, instead first constructor (String,String) argument was called. This is the Constructor injection type ambiguity.

Spring container by default converts every passing value to String value. i.e. In our example 10000 converted to String. That's why second constructor with (String,String) argument was called.

Case 2:

If you observe we didn't get expected output. i.e. third constructor with (float,String) argument was not called, instead second constructor with (String,float) argument was called. This is again ambiguity in Constructor injection type.

Solution:

We can resolve this problem by using index attribute of <constructor-arg> tag. So while using constructor injection always specify the exact datatype for constructor-arg value using type attribute and index attribute in beans.xml.

Topic 07: Stereotype Annotations | @Component Annotation | @Value Annotation

Spring provides us with three ways to define our beans and dependencies:

- XML configuration
- Java annotations
- Java Configuration Class

<context:component-scan base-package="bean.scope" />

This tag is needed when we use Annotations in Spring

Annotations:

@Component: to declare bean

@Value: to give value to a property

We have till now studied about declaring a bean in xml file for Spring/IOC container to inject objects.

In this example we will use stereotype annotations to declare a component and let IOC inject it with the help of

@Component and @value annotation

The default name of the object is camel case conversion of class name, however we can also mention a different name in @Component declaration

for ex: class Student will give student object.

If we wish other name, @Component("newName") must be used

Practical:

Inorder to inject a collection:

1. make a standalone collection in xml and give it values and id
2. Use spEL in class `#{<id>}` to mention collection

Project: 01Injection

Packages: stereotype.annotations

Topic 08: Spring Bean Scope | Singleton | Prototype | how to configure scope

Project: 01Injection

Packages: bean.scope.annotations

Bean Scope:

Spring Bean Scopes allows us to have more granular control of the bean instances creation. Sometimes we want to create bean instance as singleton but in some other cases we might want it to be created on every request or once in a session.

There are five types of spring bean scopes:

1. singleton
2. prototype
3. request
4. session
5. global-session

1. singleton – only one instance of the spring bean will be created for the spring container. This is the default spring bean scope.

whenever we configure a class in xml or through annotations, and ask the IOC to get the class instance, it will provide us with the same instance throughout the application. This is singleton scope. It is the scope of a bean set by default

2. prototype – To get different objects or to create a new instance every time,, we mention "prototype" scope

Spring MVC

Request and session scopes are used in web applications

3. request – This is same as prototype scope, however it's meant to be used for web applications. A new instance of the bean will be created for each HTTP request.

4. session – A new bean will be created for each HTTP session by the container.

5. global-session – This is used to create global session beans for Portlet applications.

Topic 09: Java Configuration (Removing Complete XML for Spring Configuration)|
@Configuration | @ComponentScan | @Bean Annotation

Project: 01Injection

package: com.java.configuration

There are 3 ways to configure beans:

1. XML
2. Annotations
3. Java Config Class

Java Config Class

@Configuration

Annotating a class with the @Configuration indicates that the class can be used by the Spring IoC container as a source of bean definitions.

@ComponentScan

Spring needs the information to locate and register all the Spring components with the application context when the application starts. Using @ComponentScan Spring can detect Spring-managed components.

@Bean

The @Bean annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

When we use @Bean on method name, the method name acts as the bean id: for example: "getSubject" & "getStudent"

However we can specify another name(s) for beans by using name array of @Bean annotation:
Example @Bean(name={"name1","name2"})

Topic 10: Spring Bean Lifecycle

Life Cycle Methods: Spring provides 2 important methods to every bean

1. public void init(): Initialization code - Loading config, connecting db, webservices etc
2. public void destroy(): Clean up code

Configure Technique: How to call these methods:

1. XML
2. Spring Interface
3. Annotation

Project: 02SpringBeanLifecycle

registerShutdownHook():

In spring non-web based applications, registerShutdownHook() method is used to shut down IoC container. It shuts down IoC container gracefully.

In non web based application like desk top application it is required to call registerShutdownHook().

In our desktop application we need to release all resources used by our spring application. So we need to ensure that after application is finished, destroy method on our beans should be called.

In web-based application, ApplicationContext already implements code to shut down the IoC container properly. But in desktop application we need to call registerShutdownHook() to shutdown IoC container properly.

1. Init and Destroy through XML

Mention the names of init and destroy methods in xml

init-method="init"

destroy-method="destroy"

and define the methods with same names in bean class

This gives us the flexibility to change the name of these methods, however we must keep in mind that the signature must be the same

Implementing bean life cycle using interfaces | InitializingBean | DisposableBean |

2. Init and Destroy using Spring Interfaces

The bean class for which we need init and destroy methods must implement:

1. InitializingBean Interface and define the method afterPropertiesSet()
2. DisposableBean Interface and define the method destroy()

Implementing Bean LifeCycle using Annotations | @PostConstruct | @PreDestroy

3. Init and Destroy using Spring Annotations

Use annotations

1. @PostConstruct for init() method
2. @PreDestroy for destroy() method

We can set any name for init() and destroy(). The important things are the annotations

We have to specify <context:annotation-config></context:annotation-config> in xml

to activate @PostConstruct & @PreDestroy

Topic 11: Autowiring

Project: 03AutoWiring

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection. Autowiring can't be used to inject primitive and string values.

Advantage of Autowiring

It requires less code because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring

- No control of programmer.
- It can't be used for primitive and string values.

Autowiring can be done in 2 ways:

1. Through XML
 2. Through Annotations
-

Topic 12: Through XML

There are many autowiring modes:

- 1) no: It is the default autowiring mode. It means no autowiring by default.
- 2) byName: The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.

```
class Employee {  
    Address ad;  
}
```

```
<bean  
    class="Address"  
    name="ad"  
/>
```

a. byName matches class Employee {Address instance name} with Address bean name in xml

b. byName is used when there are more than 1 bean of the same type

3) byType: The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.

byType can be used when there is only one bean of a particular type

It is not necessary to give name in dependency bean class and autowiring is done bytype

4) constructor: The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.

It is not necessary to give name in dependency bean class and autowiring is done by constructor

We can not mention 2 depency beans as IOC will confuse whom to inject

If 2 constructors of the same number of parameters and same type are defined, the IOC container will call the 1st constructor defined

5) autodetect: It is deprecated since Spring 3

Project: 03AutoWiring

packages: auto.wiring.by.name, auto.wiring.by.type, auto.wiring.by.constructor

Topic 13: @Autowired Annotation for Autowiring

@Autowired Annotation: byName is called implicitly

Starting with Spring 2.5, the framework introduced annotations-driven Dependency Injection. The main annotation of this feature is @Autowired. It allows Spring to resolve and inject collaborating beans into our bean.

The Spring framework enables automatic dependency injection. In other words, by declaring all the bean dependencies in a Spring configuration file, Spring container can autowire relationships between collaborating beans. This is called Spring bean autowiring.

@Qualifier Annotation

By default, Spring resolves autowired entries by type.

If more than one bean of the same type is available in the container, the framework will throw `NoUniqueBeanDefinitionException`, indicating that more than one bean is available for autowiring.

Let's imagine a situation in which two possible candidates exist for Spring to inject as bean collaborators in a given instance:

```
@Component("fooFormatter")
public class FooFormatter implements Formatter {

    public String format() {
        return "foo";
    }
}

@Component("barFormatter")
public class BarFormatter implements Formatter {

    public String format() {
        return "bar";
    }
}

@Component
public class FooService {
```

```
    @Autowired  
    private Formatter formatter;  
}
```

If we try to load FooService into our context, the Spring framework will throw a NoUniqueBeanDefinitionException. This is because Spring doesn't know which bean to inject. To avoid this problem, there are several solutions. The @Qualifier annotation is one of them.

3. @Qualifier Annotation

By using the @Qualifier annotation, we can eliminate the issue of which bean needs to be injected

By using the @Qualifier annotation, we can eliminate the issue of which bean needs to be injected.

Let's revisit our previous example and see how we solve the problem by including the @Qualifier annotation to indicate which bean we want to use:

```
public class FooService {  
  
    @Autowired  
    @Qualifier("fooFormatter")  
    private Formatter formatter;  
}
```

By including the @Qualifier annotation together with the name of the specific implementation we want to use – in this example, Foo – we can avoid ambiguity when Spring finds multiple beans of the same type.

We need to take into consideration that the qualifier name to be used is the one declared in the @Component annotation.

Note that we could've also used the `@Qualifier` annotation on the `Formatter` implementing classes, instead of specifying the names in their `@Component` annotations, to obtain the same effect:

```
@Component
```

```
  @Qualifier("fooFormatter")  
  public class FooFormatter implements Formatter {  
    //...  
  }
```

```
  @Component  
  @Qualifier("barFormatter")  
  public class BarFormatter implements Formatter {  
    //...  
  }
```

Topic 14 - Topic 24: Moved

Video 42: Spring ORM Tutorial: moved

Topic

25

-

Topic

30:

Moved

Introduction to Spring MVC

Contents

<u>Spring MVC</u>	100
<u>What is MVC?</u>	100
<u>MVC in Servlets and JSP</u>	100
<u>What is Spring MVC?</u>	100
<u>Why Spring MVC?</u>	101
<u>Problem without MVC Design Pattern</u>	101
<u>What Spring MVC Offers?</u>	101
<u>Working of Spring MVC</u>	101

www.codewitharrays.in

Spring MVC

What is MVC?

MVC stands for Model View Controller. It is a design pattern. A design pattern helps us to develop efficient code. It is a way to organize code in a nice and cleaner way in our application.

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern separates the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

1. The **Model** encapsulates the **application data** and in general they will consist of POJOs.
2. The **View** is responsible for rendering the model data and in general it **generates HTML output** that the client's browser can interpret.
3. The **Controller** is responsible for **processing user requests** and building an appropriate model and passing it to the view for rendering.

In simple, layman's terms we can say:

- ✓ Model – Data in an application
- ✓ View – Presents Data to User
- ✓ Controller – Interface between Model and View, controls the flow of application

We can implement MVC Design Pattern in every programming language and in every type of application. It is not limited just to java or to Spring.

MVC in Servlets and JSP

We did the same thing while developing a web application with Servlets, JSPs, DAOs & POJOs.

- ▶ Servlets worked has Controllers, i.e. they accepted the request, processed the data, generated the response and sent it to the JSPs
- ▶ Java Server Pages worked as Views, i.e. they presented the processed data to the viewer
- ▶ DAOs & POJOs: worked as Models, i.e. by handling the data

What is Spring MVC?

Learning and using Spring MVC is the 1st step for using Spring for Web Development

1. Spring MVC is a module/sub-framework of Spring Framework used to build a **web application**. For example: any website that we can open on any browser.
2. It is built on the top of **Servlet API**.
3. It follows the **Model-View-Controller** design pattern.
4. It implements all the basic features of Spring Core Framework like Inversion of Control, Dependency Injection etc.

Why Spring MVC?

Problem without MVC Design Pattern

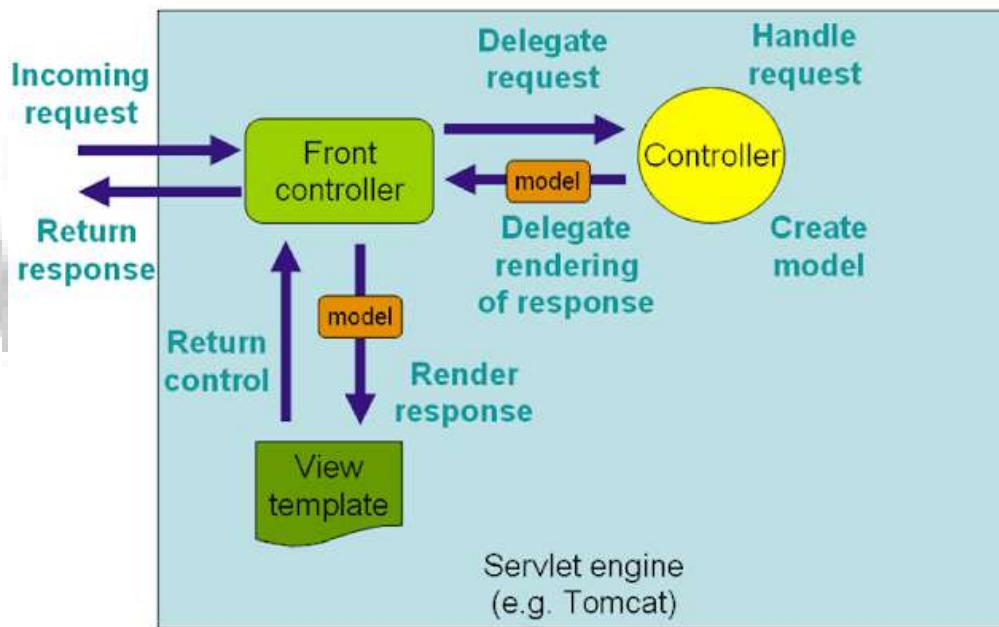
1. The code becomes tedious to maintain
2. The code becomes error prone and
3. The project requires longer development and testing cycles
4. Longer development and testing cycles means higher costs and **loss of business**

What Spring MVC Offers?

1. Separates the roles into: Model, View and Control
2. It provides xml based, annotation based or java based powerful configuration
3. It provides all the basic features of core spring framework such DI, IOC that helps us loosely couple our project
4. It helps in Rapid Application Development
5. Spring MVC is flexible, easy to test and much features. We can also use other frameworks with Spring MVC, for example: Velocity instead of JSP, Hibernate instead of JDBC etc.

Working of Spring MVC

Here is the flow of an HTTP request in Spring MVC Web application



- 1) The client sends an HTTP request to a specific URL
- 2) Front Controller (DispatcherServlet) of Spring MVC receives the request
- 3) It passes the request to a specific controller depending on the URL requested using `@Controller` and `@RequestMapping` annotations.
- 4) Spring MVC Controller then returns a logical view name and model to DispatcherServlet.

- 5) DispatcherServlet consults view resolvers and uses prefix and suffix properties to fetch the view until actual View is determined to render the output
- 6) DispatcherServlet contacts the chosen view (like Thymeleaf, Freemarker, JSP) with model data and it renders the output depending on the model data
- 7) The rendered output is returned to the client as a response



First Spring MVC Project

Contents

<u>First Spring MVC Web Application</u>	104
<u>Setup Development Environment</u>	104
<u>Create the Project</u>	104
<u>Configure the Project</u>	104
<u>Step 1: Configure the Front Controller</u>	104
<u>Step 2: Create Spring Configuration File</u>	105
<u>Step 3: Create Controller</u>	105
<u>Step 4: Configure Annotation Scanner in Config</u>	106
<u>Step 5: Create views</u>	106
<u>Step 6: Configure View Resolver</u>	106
<u>Control Flow in Spring MVC Project</u>	107
<u>Step 1: The Client initiates the Request</u>	107
<u>Step 2: The Role of the Front Controller</u>	107
<u>Step 3: Controller hands the request to handler method</u>	108
<u>Step 4: The handler method receives the request</u>	108
<u>Step 5: The handler method processes the data</u>	108
<u>Step 6: The handler method returns data and view</u>	108
<u>Step 7: DispatcherServlet sends data and view to the ViewResolver</u>	109
<u>Step 8: Front Controller sets data in View</u>	109
<u>Full control flow</u>	109

First Spring MVC Web Application

Let us create a minimalistic Spring Web MVC application to learn the configurations and the control flow:

Setup Development Environment

Install and configure latest

1. JDK
2. Eclipse for Java EE Developers
3. Apache Tomcat Server
4. Apache Maven
5. MySQL Server and Workbench if you wish to work with database

Create the Project

Create and Setup a Maven Project

1. Create a new Maven project and select – Catalog: “Internal” Filter: “maven-archetype-webapp” and other related parameters for project setup
2. Configure Tomcat Server
3. Configure the build path
4. Include Spring Web MVC dependency in pom.xml

Configure the Project

Step 1: Configure the Front Controller

Configure the front controller (DispatcherServlet) in web.xml (present in WEB-INF folder)

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

Points to remember

- ▶ The servlet name can be anything – a name of your choice. However, it must be relatable.
- ▶ The servlet class must be a fully qualified name, i.e. the class name along with the package name. We can get the class definition by pressing Ctrl + Shift + T and then searching the class by name.

Step 2: Create Spring Configuration File

1. Create an XML file in **the same folder (WEB-INF)** where web.xml is.
2. The name of the file must be: <DispatcherServlet name> “-” <servlet>.
For example: In our case **dispatcher-servlet.xml**
3. Include the <beans></beans> tags with proper namespaces

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd">
</beans>
```

Step 3: Create Controller

In the java folder, in a proper package, create a java class with **@Controller** annotation

For a starter, define a method inside this class that will help us in displaying our first JSP page

```
package employee.manager.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
@RequestMapping("/employee")
public class EmployeeController {
    @RequestMapping("/")
    public String redirectToEmpHome() {
        return "emp-home";
    }
}
```

```

    @RequestMapping("/add")
    public String addEmployee() {
        // Add Employee code
        return "add-employee";
    }
}

```

Step 4: Configure Annotation Scanner in Config

Include the following line as the first line inside <beans> tag in dispatcher-servlet.xml file:

```
<context:component-scan base-package=" employee.manager.controller"/>
```

Points to Remember:

- ▶ <context:component-scan> detects the annotations by package scanning. It tells Spring which packages need to be **scanned** to look for the annotated beans or **components**.
- ▶ As Controller class has @Controller and @RequestMapping annotations, the controller's package must be mentioned in context:component-scan for it to work properly

Step 5: Create views

1. Create a folder under WEB-INF directory with the name: "views".
2. Create 2 files: "home.jsp" & "add-employee.jsp" inside "views" folder

Step 6: Configure View Resolver

Configure the InternalResourceViewResolver bean inside the configuration file along with prefix and suffix properties

```

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
name="viewResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
</bean>

```

Points to remember:

- ▶ InternalResourceViewResolver class prepares a full path to the view page
- ▶ Set the path to JSP view files in prefix property
- ▶ Set the extension of JSP view files in suffix property
- ▶ During runtime, the view resolver will prepare the view file name by concatenating prefix & suffix with the file name returned by the controller & present it to the user.

For example: "/WEB-INF/views/" + emp-home + ".jsp", i.e. "/WEB-INF/views/emp-home.jsp"

Control Flow in Spring MVC Project

We can say that there are following important processes in a Spring Web MVC application:

1. The client triggers the request
2. The front controller delegates the request to the appropriate controller
3. The controller sends the request to the appropriate request handler
4. The request handler:
 - a. Receives the request
 - b. Collects the request parameters
 - c. Processes the data
 - d. Sends the processed data and the target view to the front controller

Let us see how Spring MVC achieves the above flow

Step 1: The Client initiates the Request

There are different ways through which a request can be triggered by the client such as:

1. When user clicks on a link
2. When a form is submitted
3. When some control passes from one server component to another
4. Through redirections

Every time any such thing happens, a request is triggered and every request is represented by a URL that specifies the request name

For example:

1. <http://localhost:8080/<Project-Name>/employee/>
2. <http://localhost:8080/<Project-Name>/employee/add>

Step 2: The Role of the Front Controller

As you can see in the above section, the requests triggered are "/employee/" and "/employee/add", in order handle them, we have to create some handler controllers and methods.

```
@Controller  
 @RequestMapping("/employee")  
 public class EmployeeController {  
     @RequestMapping("/")  
     public String redirectToEmpHome() {  
         return "emp-home";
```

```
    }  
  
    @RequestMapping("/add")  
    public String addEmployee() {  
        // Add Employee code  
        return "add-employee";  
    }  
}
```

- ▶ The request reaches the DispatcherServlet (Front Controller)
- ▶ The request will be received by DispatcherServlet (Front Controller)
- ▶ DispatcherServlet will take the help of Handler Mapping and will delegate the request to the Controller class (EmployeeController) associated with the given request: "/employee" in the above case.

Step 3: Controller hands the request to handler method

- ▶ The Controller (EmployeeController) passes the request to the appropriate handler method:
 1. "/" to "redirectToEmpHome()" and
 2. "/add/" to "addEmployee()" in the above case.

Step 4: The handler method receives the request

- ▶ The handler method receives the request
 - ▶ In case there are request parameters, the handler method extracts the data with the help of:
 1. HttpServletRequest object
 2. @RequestParams annotation
 3. @ModelAttribute annotation
- We will study about these concepts in the later sections

Step 5: The handler method processes the data

The handler method performs following actions on the data

1. Authentication
2. File IO
3. Database transaction
4. etc.

Step 6: The handler method returns data and view

The handler method after processing the data returns the following back to the DispatcherServlet:

1. The processed data
 2. The name of the view to render the data, here: "emp-home" and "add-employee"
- The handler can send the data and view name to the front controller with the help of:

1. Model Interface
2. ModelMap class
3. ModelAndView class

We will study about these concepts in the later sections

Step 7: DispatcherServlet sends data and view to the ViewResolver

1. The view resolver will prepare the path of the file by concatenating prefix and suffix
 - "emp-home" becomes "/WEB-INF/views/emp-home.jsp"
 - "add-employee" becomes "/WEB-INF/views/add-employee.jsp"
2. The view resolver will then return the full path to the front controller.

Step 8: Front Controller sets data in View

The Dispatcher Servlet will pass the data/model object to the View page to display the result

Full control flow

After configuring the project as described in the above section, run the application on server

1. When the project will be executed, observe the address bar – it will show <http://localhost:8080/<Project-Name>/>
2. This request will originate from the client and travel to the front controller (DispatcherServlet)
3. DispatcherServlet will delegate this request to the controller class (that has @Controller annotation), in our case to the EmployeeController class.
4. The request will go to the handler method that is annotated with @RequestMapping annotation and configured to handle the "/" request.
For example: In our case: @RequestMapping (value = "/", method = RequestMethod.GET) redirectToIndex method
5. The method will receive the "/" request and return the name: "emp-home" to the front controller
6. The front controller will call the view resolver declared in spring config file
7. The view resolver will prepare the path of the file by concatenating prefix and suffix
8. The view resolver will then return the full path to the front controller.
9. In case there is some data, the front controller will set it in the view
10. The front controller will send the view back to the client

Introduction to Spring Boot

Table of Contents

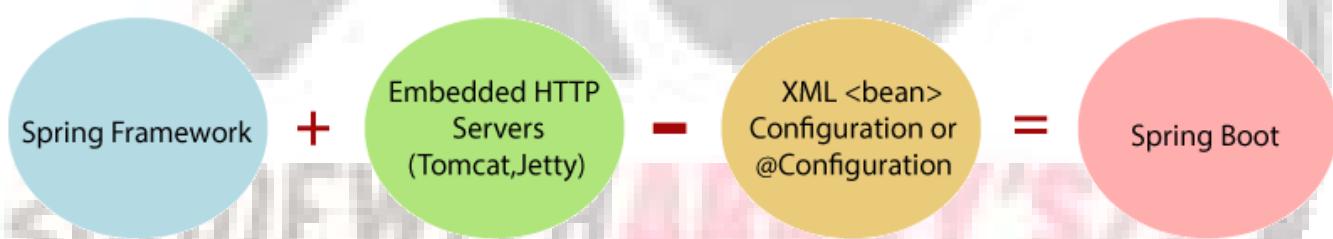
<u>What is Spring Boot?</u>	111
<u>Pros of Spring Boot</u>	111
<u>Cons of Spring Boot</u>	112
<u>Version History of Spring Boot</u>	112
<u>Spring Boot Architecture</u>	112
<u>Presentation Layer</u>	113
<u>Business Layer</u>	113
<u>Persistence Layer</u>	113
<u>Database Layer</u>	113
<u>Spring Boot Flow Architecture</u>	114

What is Spring Boot?

Spring Boot is a project that is built on top of the Spring Framework.

Funnily said - **Spring Boot is Spring on steroids**. It's a great way to get started very quickly with almost the entire Spring stack.

1. Spring Boot is a module of Spring from which we can speed up development
2. Spring Boot makes it easy to create stand-alone, **production grade** Spring based Applications that we can just run
3. It provides an easier and faster way to set up, configure and run both - simple and web based applications
4. **Spring Boot = Spring Framework + Embedded Servers - Configuration**



5. Spring boot follows "**Convention over Configuration**" software design style, i.e. if we follow spring boot's coding and project conventions, Spring boot will take care of the configurations. It decreases the effort of the developer
6. Spring Boot follows "**Opinionated Defaults Configuration**", i.e. if we include "**spring-boot-starter-data-jpa**", Spring Boot will automatically configure - in memory database, a hibernate entity manager, and a simple data-source. Spring Boot scans the class path and find the dependency and then it will automatically configure the things

Pros of Spring Boot

1. It creates stand-alone Spring Applications that can be started using Java -jar, we don't need war file

2. It **embeds** Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
3. Provide opinionated 'starter' dependencies to simplify our build configuration
4. Automatically configure Spring and 3rd party libraries whenever possible
5. Provide production ready features such as metrics, health checks, and external configuration
6. It reduces lots of development time and increases productivity.
7. It **avoids** writing lots of **boilerplate** Code, Annotations and XML Configuration.
8. It is very **easy to integrate** Spring Boot Application with its Spring Ecosystem modules like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
9. It provides **CLI (Command Line Interface)** tool to develop and test Spring Boot (Java or Groovy) Applications from command prompt very easily and quickly
10. It **provides** lots of **plugins** to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle

Cons of Spring Boot

Spring Boot can use dependencies that are not going to be used in the application. These dependencies increase the size of the application.

Version History of Spring Boot

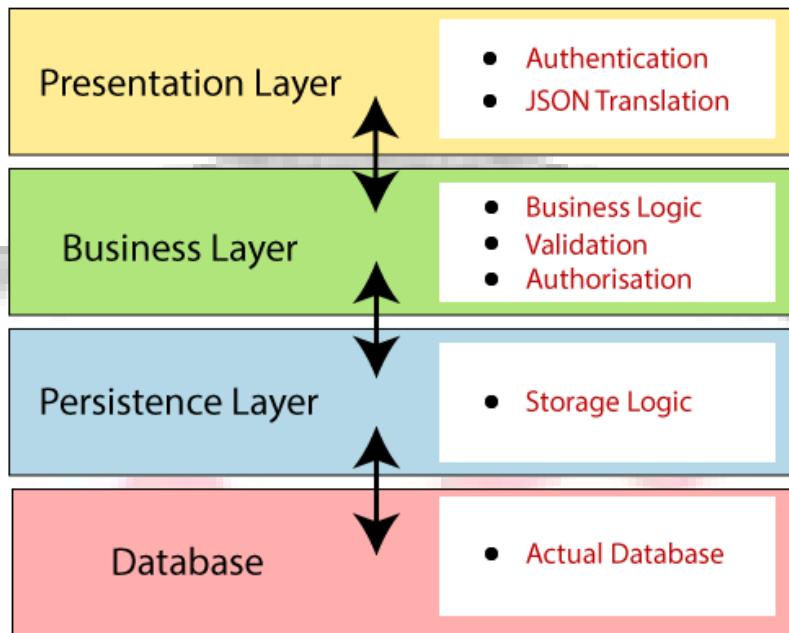
1. Spring Boot 1.0 was released in April 2014
2. Spring Boot 2.0 was released in January 2017
3. Latest current version is **Spring Boot 2.5.3**

Spring Boot Architecture

Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above (hierarchical structure) it.

Before understanding the **Spring Boot Architecture**, we must know the different layers and classes present in it. There are **four** layers in Spring Boot are as follows:

1. Presentation Layer
2. Business Layer
3. Persistence Layer
4. Database Layer



Presentation Layer

The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of **views** i.e., frontend part.

Business Layer

The business layer handles all the **business logic**. It consists of service classes and uses services provided by data access layers. It also performs **authorization** and **validation**.

Persistence Layer

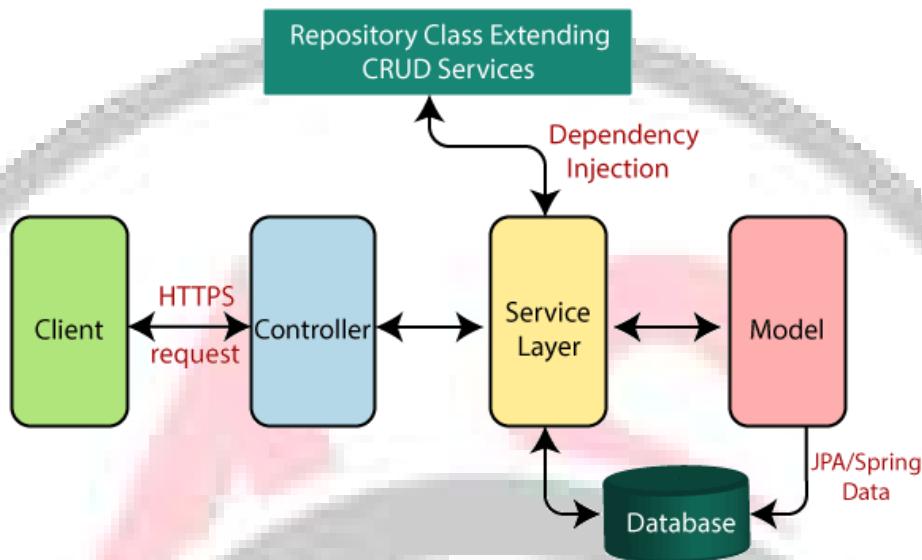
The persistence layer contains all the **storage logic** and translates business objects from and to database rows.

Database Layer

In the database layer, CRUD (create, retrieve, update, delete) operations are performed.

Spring Boot Flow Architecture

Spring Boot flow architecture



Steps:

1. The client makes the HTTP requests
2. The request goes to the controller, and the controller maps that request and handles it.
3. After that, it calls the service logic if required.
4. In the service layer, all the business logic performs.
5. Service Layer performs the logic on the data that is mapped to JPA with model classes.
6. A JSP page is returned to the user if no error occurred.

First Spring Boot Project

Table of Contents

<u>Prerequisites</u>	116
<u>Ways of creating a Spring Boot Application</u>	116
<u>Eclipse</u>	116
<u>Steps to create Spring Boot Application in Eclipse</u>	116
<u>Spring Tool Suite (STS)</u>	117
<u>Steps to create Spring Boot Application in STS</u>	117
<u>Project Structure</u>	118

Prerequisites

Following softwares and tools must be installed on the system before creating any Spring Boot Application

1. Any suitable Java version (LTS preferred)
2. Maven 3.0+
3. Eclipse/STS: **Spring Tool Suite** is recommended.

Ways of creating a Spring Boot Application

There are following ways to create a Spring Boot project:

1. With Eclipse (old method)
Create a maven project in **Eclipse** and add starter dependencies as jars
2. Use Spring Initializr and Eclipse/STS
3. **Use Spring Tool Suite IDE**
4. Spring Boot Command Line Interface
5. Using Spring plugin in Eclipse

Eclipse

1. Create a Maven project in Eclipse and add starter dependencies in the form of jar files
2. Use Spring plugin in Eclipse
3. By Spring Initializer: Eclipse is the most used IDE by Java Developers, so Spring Boot provides an initializer to make things easy for eclipse users.

Steps to create Spring Boot Application in Eclipse

1. Step 1: Search for Spring Boot Initializer on the internet
2. Step 2: Go to <https://start.spring.io/>
3. Step 3: Choose the options for the project:
 - a. Project – Maven Project,
 - b. Language – Java,

- c. Spring Boot – 2.3.5,
 - d. Project Metadata,
 - e. Packaging – Jar,
 - f. Java – 11
4. Step 4: Add Dependencies
 - a. Web
 5. Step 5: Click Generate: The project will be downloaded as a zip file. Extract it.
 6. Open Eclipse and Import the extracted project as “Existing maven Project”

Spring Tool Suite (STS)

1. Spring Tool Suite is an IDE to develop Spring applications.
2. It is an Eclipse-based development environment.
3. It provides a ready-to-use environment to implement, run, deploy, and debug the application.
4. It validates our application and provides quick fixes for the applications.

Steps to create Spring Boot Application in STS

1. Step 1: Search for STS download
2. Step 2: Go to: <https://spring.io/tools>
3. Step 3: Download 64 bit STS windows version (here 4.8.1)
4. Step 4: Execute the file by double clicking it
5. Step 5: Go to File>> New >> Spring Starter Project
6. Step 6: Choose the options for the project:
 - a. Project – Maven Project,
 - b. Language – Java,
 - c. Spring Boot – 2.3.5,
 - d. Project Metadata, such as project name and package: **spring.boot.demo**
 - e. Packaging – Jar,
 - f. Java - 11
7. Step 4: Add Dependencies
 - a. Web

Note: We will use STS in our course to gain familiarity with it

Project Structure

Java Source Code

1. All the java source code must reside inside **src/main/java** folder.
2. The package we mentioned while making the project (here `spring.boot.demo`) is important.
3. It will be considered as a base package for all the java classes hereafter, i.e. all the Java files **must** either lie inside this package or **must be** written inside the sub package of this package for Spring Boot to scan them.
4. In this example, the class `SpringDemoApplication` is the java class from where the execution will begin.
5. `SpringDemoApplication` is annotated with **@SpringBootApplication** annotation. This annotation is the combination of following annotations:
 - a. `@Configuration`,
 - b. `@EnableAutoConfiguration`, and
 - c. `@ComponentScan`
6. Just run the application as –
 - a. Java Application in Eclipse or
 - b. Run As >> Spring Boot App in STS.

Resources

1. All the static resources, configurations and themes are kept in this folder
2. **Static:** All the static resources such as HTML, CSS, JavaScript, images and other media files are contained in static folder.
3. **Themes:** It will contain any predefined themes such as for example: Thymeleaf.
4. File **application.properties**: All the properties such database configurations, context path, file path configurations must be done in `application.properties` file with the help of **key=value** pairs

Dependencies

We add **starter template jars** to our spring boot application such as: **spring-boot-starter-web**, **spring-boot-starter-data-jpa**, etc.

When we add starter jars, then Spring Boot pulls all the related jars. These Jar files contain the file `spring.factories` in META-INF folder, i.e. META-INF/spring.factories

If we use JPA, JPA configurations become active.

Spring boot scans the class path and if it finds JPA, all the configurations relating to JPA in `spring.factories` will become active, this will download spring.orm, hibernate, mysql connector etc

This is known as "Opinionated Defaults Configuration"



Rest APIs

Table of Contents

<u>What is REST ?</u>	121
<u>HTTP Methods</u>	121
<u>RESTful Web Services</u>	121
<u>Background</u>	122
<u>Restful Methods</u>	124
<u>Exercise</u>	125

What is REST ?

REST stands for REpresentational State Transfer.

REST is web standards based architecture and uses HTTP Protocol for data communication.

It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and presents the resources.

Here each resource is identified by URIs

REST uses various representations to represent a resource like text, JSON and XML. Most popular light weight data exchange format used in web services = JSON

HTTP Methods

Following well known HTTP methods are commonly used in REST based architecture.

1. GET - Provides a read only access to a resource.
2. POST - Used to create a new resource.
3. DELETE - Used to remove a resource.
4. PUT - Used to update a existing resource or create a new resource.

RESTful Web Services

A web service is a collection of open protocols and standards used for exchanging data between applications or systems.

Platform & technology independent solution.

Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.

This interoperability (e.g., between Java and Python, or Windows and Linux applications or java & .net) is due to the use of open standards.

Web services based on REST Architecture are known as RESTful web services.

These web services use HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

Background

Web services have really come a long way since its inception. In 2002, the World Wide Web consortium(W3C) had released the definition of WSDL(web service definition language) and SOAP web services. This formed the standard of how web services are implemented.

In 2004, the web consortium also released the definition of an additional standard called RESTful. Over the past couple of years, this standard has become quite popular. And is being used by many of the popular websites around the world which include Facebook and Twitter.

REST is a way to access resources which lie in a particular environment. For example, you could have a server that could be hosting important documents or pictures or videos. All of these are an example of resources. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST defines a way on how these resources can be accessed.

eg of a web application which has a requirement to talk to other applications such Facebook, Twitter, and Google.

Now if a client application had to work with sites such as Facebook, Twitter, etc. they would probably have to know what is the language Facebook, Google and Twitter are built on, and also on what platform they are built on.

Based on this, we can write the interfacing code for our web application, but this could prove to be a nightmare.

So instead , Facebook, Twitter, and Google expose their functionality in the form of Restful web services. This allows any client application to call these web services via REST.

What is Restful Web Service?

REST is used to build Web services that are lightweight, maintainable, and scalable in nature. A service which is built on the REST architecture is called a RESTful service. The underlying protocol for REST is HTTP, which is the basic web protocol. REST stands for REpresentational State Transfer

The key elements of a RESTful implementation are as follows:

1. Resources - The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is <http://www.server.com>. Now in order to access an employee record resource via REST, one can issue the command <http://www.server.com/employee/1> - This command tells the web server to please provide the details of the employee whose employee number is 1.

2. Request Verbs - These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things

like POST, PUT, and DELETE. So in the case of the example <http://www.server.com/employee/1> , the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.

3. Request Headers These are additional instructions sent with the request. These might define the type of response required or the authorization details.

4. Request Body - Data is sent with the request. Data is normally sent in the request when a POST request is made to the REST web service. In a POST call, the client actually tells the web service that it wants to add a resource to the server. Hence, the request body would have the details of the resource which is required to be added to the server.

5. Response Body This is the main body of the response. So in our example, if we were to query the web server via the request <http://www.server.com/employee/1> , the web server might return an XML document with all the details of the employee in the Response Body.

6. Response Status codes These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

Restful Methods

The below diagram shows mostly all the verbs (POST, GET, PUT, and DELETE) and an example of what they would mean.

Let's assume that we have a RESTful web service is defined at the location. <http://www.server.com/employee> . When the client makes any request to this web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. Below is what would happen If the respective verbs were sent by the client.

POST - This would be used to create a new employee using the RESTful web service

GET - This would be used to get a list of all employee using the RESTful web service

PUT - This would be used to update all employee using the RESTful web service

DELETE - This would be used to delete all employee using the RESTful web service

Exercise

Action	Request Method	URL	What it does
Get all Students	Get	/students	Get all students data from server
Get 1 student	Get	/students/2	Get 1 students data with id 2 from server
Add Student	Post	/students Student data	Server will store this data on server side
Update Student	Put	/students Student data	Server updates the data
Delete Student	Delete	/students studentId	Server will delete the student data

Explore More

Subscription : Premium CDAC NOTES & MATERIAL



Contact to Join
Premium Group



Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

codewitharrays.in freelance project available to buy contact on 8007592194

SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySQL
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySQL
3	Tour and Travel management System	React+Springboot+MySQL
4	Election commition of India (online Voting System)	React+Springboot+MySQL
5	HomeRental Booking System	React+Springboot+MySQL
6	Event Management System	React+Springboot+MySQL
7	Hotel Management System	React+Springboot+MySQL
8	Agriculture web Project	React+Springboot+MySQL
9	AirLine Reservation System / Flight booking System	React+Springboot+MySQL
10	E-commerce web Project	React+Springboot+MySQL
11	Hospital Management System	React+Springboot+MySQL
12	E-RTO Driving licence portal	React+Springboot+MySQL
13	Transpotation Services portal	React+Springboot+MySQL
14	Courier Services Portal / Courier Management System	React+Springboot+MySQL
15	Online Food Delivery Portal	React+Springboot+MySQL
16	Municipal Corporation Management	React+Springboot+MySQL
17	Gym Management System	React+Springboot+MySQL
18	Bike/Car ental System Portal	React+Springboot+MySQL
19	CharityDonation web project	React+Springboot+MySQL
20	Movie Booking System	React+Springboot+MySQL

freelance_Project available to buy contact on 8007592194

21	Job Portal web project	React+Springboot+MySQL
22	LIC Insurance Portal	React+Springboot+MySQL
23	Employee Management System	React+Springboot+MySQL
24	Payroll Management System	React+Springboot+MySQL
25	RealEstate Property Project	React+Springboot+MySQL
26	Marriage Hall Booking Project	React+Springboot+MySQL
27	Online Student Management portal	React+Springboot+MySQL
28	Resturant management System	React+Springboot+MySQL
29	Solar Management Project	React+Springboot+MySQL
30	OneStepService LinkLabourContractor	React+Springboot+MySQL

31	Vehical Service Center Portal	React+Springboot+MySQL
32	E-wallet Banking Project	React+Springboot+MySQL
33	Blogg Application Project	React+Springboot+MySQL
34	Car Parking booking Project	React+Springboot+MySQL
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySQL
36	Society management Portal	React+Springboot+MySQL
37	E-College Portal	React+Springboot+MySQL
38	FoodWaste Management Donate System	React+Springboot+MySQL
39	Sports Ground Booking	React+Springboot+MySQL
40	BloodBank mangement System	React+Springboot+MySQL
41	Bus Tickit Booking Project	React+Springboot+MySQL
42	Fruite Delivery Project	React+Springboot+MySQL
43	Woodworks Bed Shop	React+Springboot+MySQL
44	Online Dairy Product sell Project	React+Springboot+MySQL
45	Online E-Pharma medicine sell Project	React+Springboot+MySQL
46	FarmerMarketplace Web Project	React+Springboot+MySQL
47	Online Cloth Store Project	React+Springboot+MySQL
48	Train Ticket Booking Project	React+Springboot+MySQL
49	Quizz Application Project	JSP+Springboot+MySQL
50	Hotel Room Booking Project	React+Springboot+MySQL
51	Online Crime Reporting Portal Project	React+Springboot+MySQL
52	Online Child Adoption Portal Project	React+Springboot+MySQL
53	online Pizza Delivery System Project	React+Springboot+MySQL
54	Online Social Complaint Portal Project	React+Springboot+MySQL
55	Electric Vehical management system Project	React+Springboot+MySQL
56	Online mess / Tiffin management System Project	React+Springboot+MySQL
57	Online Examination Portal Project	React+Springboot+MySQL
58	Lawyer / Advocate Appointment Booking System	React+Springboot+MySQL
59	Café Management System	React+Springboot+MySQL
60	Agriculture Product Rent system Portal	React+Springboot+MySQL

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/vXz0kZQi5to?si=IiOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7uiQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/lynLouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyl4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKz

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrk7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizlyJlh
15	Employee management system project	https://youtu.be/ID1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWA OzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vn0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Sr.No	Project Name	YouTube Link
21	Online Bus ticket booking system Project	https://youtu.be/FJ0RUZfMdv8?si=auHjmNgHMrpaNzvY
22	Online Mess / Tiffin Booking System Project	https://youtu.be/NTVmHFDowyl?si=yrvClbE6fdJ0B7dQ
23		
24		
25		

TAP ON THE ICONS TO JOIN!



**STUDY
MATERIAL**

SPRING BOOT

NOTES BY ASHOK PATE

Enroll

Now!

WWW.CODEWITHARRAYS.IN

=====

Spring Boot & Microservices

=====

Pre-Requisites :

- 1) Core Java
- 2) Adv Java (JDBC + Servlets)
- 3) Oracle (SQL)
- 4) Hibernate (Basics of ORM)

Course Content :

- 1) Spring Core
- 2) Spring JDBC
- 3) Spring Boot
- 4) Spring Data JPA
- 5) Spring Web MVC
- 6) RESTful Services (Spring REST)
- 7) Spring Cloud
- 8) Microservices
- 9) Spring Security
- 10) Spring Boot Integrations
 - Apache kafka
 - Redis cache
 - Docker
 - Unit Testing (Junit)
 - Logging

Programming Language : Core Java

- Language Fundamentals
- Syntaxes
- > Standalone apps

Technologies : JDBC + Servlets + JSP

- > Database communication using JDBC
- > Servlets for web application development
- > JSP (Presentation logic)

Java Frameworks : Hibernate + Spring --> Spring Boot

=====

- 1) Programming Language (Java)
- 2) Java Technologies (JDBC + Servlets + JSP)
- 3) Java Frameworks (Hibernate + Spring & Spring Boot)

=====

Core Java : To develop Standalone applications

JDBC : To develop persistence logic

Servlets : To develop web applications

JSP : To develop Presentation logic

=====

Project Contains Several Logics

- 1) DB Logics
- 2) Business Logic
- 3) Web Logics (Request & Response)
- 4) Presentation Logics (User Interface)

1) DB Communication is mandatory

- 1.1 Load Driver
- 1.2 get connection
- 1.3 create stmt
- 1.4 excute query (it will be changed based on req)
- 1.5 close connection

Note: We have to write above steps in every program. We will end up by writing repeated code (Boiler Plate Code).

2) Web Logics (Request & Response)

- 1.1 Develop Form Display Logic
- 1.2 Form Validations
- 1.3 Capture Form data
- 1.4 Store Form data in java object

What is Framework ?

=====

=> Framework is a semi developed software

=> Frameworks will provide some common logics required for project development.

=> Frameworks provides re-usable components

=> Frameworks will reduce burden on the developers

I
Ex:

====

=> Capture form data and store into DB table

JDBC + Servlets : 20 lines of code

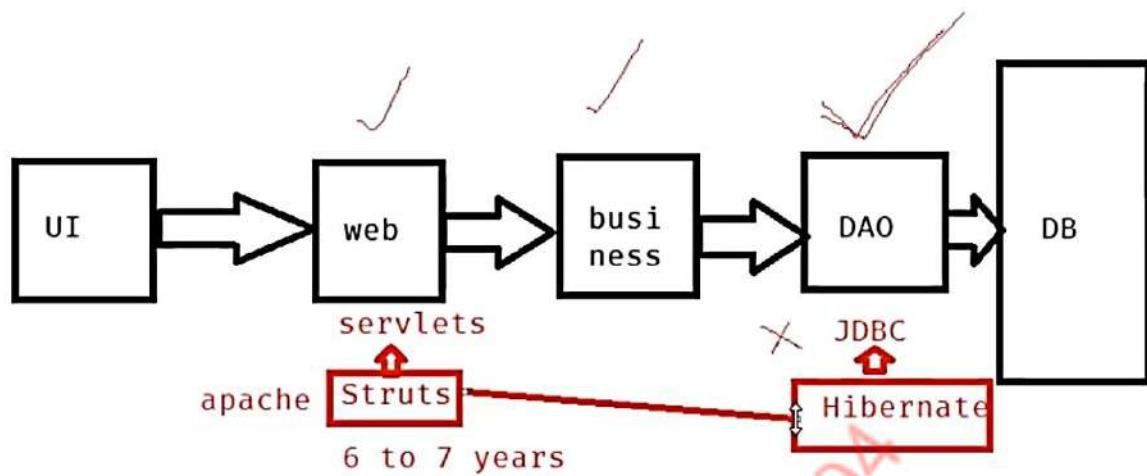
Spring : 5 lines of code

=> IN Java community we are having 2 types of frameworks

1) ORM frameworks (Ex: Hibernate)

2) Web Frameworks (Ex: Struts)

Note: Spring is called as Application development Framework.



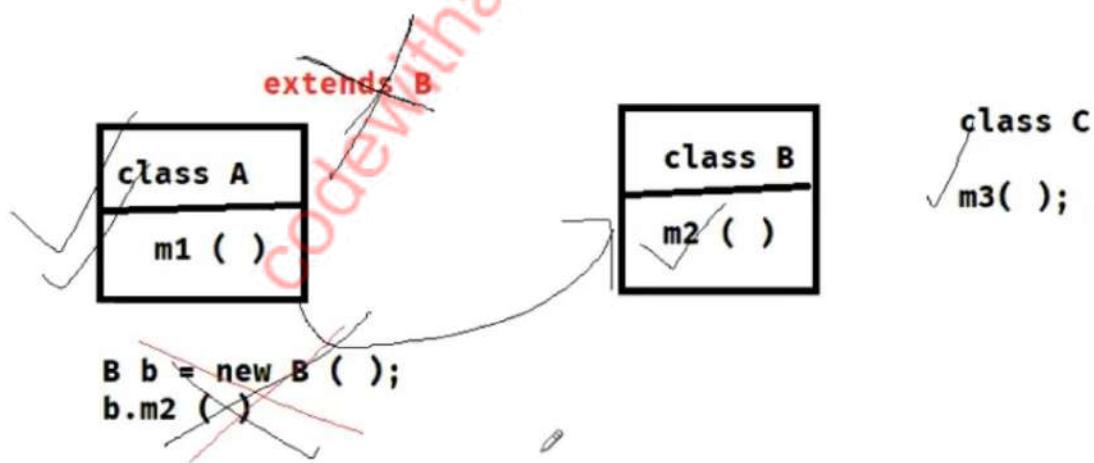
- > Spring is a java based application development framework
 - > By using spring we can develop end to end application
 - > Spring is developed by using JSE & JEE
 - > Spring framework developed by Rod Johnson
 - > First version of spring came into market in 2004 (Spring 1.x v)
 - > The current version of Spring is 6.x (2022)
 - > Spring is developed in modular fashion
 - > Spring Modules are loosely coupled
 - > Spring is versatile framework (it can integrate with any other framework)
- 1) Spring Core
 - 2) Spring Context
 - 3) Spring DAO
 - 4) Spring ORM
 - 5) Spring AOP
 - 6) Spring Web MVC
 - 7) Spring REST

- 8) Spring Data JPA
- 9) Spring Cloud
- 10) Spring Security
- 11) Spring Social
- 12) Spring Batch

=> Spring Core module is base module in Spring Framework.
=> Spring Core Providing IOC Container & Dependency Injection
=> Using IOC & DI we can develop our classes with loosely coupling.

Why need of IOC & DI ?

- I have a class A and Class B and in that I have method m1() and m2() if I want to called m2() in class A I have to create an object then I can call that method on object but this is not recommended its tightly coupled suppose I have thousand of class then tomorrow if I want to change the logic from the method it difficult to change from everywhere. So we need to loosely coupled for that we need IOC and DI
- Also we can extends class to override some method of class A in Class B but I thousand class in multiple inheritance not supported in java because of that we need IOC and DI
- Because of this the Spring Core Module come in the picture.



=====

Spring Core

=====

- > Base module of spring framework
- > Core Module providing fundamental concepts of Spring those are IOC & DI
 - IOC : Inversion Of Control
 - DI : Dependency Injection
- > IOC & DI are used to develop classes with loosely coupling
- > IOC will take care of java objects life cycle (Spring Beans)

=====

Spring Context

=====

- > To manage configurations in Spring application we will use Context Module
- > It provides configuration support required for managing classes

=====

Spring AOP

=====

- AOP : Aspect Oriented Programming
- > AOP is used to separate cross cutting logics in the application

Cross Cutting / Secondary / Helper Logics

Ex : Security, transaction, Logging, Auditing & exception handling etc...

Why we need Spring AOP?

- I have a class A and in that several methods available like deposit(), withdraw(), paybill(), in each method need security logic , then transaction logic, then auditing logic, then business logic , then logging is required suppose we have thousand method then this are repeating again and again and also code length is increase. And in future if I want to change the logic in each method I have to changes. To overcome this the Spring AOP module comes in picture. In method we write only business logic and other we can write separate for each.

Banking Application

```
void deposit ( ){  
    // security  
    // tx logic  
    // auditing  
    // b.logic  
    // logging  
}
```

```
void withdraw ( ){  
    // security  
    // tx logic  
    // auditing  
    // b.logic  
    // logging  
}
```

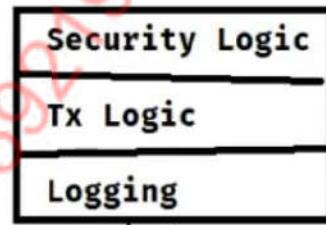
```
void payBill ( ){  
    // security  
    // tx logic  
    // auditing  
    // b.logic  
    // logging  
}
```

✓

```
deposit ( ){  
    // b logic  
}
```

✓

```
withdraw ( ){  
    // b logic  
}
```



JDBC Logic

- 1. load driver
2. get connection
3. Create statement
4. Execute query
5. process result
6. Close connection

spring jdbc

- 1. Execute query
2. process result

=> Spring JDBC provided predefined classes to perform DB operations

Ex: JdbcTemplate, NamedJdbcTemplate etc...

=====
Spring ORM
=====

=> Spring ORM module is extension for existing **ORM frameworks**

ORM - Object relational mapping

=> To support ORM integrations we have Spring ORM module

Spring ORM = Spring + ORM Framework (Ex: Hibernate)

Hibernate

- ```

1. Create Config obj
2. create session factory
3. create session
4. begin tx
5. execute methods
6. commit tx
7. close session
8. close sf
```

### spring orm

```

HibernateTemplate.save(entityObj)
```

I

### Spring Web MVC

- ```
-----  
-> To develop both web & distributed (webservices) applications  
-> It is used to simplify web layer development in applications
```

Spring Data JPA

- ```

-> It is extension for Spring ORM
-> It is providing ready made methods to perform CRUD operations in DB
```

→ We have these 5 options for to developed persistence layer but now days spring data JPA mostly used because we no need to write code the ready made methods available the classes and interfaces are available.

- 1) Jdbc
- 2) Spring JDBC
- 3) Hibernate
- 4) Spring ORM
- 5) Spring Data JPA

### Spring Security

- ```
-----  
-> It is used to secure our spring based application  
-> We can implement both Authentication & Authorization by using Spring Security
```

I

Authentication : Decide who can access our application ?

Authorization : Identify logged in user having access for the functionality or not ?

Spring Batch

=> Batch means bulk operation

Ex:

sending bulk email to customers

sending bulk sms to students regarding course update

sending bank statement to account holders

read records from file and store into DB

```
=====
Spring Cloud
=====
```

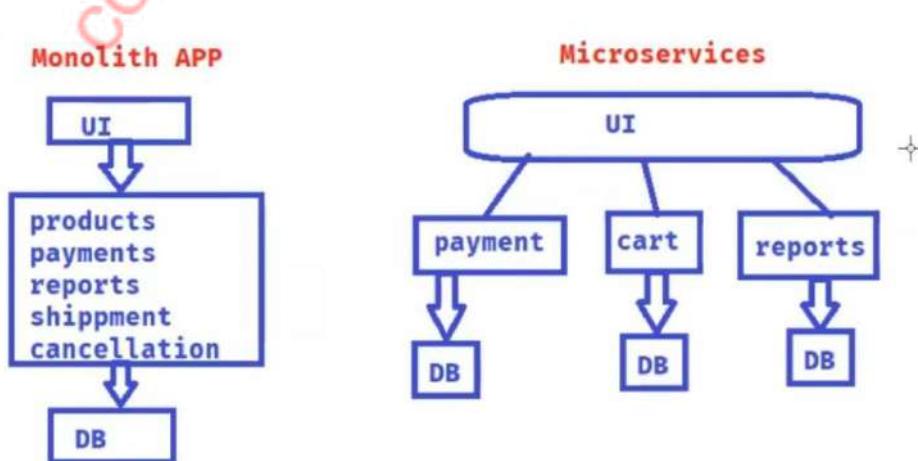
-> It provides configurations required for Microservices development

```
Service Registry
Admin Server
API Gateway
```

There are two types project in industry 1. Monolith project 2. Microservices.

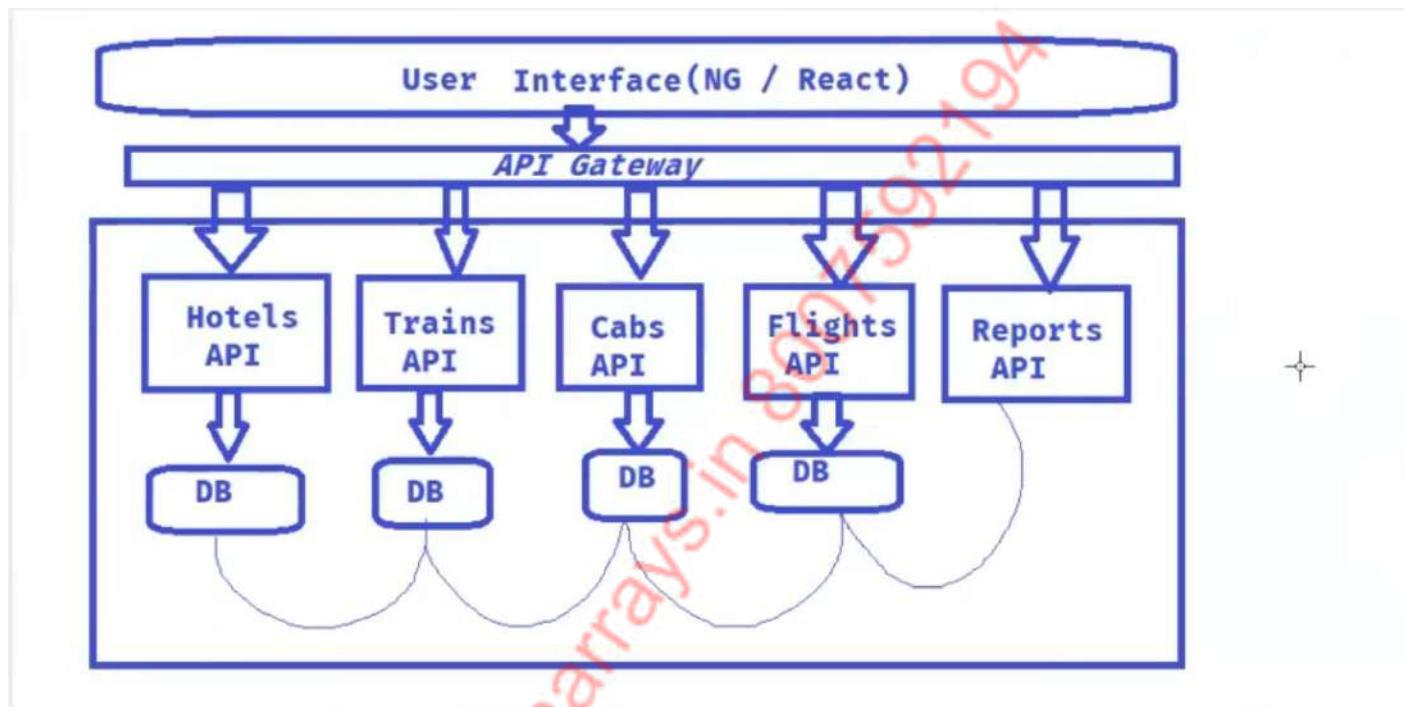
1.Monolith: means if you are developing all functionality in single project. The problem in monolith is maintainace problem. If I want to change one of the functionalities whole projects again compiled, again whole project packaged and then whole project deployed.

2.Microservices: to overcome this problem we can use the microservices. The microservices means the whole project made up in the small small project. And those small projects known as rest services. If tomorrow the maintainace occurs only that that one services we can work on that.



How to developed a full stack project?

→ For example we have to developed a project like MakeMyTrip so first we have to made different API services like Hotel Api , Tains Api, Cabs Api, Flight Api, Reports Api and the database either separated or may be same depends on you and requirement after that we have to check all this Api services on the postman tool and if all is good and running perfect then we are going to developed a UI and after that we can integrate our UI with backend with the help of API Gateway. This Api Gateway is the mediator between frontend and backend.

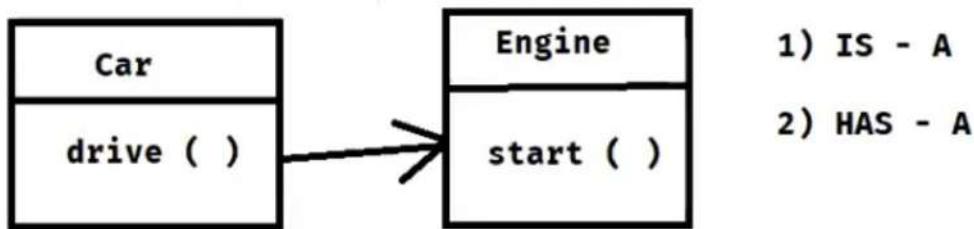


Persistence layer: means which is responsible to communicate with the database is know as persistence layer.

Distributed application: means one application is communicating with another application is known as distributed application. Ex Swiggy, Zomato, Phone Pay, MakeMyTrip etc...

Spring Core

- > Base module of Spring Framework
- > Providing IoC & DI
- > IOC & DI are used to develop classes with loosely coupling
- > First a fall we need to understand what is tightly coupling. For that we can take one small car app example.



→ I have a car class and in that drive() method and another one class engine in that start() Method. We can make this by using IS-A relation or HAS-A relation.

```

package com.codewitharrays;

public class Car extends Engine{

    public void drive() {
        int status=super.start();
        if(status>=1) {
            System.out.println("Journey Started");
        }
        else {
            System.out.println("Engine Problem");
        }
    }
}

package com.codewitharrays;

public class Engine {
    public int start() {
        //logic
        return 1;
    }
}

package com.codewitharrays;

public class App {
    public static void main (String[]arg ) {
        Car c=new Car();
        c.drive();
    }
}
  
```

- > Car is extending Engine class
- > Car class cannot use inheritance in future
- > Any changes in Engine class will effect on Car class
- > Car is tightly coupled with Engine

Note: It is not recommended to develop classes with Tightly coupling.

→ Now we can make by using Has-A Relation. Now my class is open for inheritance in future. And now I make a object and called method on object but When I do changes in future like parameter constructor make and the I get the error. So, its tightly coupled. And not recommended.

```
public class Car {

    public void drive() {
        Engine eng=new Engine();
        int status= eng.start();
        if(status>=1) {
            System.out.println("Journey Started");
        }
        else {
            System.out.println("Engine Problem");
        }
    }

}

package com.codewitharrays;

public class Engine {

    public Engine(String fuelType) {

    }

    public int start() {
        //logic
        return 1;
    }
}

package com.codewitharrays;

public class App {
    public static void main (String[]arg ) {
        Car c=new Car();
        c.drive();
    }
}
```

-> Car is directly creating Object for Engine

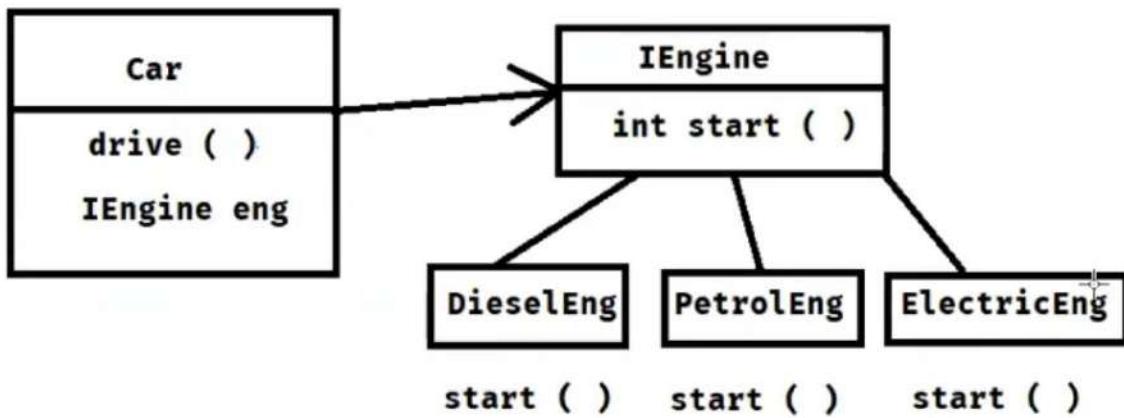
-> Any changes to Engine class will effect on Car class

-> Car is always talking to only one Engine

-> If i want to change from one Engine to another Engine then we should modify Car class code.

Note: Car is tightly coupled with Engine.

→ To overcome above problem, we make engine class as Interface class and now the other class can implement engine class. Now we achieved the loosely coupled.



```

package com.codewitharrays;

public interface IEngine {
    public int start();
}

package com.codewitharrays;

public class PetrolEngine implements IEngine{
    public int start() {
        System.out.println("Petrol Engine Started...");
        return 1;
    }
}

package com.codewitharrays;

public class DieselEngine implements IEngine {
    public int start() {
        System.out.println("Diesel engine started");
        return 1;
    }
}

public class Car {
    private IEngine eng;
    public Car(IEngine eng) {
        this.eng=eng;
    }
    public void drive() {
        int status=eng.start();
        if(status>=1) {
            System.out.println("Journey Started");
        }
        else {
            System.out.println("Engine Problem");
        }
    }
}
  
```

```
package com.codewitharrays;

public class App {

    public static void main (String[]arg ) {
        //Here we can pass the class object we can change petrol to diesel
        //also without change in car class. So we achieved loosely coupled.
        Car c=new Car(new PetrolEngine());
        c.drive();
        Car cb=new Car(new DieselEngine());
        cb.drive();
    }
}
```

What is Dependency Injection?

- The process of injecting dependent object into target object using target class variable / setter method / constructor is called as Dependency Injection.
- The process of injecting one class object into another class object is called dependency injection.

Dependency Injection Types

- 1) Field Injection (variable)
 - 2) Setter Injection (setter method)
 - 3) Constructor Injection (constructor)
-
- If I'm doing injecting object by constructor that is constructor Injection.
 - If I'm doing injecting object by setter method that is setter Injection.
 - If I'm doing injecting object by field that is field Injection.
 - Understand this concept by coding below.

```
public class Car {

    // private IEngine eng;
    IEngine eng; // field

    public Car(IEngine eng) {
        this.eng=eng; //Constructor
    }

    public void setEng(IEngine eng) {
        this.eng=eng; //Setter
    }

    public void drive() {

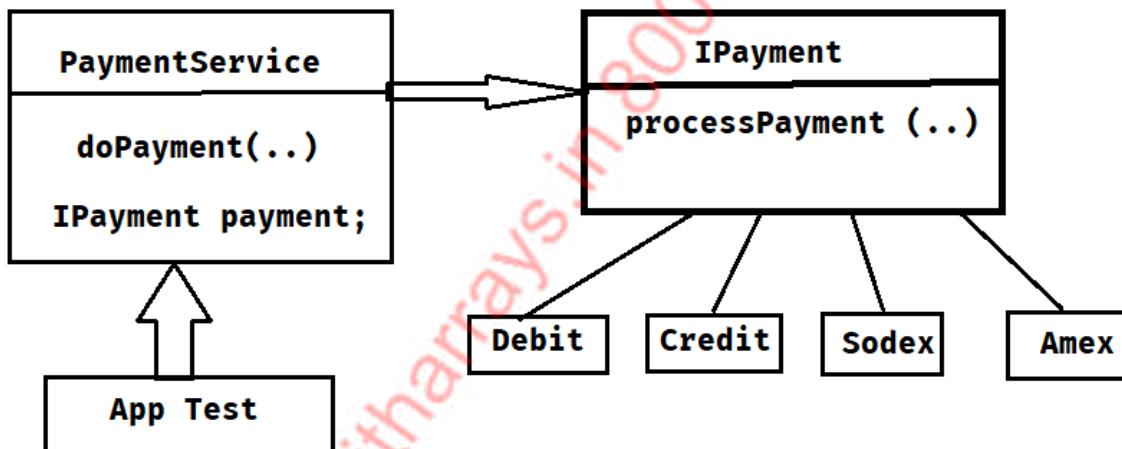
        int status=eng.start();

        if(status>=1) {
            System.out.println("Journey Started");
        }
        else {
            System.out.println("Engine Problem");
        }
    }
}
```

```
public class App {  
  
    public static void main (String[]arg ) {  
  
        Car c=new Car(new PetrolEngine()); //Constructor Injector  
        c.drive();  
        Car cb=new Car();  
        cb.setEng (new DieselEngine()); //Setter Injector  
  
        cb.eng=new PetrolEngine(); //field Injector  
  
    }  
}
```

Requirement:

Develop an application to perform bill payment. It should support for multiple Payment options (Debit card, Credit Card, Sodex & Amex....)



```
package com.codewitharrays;  
  
public interface IPayment {  
  
    public boolean processPayment(double billAmt);  
}
```

```
package com.codewitharrays;  
  
public class CreditCardPayment implements IPayment {  
  
    @Override  
    public boolean processPayment(double billAmt) {  
        System.out.println("Creditcard payment processing..");  
        return true;  
    }  
}
```

```
package com.codewitharrays;

public class DebitCardPayment implements IPayment{

    @Override
    public boolean processPayment(double billAmt) {
        System.out.println("debitcard payment processing..");
        return true;
    }
}
```

```
package com.codewitharrays;

public class SodexCardPayment implements IPayment {

    @Override
    public boolean processPayment(double billAmt) {
        System.out.println("Sodex payment processing..");
        return true;
    }
}
```

```
package com.codewitharrays;

public class AmexCardPayment implements IPayment {

    @Override
    public boolean processPayment(double billAmt) {
        System.out.println("AmexCard payment processing..");
        return true;
    }
}
```

```
package com.codewitharrays;

public class PaymentService {
    private IPayment payment;

    public PaymentService(IPayment payment) {
        this.payment=payment;
    }
    public void setPayment(IPayment payment) {
        this.payment=payment;
    }

    public void doPayment(double billAmt) {
        boolean status= payment.processPayment(billAmt);

        if(status) {
            System.out.println("Printing Receipt..");
        }else {
            System.out.println("Payment Decline.");
        }
    }
}
```

```
package com.codewitharrays;

public class App {
    public static void main1(String[] args) {
        IPayment p=new DebitCardPayment();
        PaymentService ps=new PaymentService(p); //Constructor Injector
        ps.doPayment(450.0);
    }
    public static void main(String[] args) {
        IPayment p1=new CreditCardPayment();
        IPayment p2=new DebitCardPayment();

        PaymentService ps=new PaymentService(p1); //Constructor Injector
        ps.setPayment(p2); //Setter Injector
        ps.doPayment(100.00);
    }
}

// Ans: debitcard payment processing..      Printing Receipt..
// First Constructor injection will happen then it will initialize the variable
// then setter injection will happen and it will re-initialize the same variable
// so final value be setter injection value.
// Note: Setter Injection will override Constructor injection.
```

- Right now programmer do the dependency injection but I want to IoC container inject the dependency that will saw in the next case.

What is IoC ?

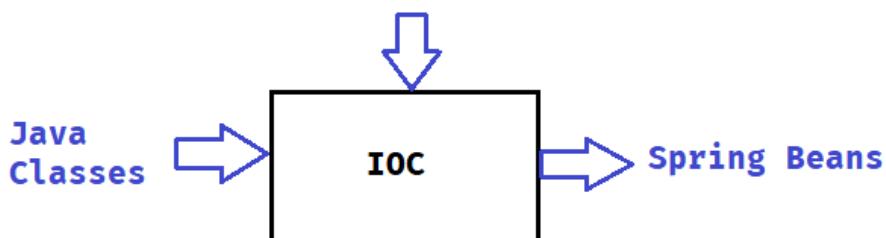
IOC: Inversion of Control

-> IOC is a principle which is used to manage and collaborate dependencies among the objects in the application.

-> In Spring, IOC is responsible for Dependency Injection.

-> What is the difference between IOC container and Dependency Injection. so many people will ask that question in the interview then tell them that the question itself is wrong we cannot compare IOC and DI. IoC is performing the DI how can we compare that.

(xml / annotations)
Configuration



- In Spring we have do configuration but in spring boot the auto configuration is available because of that spring boot came in the market.
- In Spring we can do configuration by using XML and annotation. But in spring boot we can do configuration by using annotation only.

Note: For IOC we need to pass Java Classes + Configuration as input then IOC will perform DI and it will produce Spring Beans.

Spring Bean: The class which is managed by IOC is called as Spring Bean.

How start IOC container?

=> We can start in 2 ways

1) BeanFactory (I) (outdated)

2) ApplicationContext (I) (recommended)

```
ApplicationContext ctxt = new ClassPathXmlApplicationContext(String xmlFile)
```

Creating First Spring Project

1) Open STS IDE

2) Create Maven Project

3) Open pom.xml file and add below dependency

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.25</version>
    </dependency>
</dependencies>
```

Note: After adding dependency verify project Maven Dependencies folder (jars should be displayed)

4) Create Required Java classes

```
package in.codewitharrays.beans;

public interface IPayment {
    public boolean processPayment(double billAmt);
}
```

```
package in.codewitharrays.beans;

public class DebitCardPayment implements IPayment {

    public DebitCardPayment() {
        System.out.println("DebitCardPayment :: Constructor");
    }

    public boolean processPayment(double billAmt) {
        // logic

        System.out.println("DebitCard Payment successfull...");

        return true;
    }
}

package in.codewitharrays.beans;

public class CreditCardPayment implements IPayment {

    public CreditCardPayment() {
        System.out.println("CreditCardPayment :: Constructor");
    }

    public boolean processPayment(double billAmt) {
        // logic

        System.out.println("CreditCard Payment successfull...");

        return true;
    }
}

package in.codewitharrays.beans;

public class PaymentService {

    private IPayment payment;

    public PaymentService() {
        System.out.println("PaymentService :: 0-Constructor");
    }

    public PaymentService(IPayment payment) {
        System.out.println("PaymentService :: Param-Constructor");
        this.payment = payment;
    }

    public void setPayment(IPayment payment) {
        System.out.println("setter method called...");
        this.payment = payment;
    }

    public void doPayment(double billAmt) {
        boolean status = payment.processPayment(billAmt);

        if (status) {
            System.out.println("Receipt Printing....");
        } else {
            System.out.println("Card Declined...");
        }
    }
}
```

5) Create Bean Configuration File like below

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="credit" class="in.codewitharrays.beans.CreditCardPayment" />
    <bean id="debit" class="in.codewitharrays.beans.DebitCardPayment" />
    <bean id="payment" class="in.codewitharrays.beans.PaymentService">
        <constructor-arg name="payment" ref="credit" />
        <property name="payment" ref="debit" />
    </bean>
</beans>
```

-> Constructor Injection will be represented like below or we can see in bean configuration

```
<constructor-arg name="payment" ref="credit" />
// setter injection
<property name="payment" ref="debit" />
```

Note: ref attribute represents which object should be injected.

6) Create Test Class and start IOC Container

```
package in.codewitharrays.beans;

import org.springframework.context.ApplicationContext;

public class Test {

    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("Beans.xml");

        PaymentService service =
            context.getBean(PaymentService.class);

        service.doPayment(199.00);
    }
}
```

Output:

```
CreditCardPayment :: Constructor
DebitCardPayment :: Constructor
PaymentService :: Param-Constructor
setter method called...
DebitCard Payment successfull...
Receipt Printing....
```

Spring Bean: Class Managed by IOC container is called Spring bean

Spring Bean Scopes (Refer page 19 code to understand this all concept)

- > Bean Scope will decide how many objects should be created for Spring Bean class
- > We have 4 types of scopes

- 1) singleton (default)
- 2) prototype
- 3) request
- 4) session

-> Singleton means only one object will be created

-> Prototype means every time new object will be created

Note: request & session scopes we will use in spring web MVC.

Example of Singleton Scope:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="diesel" class="in.codewitharrays.beans.DieselEngine" primary="true"/>

    <bean id="petrol" class="in.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="in.codewitharrays.beans.Car" >
        <property name="eng" ref="petrol"/></property>
    </bean>
```

Observe this carefully in this I did not mention the scope so it is a default singleton scope. And because of that the only one object is created. You can see below we created 3 objects but still give one hash code so it's only create one object. It is known as Singleton bean scope.

```
import org.springframework.context.ApplicationContext;
public class Test {
    public static void main(String[] args) {
        ApplicationContext ctxt =
            new ClassPathXmlApplicationContext('
                Car c1 = ctxt.getBean(Car.class);
                System.out.println(c1.hashCode());
                Car c2 = ctxt.getBean(Car.class);
                System.out.println(c2.hashCode());
                Car c3 = ctxt.getBean(Car.class);
                System.out.println(c3.hashCode());
            } }
```

```
<terminated> Test (1) [Java Application] C:\DieselEngine.Cons* PetrolEngine.Cons* Car::Constructor 1362546706 1362546706 1362546706
```

Example of Prototype Scope:

Observe this carefully in this I mention the scope prototype. And because of that the three object is created. You can see below we created 3 objects and complier give three different hash code so it's create three object. When every time new object will be created this known as prototype scope.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="diesel" class="in.codewitharrays.beans.DieselEngine" primary="true"/>

    <bean id="petrol" class="in.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="in.codewitharrays.beans.Car" scope="prototype" >
        <property name="eng" ref="petrol"/></property>
    </bean>
```

```
public class Test {

    public static void main(String[] args) {

        ApplicationContext ctxt =
            new ClassPathXmlApplicationContext("applicationContext.xml");

        Car c1 = ctxt.getBean(Car.class);
        System.out.println(c1.hashCode());

        Car c2 = ctxt.getBean(Car.class);
        System.out.println(c2.hashCode());

        Car c3= ctxt.getBean(Car.class);
        System.out.println(c3.hashCode());
    }
}
```

```
DieselEngine.Cons
PetrolEngine.Cons
Car::Constructor
1362546706
Car::Constructor
1496949625
Car::Constructor
236840983
```

Autowiring (Refer page 19 code to understand this all concept)

=> Auto wiring is used to identify dependent objects and inject into target objects.

=> Autowiring works based on below 3 modes

- 1) byName
- 2) byType
- 3) constructor

-> byName MODE will identify dependent bean based on variable name matching with bean id.

-> byType MODE will identify dependent bean based on variable data type.

1.ByName:

- In byName dependent bean base on variable name matching with bean id. So my variable is eng so it will check with eng and DieselEngine object will inject. And in output you can observe that.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng" class="com.codewitharrays.beans.DieselEngine" />
    <bean id="eng1" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byName" >
        </bean>
</beans>
```

DieselEngine.Constructor....)
PetrolEngine.Constructor....
Default contructor
DieselEngine started....
Journey Started...

- If the both variable name is same for DieselEngine and PetrolEngine then it will give the ambiguity error.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng" class="com.codewitharrays.beans.DieselEngine" />
    <bean id="eng" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byName" >
        </bean>
</beans>
```

ngframework.beans.factory.parsing.BeanDefinitionParsingException: Configuration problem: Bean name 'eng' is already used in this <beans> element
urce [Beans.xml]

- If bean variable is not present then the null pointer exception occurs. In below my variable is eng and in bean scope the id eng1, eng2 that is not in my class so null exception occurs.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" />

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byName" >
        </bean>
    </beans>

DieselEngine.Constructor....)
PetrolEngine.Constructor....
Default constructor
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "com.codewitharrays.beans.IEngine.start()" because "this.eng" is null
    at com.codewitharrays.beans.Car.drive(Car.java:16)
    at com.codewitharrays.beans.Test.main(Test.java:11)
```

2. byType:

- When Spring's **Autowiring byType** is enabled, Spring will search for a bean in the container that matches the type of the variable datatype or constructor parameter in the class. If it finds exactly one matching bean of that type, Spring will inject that bean. If no matching bean or more than one bean of the same type is found, Spring will throw an exception
- byType MODE will identify dependent bean based on variable data type.
- In below example the bean id only one is available and variable datatype IEngine inject with DieselEngine.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" />

    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byType" >
        </bean>
    </beans>
```

Output:

DieselEngine.Constructor....)
 Default constructor
 DieselEngine started....
 Journey Started...

- **Note:** There is a chance of getting ambiguity in byType mode.
- If variable data type is interface, then we can have multiple implementation classes in this scenario IOC cannot decide which bean it has to inject.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" />
    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byType" >
        </bean>
</beans>
```

- We can resolve byType ambiguity in 2 way

1. autowire-candidate="false":

- Agar hum autowire-candidate="false" karte hai tho use ignore kr denga aur remaining one ko inject kr denga. Example iss mai hum ne DieselEngine ko false kiya tho PetrolEngine ka object inject ho gaya hai.
- Lekin agar 10 beans hai tho hume har jagah autowire-candidate="false" karna padenga is liye dusra option hai.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" autowire-candidate="false"/>
    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byType" >
        </bean>
</beans>
```

Output:

DieselEngine.Constructor....)
PetrolEngine.Constructor....
Default constructor
PetrolEngine Started...
Journey Started...

2) primary = "true":

- Jiska object inject krna hai aur jise priority pe rakhna hai use primary="true" kro tho uska object inject ho jayenga.
- Abhi iss mai DieselEngine bean mai primary="true" kr diya issliye uska object inject ho gaya check output carefully.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemalocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" primary="true"/>
    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="byType" >
        </bean>
</beans>
```

Output:

DieselEngine.Constructor....)
PetrolEngine.Constructor....
Default constructor
DieselEngine started....
Journey Started...

Note: Why this byType is better than byName agar koi developer ne ek eng variable banaya aur bean me inject kr diya byName. Lekin In future agar kisi dusre developer ne variable name change kr diya tho. Code may gets failed. But in byType the id is anything injection is happened on variable datatype.

3. Constructor: (Refer page 19 code to understand this all concept / only one constructor is added extra)

- In constructor auto wire first it will check byName if the byName variable is available the bean is inject. If not then it will go for byType and the multiple class is implement by interface then the primary="true" is necessary to give for high priority to inject the bean. Constructor is the combination of byName & byType.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemalocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng" class="com.codewitharrays.beans.DieselEngine" />
    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="constructor" >
        </bean>
</beans>
```

```
package com.codewitharrays.beans;

public class Car {
    private IEngine eng;
    public Car() {
        System.out.println("Default constructor");
    }

    public Car(IEngine eng) {
        System.out.println("parameter constructor");
        this.eng=eng;
    }

    // public void setEng(IEngine eng) {
    //     this.eng=eng;
    // }

    public void drive() {
        int status = eng.start();
        if (status >= 1) {
            System.out.println("Journey Started...");
        } else {
            System.out.println("Engine Trouble...");
        }
    }
}
```

Output:

```
DieselEngine.Constructor....)
PetrolEngine.Constructor....
parameter constructor
DieselEngine started...
Journey Started...
```

- In above example we make one constructor and in bean scope we give autowire as constructor and because of that it will go for byName first and variable eng found and the DieselEngine bean in injected the output observed.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="constructor" >
        </bean>
</beans>
```

Output:

```
PetrolEngine.Constructor....
parameter constructor
PetrolEngine Started...
Journey Started...
```

- In above example the first check for byName and it will not found then go for byType and the variable datatype is inject PetrolEngine because other interface not mentioned here. If present then ambiguity error occurs.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="eng1" class="com.codewitharrays.beans.DieselEngine" primary="true" />
    <bean id="eng2" class="com.codewitharrays.beans.PetrolEngine" />
    <bean id="car" class="com.codewitharrays.beans.Car" autowire="constructor" >
        </bean>
</beans>
```

Output:

PetrolEngine.Constructor....
parameter contructor
DieselEngine started....
Journey Started...

- In this above example the we autowire as constructor and first go for byName and not found then it will go for byType and the multiple implements by interface. So the primary="true" is inject because of high priority.

This xml based is outdated so we go for annotation based injection and spring boot doesn't support for xml.

Spring Annotations

- Annotation is used to provide some metadata.

1. **@Configuration:** To represent java class as config class.

- @Configuration is used on classes that define beans. @Configuration is an analog for an XML configuration file
- it is configured using Java classes. A Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.

2. **@Component:** To represent java class as Spring Bean class

- @Component is used on classes to indicate a Spring component. The @Component annotation marks the Java class as a bean or component so that the component scanning mechanism of Spring can add it into the application context.

3. **@Service:** To represent java class as Spring Bean class

- @Service marks a Java class that performs some service, such as executing business logic, performing calculations, and calling external APIs. This annotation is a specialized form of the @Component annotation intended to be used in the service layer.

4. @Repository: To represent java class as Spring Bean class

- This annotation is used on Java classes that directly access the database. The @Repository annotation works as a marker for any class that fulfils the role of repository or Data Access Object.
- This annotation has an automatic translation feature. For example, when an exception occurs in the @Repository, there is a handler for that exception and there is no need to add a try-catch block.

5. @Scope: To represent scope of spring bean (default: singleton)

- @Scope is used to define the scope of a @Component class or a @Bean definition. It can be either singleton, prototype, request, session, global Session or some custom scope.

6. @Autowired: Inject dependent into target

@Autowired is used to mark a dependency which Spring is going to resolve and inject automatically. We can use this annotation with a constructor, setter, or field injection.

7. @Bean: To customize bean object creation.

- @Bean is a method-level annotation and a direct analog of the XML element.
- @Bean marks a factory method which instantiates a Spring bean.
- Spring calls these methods when a new instance of the return type is required.

8. @Qualifier: To identify bean based on the given name for DI

- @Qualifier helps fine-tune annotation-based auto wiring. There may be scenarios when we create more than one bean of the same type and want to wire only one of them with a property.
- This can be controlled using @Qualifier annotation along with the @Autowired annotation.
- We use @Qualifier along with @Autowired to provide the bean ID or bean name we want to use in ambiguous situations.

9. @Primary: To give priority for the bean for auto wiring

- We use @Primary to give higher preference to a bean when there are multiple beans of the same type.
- When a bean is not marked with @Qualifier, a bean marked with @Primary will be served in case of ambiguity.

What is Component Scanning ?

=> It is used to identify Spring Bean classes available in the Project.

=> It will start scanning from current package.

se package name: com.codewitharrays

 AppConfig

 - @Configuration

 - @ComponentScan

 com.codewitharrays.service ----- scanned

 com.codewitharrays.util ----- scanned

 com.codewitharrays.dao ----- not scanned

We can understand this by the example:

```
package com.codewitharrays.beans;

import org.springframework.stereotype.Component;

@Component
public class Chip {

    public Chip() {
        System.out.println("Chip constructor");
    }

    public static String Chips() {
        return "32-bits";
    }
}

package com.codewitharrays.beans;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Robot {

    @Autowired
    private Chip chip;

    public Robot() {
        System.out.println("Robot constructor");
    }

    public void doWork() {
        String type = chip.Chips();
        if (type.contains("32-bits")) {
            System.out.println("performance is low with this: "+type);
        }else {
            System.out.println("chip is not found: ");
        }
    }
}

package com.codewitharrays.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.codewitharrays")
public class AppConfig {
```

```

package com.codewitharrays.config;

import org.springframework.stereotype.Component;

@Component
public class Car {

    public Car() {
        System.out.println("Car constructor");
    }
}

package com.codewitharrays.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.codewitharrays.config.AppConfig;

public class App {
    public static void main(String[] args) {
        ApplicationContext ctx= new AnnotationConfigApplicationContext(AppConfig.class);
    }
}

```

With base Packages write Output:	without base Packages output:	without component Scan
Chip constructor	Car constructor	No output
Robot constructor		
Car constructor		

--ComponentScan first scan the component from the same package only and create object for component class bean. To resolve this then we can give the basePackages = "in.codewitharrays" then it will check for all component present in this package and create object for that component class bean. And you can give multiple package name also. Ex: { com.tcs, com.wipro, in.codewitharrays}

Right now, the all-component object is created because of singleton scope if I want to change the scope then @scope annotation is used.

@Scope: To represent scope of spring bean (default: singleton)

- @Scope is used to define the scope of a @Component class or a @Bean definition. It can be either singleton, prototype, request, session, global Session or some custom scope.

Let's understand by following example:

Refer above 30 page no code. Only change In car class. I change the scope of car class that I prototype and now the object is not created. Until I called.

```
package com.codewitharrays.config;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope("prototype")
public class Car {

    public Car() {
        System.out.println("Car constructor");
    }
}
```

Output:

Chip constructor
Robot constructor

@Bean: To customize bean object creation.

- @Bean is a method-level annotation and a direct analog of the XML element.
- @Bean marks a factory method which instantiates a Spring bean.
- Spring calls these methods when a new instance of the return type is required.

-When we mentioned a class as @component IoC create the object but if I want to make the object because I have special requirement that time the @Bean annotation we can used.

Let's understand by the following example: Refer above 30-page code additional are below

```
package com.codewitharrays.config;

public class Engine {
    public Engine() {
        System.out.println("Engine constructor");
    }
}
```

```
package com.codewitharrays.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.codewitharrays")
public class AppConfig {

    @Bean
    public Engine getEngine() {
        Engine engine=new Engine();
        return engine;
    }
}
```

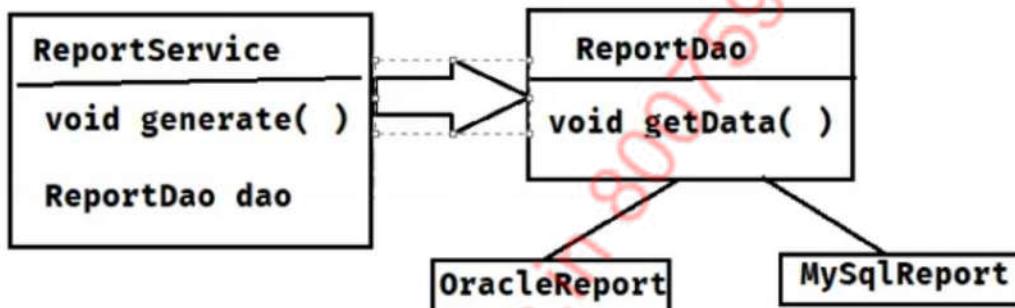
- What happened here I want to make object by me not a IoC so for that I used @Bean annotation on that getEngine() method without @Bean the object is not created you can check by compiling the code. After @bean the output is:

Output:

Chip constructor
Robot constructor
Engine constructor

Auto wiring with annotation:

--Understand some concept



```
package com.codewitharrays.bean;

public interface ReportDao {
    public void getData();
}

package com.codewitharrays.bean;

import org.springframework.stereotype.Repository;

@Repository("dao")
public class OracleReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the oracle db");
    }
}

package com.codewitharrays.bean;

import org.springframework.stereotype.Repository;

@Repository("dao")
public class MySqlReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the mysql db");
    }
}
```

```

package com.codewitharrays.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.codewitharrays.AppConfig;
import com.codewitharrays.bean.ReportService;

public class App {
    public static void main(String[] args) {
        ApplicationContext ctxt=
            new AnnotationConfigApplicationContext(AppConfig.class);

        ReportService service=ctxt.getBean(ReportService.class);
        service.generatereport();
        //Now it will go for byType but the interface implements multiple class so
        //give me ambiguity error
    }
}

package com.codewitharrays;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
public class AppConfig {

}

package com.codewitharrays.bean;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class ReportService {

    //1. Right now the autowired happened byName and it will not find so going for byType and
    //interface implement multiple class so it will gave me ambiguity error

    //@Autowired
    // private ReportDao dao;

    //2. Now in oracle class i mentioned the Repository("dao") then the byName the object created
    //Getting data from the oracle db Report Generated this output get.
    //if in mysql class i mentioned same then ambiguity error occurs.
    @Autowired
    private ReportDao dao;

    public void generatereport() {
        dao.getData();
        System.out.println("Report Generated...");
    }
}

```

-- To resolve this above code problem, we can used @Qualifier annotation

@Qualifier: To identify bean based on the given name for DI

- @Qualifier helps fine-tune annotation-based auto wiring. There may be scenarios when we create more than one bean of the same type and want to wire only one of them with a property.
- This can be controlled using @Qualifier annotation along with the @Autowired annotation.
- We use @Qualifier along with @Autowired to provide the bean ID or bean name we want to use in ambiguous situations.
- So in above code we can do small changes in Repository oracle and mysql class we give name @Repository("oracleDB") & and @Repository("MySqlDB") and with @autowire we can @Qualifier annotation. And the name of the class that I want to create the object Example in below code I create object for mysql.

```
package com.codewitharrays.bean;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service
public class ReportService {

    @Autowired
    @Qualifier("MySqlDB")
    private ReportDao dao;

    public void generatereport() {
        dao.getData();
        System.out.println("Report Generated...");
    }
}

package com.codewitharrays.bean;

import org.springframework.stereotype.Repository;

//@Repository("dao")
@Repository("MySqlDB")
public class MySqlReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the mysql db");
    }
}

package com.codewitharrays.bean;

import org.springframework.stereotype.Repository;

//@Repository("dao")
@Repository("OracleDB")
public class OracleReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the oracle db");
    }
}
```

Output:

Getting data from the mysql db

Report Generated...

- We can create object byName if its variable name not match. With @Qualifier annotation.
- But you don't want go with byName and you want go with byType then you can used @primary annotation

```
package com.codewitharrays.bean;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Repository;

//@Repository("dao")
@Repository("MySqlDB")
@Primary
public class MySqlReportDB implements ReportDao{

    public void getData() {
        System.out.println("Getting data from the mysql db");
    }
}
```

- Right now the auto wiring is done by field. In below code we can do by using Setter.

```
package com.codewitharrays.bean;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service
public class ReportService {

    private ReportDao dao;

    @Autowired
    public void setDao(ReportDao dao) {
        System.out.println("Setter method called");
        this.dao=dao;
    }

    public void generatereport() {
        dao.getData();
        System.out.println("Report Generated...");
    }
}
```

- when we want go for constructor injection autowired

```
package com.codewitharrays.bean;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service
public class ReportService {

    private ReportDao dao;

    //4. when we want go for constructor injection autowired
    @Autowired
    public ReportService(ReportDao dao) {
        System.out.println("report service constructor");
        this.dao = dao;
    }

    public void generatereport() {
        dao.getData();
        System.out.println("Report Generated...");
    }
}
```

- Okay now if I remove the @Autowired from the constructor it will work or not it will work because the @service is the spring bean class and for that IoC will create and object and to create object the constructor is needed and the parameterized constructor is available so here @Autowired is optional.
- But in case I have 2 constructors. One is default constructor and second is parameterized constructor then the @Autowired is given compulsory.

```
package com.codewitharrays.bean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service
public class ReportService {

    private ReportDao dao;

    public ReportService() {
        System.out.println("report service : 0 constructor");
    }
    //If two constructor is present then the @autowired is given compulsory else its optional
    @Autowired
    public ReportService(ReportDao dao) {
        System.out.println("report service: Param constructor");
        this.dao = dao;
    }

    public void generatereport() {
        dao.getData();
        System.out.println("Report Generated...");
    }
}
```

Spring Bean Life Cycle

- => The class which is managed by IOC is called as Spring Bean.
- => We can perform some operations when bean object created and before bean object is removed.

- post construct
- pre destroy

=> To achieve above requirement we can use bean life cycle methods

=> Bean Life cycle methods we can execute in 3 ways

- 1) Declarative approach (xml)
- 2) Programmatic approach
- 3) Annotation approach

1. Declarative Approach

```
public class UserDao {  
    public void init() {  
        System.out.println("getting db connection...");  
    }  
    public void getData() {  
        System.out.println("getting data from db....");  
    }  
    public void destroy() {  
        System.out.println("closing db connection...");  
    }  
}  
<bean id="dao"  
      class="in.codewitharrays.UserDao"  
      init-method="init"  
      destroy-method="destroy"  
      />
```

2. Programmatic Approach

=> We need to implement 2 interfaces here

- 1) InitializingBean ----> afterPropertiesSet ()
- 2) DisposableBean ---> destroy ()

```
public class UserDao implements InitializingBean, DisposableBean{  
    public void afterPropertiesSet() throws Exception {  
        System.out.println("init method....");  
    }  
    public void destroy() throws Exception {  
        System.out.println("destroy method....");  
    }  
}
```

```
public void getData() {  
    System.out.println("getting data from db....");  
}  
}
```

3. Annotation Approach

=> We have below 2 annotations to achieve life cycle methods execution

- 1) @PostConstruct
- 2) @PreDestroy

@DependsOn: It is used to specify one class is indirectly dependent on another class.

```
@Component  
public class UserDao {  
    @PostConstruct  
    public void init() throws Exception {  
        System.out.println("init method....");  
    }  
  
    @PreDestroy  
    public void destroy() throws Exception {  
        System.out.println("destroy method....");  
    }  
    public void getData() {  
        System.out.println("getting data from db....");  
    }  
}  
  
@Component("userdao")  
public class UserDao implements InitializingBean {  
    @Override  
    public void afterPropertiesSet() throws Exception {  
        System.out.println("get data from db...");  
        System.out.println("store data into redis...");  
    }  
}
```

Project Lombok

- > Project Lombok is a third party library
- > It is used to avoid boiler-plate-code in project
- > Project Lombok will generate below things for our classes
- 1) setter methods
- 2) getter methods
- 3) 0-param constructor
- 4) param-constructor
- 5) `toString()` method
- 6) `equals()` method
- 7) `hashCode()`

Project Lombok Setup

Step-1) Add Lombok Dependency in pom.xml file

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.26</version>
</dependency>
```

Step-2) Install Lombok jar in our IDE (Eclipse / STS / IntelliJ)

- > Goto lombok jar location
- > execute below command
 - \$ java -jar <lombok-jar-file-name>
- > Specify IDE location (eclipse.exe / STS.exe)
- > Click on install
- > Close installer
- > Re-Start IDE

Note: Step-2 is required only first time.

Project Lombok Annotations

=> Project Lombok provided annotations to generate boiler plate code.

- 1) `@Setter` : To generate setter methods for variables
- 2) `@Getter` : To generate getter methods for variables
- 3) `@ToString` : To generate `toString()` method
- 4) `@NoArgsConstructor` : To generate 0-param constructor

- 5) @AllArgsConstructor : To generate param-constructor
- 6) @EqualsAndHashCode : To generate equals () & hashCode () methods
- 7) @Data =
 @Setter + @Getter + @NoArgsConstructor + @ToString + @EqualsAndHashCode

```
@Setter  
@Getter  
@ToString  
@AllArgsConstructor  
@NoArgsConstructor  
public class Person {  
  
    private Integer personId;  
    private String personName;  
    private String gender;  
    private Long phno;  
    private Date dob;  
}  
  
import lombok.Data;  
  
@Data  
public class User {  
  
    private Integer userid;  
    private String username;  
    private String userEmail;  
    private String pwd;  
    private String gender;  
    private Long phno;  
}  
  
public class Test {  
  
    Run | Debug  
    public static void main(String[] args) {  
        Person p = new Person();  
        p.setPersonId(personId:101);  
        p.setGender(gender:"male");  
        p.setPersonName(personName:"codewitharrays");  
  
        System.out.println(p);  
    }  
}
```

Spring Boot

-> Spring Boot is one approach to develop Spring Based applications with less configurations.

-> Using Spring Boot we can develop below types of applications

- 1) Standalone app
- 2) Web
- 3) Distributed (webservices)

-> We have below advantages with Spring Boot

1) Starter Pom (simplifies maven/Gradle build configuration)

- > web-starter
- > datajpa-starter
- > security-starter
- > mail-starter

2) Auto Configuration (boot will identify required configs for our app)

- > Creating DB connection pool
- > deploy web app in embedded server
- > Start IOC container
- > Component Scanning

First Spring Boot App: (We can learn this advantages by using simple code)

1. Click on file -> New -> Spring Starter Project -> Artifact name -> maven -> jar -> version
2. Add dependency Spring Web, Spring Security etc. Finished.

```
package com.codewitharrays;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @GetMapping("/")
    public String Welcome() {
        return "Welcome to codewitharrays";
    }
}
```

Now we write some code that. And instantly we created the web MVC project. By using localhost:8080 we can check on the chrome.

Output: Welcome to codewitharrays

But now directly access I want to add some security so for that spring security dependency added.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

We cannot access directly we have to provide the username and password. In application. Properties we must set this configuration

`spring.security.user.name=codewitharrays`

`spring.security.user.password=codewitharrays`



now I created one class Report Service and make one constructor and annotated at @Service and now the IoC make an object for that class. And here the @componentScan is also not required that is the magic of Spring boot is auto configure lot of things.

```
package com.codewitharrays.beans;

import org.springframework.stereotype.Service;

@Service
public class ReportService {

    public ReportService() {
        System.out.println("Report service constructor");
    }
}
```

3) Actuators (To monitor and manage our application)

- > how many beans loaded ?
- > How many url-patterns mapped
- > What configuration properties loaded ?
- > What is health of app?
- > Heap Dump
- > Thread Dump

4) Embedded Servers: It provides server to run our boot application

- > Apache Tomcat (default)
 - > Jetty
 - > netty
- Spring boot default running server on 8080 port no. But if you want to change the port no then in application properties we have to configure **server.port=9090**.

Creating Spring Boot Application

- 1) Initializer website (start.spring.io)
- 2) IDE

Note: IDE will internally connect with start.spring.io website to create Spring Boot application (Spring Starter Project)

Spring Boot = Spring - xml configurations + Auto Config + Embedded Servers + Actuators

The @SpringBootApplication annotation is equal to below 3 annotations

- 1) @SpringBootConfiguration ----> @Configuration
- 2) @EnableAutoConfiguration
- 3) @ComponentScan

How IOC starting in Spring Boot ?

=> SpringApplication.run () method contains logic to start IOC container

boot-starter :: AnnotationConfigApplicationContext

web-starter :: AnnotationConfigServletWebServerApplicationContext

starter-webflux :: AnnotationConfigReactiveWebServerApplicationContext

What is Banner in Spring Boot?

-> Spring Logo which is printing on console is called as banner

-> We have below 3 modes for banner

- 1) console (default) ---> prints on console
- 2) log ----> prints on log file
- 3) off ---> don't print banner

-> We can set banner mode in application.properties file

```
spring.main.banner-mode=off
```

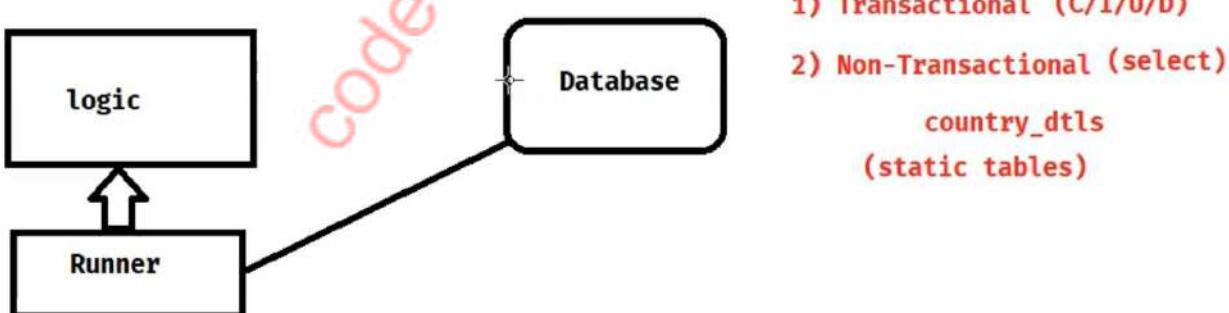
-> We can customize banner text in spring boot by creating banner.txt file under src/main/resources folder.

ASCII Text Gen URL: <https://patorjk.com/software/taag/#p=display&f=Graffiti&t=SBI%20API>

What is Runner in spring boot?

- Spring boot provided runner to execute logic only once when application starts.
- We have 2 types of Runners in spring boot
 - a) ApplicationRunner --> run ()
 - b) CommandLineRunner --> run ()

Note: Both are functional interfaces. Only one abstract method



The one is transactional data who can be modified and second one is non-transactional data who's static and cannot be modified. If in my application if I called every time static data even data is same there is no use of that. To resolve this issue we can use runner. Spring boot provided runner to execute logic only once when application starts.

```
package com.codewitharrays.runners;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

```
package com.codewitharrays.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.*;

public class MyCmdRunner implements CommandLineRunner {
    @Override
    public void run(String... args) throws Exception {
        System.out.println("Command line runner run() method:");
    }
}
```

Use Cases in which scenario we can used Runner

1. Load Static tables data when application starts.
2. Delete data from staging tables (temporary tables)
3. Send notification regarding application startup

What is springApplication.run() method?

1. **Load Initializers:**
 - Spring Boot loads any ApplicationContextInitializer classes that have been registered. These initializers are used to customize the ApplicationContext before it is refreshed.
2. **Start Listeners:**
 - Any ApplicationListener beans are registered and started. These listeners are used to listen for various events during the lifecycle of the Spring Boot application (such as context refresh events, start events, etc.).
3. **Print Banner:**
 - Before the application starts, Spring Boot prints a banner to the console (the banner can be customized or disabled).
4. **Create IOC Container (ApplicationContext):**
 - It creates and refreshes the ApplicationContext (which is the IoC container in Spring). The ApplicationContext is responsible for managing all the beans and dependencies of the application.

5. Call Runners:

- Spring Boot calls any CommandLineRunner or ApplicationRunner beans that have been defined. These beans are used to run specific logic right after the application context is loaded.

6. Return IoC Object (ApplicationContext):

- Finally, the SpringApplication.run() method returns the ApplicationContext object, which represents the IoC container. This can be used to interact with the application context programmatically.

Spring Data JPA

Spring Data JPA is one module in spring framework

It is used to develop Persistence Layer (DB Logic). We can develop Persistence layer by using this 5

1. Java JDBC
2. Spring JDBC
3. Hibernate Framework
4. Spring ORM

5. Spring Data JPA

- Spring Data JPA is simplifying CRUD operations implementation in Project. Lets understand by example: I have multiple classes in my project and for each class I have the 4 method mandatory to implement that insert record, update record, delete record, select record.
- So if I have 5000 tables then the 5000 classes in my project and for that this 4 method so the line of code is around 20000. And that is why Spring Data JPA come in picture by using the data JPA i don't have write single line of code everything take care by Data JPA.

Ex:

USER_MASTER -> UserMasterDao -> 4 methods (insert/update/delete/select)

ROLE_MASTER -> RoleMasterDao -> 4 methods (insert/update/delete/select)

PRODUCT_DTLS -> ProductDtlsDao -> 4 methods (insert/update/delete/select)

PAYMENT_DTLS -> PaymentDtlsDao -> 4 methods (insert/update/delete/select)

Like wise I have 5000 class

5000 *4 = 20000 method

But if I used Data JPA 50000 * 0= 0 methods to write.

- To avoid boiler plate code in DAO classes we can use spring Data JPA.
- Spring Data JPA will use Hibernate framework internally.
- Hibernate framework internally uses JDBC

Spring Data JPA Repositories

- To simplify Persistence Layer Development Data JPA provided Repository interfaces.
 1. **CrudRepository** (CRUD methods)
 2. **JpaRepository** (CRUD methods + Sorting + pagination + Query by example)

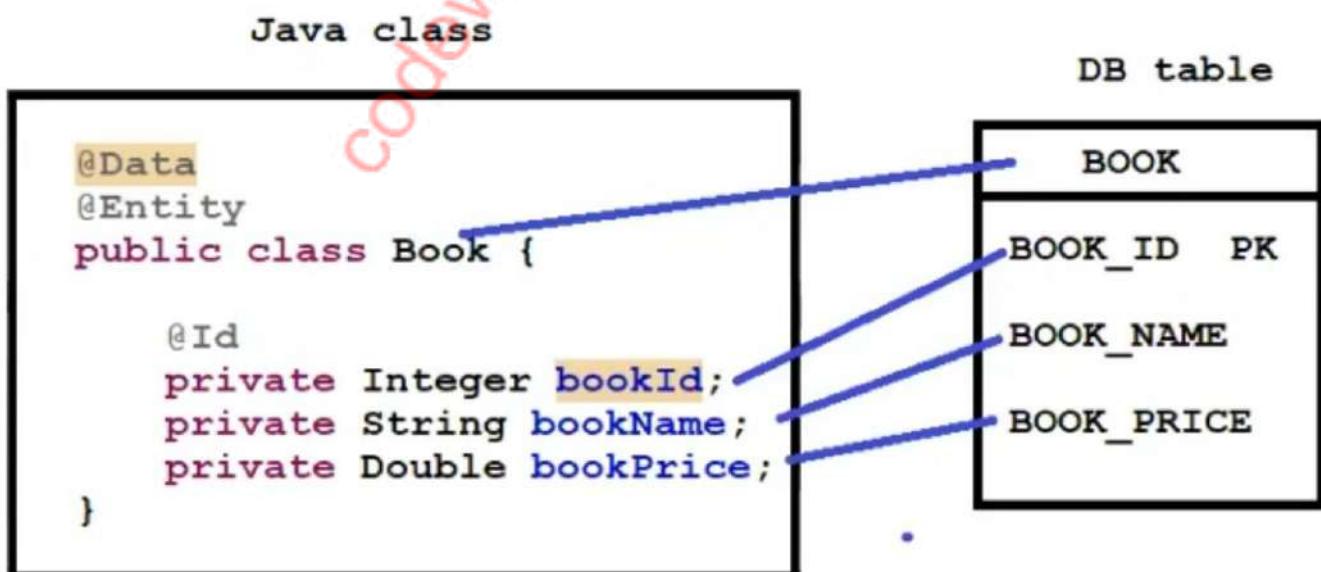
Note: To perform DB operations, we need to create an interface by extending properties from JPA repository interface.

Developing First Data JPA Application

- 1) Create Spring boot application with dependencies
 - a) Data-jpa-starter
 - b) MySQL-driver
 - c) Project-Lombok
- 2) Configure Data source properties in application.properties file
- 3) Create Entity class (Java class to DB table mapping)
- 4) Create Repository interface by extending JPA Repository.
- 5) Test the application by calling Repository interface methods.

```
#This datasource is used to connection with database
spring.datasource.username=root
spring.datasource.password=cdac
spring.datasource.url=jdbc:mysql://localhost:3306/sbms
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```



```
package com.codewitharrays.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Data;

@Data
@Entity //To represent the class as entity class we can use entity annotation
public class Book {

    //If we want to change the table name then we can use the @Table annotation.
    //If we want to change the column name then we can use the @Column annotation
    //In database the primary key require to make the pk we can use the @Id annotation.
    @Id
    private Integer bookId;
    private String bookName;
    private Double bookPrice;
}

package com.codewitharrays.repo;

import org.springframework.data.repository.CrudRepository;

import com.codewitharrays.entity.Book;

//In Repository the class extends by CrudRepository or JpaRepository
//the repository class is the interface class
//The two parameter can take JpaRepository class first is the entity class name and second is the primary key datatype
//For example in this the Book is the entity class name and the Integer is primary key datatype
public interface BookRepository extends CrudRepository<Book, Integer>{

}

package com.codewitharrays;

import java.util.Optional;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ConfigurableApplicationContext;

import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Book b=new Book();
        b.setBookId(101);
        b.setBookName("Springboot");
        b.setBookPrice(1000.00);
        repo.save(b);

        Optional <Book> findById=repo.findById(101);
        System.out.println(findById);
    }
}
```

Methods Present in the CrudRepository

1. save (E): Saves a given entity. (To use insert and update)

- How save method execute the insert and update both the operation by example

```
package com.codewitharrays;

import java.util.Optional;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Book b=new Book();
        b.setBookId(102);
        b.setBookName("Core java");
        b.setBookPrice(2000.00);
        repo.save(b);
    }
}

2024-10-23T11:23:58.071+05:30  INFO 15928 --- [10-Spring-Data-JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_id=?
Hibernate: insert into book (book_name,book_price,book_id) values (?,?,?)
2024-10-23T11:23:58.178+05:30  INFO 15928 --- [10-Spring-Data-JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

- Here we create the new book with id 102 and when code is compiling the hibernate generate the query first with select query check the record is available or not if not then second, he generates the insert query and add the record in database.

```
package com.codewitharrays;

import java.util.Optional;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Book b=new Book();
        b.setBookId(102);
        b.setBookName("Core java");
        b.setBookPrice(4000.00);
        repo.save(b);
    }
}

2024-10-23T11:25:37.469+05:30  INFO 15160 --- [10-Spring-Data-JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_id=?
Hibernate: update book set book_name=?,book_price=? where book_id=?
2024-10-23T11:25:37.580+05:30  INFO 15160 --- [10-Spring-Data-JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

- Here we again create the book with same id 102 and when code is compiling the hibernate generate the query first with select query check the record and the record is found so then the hibernate generate the update query and add the record.
- This is how the save method execute both update and insert record.

2. **saveAll (Iterable)**: Saves all given entities

```
package com.codewitharrays;

import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
            SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Book b1=new Book();
        b1.setBookId(103);
        b1.setBookName("React js");
        b1.setBookPrice(5000.00);

        Book b2=new Book();
        b2.setBookId(104);
        b2.setBookName("HTML");
        b2.setBookPrice(6000.00);

        List<Book> books=new ArrayList<>();
        books.add(b1);
        books.add(b2);
        repo.saveAll(books);
    }
}
```

- We can make b1 record and b2 record and these two records add at a time for that we use saveAll method.
- Also we can use repo.saveAll(Arrays.asList(b1,b2)) if we don't want use this List collection.

3. **findById (Id):** Retrieves an entity by its id. To retrieve record based on given PK

```
package com.codewitharrays;

import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
            SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        //Optional is introduced in java 8 to avoid the null pointer exception and check multiple nulls
        Optional<Book> findById =repo.findById(102);
        if (findById.isPresent()) {
            System.out.println(findById.get());
        }
    }
}

2024-10-25T12:10:56.200+05:30  INFO 19652 --- [10-Spring-Data-JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_id=?
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
2024-10-25T12:10:56.247+05:30  INFO 19652 --- [10-Spring-Data-JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

5. **findAll ():** Returns all instances of the type. To retrieve all records from table

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
            SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Iterable<Book> findAll = repo.findAll();
        for (Book b : findAll) {
            System.out.println(b);
        }
    }
}

2024-10-25T12:32:39.969+05:30  INFO 11084 --- [10-Spring-Data-JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0
Book(bookId=101, bookName=Springboot, bookPrice=1000.0)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
Book(bookId=104, bookName=HTML, bookPrice=6000.0)
2024-10-25T12:32:40.091+05:30  INFO 11084 --- [10-Spring-Data-JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

6. findAllById (Iterable Ids): Returns all instances of the type {@code T} with the given IDs. To retrieve record based on given PKs (multiple pk)

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        Iterable<Book> findAllById = repo.findAllById(Arrays.asList(102,103));

        for (Book b : findAllById) {
            System.out.println(b);
        }
    }
}
```

```
2024-10-25T12:29:04.699+05:30  INFO 10880 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application      :
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_id in (?,?)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
2024-10-25T12:29:04.827+05:30  INFO 10880 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

4. boolean existsById (): Returns whether an entity with the given id exists. To check Presence of record

```
package com.codewitharrays;

import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        boolean status= repo.existsById(102);
        System.out.println("Record present:: "+status);
    }
}

2024-10-25T11:50:55.502+05:30  INFO 21412 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application      :
Hibernate: select count(*) from book b1_0 where b1_0.book_id=?
Record present:: true
2024-10-25T11:50:55.778+05:30  INFO 21412 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

- If the record is present then return the true if not available then return false.

7. count (): Returns the number of entities available. To get records count in table

```
package com.codewitharrays;

import java.util.ArrayList;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        // It will return the count from record
        System.out.println("Records:: "+repo.count());
    }
}

2024-10-25T11:58:53.854+05:30  INFO 22412 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select count(*) from book b1_0
Records:: 4
2024-10-25T11:58:54.086+05:30  INFO 22412 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean
```

8. deleteById (Id): Deletes the entity with the given id. To delete record based on given PK (primary key)

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        if (repo.existsById(104)) {
            repo.deleteById(104);
        }else
            System.out.println("Record not found");
    }
}

2024-10-25T12:59:08.726+05:30  INFO 2272 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application
Hibernate: select count(*) from book b1_0 where b1_0.book_id=?
Record not found
2024-10-25T12:59:08.964+05:30  INFO 2272 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

10. deleteAllById (Iterable Ids): Deletes all instances of the type {@code T} with the given IDs.

Delete records based on given PKs

11. delete (E): Deletes a given entity. Delete record based on given Entity Object

12. deleteAll (Iterable entities): Deletes a given entity. Delete records based on given entity objects

13. deleteAll (): Deletes all entities managed by the repository. Delete all the record from table.

1) Find By Method are available in JPA

2) Custom Queries

```
import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository
        List<Book> findByBookPrice = repo.findByBookPriceGreaterThan(2000);
        for (Book b : findByBookPrice) {
            System.out.println(b);
        }
    }
}

public interface BookRepository extends CrudRepository<Book, Integer>{
    List<Book> findByBookPriceGreaterThan(double price);
}
```

```
2024-10-25T15:12:56.197+05:30  INFO 20376 --- [10-Spring-Data_JPA-APP] [           main] com.codewitharrays.Application      :
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_price>?
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
2024-10-25T15:12:56.357+05:30  INFO 20376 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

- findBy methods are used to perform select operations
- Using non-primary key columns, we can select records
- In findBy methods method name is very important because based on method name JPA will construct the query for execution.
- Find By methods should represent entity class variables

- Here in this example we find the book price less than

```

import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository

        List<Book> lessThan = repo.findByBookPriceLessThan(5000);
        for (Book b : lessThan) {
            System.out.println(b);
        }
    }
}

public interface BookRepository extends CrudRepository<Book, Integer>{
    List<Book> findByBookPriceGreaterThan(double price);
    List<Book> findByBookPriceLessThan(double price);
}

```

2024-10-25T15:29:34.473+05:30 INFO 2188 --- [10-Spring-Data_JPA-APP] [main] com.codewitharrays.Application
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_price<?
Book(bookId=101, bookName=Springboot, bookPrice=1000.0)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
2024-10-25T15:29:34.622+05:30 INFO 2188 --- [10-Spring-Data_JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean

- Here in these examples, we find book by name

```

import com.codewitharrays.entity.Book;
import com.codewitharrays.repo.BookRepository;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext ctx=
        SpringApplication.run(Application.class, args);

        BookRepository repo =ctx.getBean(BookRepository.class);
        //Here we can call the some methods from the CrudRepository

        List<Book> bookName = repo.findByBookName("Core java");
        for (Book b : bookName) {
            System.out.println(b);
        }
    }
}

```

```
public interface BookRepository extends CrudRepository<Book, Integer>{  
    List<Book> findByBookPriceGreaterThan(double price);  
    List<Book> findByBookPriceLessThan(double price);  
    List<Book> findByBookName(String bookName);  
}
```

```
2024-10-25T15:35:04.045+05:30 INFO 15140 --- [10-Spring-Data-JPA-APP] [main] com.codewitharrays.Application :  
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0 where b1_0.book_name=?  
Book(bookId=102, bookName=Core java, bookPrice=4000.0)  
2024-10-25T15:35:04.211+05:30 INFO 15140 --- [10-Spring-Data-JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

Custom Queries

- If we want to execute our query in JPA Repo then we can go for Custom Queries. To Represent custom query we are using @Query annotation.
- Custom Queries we can write in 2 ways
- 1. HQL Queries 2. Native SQL Queries

Following example is native sql query

```
import com.codewitharrays.entity.Book;  
import com.codewitharrays.repo.BookRepository;  
  
@SpringBootApplication  
public class Application {  
  
    public static void main(String[] args) {  
        ConfigurableApplicationContext ctx=  
        SpringApplication.run(Application.class, args);  
  
        BookRepository repo = ctx.getBean(BookRepository.class);  
        //Here we can call the some methods from the CrudRepository  
        List<Book> allBooks = repo.getAllBooks();  
        for (Book b : allBooks) {  
            System.out.println(b);  
        }  
    }  
}  
  
public interface BookRepository extends CrudRepository<Book, Integer>{  
    List<Book> findByBookPriceGreaterThan(double price);  
    List<Book> findByBookPriceLessThan(double price);  
    List<Book> findByBookName(String bookName);  
  
    @Query(value = "Select * from book", nativeQuery = true)  
    List<Book> getAllBooks();  
}
```

```
2024-10-25T15:56:35.381+05:30 INFO 21368 --- [10-Spring-Data-JPA-APP] [main] com.codewitharrays.Application :  
Hibernate: Select * from book  
Book(bookId=101, bookName=Springboot, bookPrice=1000.0)  
Book(bookId=102, bookName=Core java, bookPrice=4000.0)  
Book(bookId=103, bookName=React js, bookPrice=5000.0)  
2024-10-25T15:56:35.485+05:30 INFO 21368 --- [10-Spring-Data-JPA-APP] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

Difference between HQL and SQL Native Queries

- #1
HQL queries are DB independent queries
SQL queries are DB dependent queries
- #2
In HQL query, we will use entity class name & variables
In SQL query, we will use table name & column names
- #3
HQL query can't execute in DB directly (dialect class execute and then convert into sql query).
- SQL query can execute in DB directly.

#4
HQL: Easy maintenance to convert one database to another database in future
SQL: you want fast performance then go for SQL query conversion not required
Note: Every HQL query should be converted to SQL query before execution that conversion will done by Dialect class

Every Database will have its own dialect class.

Ex: oracleDialect , MySqlDialect , DB2Dialect , PostgresDialect etc.

Note: Dialect class will be loaded along with DB driver class.

Example of HQL Query:

```
import java.util.ArrayList;  
  
@SpringBootApplication  
public class Application {  
  
    public static void main(String[] args) {  
        ConfigurableApplicationContext ctx=  
        SpringApplication.run(Application.class, args);  
  
        BookRepository repo = ctx.getBean(BookRepository.class);  
        //Here we can call the some methods from the CrudRepository  
        List<Book> book = repo.getBooks();  
        for (Book b : book) {  
            System.out.println(b);  
        }  
    }  
}  
  
public interface BookRepository extends CrudRepository<Book, Integer>{  
  
    @Query(value = "Select * from book", nativeQuery = true)  
    List<Book> getAllBooks(); //Native Query  
  
    @Query("SELECT b FROM Book b") // HQL query  
    List<Book> getBooks();  
}
```

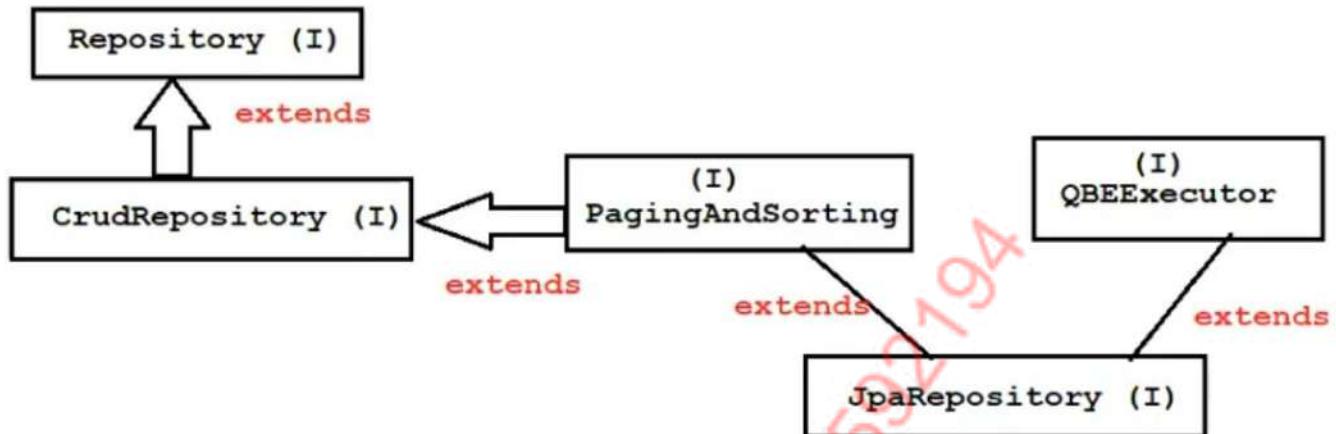
```
2024-10-25T19:04:23.510+05:30  INFO 9220 --- [10-Spring-Data_JPA-APP] [main]
Hibernate: select b1_0.book_id,b1_0.book_name,b1_0.book_price from book b1_0
Book(bookId=101, bookName=Springboot, bookPrice=1000.0)
Book(bookId=102, bookName=Core java, bookPrice=4000.0)
Book(bookId=103, bookName=React js, bookPrice=5000.0)
2024-10-25T19:04:23.614+05:30  INFO 9220 --- [10-Spring-Data_JPA-APP] [ionShutdownHook]
```

Spring boot shortcut key for Java

- 1. Ctrl + Shift + R :** To find all resource files, including the config XML files from the workspace
 - 2. Ctrl + Shift + T :** To find class in an application or from inside a jar
 - 3. Ctrl + W :** To close the current file
 - 4. Ctrl + Shift + W :** To close all the open files
 - 5. Ctrl + Shift + O :** To organise the missing imports
 - 6. Ctrl + Shift + F :** For auto-formatting
 - 7. Ctrl + / :** To add/remove single line comment for selected line
 - 8. Ctrl + Shift + Enter :** To add blank line before the current line
 - 9. Alt + Shift + S, R :** To generate Setters & Getters
 - 10. Alt + Shift + S, S :** To generate toString() method
 - 11. Alt + Shift + S, o :** To add constructor with fields
 - 12. Ctrl + 1 + Enter :** To store method return value to variable
 - 13. Ctrl + D :** To delete current line
 - 14. Ctrl + O :** Display all methods of current class, Pressing Ctrl + O again will display all the inherited methods
 - 15. Ctrl + Shift + P :** Go to matching bracket
 - 16. Alt + Shift + R :** To rename
 - 17. Alt + Shift + T :** To open the context-sensitive refactoring menu
 - 18. Alt + Shift + Z :** To surround a block with try and catch
 - 19. Ctrl + Shift + / :** To comment and uncomment lines with a block comment
 - 20. Ctrl + Q :** To go to the last edited position
-

JpaRepository

- It is a predefined interface available in data jpa.
- Using JpaRepository also we can perform CRUD Operations
- We can perform Crud Operations + Sorting + Pagination + QBE



Understand the JPA Repository By coding Example:

```

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "emp_tbl")
public class Employee {

    @Id
    private Integer empId;
    private String empName;
    private String gender;
    private Double salary;
    private String deptName;
}

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Employee e1=new Employee(1, "Ashok", "Male", 5000.00, "Backend");
        repo.save(e1);
    }
}

import org.springframework.data.jpa.repository.JpaRepository;
import com.codewitharrays.entity.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {
}
  
```

- First record is inserted by e1 and we use save() method present in JpaRepository.
- All methods from crud Repository and also additional methods are available in the JPA Repository.
- Now we can add multiple employee records by using saveAll() methods.

```
package com.codewitharrays;

import java.util.Arrays;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Employee e2=new Employee(2, "Suraj", "Male", 7000.00, "Frontend");
        Employee e3=new Employee(3, "Shubham", "Male", 8000.00, "Database");
        Employee e4=new Employee(4, "Vikas", "Male", 11000.00, "Production");
        Employee e5=new Employee(5, "Niraj", "Male", 15000.00, "Fullstack");
        Employee e6=new Employee(6, "Saurabh", "Male", 9000.00, "HR");

        repo.saveAll(Arrays.asList(e2,e3,e4,e5,e6));
    }
}
```

- Now we can find the record by using findAll() method.

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        List<Employee> findAll = repo.findAll();
        for (Employee emp : findAll) {
            System.out.println(emp);
        }
    }
}

Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)
Employee(empId=2, empName=Suraj, gender=Male, salary=7000.0, deptName=Frontend)
Employee(empId=3, empName=Shubham, gender=Male, salary=8000.0, deptName=Database)
Employee(empId=4, empName=Vikas, gender=Male, salary=11000.0, deptName=Production)
Employee(empId=5, empName=Niraj, gender=Male, salary=15000.0, deptName=Fullstack)
Employee(empId=6, empName=Saurabh, gender=Male, salary=9000.0, deptName=HR)
2024-10-26T12:41:15.102+05:30  INFO 18872 --- [11-DataJPA_Repo-App] j.LocalContainerEntityManagerFactoryBean :
```

- But if we observed carefully the record I get directly the way present in the database by I want to sort the record byName , bySalary etc. then the JpaRepository has PagingAndSorting interface class methods extending and we can use those methods.
- Iterable<T> findAll(Sort sort); For sorting ...
- Page<T> findAll(Pageable pageable); for pagination...

- Sort.by(" ") these can be used. Lets understand by coding example how to sort by ascending and descending order.

1. Ascending Order

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Sort ascending = Sort.by("empName").ascending();

        List<Employee> findAll = repo.findAll(ascending);
        for (Employee emp : findAll) {
            System.out.println(emp);
        }
    }
}
```

2024-10-26T12:53:13.331+05:30 INFO 13184 --- [11-DataJPA_Repo-App] [main] com.codewitharrays.Application : Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0 order by e1_0.emp_name Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)
Employee(empId=5, empName=Niraj, gender=Male, salary=15000.0, deptName=Fullstack)
Employee(empId=6, empName=Saurabh, gender=Male, salary=9000.0, deptName=HR)
Employee(empId=3, empName=Shubham, gender=Male, salary=8000.0, deptName=Database)
Employee(empId=2, empName=Suraj, gender=Male, salary=7000.0, deptName=Frontend)
Employee(empId=4, empName=Vikas, gender=Male, salary=11000.0, deptName=Production)

2024-10-26T12:53:13.503+05:30 INFO 13184 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :

2. Descending order

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Sort descending = Sort.by("empName").descending();

        List<Employee> findAll = repo.findAll(descending);
        for (Employee emp : findAll) {
            System.out.println(emp);
        }
    }
}
```

2024-10-26T12:56:23.386+05:30 INFO 7460 --- [11-DataJPA_Repo-App] [main] com.codewitharrays.Application : Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0 order by e1_0.emp_name desc Employee(empId=4, empName=Vikas, gender=Male, salary=11000.0, deptName=Production)
Employee(empId=2, empName=Suraj, gender=Male, salary=7000.0, deptName=Frontend)
Employee(empId=3, empName=Shubham, gender=Male, salary=8000.0, deptName=Database)
Employee(empId=6, empName=Saurabh, gender=Male, salary=9000.0, deptName=HR)
Employee(empId=5, empName=Niraj, gender=Male, salary=15000.0, deptName=Fullstack)
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)

2024-10-26T12:56:23.935+05:30 INFO 7460 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :

Also you can do it by Salary. But I want to sort by passing the multiple properties also that can understand by coding example:

```
Sort sort = Sort.by("empName", "salary").descending();
```

Pagination

- Dividing total records into multiple pages is called as pagination.
- Page_Size : 2 Total Records: 6 Total page : 3 pages
- What is page number : ?
- 1st Page first 2 records
- 2nd page next 2 record
- 3rd page next 2 record
- To implement this we have to create object of PageRequest.of(pageNo, pageSize) the parameter is pageNo and the second parameter pageSize it will take.
- In Jpa pageNo should start with 0.

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Sort sort = Sort.by("empName", "salary").descending();
        int pageNo=1;
        PageRequest page = PageRequest.of(pageNo-1, 2);

        Page<Employee> findAll = repo.findAll(page);
        List<Employee> content = findAll.getContent();
        for (Employee emp : content) {
            System.out.println(emp);
        }
    }
}

2024-10-26T14:33:31.654+05:30  INFO 5920 --- [11-DataJPA_Repo-App] [           main] com.codewitharrays.Application      :
Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0 limit ?,?
Hibernate: select count(e1_0.emp_id) from emp_tbl e1_0
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)
Employee(empId=2, empName=Suraj, gender=Male, salary=7000.0, deptName=Frontend)
2024-10-26T14:33:31.795+05:30  INFO 5920 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean :
```

Query By Example: It is used to prepare Dynamic Query based on data available in Entity

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
        EmployeeRepository repo = context.getBean(EmployeeRepository.class);

        Employee e1=new Employee();
        e1.setGender("Male");
        e1.setDeptName("Backend");
        Example<Employee> example = Example.of(e1);
        List<Employee> emp = repo.findAll(example);
        for (Employee e : emp) {
            System.out.println(e);
        }
    }
}
```

```
2024-10-26T15:00:59.077+05:30 INFO 12320 --- [11-DataJPA_Repo-App] [main] com.codewitharrays.Application : Started Application  
Hibernate: select e1_0.emp_id,e1_0.dept_name,e1_0.emp_name,e1_0.gender,e1_0.salary from emp_tbl e1_0 where e1_0.gender=? and e1_0.dept_name=?  
Employee(empId=1, empName=Ashok, gender=Male, salary=5000.0, deptName=Backend)  
2024-10-26T15:00:59.217+05:30 INFO 12320 --- [11-DataJPA_Repo-App] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'Default'  
2024-10-26T15:00:59.219+05:30 INFO 12320 --- [11-DataJPA_Repo-App] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 -
```

- Right now, the data set hardcoded but in web mvc this data set by UI.

TimeStamping In JPA

- It is used to insert CREATED_DATE & UPDATED_DATE columns values for the record
- @CreationTimeStamp: To populate record created date
- @UpdateTimeStamp: To populate record updated date
- We can understand by the coding example.

```
package com.codewitharrays.repo;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.codewitharrays.entity.User;  
  
public interface UserRepository extends JpaRepository<User, Integer> {  
  
}  
  
  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@NoArgsConstructor  
@Entity  
@Table(name = "user_tbl")  
public class User {  
  
    @Id  
    private Integer userId;  
    private String userName;  
    private String userGender;  
    private String deptName;  
    private Double salary;  
  
    @CreationTimestamp  
    private LocalDate dateCreated;  
  
    @UpdateTimestamp  
    private LocalDate lastUpdated;  
}
```

```

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "user_tbl")
public class User {

    @Id
    private Integer userId;
    private String userName;
    private String userGender;
    private String deptName;
    private Double salary;

    @CreationTimestamp
    private LocalDate dateCreated;

    @UpdateTimestamp
    private LocalDate lastUpdated;
}

```

	user_id	date_created	dept_name	last_updated	salary	user_gender	user_name
▶	101	2024-10-26	Admin	2024-10-26	NULL	male	codewitharrays
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- In Above code When we used this `@CreationTimeStamp` Annotation and `@UpdateTimeStamp` And insert one record in update column the date is inserted we don't want like that. When the new record inserted the timestamp create and when we update record then update column execute.
- To resolve this problem, we can use attribute in Creation `Updateable=false;` and in updating `insertable=false;`

```

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "user_tbl")
public class User {

    @Id
    private Integer userId;
    private String userName;
    private String userGender;
    private String deptName;
    private Double salary;

    @CreationTimestamp
    @Column(name = "date_create", updatable = false)
    private LocalDate dateCreated;

    @UpdateTimestamp
    @Column(name = "last_update", insertable = false)
    private LocalDate lastUpdated;
}

```

```

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        UserRepository repo = context.getBean(UserRepository.class);

        User user=new User();
        user.setUserId(102);
        user.setDeptName("Super Admin");
        user.setUserName("Ashok Pate");
        user.setUserGender("male");
        repo.save(user);
    }
}

```

	user_id	date_create	dept_name	last_update	salary	user_gender	user_name
▶	102	2024-10-26	Super Admin	NULL	NULL	male	Ashok Pate
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- Now the problem is resolved. We update some value now.

```

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        UserRepository repo = context.getBean(UserRepository.class);

        User user=new User();
        user.setUserId(102);
        user.setDeptName("HR");
        user.setUserName("Ashok Pate");
        user.setUserGender("male");
        repo.save(user);
    }
}

```

	user_id	date_create	dept_name	last_update	salary	user_gender	user_name
▶	102	2024-10-26	HR	2024-10-26	NULL	male	Ashok Pate
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Generators

- Generators are used to generate the value for Primary key Columns
- For generators we use @GeneratedValue annotation
- Why we need generators. Right now, we passed the id but while dynamic user don't know about the primary key value so it will be auto generated and that will take care spring boot.
- Also, while defining entity variable, we used Wrapper classes instead of primitive type. Because when we use primitive data type and run the application the default value is stored in data base i.e not recommended. And when we use Wrapper class the null value is store. i.e recommended. If without @GeneratedValue run application then null value stored in primary key column and it will throw null pointer exception.
- Primary Key is a constraint in database it is used to maintain unique data in column
- Primary Key constraint is combination of below 2 constraints
 - 1) UNIQUE
 - 2) NOT NULL

We can specify below strategies for Generator

- 1) AUTO: it will take automatically if its MySQL database then take IDENTITY and if oracle then take SEQUENCE.
- 2) IDENTITY (MySQL)
- 3) SEQUENCE (oracle)
- 4) TABLE: It will maintain another table to get primary key column values

Example:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "user_tbl")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer userId;
    private String userName;
    private String userGender;
    private String deptName;
    private Double salary;

    @CreationTimestamp
    @Column(name = "date_create", updatable = false)
    private LocalDate dateCreated;

    @UpdateTimestamp
    @Column(name = "last_update", insertable = false)
    private LocalDate lastUpdated;
}
```

```

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        UserRepository repo = context.getBean(UserRepository.class);

        User user=new User();

        user.setDeptName("Fullstack Developer");
        user.setUserName("codewitharrays");
        user.setUserGender("male");
        repo.save(user);
    }
}

```

CodeWithArrays.in 8007592194

	user_id	date_create	dept_name	last_update	salary	user_gender	user_name
▶	1	2024-10-26	Fullstack Developer	NULL	NULL	male	codewitharrays
	2	2024-10-26	Fullstack Developer	NULL	NULL	male	codewitharrays
*	3	2024-10-26	Fullstack Developer	NULL	NULL	male	codewitharrays
	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Custom Generator

- We want to generate primary key columns values as per project requirement like below.

ORDER_DTLS table: OD_1, OD_2, OD_3 OD_100

- To achieve this requirement, we need to develop our own generator. That is called as Custom Generator.

```

public class OrderIdGenerator implements IdentifierGenerator {

    public void generate(){
        String prefix = "OD_"
        String suffix = get from db; [1,2,3...]
        String pk = prefix+suffix;
        //set pk to entity obj
    }
}

```

Composite Primary Keys

- More than one Primary Key column in table is known as composite primary key
- Below is SQL query to create a table with multiple primary key columns

```
create table account_tbl (
    acc_num bigint not null,
    acc_type varchar(255) not null,
    branch varchar(255),
    holder_name varchar(255),
    primary key (acc_num, acc_type)
);
```

- To work with Composite PKs we will use below 2 annotations
 1. **@Embeddable**: Class level annotation to represent primary key columns
 2. **@EmbeddedId**: Variable level annotation to present Embeddable class

Note: Generators will not support for Composite Primary keys

Note: The class which is representing Composite Keys should implement Serializable interface.

```
@Data
@Entity
@Table(name = "acc_tbl")
public class Account {

    private String branchName;

    private String holderName;
    @EmbeddedId
    private AccountPK accountPK;
}

@Data
@Embeddable
public class AccountPK implements Serializable{

    private Integer accNumber;
    private String accType;
}

package com.codewitharrays.entity;

import org.springframework.data.jpa.repository.JpaRepository;

public interface AccountRepo extends JpaRepository<Account, AccountPK> {
}
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@Configuration
public class AppConfig {
    @Bean
    public AccountRepo accountRepo() {
        return new AccountRepo();
    }
}

@RestController
public class AccountController {
    private final AccountRepo accountRepo;

    public AccountController(AccountRepo accountRepo) {
        this.accountRepo = accountRepo;
    }

    @PostMapping("/accounts")
    public void save(@RequestBody Account account) {
        accountRepo.save(account);
    }
}

@Repository
public interface AccountRepo extends JpaRepository<Account, Long> {
}
```

- To retrieve the data, we have to set the Composite key

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        AccountRepo repo = context.getBean(AccountRepo.class);

        AccountPK pk=new AccountPK();
        pk.setAccNumber(389649535);
        pk.setAccType("saving");

        Optional<Account> findById = repo.findById(pk);
        if (findById.isPresent()) {
            System.out.println(findById);
        }
    }
}
```

Profiles in Spring Boot

Every Project will have multiple Environments in the Realtime.

Environments are used to test our application before delivering to client

Note: Environment means platform to run our application

(Linux VM, DB Server, Log Server etc...)

We can see below Environments in the Realtime

- 1) Dev Env: Developers will use it for code Integration testing
- 2) SIT Env: Testers will use it application end to end testing
- 3) UAT Env: Client / Client-Side Team will use it for acceptance testing

Note: In UAT client / client-side team will decide GO or NO-GO

GO: Approved for Production deployment

NO-GO: Denied for Production Deployment

- 4) PILOT Env: Pre-Prod Env for testing with live data
- 5) PROD Env: Live Environment (end users will access our prod app)

- Every Environment will have its own Database and every database will have separate configuration properties (uname, pwd and URL)
- If we want to deploy our code into multiple environments then we have to change configuration properties in application.properties file for every deployment. (It is not recommended).
 - 1) DB Props
 - 2) SMTP PProps
 - 3) Kafka Props
 - 4) Redis Props
 - 5) REST API Endpoint URLs
- To avoid this problem we will use Profiles in Spring Boot.
- Profiles are used to configure Environment Specific properties
 1. application-dev.properties
 2. application-sit.properties
 3. application-uat.properties
 4. application-pilot.properties
 5. application-prod.properties

In application.properties file we need to activate profile

```
# Activating dev profile
```

```
spring.profiles.active=dev
```

Note: Above represents application should load properties from Dev properties file.

Spring Web MVC

It is one module in Spring Framework to develop web applications.

Web MVC module simplified web application development process.

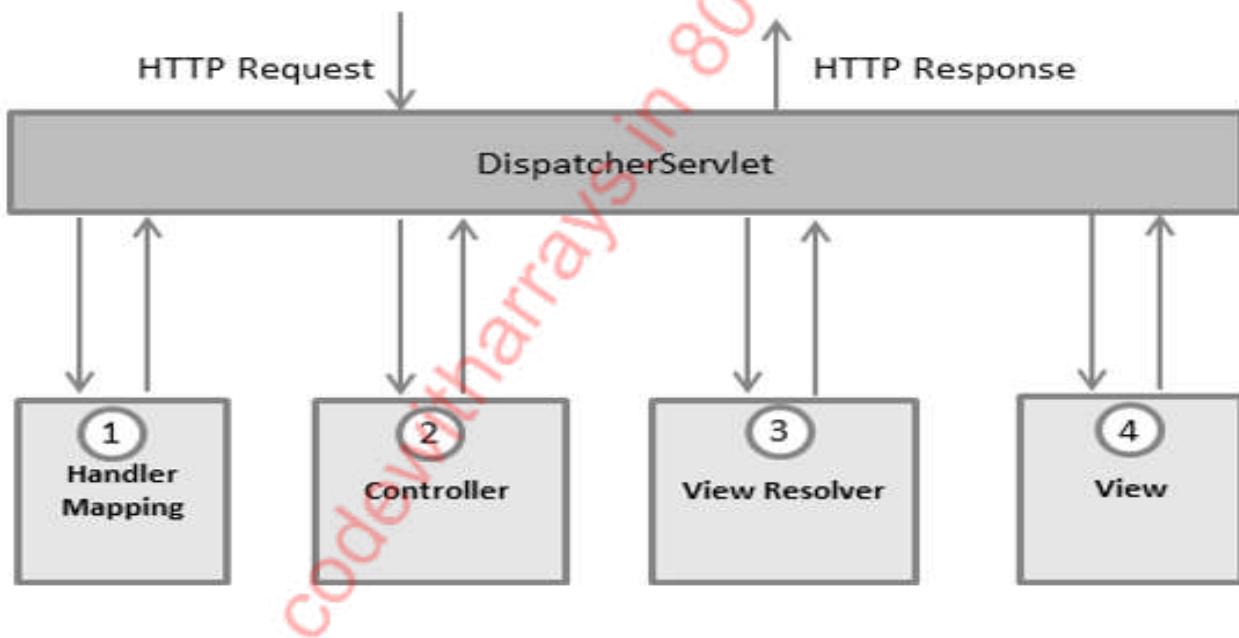
- 1) Form Binding (form <---> java object)
- 2) Flexibility in Form Binding (type conversion)
- 3) Multiple Presentation Technologies (JSP & Thyme leaf)
- 4) Form Tag Library (ready-made tags support)

Note: To develop web application using spring-boot we need to add below starter in pom.xml

spring-boot-starter-web

The above starter provides support for below things

- 1) MVC based web applications
- 2) Restful Services
- 3) Embedded Container (Tomcat)



- **Dispatcher Servlet:** Framework Servlet / Front Controller.
Responsible to perform Pre-Processing and Post-Processing of request
- **Handler Mapper:** Responsible to identify Request Handler class (controller)
- **Controller:** Java class which is responsible to handle request & response
Controller will return **ModelAndView** object to Dispatcher Servlet.
- **Model:** Represents data in key-value format
- **View:** Logical File Name

Note: Controllers are loosely coupled with Presentation technology.

- **View Resolver:** To identify presentation file location and technology
- **View:** It is responsible to render Model data in view file.

Building First Web app with spring boot

- 1) Create Spring Starter Project with below dependencies
 - a) spring-boot-starter-web
 - b) spring-boot-devtools
 - c) tomcat-embed-jasper (mvnrepository.com)
- 2) Create controller class with required methods & map controller methods to URL pattern
- 3) Create View File with presentation logic
- 4) Configure View Resolver in application.properties file
- 5) Run the application and test it.

Observations

- devtools dependency is used to restart our server when we make code changes.
- To represent java class as controller we are using @Controller annotation
- Controller methods we need to map with HTTP methods using unique URL pattern
 - GET --> @GetMapping
 - POST --> @PostMapping
- Apache Tomcat is coming as default embedded container.
- Embedded container port number is 8080. We can change that port number using application.properties file Server.port = 9090
- Spring Boot web apps will not have context path. We can add context-path using application.properties file. server.servlet.context-path=/codewitharrays

First Web MVC Project to Display Welcome message:

```
@Controller
public class MsgController {

    @GetMapping("/welcome")
    public ModelAndView getWelcomeMsg() {
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "Hi, Welcome to Codewitharrays..!!!");
        mav.setViewName("message");
        return mav;
    }

    @GetMapping("/greet")
    public ModelAndView getGreetMsg() {
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "Good Evening..!!!");
        mav.setViewName("message");
        return mav;
    }
}
```

```
spring.mvc.view.prefix=/views/  
spring.mvc.view.suffix=.jsp  
server.servlet.context-path=/codewitharrays
```

Message.jsp \${msg}

02- Project using Web MVC

Requirement: Retrieve book record based on given id and display in web page.

1) Create Spring Starter Project with below dependencies

- a) web-starter
- b) data-jpa
- c) mysql-connector-j
- d) lombok
- e) devtools
- f) tomcat-embed-jasper

2) Configure View Resolver & Data Source properties in application.properties file

3) Create Entity class (table mapping)

4) Create Jpa Repository interface

5) Create Controller class with methods to handle request & response

6) Create View Page

7) Run the application and test it

@RequestParam: To capture request data using key

Model And View:

```
package com.codewitharrays.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
import lombok.Data;  
@Data  
@Entity  
public class Book {  
  
    @Id  
    private Integer bookId;  
    private String bookName;  
    private Double bookPrice;  
}  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.codewitharrays.entity.Book;  
  
public interface BookRepository extends JpaRepository<Book, Integer>{  
}
```

```
@Controller
public class BookController {

    @Autowired
    private BookRepository repo;

    @GetMapping("/book")
    public ModelAndView getBookById(@RequestParam("id") Integer id) {
        System.out.println("id: " + id);
        ModelAndView mav=new ModelAndView();

        Optional<Book> findById = repo.findById(id);
        if (findById.isPresent()) {
            Book bookObj = findById.get();
            System.out.println(bookObj);
            mav.addObject("book", bookObj);
        }
        mav.setViewName("index");
        return mav;
    }

    <html >
    <head>

    </head>
    <body>

        <h1>Book Details</h1>
        <form action="book">
            Book Id: <input type="text" name="id">
            <input type="submit" value="Search">
        </form>
        <hr>
        Book id: ${book.bookId} <br>
        Book Name: ${book.bookName} <br>
        Book Price: ${book.bookPrice}

    </body>
    </html>
```

```
spring.application.name=14-Spring_MVC-Book-App-1
#This datasource is used to connection with database
spring.datasource.username=root
spring.datasource.password=cdac
spring.datasource.url=jdbc:mysql://localhost:3306/sbms
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp
```

Output: Book Details

Book Id:

Book id: 101
Book Name: Springboot
Book Price: 1000.0

We can write the same controller method in different way. This time we take String as return type and directly return index page. Model pass as argument on model we call addAttribute to set the view

```
@GetMapping("/book")
public String getBookById(@RequestParam("id") Integer id, Model model) {
    Optional<Book> findById = repo.findById(id);
    if (findById.isPresent()) {
        Book bookObj = findById.get();
        model.addAttribute("book", bookObj);
    }
    return "index";
}
```

03 – Web Application Requirement: Develop Student Enquiry Form like below

1) Course name drop down values should come from database table

2) Timings checkboxes options should come from database table

Note: When we click on submit button record should inserted into database table (STUDENT_ENQUIRIES) and display success message on the same page.

— Student Enquiry Form —

Name :

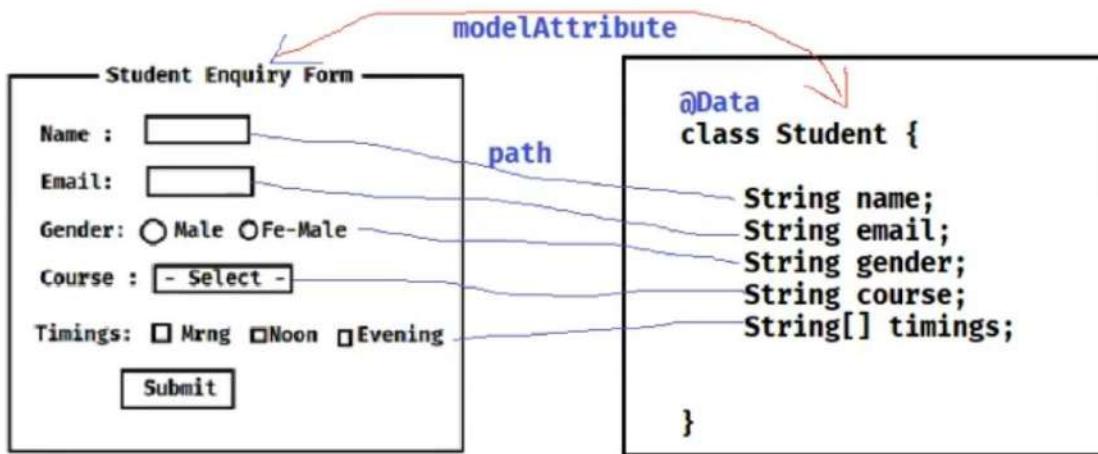
Email:

Gender: Male Fe-Male

Course :

Timings: Mrng Noon Evening

- For above project we map student with modelAttribute and the variable and fields map with path attribute. First we take the hardcoded value for courses and timing to solve.
- Predefined tags provided to simplify Forms development
- To use Spring MVC form tag library in jsp we have to add below taglib url
- <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>



```

package com.codewitharrays.entity;

import lombok.Data;

@Data
public class Student {

    private String name;
    private String email;
    private String gender;
    private String courses;
    private String [] timing;
}



---


package com.codewitharrays.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

import com.codewitharrays.entity.Student;

@Controller
public class StudentController {

    @GetMapping("/")
    public String loadIndex(Model model) {
        Student std=new Student();
        model.addAttribute("student", std);

        return "index";
    }

    @PostMapping("/save")
    public String handleSaveMethod(Student s,Model model) {
        System.out.println(s);
        model.addAttribute("msg", "Data save");
        return "index";
    }
}

```

```

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <h1>Student Form</h1>
    <p> <font color="green">${msg}</font> </p>
    <form:form action="save" modelAttribute="student" method="POST">
        <table>
            <tr>
                <td>Name: </td>
                <td> <form:input path="name" /> </td>
            </tr>
            <tr>
                <td>Email:</td>
                <td><form:input path="email"/> </td>
            </tr>
            <tr>
                <td>Gender:</td>
                <td><form:radiobutton path="gender" value="M"/> Male </td>
                <td><form:radiobutton path="gender" value="F"/> Fe-Male </td>
            </tr>
            <tr>
                <td>Course</td>
                <td><form:select path="courses">
                    <form:option value="">-Select-</form:option>
                    <!-- This is hardcoded value -->
                    <form:option value="java">Java</form:option>
                    <form:option value="springboot">SpringBoot</form:option>
                    <form:option value="react">React</form:option>
                </form:select>
                </td>
                <td>Timing</td>
                <td>
                    <!-- This is hardcoded value -->
                    <form:checkbox path="timing" value="mrng"/>Morning
                    <form:checkbox path="timing" value="noon"/>Afternoon
                    <form:checkbox path="timing" value="evening"/>Evening
                </td>
            <tr>
                <td><input type="submit" value="save"></td>
            </tr>
        </table>
    </form:form>
</body>
</html>

```

Student Form

Data save

Name:	<input type="text" value="ashok"/>
Email:	<input type="text" value="ashok@gmail.com"/>
Gender:	<input checked="" type="radio"/> Male <input type="radio"/> Fe-Male
Course	<input type="button" value="SpringBoot"/> <input checked="" type="checkbox" value="mrng"/> Morning <input type="checkbox" value="noon"/> Afternoon <input type="checkbox" value="evening"/> Evening
<input type="button" value="save"/>	

- Now we make service class and pass the course and timing dynamically.

```

package com.codewitharrays.service;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Service;

@Service
public class StudentService {

    public List<String> getcourses() {
        return Arrays.asList("Java", "Springboot", "react");
    }

    public List<String> getTiming() {
        return Arrays.asList("Morning", "Afternoon", "Evening");
    }
}

@Controller
public class StudentController {

    @Autowired
    private StudentService service;

    @GetMapping("/")
    public String loadIndex(Model model) {
        Student std=new Student();
        model.addAttribute("student", std);
        List<String> courses = service.getcourses();
        model.addAttribute("course", courses);
        List<String> timing = service.getTiming();
        model.addAttribute("time", timing);
        return "index";
    }

    @PostMapping("/save")
    public String handleSaveMethod(Student s, Model model) {
        System.out.println(s);
        model.addAttribute("msg", "Data save");
        Student std=new Student();
        model.addAttribute("student", std);
        List<String> courses = service.getcourses();
        model.addAttribute("course", courses);
        List<String> timing = service.getTiming();
        model.addAttribute("time", timing);
        return "index";
    }
}

```

```

<td>Course</td>
<td><form:select path="courses">
    <form:option value="">-Select-</form:option>
    <form:options items="${course}" />
    <!-- Dynamically taken the values -->
</form:select>
</td>
<td>Timing</td>
<td>
    <!-- Dynamically taken the values -->
    <form:checkboxes path="timing" items="${time}" />
</td>
<tr>
    <td><input type="submit" value="save"></td>
</tr>

```

- Now we can improve our controller code. That formInitBinding method again we write in handleSaveMethod otherwise we get the items null exception. Because the after data save page doesn't reload and we need that courses , timing value in that for that we write this method.

```

@Controller
public class StudentController {

    @Autowired
    private StudentService service;

    private void formInitBinding(Model model) {
        model.addAttribute("student", new Student());
        model.addAttribute("course", service.getCourses());
        model.addAttribute("time", service.getTiming());
    }

    @GetMapping("/")
    public String loadIndex(Model model) {
        formInitBinding(model);
        return "index";
    }

    @PostMapping("/save")
    public String handleSaveMethod(Student s, Model model) {
        System.out.println(s);
        model.addAttribute("msg", "Data save");
        formInitBinding(model);
        return "index";
    }
}

```

codewitharrays.in 8007592194

Explore More

Subscription : Premium CDAC NOTES & MATERIAL



Contact to Join
Premium Group



Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

codewitharrays.in freelance project available to buy contact on 8007592194		
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySQL
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySQL
3	Tour and Travel management System	React+Springboot+MySQL
4	Election commition of India (online Voting System)	React+Springboot+MySQL
5	HomeRental Booking System	React+Springboot+MySQL
6	Event Management System	React+Springboot+MySQL
7	Hotel Management System	React+Springboot+MySQL
8	Agriculture web Project	React+Springboot+MySQL
9	AirLine Reservation System / Flight booking System	React+Springboot+MySQL
10	E-commerce web Project	React+Springboot+MySQL
11	Hospital Management System	React+Springboot+MySQL
12	E-RTO Driving licence portal	React+Springboot+MySQL
13	Transpotation Services portal	React+Springboot+MySQL
14	Courier Services Portal / Courier Management System	React+Springboot+MySQL
15	Online Food Delivery Portal	React+Springboot+MySQL
16	Municipal Corporation Management	React+Springboot+MySQL
17	Gym Management System	React+Springboot+MySQL
18	Bike/Car ental System Portal	React+Springboot+MySQL
19	CharityDonation web project	React+Springboot+MySQL
20	Movie Booking System	React+Springboot+MySQL

freelance_Project available to buy contact on 8007592194

21	Job Portal web project	React+Springboot+MySQL
22	LIC Insurance Portal	React+Springboot+MySQL
23	Employee Management System	React+Springboot+MySQL
24	Payroll Management System	React+Springboot+MySQL
25	RealEstate Property Project	React+Springboot+MySQL
26	Marriage Hall Booking Project	React+Springboot+MySQL
27	Online Student Management portal	React+Springboot+MySQL
28	Resturant management System	React+Springboot+MySQL
29	Solar Management Project	React+Springboot+MySQL
30	OneStepService LinkLabourContractor	React+Springboot+MySQL

31	Vehical Service Center Portal	React+Springboot+MySQL
32	E-wallet Banking Project	React+Springboot+MySQL
33	Blogg Application Project	React+Springboot+MySQL
34	Car Parking booking Project	React+Springboot+MySQL
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySQL
36	Society management Portal	React+Springboot+MySQL
37	E-College Portal	React+Springboot+MySQL
38	FoodWaste Management Donate System	React+Springboot+MySQL
39	Sports Ground Booking	React+Springboot+MySQL
40	BloodBank mangement System	React+Springboot+MySQL
41	Bus Tickit Booking Project	React+Springboot+MySQL
42	Fruite Delivery Project	React+Springboot+MySQL
43	Woodworks Bed Shop	React+Springboot+MySQL
44	Online Dairy Product sell Project	React+Springboot+MySQL
45	Online E-Pharma medicine sell Project	React+Springboot+MySQL
46	FarmerMarketplace Web Project	React+Springboot+MySQL
47	Online Cloth Store Project	React+Springboot+MySQL
48	Train Ticket Booking Project	React+Springboot+MySQL
49	Quizz Application Project	JSP+Springboot+MySQL
50	Hotel Room Booking Project	React+Springboot+MySQL
51	Online Crime Reporting Portal Project	React+Springboot+MySQL
52	Online Child Adoption Portal Project	React+Springboot+MySQL
53	online Pizza Delivery System Project	React+Springboot+MySQL
54	Online Social Complaint Portal Project	React+Springboot+MySQL
55	Electric Vehical management system Project	React+Springboot+MySQL
56	Online mess / Tiffin management System Project	React+Springboot+MySQL
57	Online Examination Portal Project	React+Springboot+MySQL
58	Lawyer / Advocate Appointment Booking System	React+Springboot+MySQL
59	Café Management System	React+Springboot+MySQL
60	Agriculture Product Rent system Portal	React+Springboot+MySQL

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/vXz0kZQi5to?si=IiOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/lynLouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyl4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKz

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrk7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizlyJlh
15	Employee management system project	https://youtu.be/ID1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWA OzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vn0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Sr.No	Project Name	YouTube Link
21	Online Bus ticket booking system Project	https://youtu.be/FJ0RUZfMdv8?si=auHjmNgHMrpaNzvY
22	Online Mess / Tiffin Booking System Project	https://youtu.be/NTVmHFDowyl?si=yrvClbE6fdJ0B7dQ
23		
24		
25		

TAP ON THE ICONS TO JOIN!



**STUDY
MATERIAL**

SERVLET

NOTES BY ASHOK PATE

Enroll

Now!

WWW.CODEWITHARRAYS.IN

Servlet Tutorial

Introduction

Servlet is a server side Java based programming language widely used to create dynamic web applications. Servlet runs inside a servlet container. It is a portable language so, it will not rely only on one type of web servers. Servlets can be seen as an improved variant of Common Gateway Interface (CGI).

What is servlet container?

Servlet container is a part of web server used to handle various server side technologies request. Servlet container is used to manage the lifecycle and provide runtime environment.

Common Gateway Interface (CGI)

CGI is an initial server side scripting language used to generate webpages dynamically. Unlike servlets, it is not based on Java. CGI programs can be written in many other programming languages like C, C++ and Perl etc. It has some drawbacks which are overcome by Java servlets.

Advantages of servlet over CGI

Servlet	CGI
Servlets are platform independent.	CGI is platform dependent.
In servlets, each request is handled by a thread which provides better performance.	In CGI, each request is handled by a process that degrades the performance.
As servlets are based on Java, they provide more security.	CGI provides less security as its programs are written in programming languages like C, C++ etc.
Servlets can easily handle cookies for session tracking.	CGI is not capable of handling cookie.

Features

Servlet provides various features. Some of them are listed below.

- **Server side:** Servlet is a server side technology. So, it is capable to handle server requests and responses.
- **Dynamic:** Servlet creates dynamic webpages. So the content of these pages can be updated dynamically with time and requirements.
- **Portable:** Java is a platform independent language. So being based on Java, servlets created on one operating system having any web server can easily run on another operating system having any other web server.
- **Secured and Robust:** Since Servlet is based on Java programming language so, it are secure and robust like Java.
- **Better performance:** Servlet uses thread instead of a process. So, we can perform multiple tasks more efficiently.

Life Cycle of Servlet

Servlet container is responsible to manage the lifecycle of a servlet. Servlet provides three methods as a part of its lifecycle. Let's study the lifecycle of servlets with the following steps.

- **Loading class:** This is an initial stage of servlet in which a servlet class is loaded whenever a request is made.
- **Creating instance:** As soon as the class is loaded, servlet container created the instance of that class.
- **init():** In this step, the servlet container invokes an init() method to initialize the servlet instance. An object of ServletConfig interface is passed within this method. Note that **init() method** is invoked **once in a lifetime of a servlet**.
- **service():** After initialization, servlet container invokes service() method for every request. Each request is handled by a separate thread. This method is used to perform various operations.

- **destroy()**: This is the final stage of the servlet. In this phase, servlet container invokes **destroy()** method for closing the servlet. As soon as **destroy()** method is invoked the memory allocated to servlet and its object is collected by the automatic garbage collector. Note that **destroy() method** is invoked **once in a lifetime** of a servlet.

Servlet Packages

There are two packages in Java Servlet that provide various features to servlet. These two packages are javax.servlet and javax.servlet.http.

1. **javax.servlet package**: This package contains various servlet interfaces and classes which are capable of handling any type of protocol.
2. **javax.servlet.http package**: This package contains various interfaces and classes which are capable of handling a specific http type of protocol.

Overview of some important interfaces and classes

1. javax.servlet package interface

Some of the important interfaces are listed below.

Interface	Overview
Servlet	This interface is used to create a servlet class. Each servlet class must require to implement this interface either directly or indirectly.
ServletRequest	The object of this interface is used to retrieve the information from the user.
ServletResponse	The object of this interface is used to provide response to the user.
ServletConfig	ServletConfig object is used to provide the information to the servlet class explicitly.
ServletContext	The object of ServletContext is used to provide the information to the web application explicitly.

2. javax.servlet package classes

Some of the important classes are listed below.

Classes	Overview
GenericServlet	This is used to create servlet class. Internally, it implements the Servlet interface.
ServletInputStream	This class is used to read the binary data from user requests.
ServletOutputStream	This class is used to send binary data to the user side.
ServletException	This class is used to handle the exceptions occur in servlets.
ServletContextEvent	If any changes are made in servlet context of web application, this class notifies.

1. javax.servlet.http package interface

Some of the important interface of this package are listed below:

Interface	Overview
HttpServletRequest	The object of this interface is used to get the information from the user under http protocol.
HttpServletResponse	The object of this interface is used to provide the response of the request under http protocol.
HttpSession	This interface is used to track the information of users.
HttpSessionAttributeListener	This interface notifies if any change occurs in HttpSession attribute.
HttpSessionListener	This interface notifies if any changes occur in HttpSession lifecycle.

2. javax.servlet.http package classes

Some of the important interface of this package are listed below.

Class	Overview
HttpServlet	This class is used to create servlet class.
Cookie	This class is used to maintain the session of the state.
HttpSessionEvent	This class notifies if any changes occur in the session of web application.
HttpSessionBindingEvent	This class notifies when any attribute is bound, unbound or replaced in a session.

web.xml file

Creating a servlet class is not enough. We have to deploy that class also. So, to deploy the class we use web.xml file. In this file we have to map our class configurations.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>Filename</display-name>
  <servlet>
    <servlet-name>ServletName</servlet-name>
    <servlet-class>ClassName</servlet-class>
  </servlet>
  <servlet-mapping>
```

```
<servlet-name>ServletName</servlet-name>
<url-pattern>/PathName</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

We just required to edit and add some tag in our web.xml file through a text editor. Each tag contains some unique specification.

<servlet-name>: We can provide any specific name that represents a servlet class. The name can be same as of class name.

<servlet-class>: A name of servlet class is given with this tag.

<welcome-file-list>: If we want to attach other files such as html, htm, jpg with our servlet class then we have to deploy that file in this tag. Here each file is mapped separately in <welcome-file> tag.

Servlet Interface

Java provides a servlet interface in javax.servlet package that is implemented by servlet program either directly or indirectly. This interface provides five methods and these methods must be declared within the servlet program if we are implementing a servlet interface. So implementing the servlet interface is a direct approach.

We also discuss about the indirect approach later

Methods of Servlet interface

Methods provided by servlet interface are mentioned below.

Method	Description
public void init(ServletConfig con)	This method is invoked by servlet container once only. The purpose of it is to indicate that the servlet is ready to use.
public void service(ServletRequest req,ServletResponse) throws ServletException	This method is invoked every time whenever a response is given to a request. So this method is used to perform actual tasks.
public ServletConfig getServletConfig()	This method return an object of ServletConfig that contains initialization parameters of the servlet.
public String getServletInfo()	This method is used to provide the information about the servlet.
public void destroy()	This method is invoked by servlet container once only. It is used to indicate that now the server is terminated and no more tasks will execute.

First servlet program using Servlet interface This is the simple example of servlet interface with its methods. **FirstServlet.java**

```
import javax.servlet.*;

public class FirstServlet implements Servlet
{
    public void init(ServletConfig con)
    {
```

```
        System.out.println("init method is invoked once only");
    }

    public void service(ServletRequest req,ServletResponse res) throws
ServletException

    {
        System.out.println("service method is invoked");
    }

    public ServletConfig getServletConfig()
    {
        return null;
    }

    public String getServletInfo()
    {
        return "info";
    }

    public void destroy()
    {
        System.out.println("destroy method is invoked once only");
    }
}
```

Web.xml Now we have to deploy are servlet class in web.xml file. So, open your web.xml file and edit the required tag.

```
<web-app>

<servlet>

<servlet-name>FirstServlet</servlet-name>

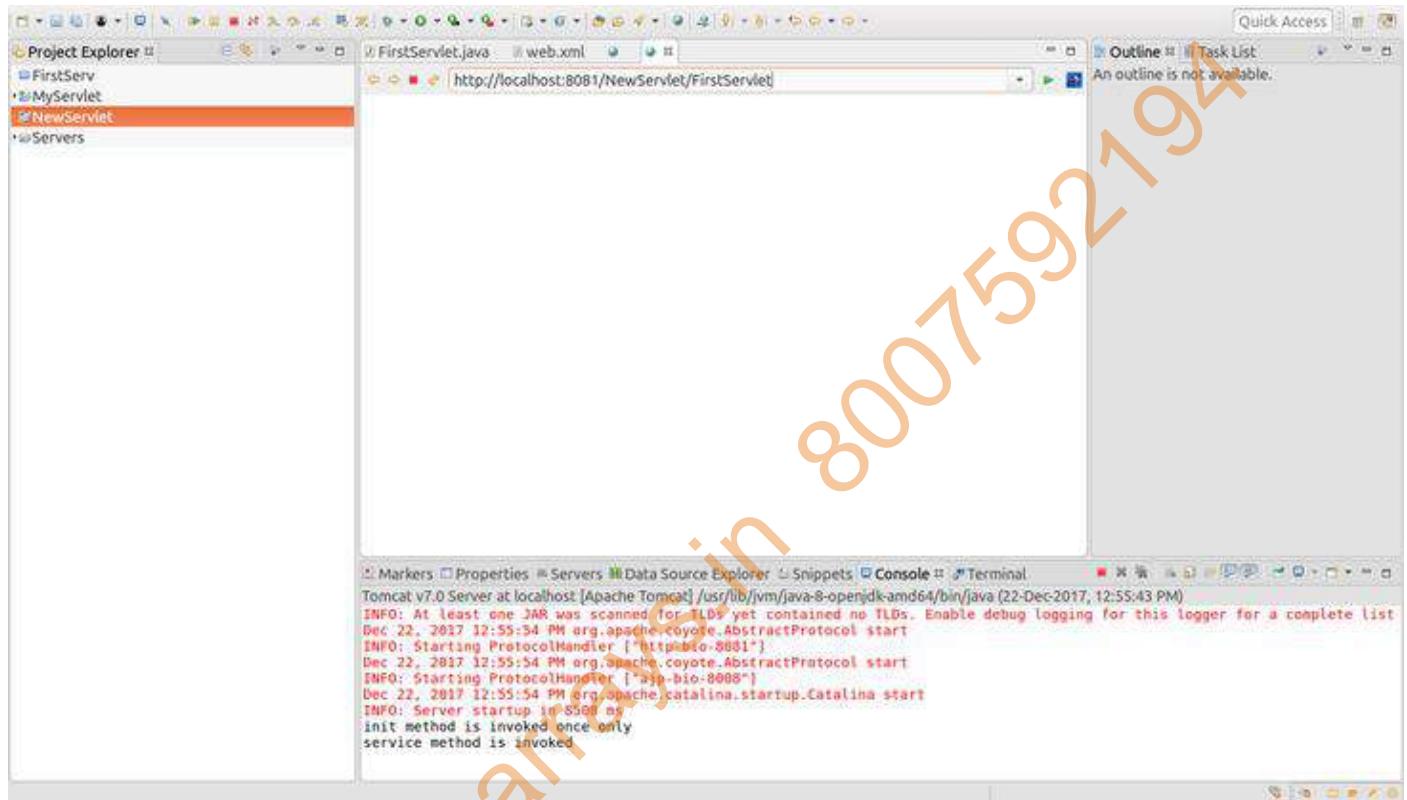
<servlet-class>FirstServlet</servlet-class>

<servlet>
```

```

<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>
</web-app>

```



See here init() method and service() method is invoked but destroy() method is still not invoked. This is because server is still in running mode.

Output:

```

Dec 22, 2017 1:48:26 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 4582 ms
init method is invoked once only
service method is invoked
service method is invoked
service method is invoked
Dec 22, 2017 1:49:34 PM org.apache.catalina.core.StandardServer await
INFO: A valid shutdown command was received via the shutdown port. Stopping the Server instance.
Dec 22, 2017 1:49:34 PM org.apache.coyote.AbstractProtocol pause
INFO: Pausing ProtocolHandler ["http-bio-8081"]
Dec 22, 2017 1:49:34 PM org.apache.coyote.AbstractProtocol pause
INFO: Pausing ProtocolHandler ["ajp-bio-8008"]
Dec 22, 2017 1:49:34 PM org.apache.catalina.core.StandardService stopInternal
INFO: Stopping service Catalina
destroy method is invoked once only
Dec 22, 2017 1:49:34 PM org.apache.coyote.AbstractProtocol stop

```

Everytime when we refresh our page service() method is invoked. As soon as we stop the server destroy() method is also invoked which indicates that now no more task will execute.

Servlet Request and Response

Servlet handles various client requests and provides their responses. It provides two interfaces i.e ServletRequest and ServletResponse for this purpose. Let's discuss about these interfaces in detail.

1. ServletRequest interface

ServletRequest is an interface whose object is used to provide the information of each request to servlet. This information may be any name, type, value or other attribute. This interface is present in javax.servlet package. Servlet container is responsible to create ServletRequest object which is given with service() method argument.

Methods of ServletRequest

These are some important methods provided by ServletRequest interface.

Method	Description
<code>String getParameter(String name)</code>	This method returns the value given in request as a String.
<code>Object getAttribute(String name)</code>	This method provides the attribute of request as an Object.
<code>String getServerName()</code>	This method provides the server name to which request is sent.
<code>int getServerPort()</code>	This method returns the port number to which request is sent.
<code>boolean isSecure()</code>	This method indicates whether the server is secure or not.

2. ServletResponse interface

The object of ServletResponse interface is used to send the responses to the clients. The information send in responses can be a binary or character data. ServletResponse interface is present in javax.servlet package and passes as an argument of service() method.

Methods of ServletResponse

Method	Description
PrintWriter <code>getWriter()</code>	This method is used to send character data in response.
int <code>getBufferSize()</code>	This method returns the capacity of buffer in sent response.
ServletOutputStream <code>getOutputStream()</code>	This method is used to send binary data in response.
boolean <code>isCommitted()</code>	This method indicates the completion of response.
void <code>reset()</code>	This method is used to remove the data present in buffer.
void <code>setContentType(String type)</code>	This method is used to set the type of content.

Servlet Request and Response example with HTML

In this example, the name is given as a request in HTML form. The object of HttpServletRequest interface is used to handle the request and HttpServletResponse interface is used to provide the response.

index.html

```
<form action="serv" method="get">
<h3>Enter your Name:</h3>
<input type="text" name="username">
```

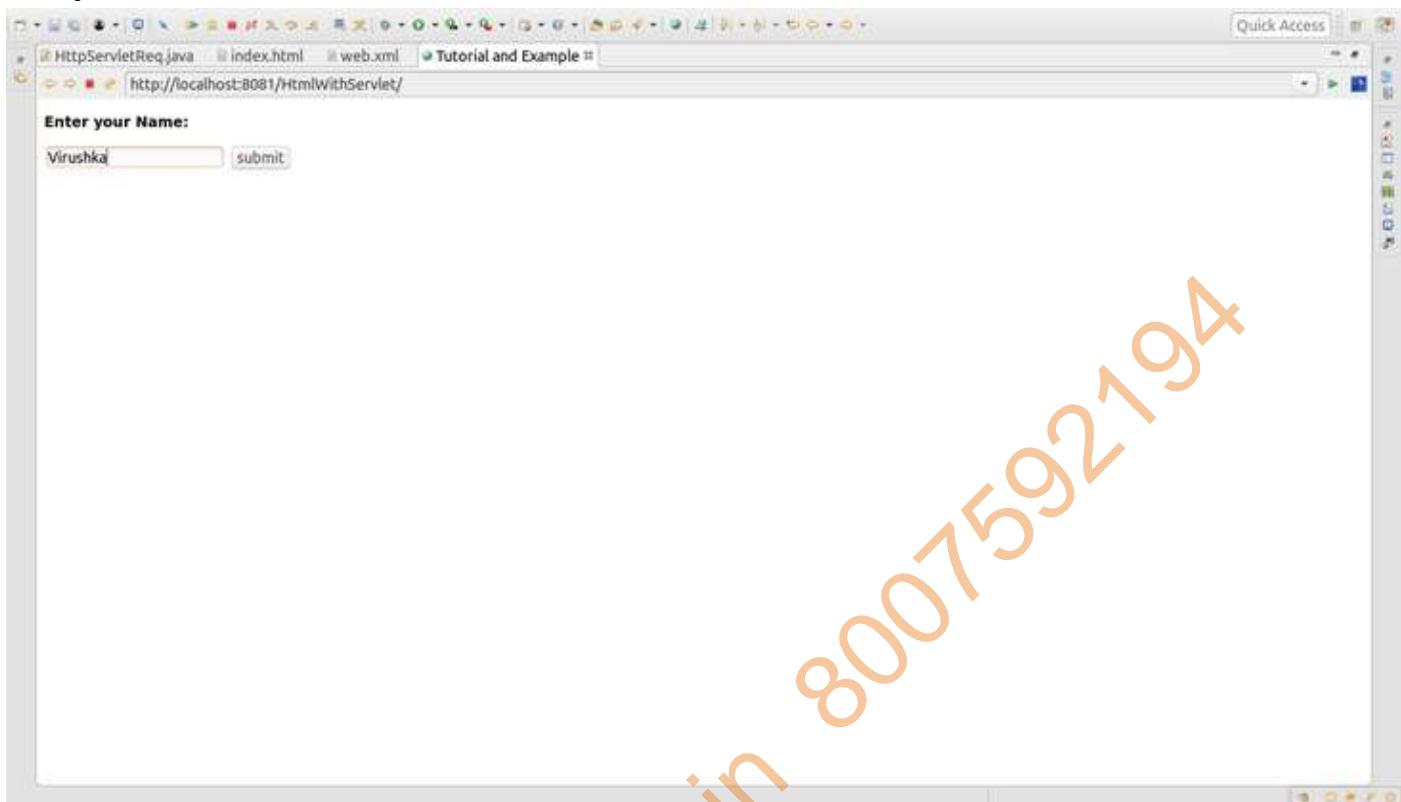
```
<input type="submit" value="submit">  
</form>
```

HttpServletReq.java

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
import javax.servlet.annotation.WebServlet;  
  
@WebServlet("/serv")  
  
public class HttpServletReq extends HttpServlet  
{  
  
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws  
    ServletException, IOException  
    {  
        res.setContentType("text/html");  
        PrintWriter pw = res.getWriter();  
        String s1 = req.getParameter("username");  
        pw.println("<h1>Welcome:" + s1 + "</h1>");  
        String s2 = req.getServerName();  
        pw.println("<h1>Server name:" + s2 + "</h1>");  
        int i = req.getServerPort();  
        pw.println("<h1>Server Port:" + i + "</h1>");  
        boolean b = req.isSecure();  
        pw.println("<h1>Is the server secure:" + b + "</h1>");  
    }  
}
```

```
}
```

Output:



ServletConfig

Servlet container uses the object of ServletConfig interface to provide the information to servlet during initialization. **ServletConfig object fetch this information from web.xml file**. So if we have to provide some dynamic information to servlet then we can pass it from web.xml file without making any change in our servlet class.

From servlet 3.0 we can also declare this information in servlet class itself. This can be done via annotations. Now we just required to pass the name with its value within **@WebInitParam annotation**. This annotation is present in javax.servlet.annotation.WebInitParam package.

Methods of ServletConfig

Method	Description
<code>String getInitParameter(String name)</code>	This method returns the value of specific parameter name.
<code>Enumeration getInitParameterNames()</code>	This method provides the enumeration of names.
<code>ServletContext getServletContext()</code>	This method returns the object of ServletContext.
<code>String getServletName()</code>	This method returns the name of servlet instance.

Example of ServletConfig interface

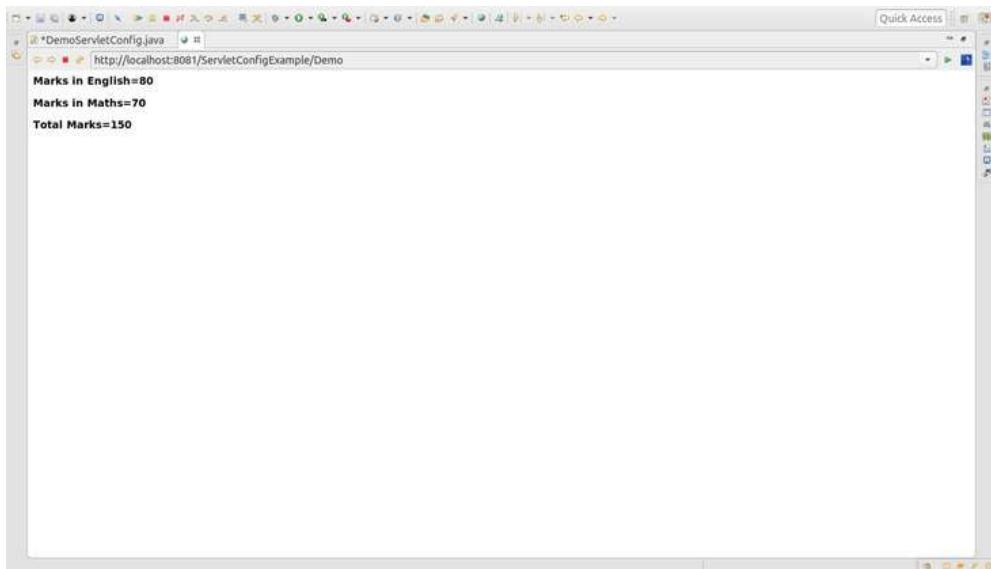
In this example, we are using annotations that contains subject as name and marks as it's specific value.

DemoServletConfig.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.annotation.WebInitParam;
@WebServlet(
urlPatterns = {" /Demo"}, initParams = {
@WebInitParam(name= "English", value="80"),
@WebInitParam(name= "Maths", value= "70"),
})
public class DemoServletConfig extends HttpServlet
```

```
{  
public void doGet(HttpServletRequest req,HttpServletResponse res) throws  
ServletException,IOException  
{  
    res.setContentType("text/html");  
    PrintWriter pw=res.getWriter();  
    ServletConfig con=getServletConfig();  
    String nm1=con.getInitParameter("English");  
    String nm2=con.getInitParameter("Maths");  
    Integer int1=new Integer(nm1);  
    Integer int2=new Integer(nm2);  
    int sum=int1+int2;  
    pw.println("<html><body>");  
    pw.println("<h3>Marks in English="+int1+"</h3>");  
    pw.println("<h3>Marks in Maths="+int2+"</h3>");  
    pw.println("<h3>Total Marks="+sum+"</h3>");  
    pw.println("</body></html>");  
}  
}
```

Output:



ServletContext

ServletContext interface is used to provide the configuration information per web application explicitly. Thus unlike ServletConfig, ServletContext interface can be used to provide information to more than one servlet of web application from web.xml file.

So if we want to pass the same information to all servlets of a web application then passing it from web.xml file at once is much efficient way than providing the same information in each servlet separately.

For sharing information from web.xml we need to pass the name in <param-name> and value in <param-value> tags. <context-param> contains these tags.

Methods of ServletContext interface

Methods	Description
Object getAttribute(String name)	This method returns the attribute of specific name.
Enumeration getAttribute()	This method returns the enumeration of attributes names.
String getInitParameter(String name)	This method returns the parameter of name in the form of string.

<code>String getServletInfo()</code>	This method returns the information about servlet container such as its name and version.
<code>void removeAttribute(String name)</code>	We can remove the attributes of name using this method

Example of ServletContext interface

In this example, two java servlet classes are taken. Both classes contains different information of students of same college. So instead of providing college name every time we are sharing this information from web.xml file.

DemoServletContext.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServletContext extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res) throws
    ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        ServletContext con=getServletContext();
        String st=con.getInitParameter("College");
        pw.println("<html><body>");
        pw.println("<h1>Name:Anuj</h1>");
        pw.println("<h1>Roll number:101</h1>");
        pw.println("<h1>College:"+st+"</h1>");
        pw.println("</body></html>");
    }
}
```

```
 }  
}
```

DemoServletContext1.java

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
  
public class DemoServletContext1 extends HttpServlet  
{  
  
    public void doGet(HttpServletRequest req,HttpServletResponse res) throws  
    ServletException,IOException  
{  
  
        res.setContentType("text/html");  
  
        PrintWriter pw=res.getWriter();  
  
        ServletContext con=getServletContext();  
  
        String st=con.getInitParameter("College");  
  
        pw.println("<html><body>");  
        pw.println("<h1>Name:Abhishek</h1>");  
        pw.println("<h1>Roll number:102</h1>");  
        pw.println("<h1>College:"+st+"</h1>");  
        pw.println("</body></html>");  
    }  
}
```

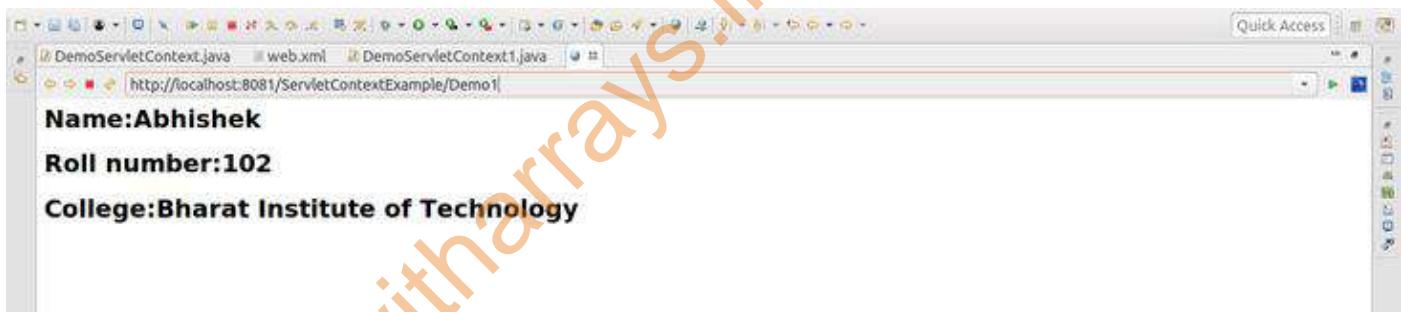
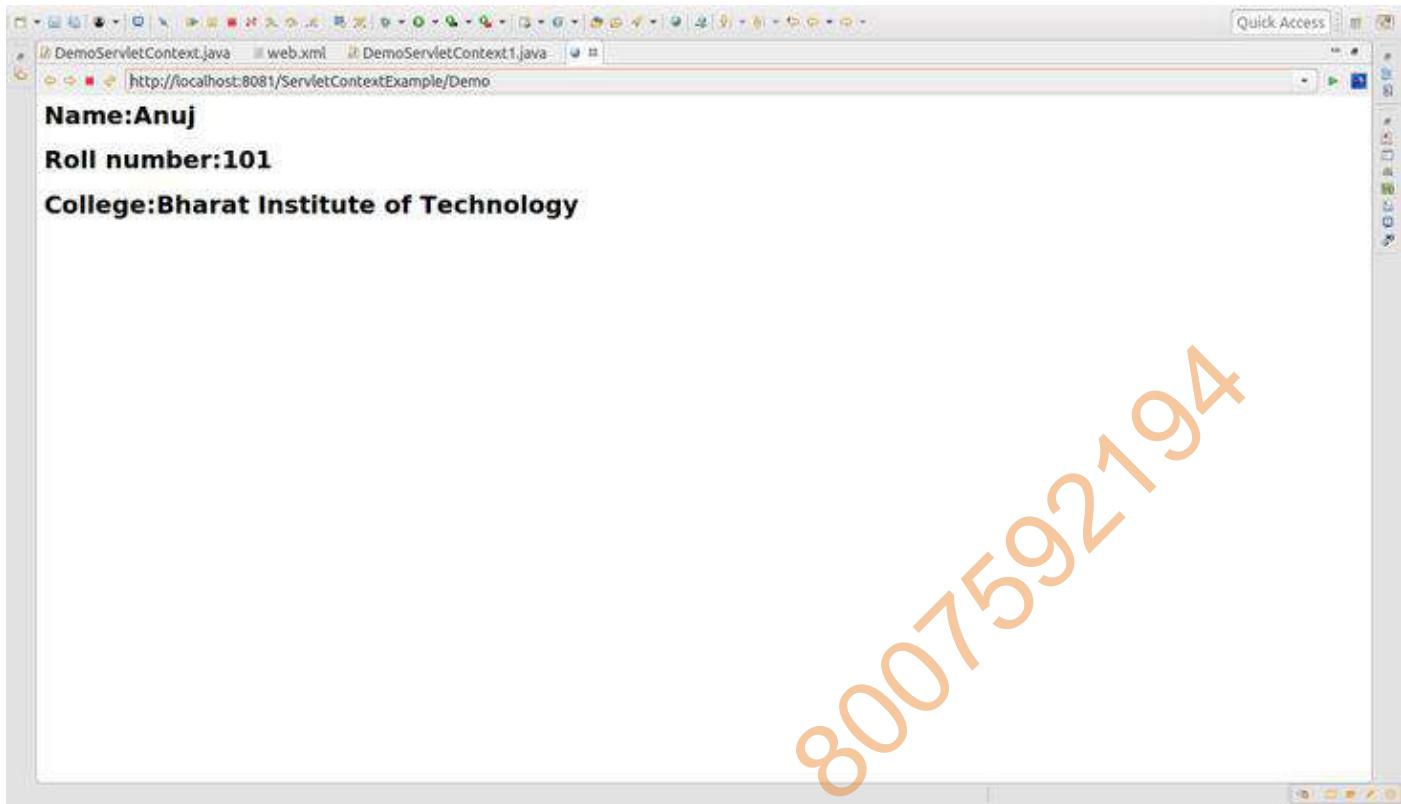
Web.xml

```
<servlet>  
    <servlet-name>DemoServletContext</servlet-name>  
    <servlet-class>DemoServletContext</servlet-class>  
</servlet>
```

```
<servlet>
<servlet-name>DemoServletContext1</servlet-name>
<servlet-class>DemoServletContext1</servlet-class>
</servlet>
<context-param>
<param-name>College</param-name>
<param-value>Bharat Institute of Technology</param-value>
</context-param>
<servlet-mapping>
<servlet-name>DemoServletContext1</servlet-name>
<url-pattern>/Demo1</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>DemoServletContext</servlet-name>
<url-pattern>/Demo</url-pattern>
</servlet-mapping>
```

Code with arrays in 8001592194

Output:



RequestDispatcher

RequestDispatcher is an interface used to receive requests from the users and bind it with other files such as HTML file, Servlet file, JSP file etc. Servlet container is responsible to create RequestDispatcher object.

RequestDispatcher provides **forward()** and **include()** methods. These methods are used to call RequestDispatcher.

Methods of RequestDispatcher

Method	Description
void forward(ServletRequest req, ServletResponse res)	If this method is invoked then the request of current file is send forward to the another file of any type such as HTML, Servlet, JSP etc. and the response of that file is provided by the server.
void include(ServletRequest req, ServletResponse res)	If this method is invoked then the content of current file such as HTML, Servlet, JSP is included with the response

Example of RequestDispatcher interface

In this example we create a form in html file. ServletChecker.java file checks the password field of that form. If the entered password length is less than 8 then include() method is invoked and generates error else forward() method is invoked and forward that request to another java file.

index.html

```
<form action="serv" method="post">
<table>
<h1>
<tr><td><h3>Name:</h3><td><input type="text"
name="name"></td></tr> <br>
<tr><td><h3>Password:</h3><td><input type="password" name="pass"
placeholder="Must be of 8 characters"></td></tr> <br>
<tr><td><h1><input type="submit" value="sign up"></h1></td></tr>
</h1>
</table>
</form>
```

ServletChecker.java

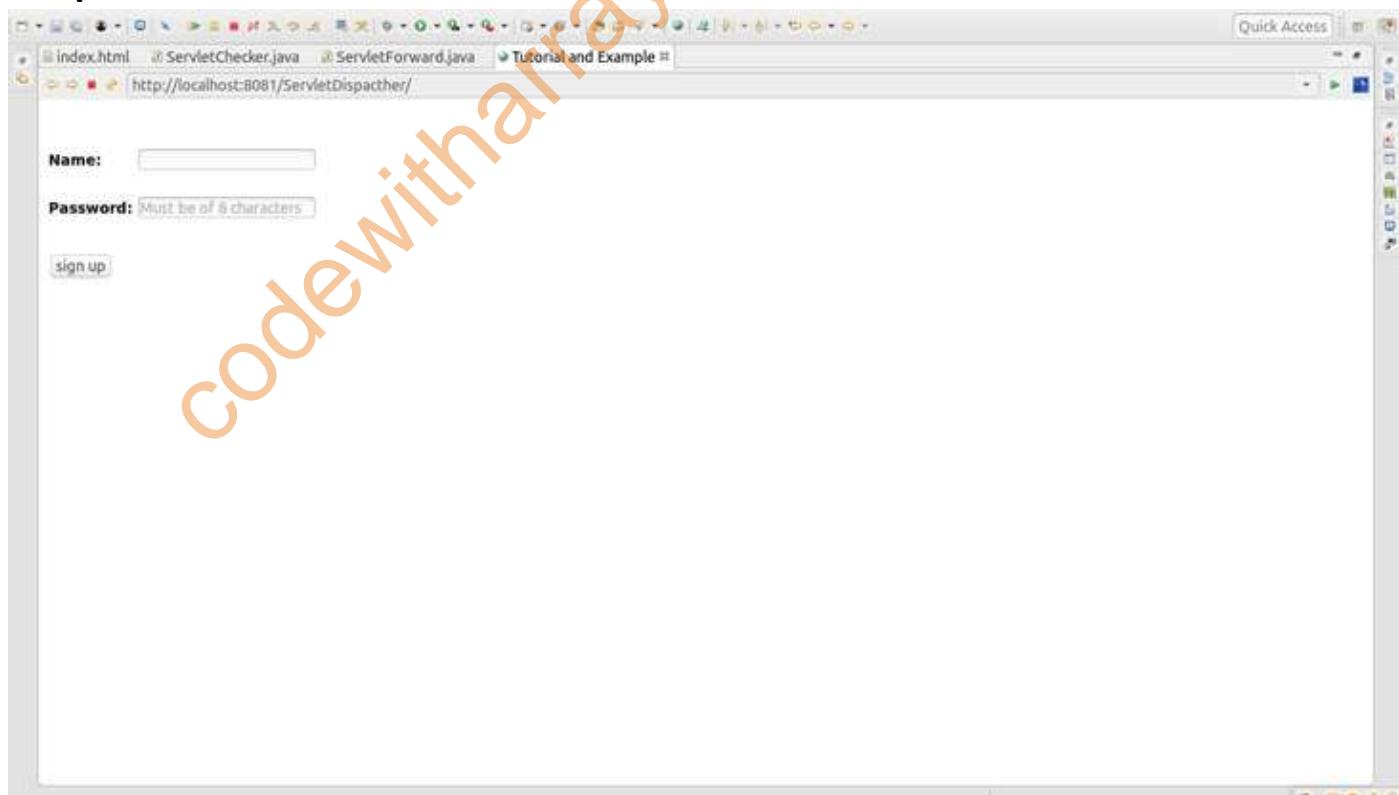
```
import java.io.*;
import javax.servlet.*;
```

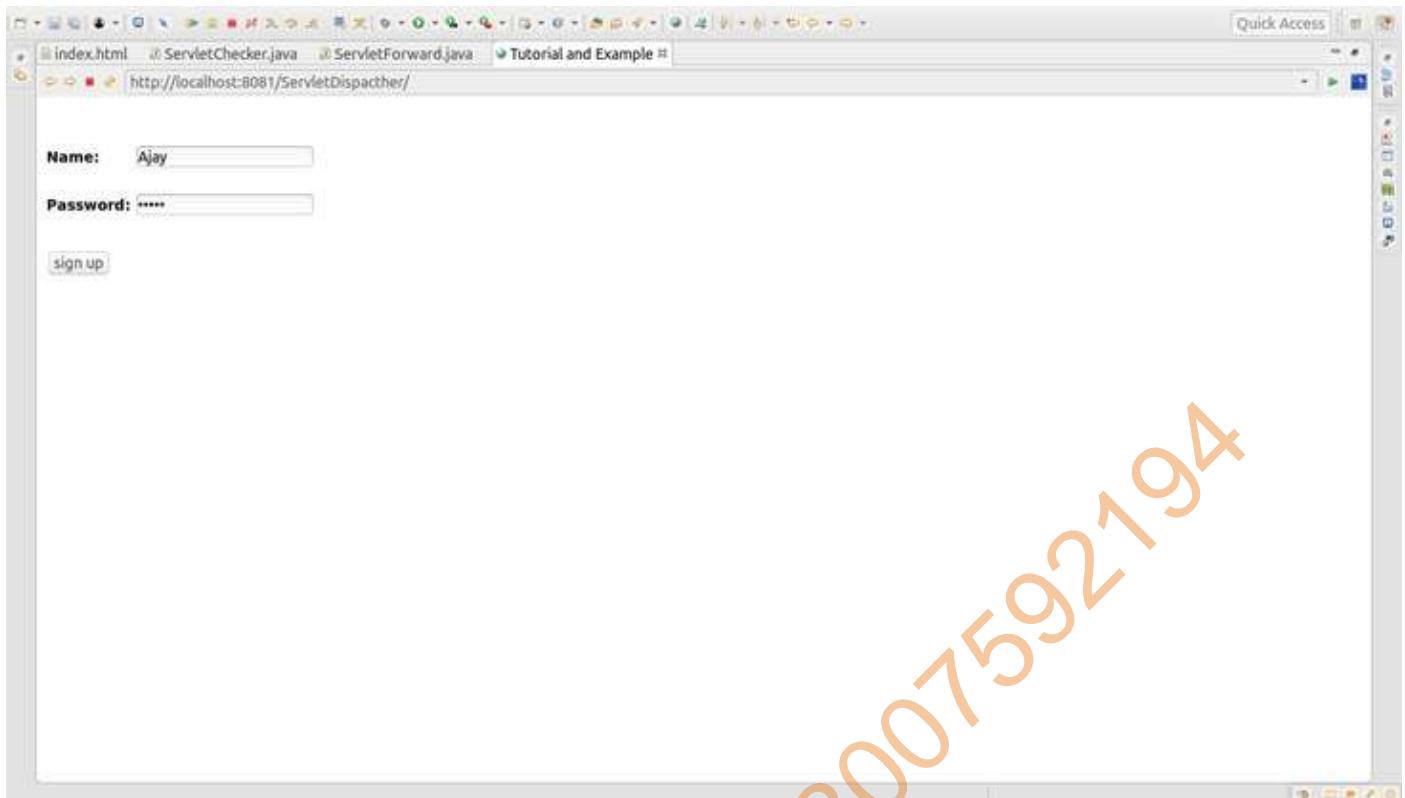
```
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/serv")
public class ServletChecker extends HttpServlet
{
    public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        String st1=req.getParameter("pass");
        int i1=st1.length();
        if(i1<8)
        {
            pw.println("<h1>Error:Password must be of 8 character</h1>");
            RequestDispatcher rd=req.getRequestDispatcher("/index.html");
            rd.include(req,res);
        }
        else
        {
            RequestDispatcher rd1=req.getRequestDispatcher("ServletForward");
            rd1.forward(req,res);
        }
    }
}
```

ServletForward.java

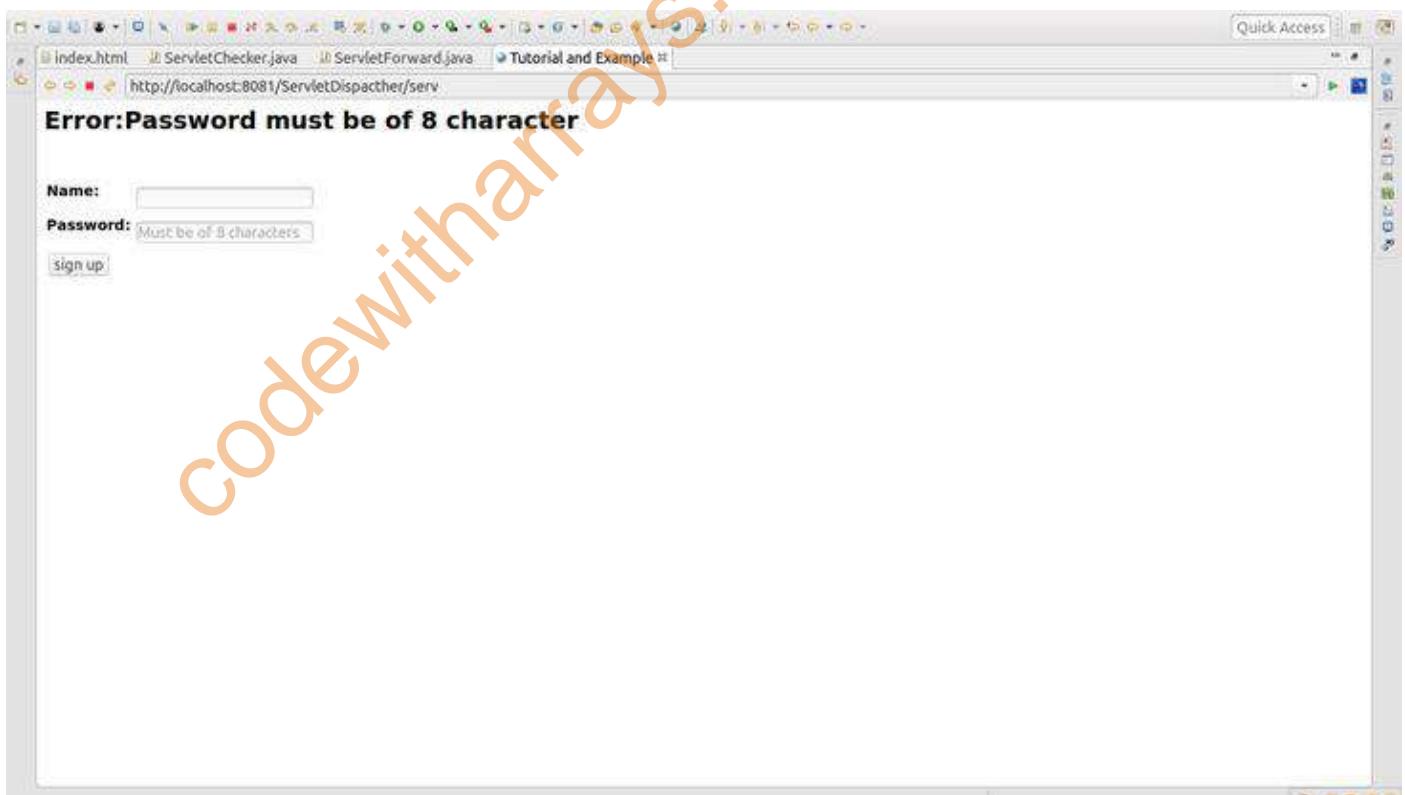
```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/ServletForward")
public class ServletForward extends HttpServlet
{
    public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        String st1=req.getParameter("name");
        pw.println("<h1>Welcome "+st1+"</h2>");
        pw.println("<h3>You registered successfully</h3>");
    }
}
```

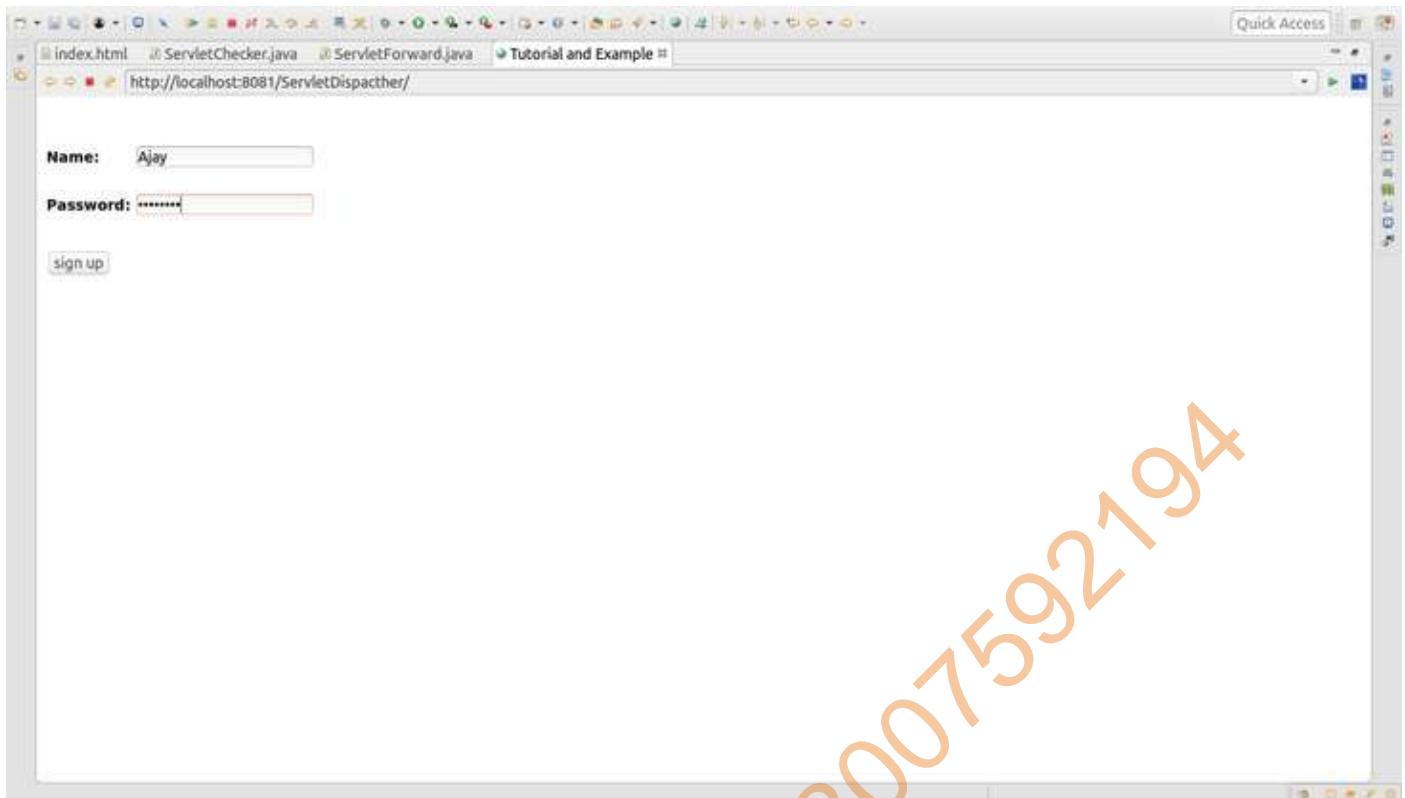
Output:



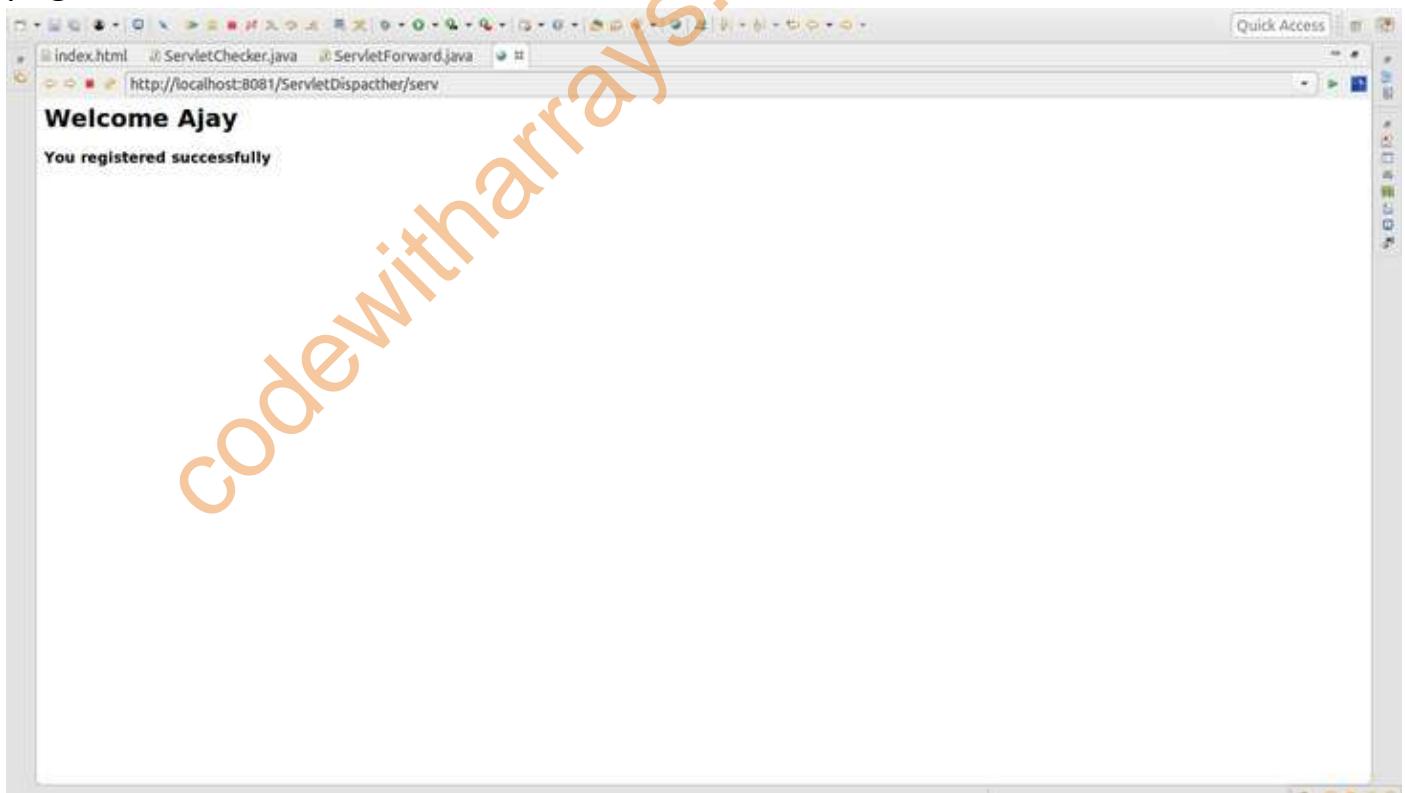
The inserted length of the password is less than 8 character. So it will generate error message and include the current form within it.



As expected error message generated.



Now inserted length of the form is greater than 8 character. So forward() method is invoked and sent as a request to another page.



The following response generated.

Servlet Session Management

What is session?

A session is a link between a client and the server. This link represents some identity that makes client unique so that it is known by the server whenever a request is made.

Need to maintain this session?

Whenever a client made a request, the server treats it as a new request. In some applications we require that a series of requests made by a client will associate with one another. So we need to maintain the state of a user.

How to maintain a session?

To maintain a session, servlet container uses various methods to make a particular client unique so that server can easily identify it.

Different ways to maintain session

There are several ways in servlet to maintain session:

- HttpSession
- Cookies
- Hidden Form Field
- URL Rewriting

1. Servlet HttpSession

- HttpSession is an interface used by servlet container to create a session between Http client and Http server. So using this interface we can maintain the state of a user.
- The object of HttpSession stores various information about the user. This interface is present in javax.servlet.http package.
- **Methods of HttpSession**
- These are some important methods provided by HttpSession interface.

Methods	Description
String getId()	This method returns a unique id of client used to identify it.

long getLastAccessedTime()	This method provides the time when last request is made. It returns the time in milliseconds.
Long getCreationTime()	This method provides the time when first request is made. It returns the time in milliseconds.
void setMaxInactiveInterval()	This method sets time in seconds after which the session becomes invalid.
Object getAttribute(String name)	This method provides the value of attribute bind with the name passed within it.

- **Example of HttpSession**

- In this example, we will maintain the state of a particular client. When a client makes first request then its unique id is generated. With each request, number of visit of a user is incremented by one unless request will be made within a specific period of time given in setMaxInactiveInterval() method unless the session becomes expire.

- **HttpSessionExample.java**

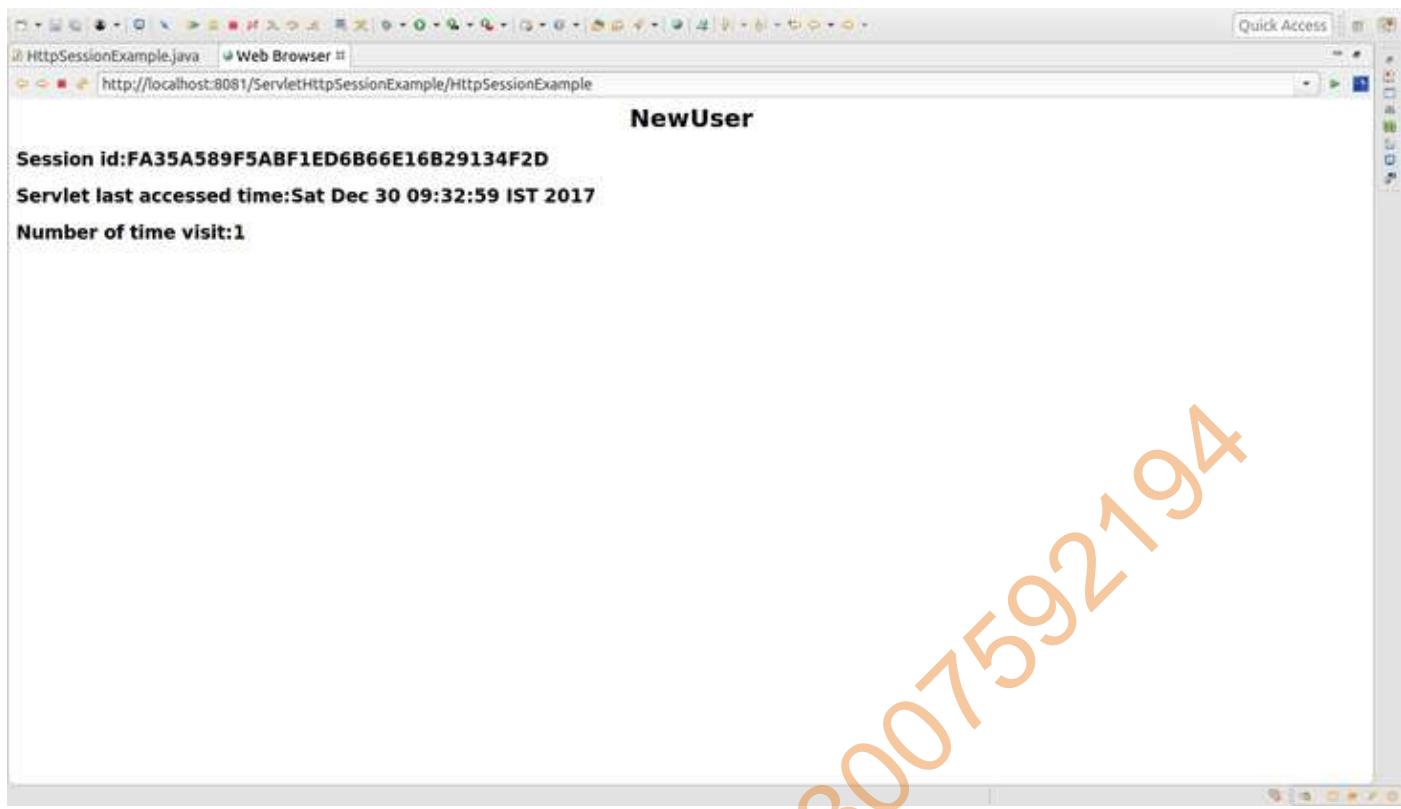
```

• import java.io.*;
• import javax.servlet.*;
• import javax.servlet.http.*;
• import java.util.*;
• import javax.servlet.annotation.WebServlet;
• @WebServlet("/HttpSessionExample")
• public class HttpSessionExample extends HttpServlet {
•     public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
•         res.setContentType("text/html");
•         HttpSession session = req.getSession();
•         res.setContentType("text/html");
•         PrintWriter pw = res.getWriter();
•         Integer attribute = (Integer) session.getAttribute("attribute");
•         if (attribute == null)
•             {
•                 attribute = new Integer(1);
•                 pw.println("<h1><center>NewUser</center></h1>");

```

```
• } else {  
•     pw.println("<h1><center>Welcome back</center></h1>");  
•     attribute = new Integer(attribute.intValue() + 1);  
• }  
• session.setAttribute("attribute",attribute);  
• session.setMaxInactiveInterval(60);  
• pw.println("<h2>Session id:"+session.getId()+"</h2>");  
• pw.println("<h2>Servlet last accessed time:"+new  
Date(session.getLastAccessedTime())+"</h2>");  
• pw.println("<h2>Number of time visit:"+attribute+"</h2>");  
• }{
```

Codewitharrays.in 8007592194



2. Servlet Cookie

Cookie is a class that is used for session management. It contains a small amount of information which persists between the server and client. This information includes name, value and other attributes.

Servlet send this information to web browser and browser saved it so that whenever browser makes a request then it also sends this information along with it. Thus through cookies a server can easily identify a client.

Methods of Cookie

Method	Description
String getName()	This method returns the name of cookie.
String getValue()	This method returns the current value of cookie.
void setValue(String name)	We can set the value of cookie using this method.
void setMaxAge(int i)	This method is used to set the maximum age of cookie after which the session becomes expire.
int getMaxAge()	This method returns the maximum age of a cookie.

3. Servlet URL Rewriting

This is another approach to maintain the session of a user. In this mechanism, when a client make a request then some extra information is appended in the URL of that request.

This extra information uniquely identifies a user and it may be a path info, parameter attribute etc. Unlike cookies, URL Rewriting will also work when cookies are disabled in our browser.

Example of URL Rewriting

Let's see an example of URL Rewriting.

Welcome.jsp

```
<html>
```

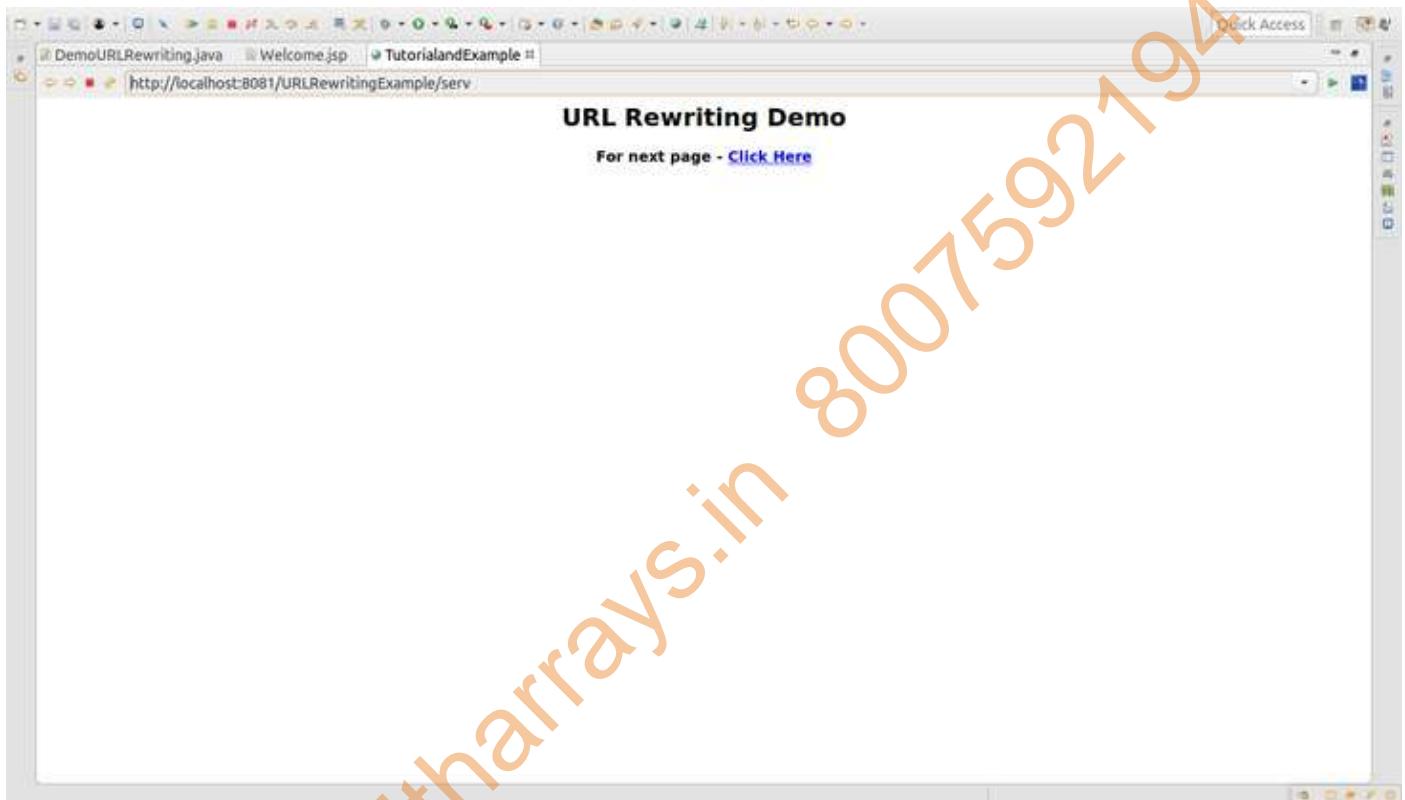
```
<body>
<center>
<h1>Welcome to Tutorial and Example</h1>
</center>
</body>
</html>
```

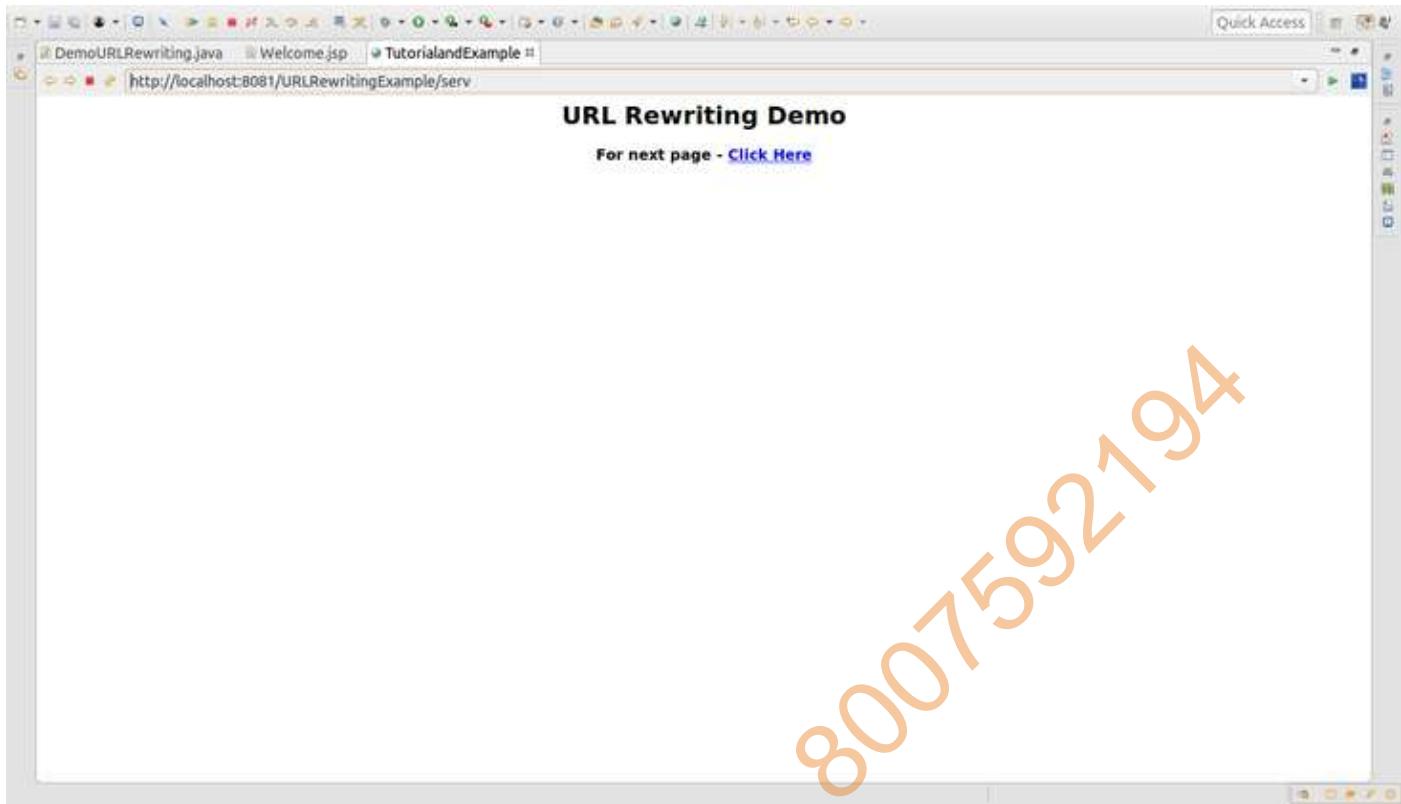
DemoURLRewriting.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/serv")
public class DemoURLRewriting extends HttpServlet {
protected void doGet(HttpServletRequest req,
HttpServletResponse res) throws ServletException, IOException {
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
String path = req.getContextPath();
String addURL = res.encodeURL(path + "/Welcome.jsp");
pw.println("<html>");
pw.println("<head>");
pw.println("<title>TutorialandExample</title>");
pw.println("</head>");
pw.println("<body><center>");
pw.println("<h1>URL Rewriting Demo</h1>");
pw.println("<h3>For next page - <a href=\""+ + addURL
```

```
+ "\"> Click Here</a></h3>");  
pw.println("</center></body>");  
pw.println("</html>");  
}  
}
```

Output:





4.Servlet Hidden Form Field

Hidden form field is an invisible field that saves the information regarding the state in client browser. As this information is invisible so client can't see it but it is visible to servlet.

These fields are added to HTML form and when client submit form then these fields are also sent to the server with form.

Example of Hidden Form Field

index.html

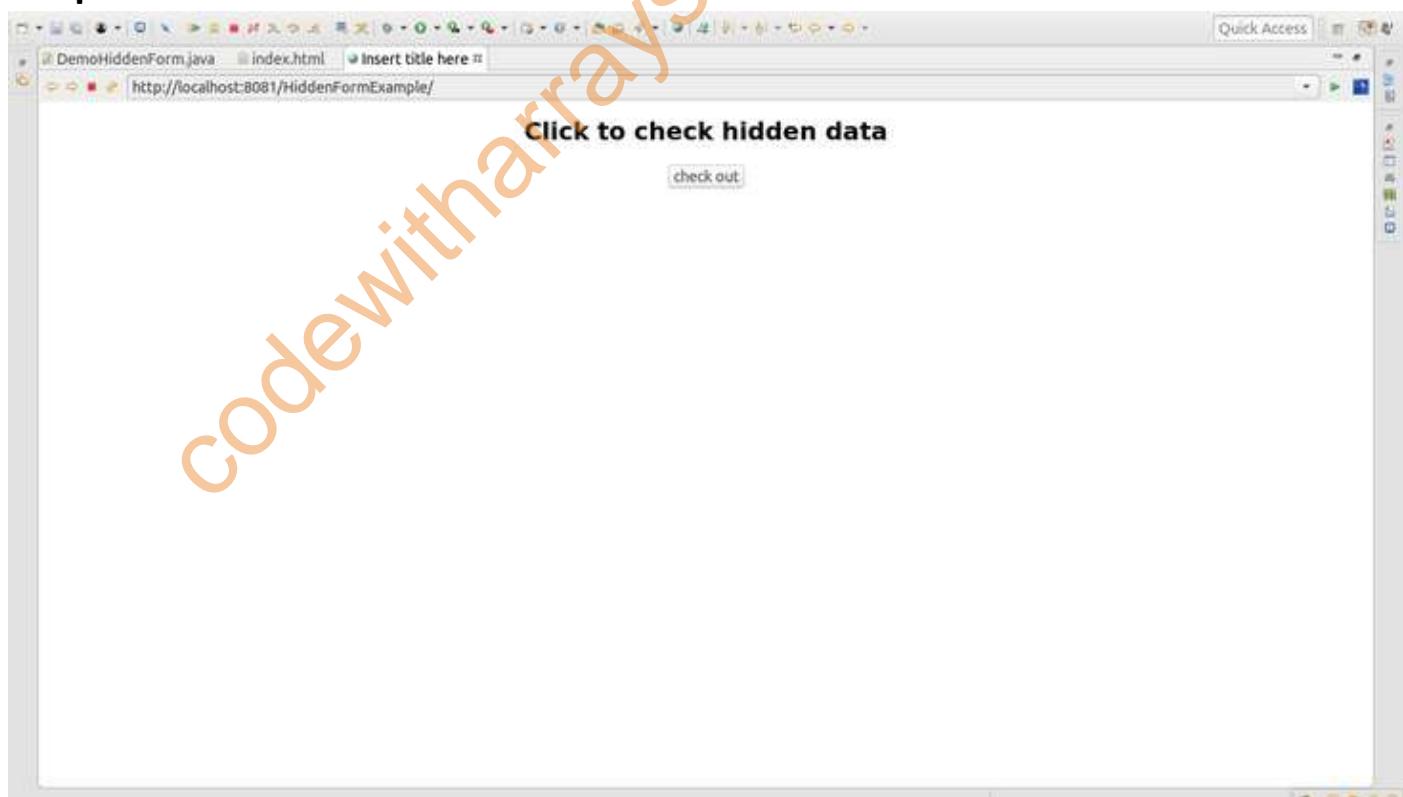
```
<center>
<h1>Click to check hidden data</h1>
<form action="serv" method="post">
<input type="hidden" name="website" value="TutorialandExample">
<input type="submit" value="check out">
</form>
</center>
```

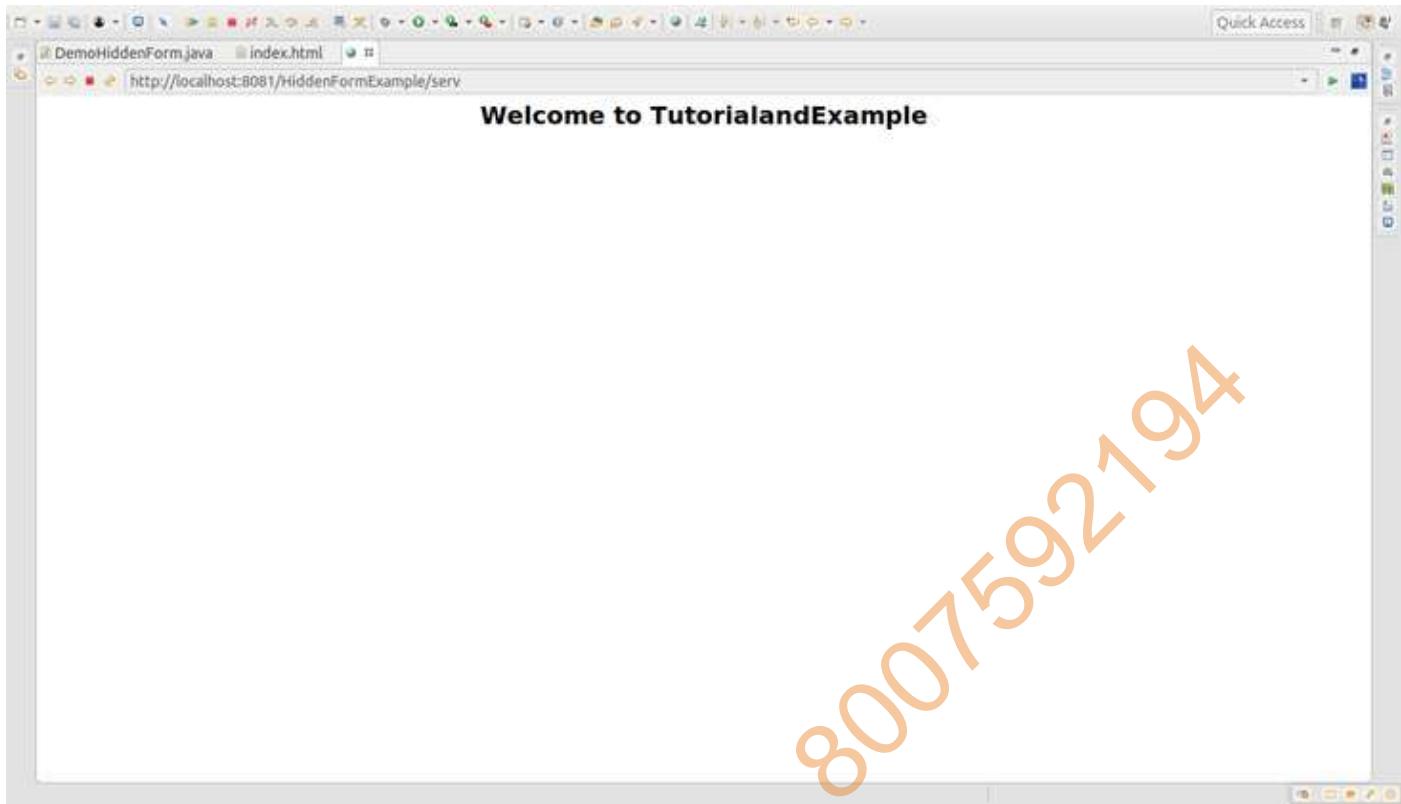
DemoHiddenForm.java

```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/serv")
public class DemoHiddenForm extends HttpServlet {
protected void doPost(HttpServletRequest req,
HttpServletResponse res) throws ServletException, IOException {
String web = req.getParameter("website");
res.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println("<h1><center>Welcome to " + web+"</center></h1>");
}
}
```

Output:





Asynchronous Servlet

In servlets, each request is handled by a thread. Thus the large amount of threads are required for heavy applications. Sometimes a situation arises when a thread waits for a resource or wait for an event. In this case thread may become blocked and degrades the performance of the servlet.

To overcome these situations servlet provides asynchronous support. This ensures that no threads associated with requests become idle.

Let's discuss about asynchronous interfaces and classes.

AsyncContext

AsyncContext is an interface present in javax.servlet package. The object of AsyncContext is used to perform various asynchronous operations and for that ServletRequest.startAsync() method is invoked.

Syntax used to enable asyncSupport:

`@WebServlet(urlPatterns="/pathname", asyncSupported=true)`

Methods of AsyncContext

Method	Description

ServletRequest getRequest()	This method provides the request associated with AsyncContext when startAsync() method is invoked.
ServletResponse getResponse()	This method provides the response associated with AsyncContext when startAsync() method is invoked.
void dispatch()	Through this method we can dispatch the request and response of AsyncContext object to servlet container.
void setTimeout()	We can set a bound of time to AsyncContext by using this method.
long getTimeout()	It provides the time value if declared through setTimeout() method otherwise provide a default value of set by container.
void complete()	The purpose of this method it to complete asynchronous operation.

Explore More

Subscription : Premium CDAC NOTES & MATERIAL



Contact to Join
Premium Group



Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

codewitharrays.in freelance project available to buy contact on 8007592194

SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySQL
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySQL
3	Tour and Travel management System	React+Springboot+MySQL
4	Election commition of India (online Voting System)	React+Springboot+MySQL
5	HomeRental Booking System	React+Springboot+MySQL
6	Event Management System	React+Springboot+MySQL
7	Hotel Management System	React+Springboot+MySQL
8	Agriculture web Project	React+Springboot+MySQL
9	AirLine Reservation System / Flight booking System	React+Springboot+MySQL
10	E-commerce web Project	React+Springboot+MySQL
11	Hospital Management System	React+Springboot+MySQL
12	E-RTO Driving licence portal	React+Springboot+MySQL
13	Transpotation Services portal	React+Springboot+MySQL
14	Courier Services Portal / Courier Management System	React+Springboot+MySQL
15	Online Food Delivery Portal	React+Springboot+MySQL
16	Municipal Corporation Management	React+Springboot+MySQL
17	Gym Management System	React+Springboot+MySQL
18	Bike/Car ental System Portal	React+Springboot+MySQL
19	CharityDonation web project	React+Springboot+MySQL
20	Movie Booking System	React+Springboot+MySQL

freelance_Project available to buy contact on 8007592194

21	Job Portal web project	React+Springboot+MySQL
22	LIC Insurance Portal	React+Springboot+MySQL
23	Employee Management System	React+Springboot+MySQL
24	Payroll Management System	React+Springboot+MySQL
25	RealEstate Property Project	React+Springboot+MySQL
26	Marriage Hall Booking Project	React+Springboot+MySQL
27	Online Student Management portal	React+Springboot+MySQL
28	Resturant management System	React+Springboot+MySQL
29	Solar Management Project	React+Springboot+MySQL
30	OneStepService LinkLabourContractor	React+Springboot+MySQL

31	Vehical Service Center Portal	React+Springboot+MySQL
32	E-wallet Banking Project	React+Springboot+MySQL
33	Blogg Application Project	React+Springboot+MySQL
34	Car Parking booking Project	React+Springboot+MySQL
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySQL
36	Society management Portal	React+Springboot+MySQL
37	E-College Portal	React+Springboot+MySQL
38	FoodWaste Management Donate System	React+Springboot+MySQL
39	Sports Ground Booking	React+Springboot+MySQL
40	BloodBank mangement System	React+Springboot+MySQL
41	Bus Tickit Booking Project	React+Springboot+MySQL
42	Fruite Delivery Project	React+Springboot+MySQL
43	Woodworks Bed Shop	React+Springboot+MySQL
44	Online Dairy Product sell Project	React+Springboot+MySQL
45	Online E-Pharma medicine sell Project	React+Springboot+MySQL
46	FarmerMarketplace Web Project	React+Springboot+MySQL
47	Online Cloth Store Project	React+Springboot+MySQL
48	Train Ticket Booking Project	React+Springboot+MySQL
49	Quizz Application Project	JSP+Springboot+MySQL
50	Hotel Room Booking Project	React+Springboot+MySQL
51	Online Crime Reporting Portal Project	React+Springboot+MySQL
52	Online Child Adoption Portal Project	React+Springboot+MySQL
53	online Pizza Delivery System Project	React+Springboot+MySQL
54	Online Social Complaint Portal Project	React+Springboot+MySQL
55	Electric Vehical management system Project	React+Springboot+MySQL
56	Online mess / Tiffin management System Project	React+Springboot+MySQL
57	Online Examination Portal Project	React+Springboot+MySQL
58	Lawyer / Advocate Appointment Booking System	React+Springboot+MySQL
59	Café Management System	React+Springboot+MySQL
60	Agriculture Product Rent system Portal	React+Springboot+MySQL

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/vXz0kZQi5to?si=IiOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7uiQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/lynLouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyl4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHxE?si=gDiAhKBowi2gNUKz

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrk7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizlyJlh
15	Employee management system project	https://youtu.be/ID1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vn0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Sr.No	Project Name	YouTube Link
21	Online Bus ticket booking system Project	https://youtu.be/FJ0RUZfMdv8?si=auHjmNgHMrpaNzvY
22	Online Mess / Tiffin Booking System Project	https://youtu.be/NTVmHFDowyl?si=yrvClbE6fdJ0B7dQ
23		
24		
25		

TAP ON THE ICONS TO JOIN!



**STUDY
MATERIAL**

JSP

NOTES BY ASHOK PATE

Enroll

Now!

WWW.CODEWITHARRAYS.IN

SR.NO	Title
1	JSP Life Cycle
2	Scripting in JSP
3	JSP Script let Tag
4	JSP Declaration Tag
5	JSP Expression Tag
6	JSP Comments
7	JSP Directives
8	JSP Taglib Directive
9	JSP Control Statements
10	JSP Implicit Objects
11	JSP out
12	JSP Request
13	JSP Response
14	JSP Config
15	JSP Application
16	JSP Session
17	JSP Page Context
18	JSP Page
19	JSP exception
20	JSP Action Tags
21	JSP forward and include tag
22	JSP useBean , getProperty and setProperty tags
23	JSP Cookie Handling
24	JSP Expression Language
25	JSP EL Implicit Objects
26	JSTL – Java Standard tag library
27	JSTL Core Tags
28	JSTL Formatting Tags
29	JSTL SQL Tags
30	JSTL XML Tags
31	JSTL function Tags
32	JSP Custom Tag

JSP Tutorial

Java Server Pages (JSP) is a server-side technology used to create static and dynamic web applications. The static content is expressed by text-based format files such as HTML, XML, SVG whereas JSP elements are used to construct dynamic content.

JSP is a convenient way of writing servlets. It enables you to insert Java code into HTML pages through simple JSP tags. JSP source files are represented by .jsp extension.

Difference between Servlet and JSP

JSP	Servlet
JSP files are represented by .jsp extension.	Servlet files are represented by .java extension.
JSP code can be inserted into HTML page.	Servlet code cannot be inserted into HTML page whereas vice-versa is possible.
JSP contains in-built implicit objects.	In Servlet, the required objects are called explicitly.
JSP is an easy language as it contains less code and deals maximum with tags.	Servlet contains a heavy bug of java code.
While compilation, JSP code is initially converted into Servlet. So, the execution of JSP page takes much more time than Servlet.	The execution of Servlet code is faster than JSP.

JSP Features

Some key features of JSP technology are as follows:

- **Portable** - JSP is a platform independent technology. So, JSP web pages can run on any browser and web container independently.
- **Simple**- It provides an easy way to develop and deploy web applications.
- **Powerful** - JSP is a Java based technology. Thus, it is secured and robust.
- **Tag based approach** - Instead of heavy bug of java code, JSP uses simple pre-defined tags.
- **Customized tag** - JSP also allows users to define their own tag.

JSP Life Cycle

JSP life cycle

The life cycle of JSP page internally implemented as a Servlet. These are the following stages through which a JSP page has to pass:

- **Translation** - This is an initial phase of JSP lifecycle that exist when first request of the JSP page is made. In this phase, web container translates the JSP page into servlet class.
- **Compilation** - As soon as the JSP page is translated, web container compiles the servlet class. Both translation and compilation have been proceeded only when JSP page's servlet is older than JSP page.
- **Loading and Instantiating** - Once the translation and compilation have been done, JSP page servlet follows the Servlet lifecycle. Hence, class is loaded and instance is created.
- **Execution** - In this phase, the actual task is performed. Web container invokes `jspInit()` method to initialize servlet instance. To pass request and response objects, web container invokes `jspService()` method.
- **Destruction** - This is the final phase of lifecycle in which web container invokes `jspDestroy()` method to remove JSP page servlet, if it is no more further required.

Features Some key features of JSP technology are as follows:

- **Portable** - JSP is a platform independent technology. So, JSP web pages can run on any browser and web container independently.
- **Simple** - It provides an easy way to develop and deploy web applications.
- **Powerful** - JSP is a Java based technology. Thus, it is secured and robust.
- **Tag based approach** - Instead of heavy bug of java code, JSP uses simple pre-defined tags.
- **Customized tag** - JSP also allows users to define their own tag.

Scripting in JSP

JSP scripting provides an easy and efficient way to insert Java programming language in JSP pages. Here, Java is a default language.

JSP also allows you to use any other scripting language that is capable to call Java objects. In this case, you have to specify that language in the page directive. The following syntax is used to implement it.

```
<%@ page language = "Scripting-language" %>
```

Disabling Scripting

It is not mandatory to use scripting in JSP page. So, JSP allows you to disable scripting if it is not required. Although, by default scripting is always enable.

Scriptlet elements

In JSP scripting, scriptlet elements perform the actual task. Hence, Java statements are enclosed within these elements.

JSP scripting elements are capable to create and manipulate Java objects, declaring variables and methods, catching Java Exceptions etc. It enables the JSP page to connect with database and fetch queries.

JSP provides three types of scripting elements to embed Java code in JSP page. Each scripting element has its own purpose. The following scripting elements are:

- Scriptlet tag
- Declaration tag
- Expression tag

JSP Scriptlet Tag

JSP scriptlet tag allows you to insert programming logic inside JSP page. Hence, you can put valid Java code within scriptlet tags. Each Java statement must be followed by a semicolon.

Syntax: - <% Java code %>

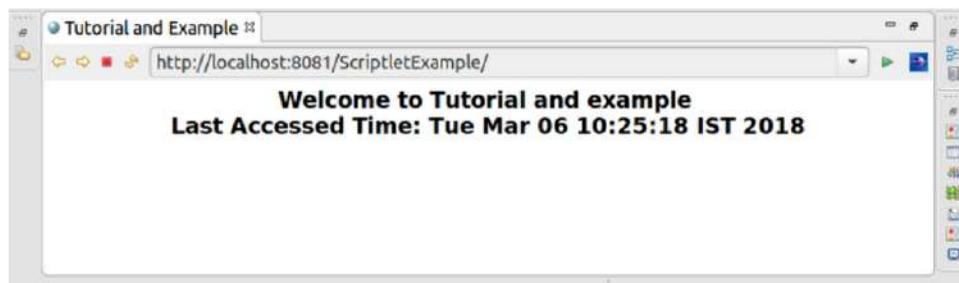
Whatever the code written inside scriptlet tag <% %> is compiled as Java code. This code can be accessible anywhere inside the JSP page. Any number of Java statements and local variables can be inserted in scriptlet tag. Although, scriptlet doesn't allow method declaration within it.

Example of JSP Scriptlet Tag

Here is a simple example of JSP scriptlet tag having a Java code within it.

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<h2><center>Welcome to Tutorial and example
<%
java.util.Date date = new java.util.Date();
out.print("<br>Last Accessed Time: "+date);
%>
</center></h2>
</body>
</html>
```



Output:

Internal Procedure

Internally, JSP container inserts the content of scriptlet tag into the `jspService()` method.

Internal code:

```
public void jspService(HttpServletRequest request, HttpServletResponse response) throws ServletException
{
    PrintWriter out=response.getWriter();
    response.setContentType("text/html");
    java.util.Date date = new java.util.Date();
    out.print("<br>Last Accessed Time: "+date);
}
```

JSP Declaration Tag

JSP declaration tag is used to declare variables and methods in JSP page. Unlike scriptlet tag, JSP container placed the content of declaration tag outside the `jspService()` method. So, variables and methods declared in declaration tag are static or instance.

The syntax of declaration tag is as follows:

```
<%! variable and method declaration %>
```

The variable and method inside declaration tag is up to the class level. We can put any number of variables and methods inside declarative tag.

Example of declaring variables in JSP Declaration Tag

Along with declaring variable, this example also shows that declaration tag doesn't get memory at each request.

index.jsp

```
<html>
<head>
<title>Insert title here</title>
</head>
<body>
<h1><center>
<%!
String name="Tutorial and Example";
int count=0;
%>
<%out.print("Website name:"+name);
out.print("<br>Number of time access:"+ ++count);
%>
</center></h1>
</body>
</html>
```

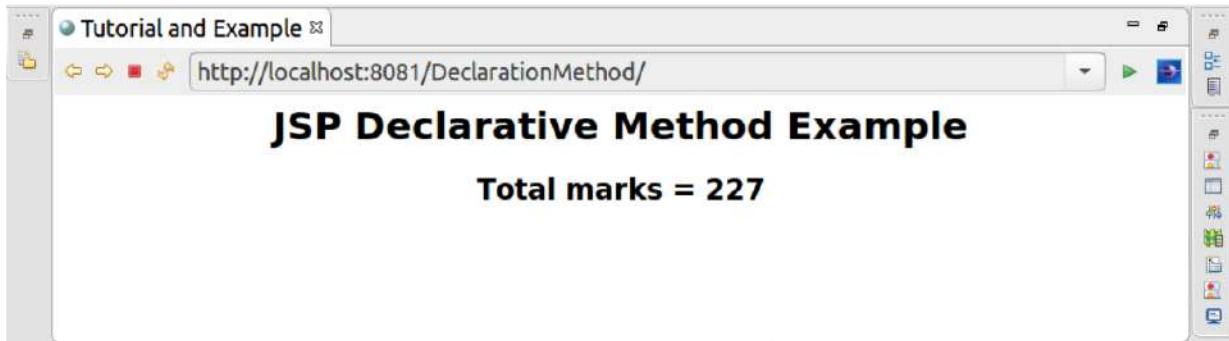
**Output:**

Example of declaring methods in JSP Declaration Tag

In this example, a method is declared inside declaration tag and has been accessing from scriptlet tag.

Index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<center>
<h1>JSP Declarative Method Example </h1>
<h2>
<%!
int marks(int english,int math,int science)
{
    return english+math+science;
}
%>
<%out.print("Total marks = "+marks(72,75,80));%>
</h2>
</center>
</body>
</html>
```



JSP Expression Tag

JSP expression tag allows you to place the result of Java expression in a convenient way. It can be seen as an alternative of `out.print()` method.

Thus, if we want to print any statement or result directly then we can use the below syntax:

```
<%=result%>
```

Note: Java statement in expression tag doesn't require a semicolon to terminate.

Example of JSP Expression Tag

In this example, we are just printing a string without using `out.print()` method.

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<h1><center>
<%="Welcome to JSP tutorial" %>
</center></h1>
</body>
</html>
```



Output:

Comments JSP provides separate tag for comment section. The content inside these tag is completely ignored by JSP engine. The syntax of comment tag is mentioned below. `<%-- This is a comment section --%>` In JSP, a tag inside another tag is not allowed. So, comment tag cannot be used inside the scriptlet elements. Instead of JSP comment, you can also use HTML comment.

JSP Comments

JSP provides separate tag for comment section. The content inside these tag is completely ignored by JSP engine.

The syntax of comment tag is mentioned below.

```
<%-- This is a comment section --%>
```

In JSP, a tag inside another tag is not allowed. So, comment tag cannot be used inside the scriptlet elements. Instead of JSP comment, you can also use HTML comment.

JSP Directives

In JSP, the role of directive is to provide directions to the JSP container regarding the compilation of JSP page. Directives convey information from JSP page to container that give special instruction at translation time.

The three types of directive provided by JSP is as follows:-

- Taglib Directive
- Include Directive
- Page Directive

Taglib Directive Taglib directive is used to define tag libraries in JSP page. It enables user to use custom tags within JSP file. A JSP page can contains more than one taglib directives. **Syntax of Taglib Directive** <%@ taglib uri="uri" prefix="value" %> Here, uri represents the path of tag library description and prefix represent the name of custom tag. **Example of Taglib directive** Here is a simple example of taglib directive. To run this code you need to add jstl jar file within lib directory of your project. It is better to run this code after learning JSTL. **index.jsp**

```
</html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<h2 align="center">
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:out value="Welcome to JSP Taglib Directive"/>
</h2>
</body>
</html>
```



We will learn more about taglib directive in

JSP custom tag.

JSP Taglib Directive

Taglib directive is used to define tag libraries in JSP page. It enables user to use custom tags within JSP file. A JSP page can contains more than one taglib directives.

Syntax of Taglib Directive

```
<%@ taglib uri="uri" prefix="value">
```

Here, uri represents the path of tag library description and prefix represent the name of custom tag.

Example of Taglib directive

Here is a simple example of taglib directive. To run this code you need to add jstl jar file within lib directory of your project. It is better to run this code after learning JSTL.

index.jsp

```
</html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<h2 align="center">
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:out value="Welcome to JSP Taglib Directive"/>
</h2>
</body>
</html>
```

We will learn more about taglib directive in JSP custom tag.

JSP Include Directive

As the name implies, include directive is used to include the content of other files such as HTML, JSP, text into the current JSP page. These files are included during translation phase.

Syntax of include directive

```
<%@ include file="file-name">
```

Include directive tag can be placed anywhere in JSP page.

Example of Include Directive

This example expresses the simple way to include more than one file of different types within JSP page through include directive.

header.html

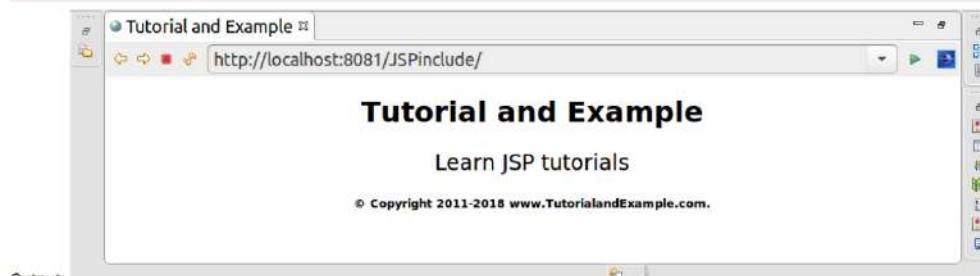
```
<h1 align="center">  
Tutorial and Example  
</h1>
```

footer.jsp

```
<h5 align="center">  
© Copyright 2011-2018 www.TutorialandExample.com.  
</h5>
```

index.jsp

```
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
<title>Tutorial and Example</title>  
</head>  
<body>  
<%@ include file="header.html">  
<font size="5">  
<p align="center">Learn JSP tutorials</p></font>  
<%@ include file="footer.jsp">  
</body>  
</html>
```



Output:

JSP Page Directive

JSP page directive provides various attributes with unique properties. These attributes can be applied to an entire JSP page. The syntax of Page directive is as follows: -

```
<%@page attribute="value">
```

Page directive tag can be placed anywhere inside the JSP page but placing it as the first statement is more preferable.

Attributes of Page Directive

These are the following attributes of page directive: -

language

This attribute defines the scripting language used in JSP page. By default, Java is used as a scripting language.

Syntax: -

```
<%@page language="Scripting-language">
```

contentType

In servlets, we required to set the type of content on response attribute.

```
response.setContentType("text/html");
```

Like servlets, JSP use contentType attribute to define MIME type and character set of the response message. The default expression is: -

```
<%@page contentType="text/html; charset=UTF-8">
```

buffer

This attribute is use to buffer the response objects. Here, value represents the size of buffer. Instead of passing numeric value, it also allows to declare none. In this case, buffer uses its default size that is 8kb.

Syntax: -

```
<%@page buffer="value">
```

info

In servlets, getServletInfo() method returns the information about the servlet. The similar role is played by info attribute in JSP. Hence, this attribute provides any type of description or information in a form of string.

Syntax: -

```
<%@page info="value">
```

autoFlush

The purpose of this attribute is to flush the buffer automatically. For this, the value of autoFlush must be true.

Syntax: -

```
<%@page autoFlush="true/false">
```

The default value of autoFlush is always true.

isThreadSafe

The role of isThreadSafe attribute is similar to SingleThreadModel interface in Servlet. Thus, it ensures that JSP handle only one type of request at a time.

Syntax: -

```
<%@page isThreadSafe="true/false">
```

Here, the default value is true.

extends

The extends attribute inherits the superclass to serve its properties in current class. It is similar to extends keyword in Java.

```
<%@page extends="package.class">
```

import

This attribute is used to import packages in JSP page. The package consists of similar type of classes and interfaces. It is similar to import keyword in Java.

```
<%@page import="package-name">
```

session

Sessions are used to recognize the user. By default, the value of session is true in JSP. Hence, JSP always establish a session unless we make the value false.

```
<%@page session="true/false">
```

errorPage

This attribute redirects the current page to JSP exception page if any exception occurs.

Syntax: -

```
<%page errorPage="value">
```

Here, URL of JSP exception page is passed.

isErrorPage

This attribute specifies that the current JSP page contain an error page. This error page can be utilized only when exception occur.

Syntax: -

```
<%page isErrorPage="true/false">
```

The default value of isErrorPage is always false.

isELIgnored

This attribute can be used to enable or disable the usage of Expression Language tags. By default, the value is true.

```
<%page isELIgnored="true/false">
```

Thus, if there is no requirement of Expression Language tag then it can be disabled from JSP page.

JSP Control Statements

There are various standard programming languages like C, C++, Java etc. that supports control statements. On the basis of that, JSP also follows the same methodology. Let's study the flow of control statements in Java Server Pages. Here, are some conditional statements about which we will discuss: -

- if else
- for loop
- while loop
- switch

IF...ELSE

JSP allows to use IF ELSE conditional statement as a programming logic of scriptlet tag. Apart from that, IF can also be used in many other ways like nested IF or standalone IF. Here is a simple example of IF ELSE statement.

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<%! int num=8; %>
<% if(num%2==0)
{
    out.println("Number is even");
}
else
{
    out.println("Number is odd");
%>
</body>
</html>
```

For Loop

Here, is a simple example of for loop.

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<%
for(int i=0;i<5;i++)
{
    for(int j=1;j<=i+1;j++)
    {
        out.print(j);
    }
    out.print("<br>");
}
%>
</body>
</html>
```

While loop

Here, is a simple example of while loop.

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<%
int i=0;
while(i<4)
{
    i++;
    out.print(i+"<br>"); 
}
%>
</body>
</html>
```

Switch Statement

Switch is used to maintain the flow of control statement. It contains various cases with one default case. The default case will execute only once, when none other case satisfies the condition. Here, is a simple example of switch statement.

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<% int weekday=4; %>
<%
switch(weekday)
{
case 1:
    out.print("Monday");
    break;
case 2:
    out.print("Tuesday");
    break;
case 3:
    out.print("Wednesday");
    break;
case 4:
    out.print("Thursday");
    break;
case 5:
    out.print("Friday");
    break;
case 6:
    out.print("Saturday");
    break;
default:
    out.print("Sunday");
    break;
}
%>
</body>
</html>
```

JSP Implicit Objects

In JSP, implicit objects are pre-defined objects, created by the web container. These objects are created during the translation phase from JSP to Servlet. JSP allows you to call these objects directly without initializing them.

Implicit objects are required to be declared in scriptlet tags only. Hence, in translation phase these objects are embedded within service() method of servlet.

Note: - Servlet provide these objects to enhance the ease of coding in JSP.

List of JSP implicit objects

JSP provides 9 implicit objects that can be used in JSP pages. Each object represents some unique identity. Here, is the list of all implicit objects with their classes.

Object	Classes
out	javax.servlet.jsp.JspWriter
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
config	javax.servlet.ServletConfig
application	javax.servlet.ServletContext
session	javax.servlet.http.HttpSession
exception	javax.servlet.http.HttpException
Page	java.lang.Object
PageContext	javax.servlet.jsp.PageContext

We will learn more about each object with simple examples.

JSP out

In JSP, out is an implicit object that is associated with the response object. This object is used to write content to the output stream.

Methods of JSP out

Following are some important methods of JSP out: -

Method	Description
void print()	This is one of the most frequently used method. It prints the statement.
void clear()	This method is used to remove the content of buffer.
boolean isAutoFlush()	This method specifies whether the buffer is flushed or not.
int getBufferSize()	This method returns the size of buffer in bytes.
int getRemaining()	This method returns the unused size of buffer in bytes.

Example of JSP out

This example shows the functionality of all the above methods on JSP out object.

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<%
String s="Tutorial and Example";
out.println("Learn JSP");
out.clear();
Boolean b=out.isAutoFlush() ;
out.println("<br><h2 align=center>IsAutoFlush:"+b);
out.println("<br>Website:"+s);
int i1=out.getBufferSize();
out.println("<br>Buffer Size:"+i1);
int i2=out.getRemaining();
out.println("<br>Remaining size:"+i2+"</h2>");
%>
</body>
</html>
```

JSP Request

The request object is used to fetch the user's data from the browser and pass this data to the server. The working of JSP request object is similar to the servlet's HttpServletRequest interface object.

Example of JSP Request

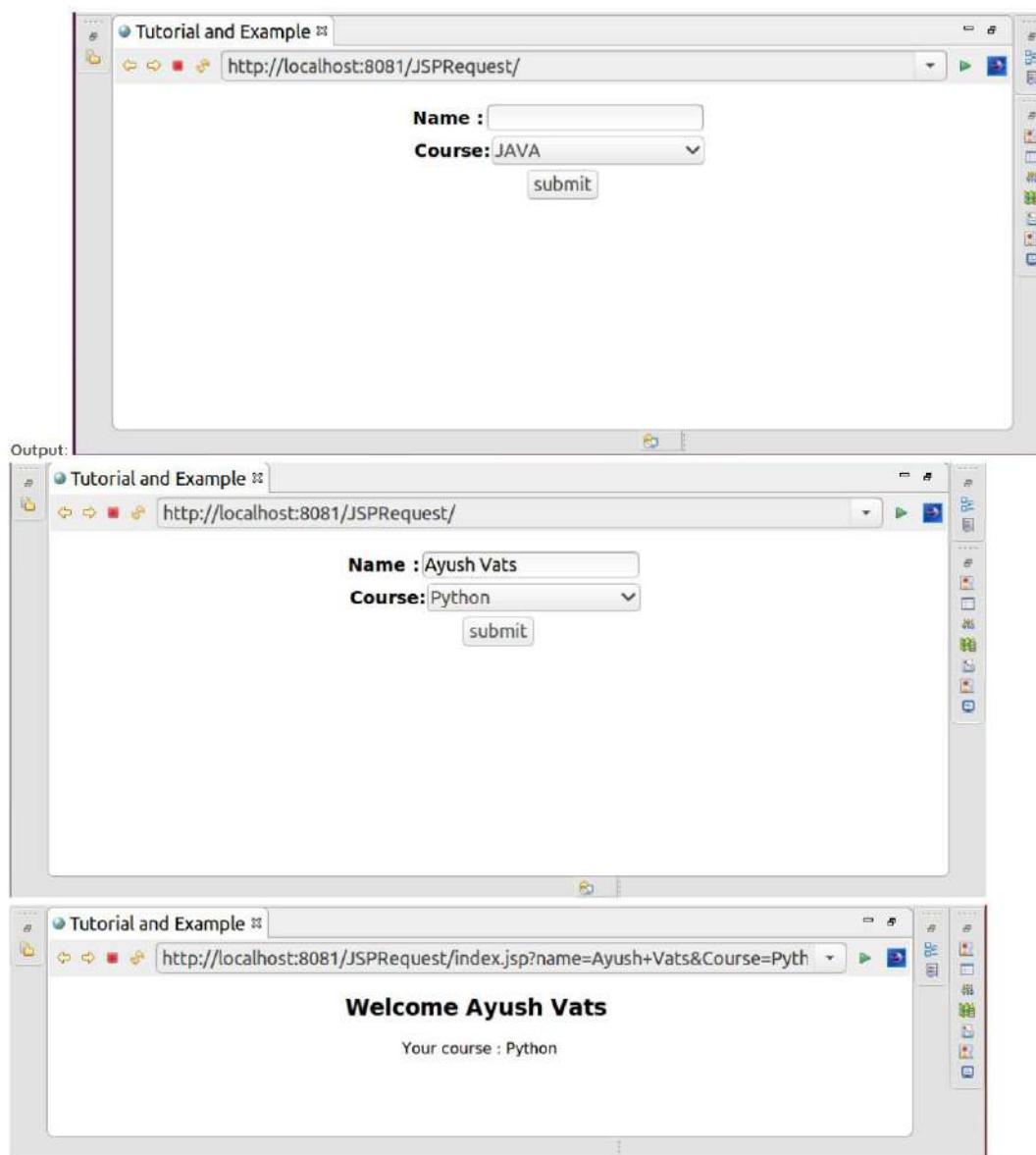
In this example, request object fetches the form data.

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Tutorial and Example</title>
</head>
<body>
<h3 align="center">
<form action="index.jsp">
Name :<input type="text" name="name"> <br>
Course:<select name="Course" style="width:230px">
<option value="JAVA">JAVA</option>
<option value="PHP">PHP</option>
<option value="Python">Python</option>
<option value="Other">Other</option>
</select> <br>
<input type="submit" value="submit">
</form>
</h3>
</body>
</html>
```

index.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tutorial and Example</title>
</head>
<body>
<center>
<%
String str1=request.getParameter("name");
String str2=request.getParameter("course");
%>
<h2>
<%="Welcome "+str1%> <br>
</h2>
<%="Your course : "+str2 %>
</center>
</body>
</html>
```



JSP response

Basically, JSP response object is an instance of servlet's HttpServletResponse interface. It is used to resolve client request.

Response object can be used to perform various functions such as encode URL, add cookies, add headers, send errors, set content type etc.

Example of JSP Response

In this example, user's password is set in index.jsp file. If the password entered in the form is same then valid.jsp file will execute. If entered password is unmatched then invalid.jsp file will execute.

index.html

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<form action="index.jsp">
Name:<input type="text" name="name"> <br>
Password:<input type="password" name="pass"> <br>
<input type="submit" name="submit">
</form>
</body>
</html>
```

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<%
String name=request.getParameter("name");
String pass=request.getParameter("pass");
if(pass.equals("abcd"))
{
    response.sendRedirect("valid.jsp");
}
else
{
    response.sendRedirect("invalid.jsp");
}
%>
</body>
</html>
```

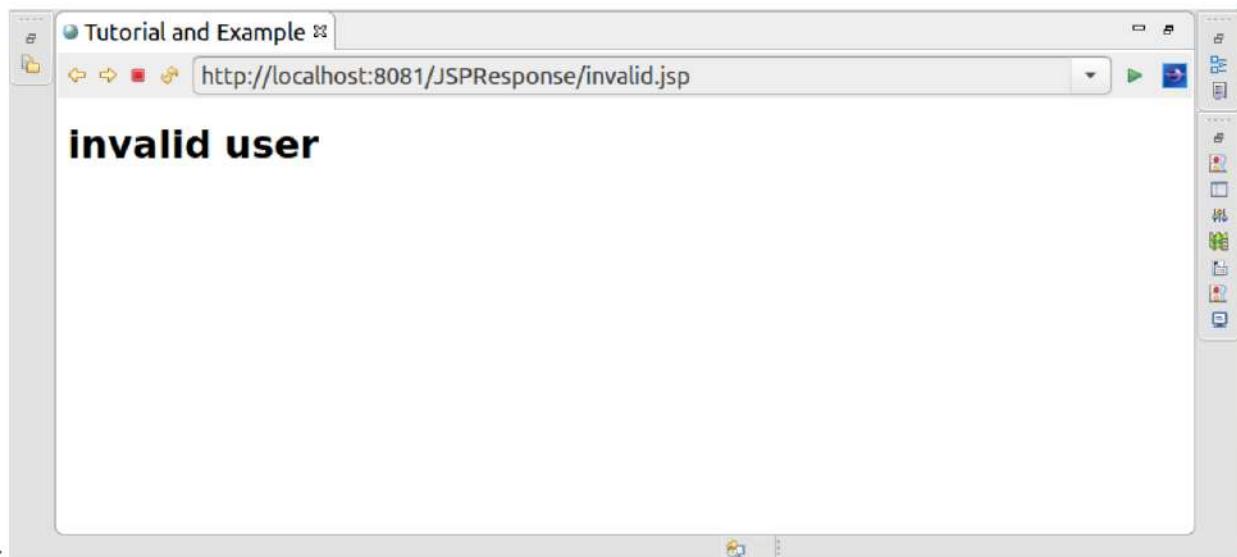
Valid.jsp

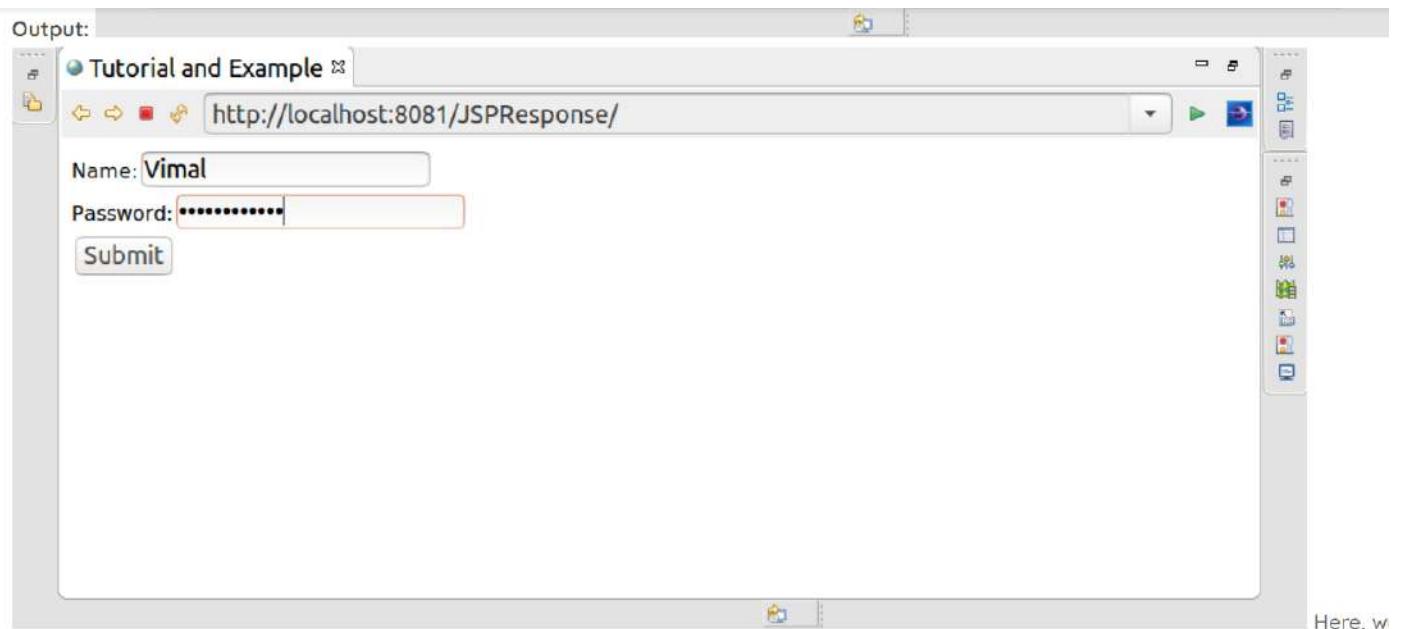
```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<h1>
<%="Valid user"%>
</h1>
</body>
</html>
```

invalid.jsp

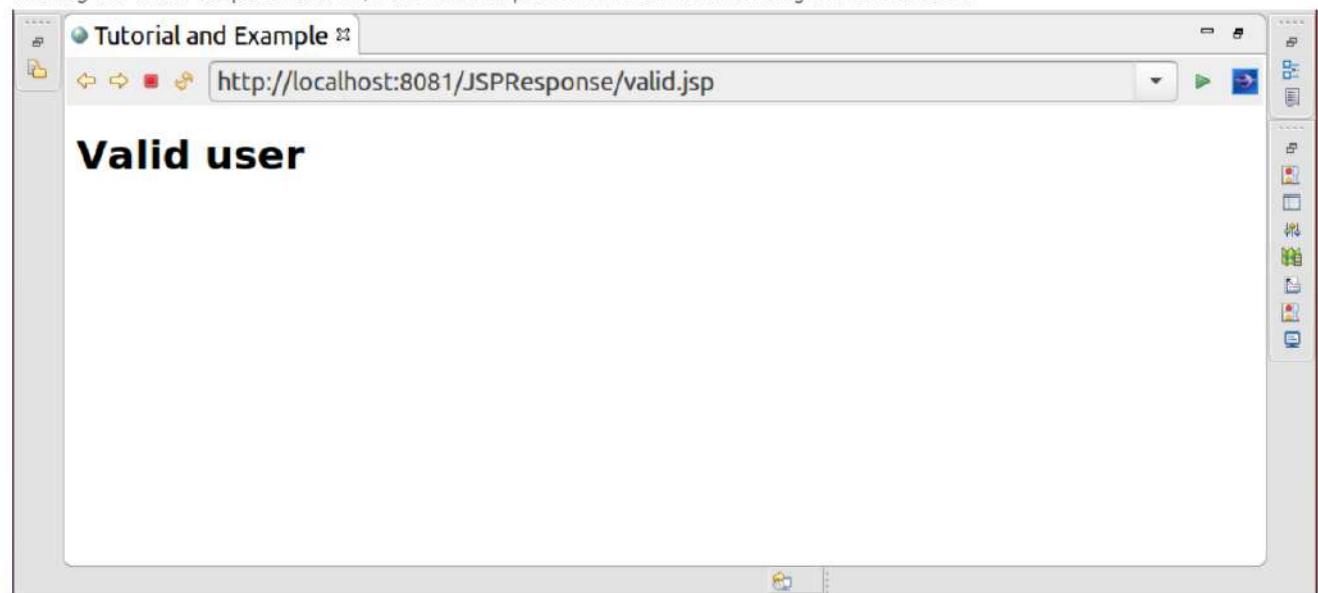
```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<h1>jsp response object method
<%="invalid user" %>
</h1>
</body>
</html>
```

Here, we already set "abcd" as password. So, if the entered password is correct then only the user is valid.





already set "abcd" as password. So, if the entered password is correct then only the user is valid.



JSP config

JSP config object is used to send the configuration information to JSP page. Here, config is an instance of servlet's ServletConfig interface.

In this case, the information is send to JSP page through web.xml file. Config object is used to fetch this information. It is used widely for the initialization of parameters.

Note: - The scope of JSP config object is up to a single JSP page only.

Example of JSP Config

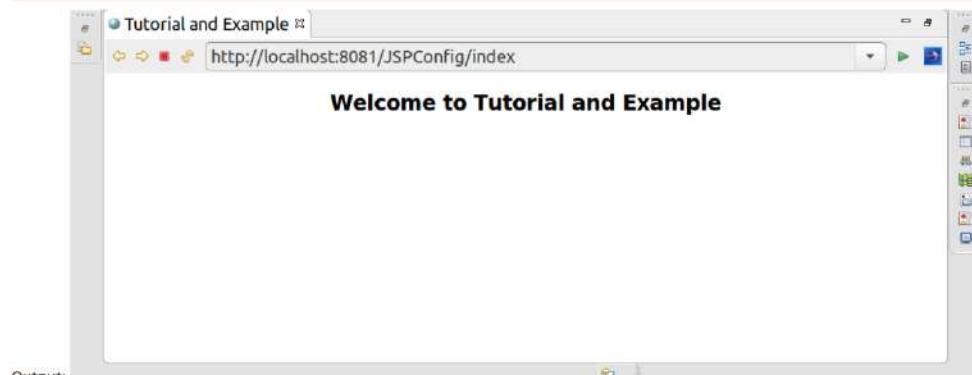
In this example, web.xml file contains the information. The config object in index.jsp file fetches that information.

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<%
String name=config.getServletName();
out.print("<h2 align=center>Welcome to "+name+"</h2>");
%>
</body>
</html>
```

web.xml

```
<web-app>
<servlet>
<servlet-name>Tutorial and Example</servlet-name>
<jsp-file>/index.jsp</jsp-file>
</servlet>
<servlet-mapping>
<servlet-name>Tutorial and Example</servlet-name>
<url-pattern>/index</url-pattern>
</servlet-mapping>
</web-app>
```



JSP application

JSP application object is an instance of servlet's ServletContext interface. It is used initialize parameter of one or more JSP pages in an application. Thus, the scope of application object is wider than config object.

Example of JSP Application

This example shows the functionality of various methods of application object.

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<%
String tut="JSP Tutorial";
application.setAttribute("var",tut);
String fetch=(String)application.getAttribute("var");
String info=application.getServerInfo();
int majversion=application.getMajorVersion();
int minversion=application.getMinorVersion();
%>
<h2 align="center">
<%=fetch %>
<br>
<%="Servlet Info:"+info %>
<br>
<%="Major Version:"+majversion %>
<br>
<%="Minor Version:"+minversion %>
</h2>
</body>
</html>
```



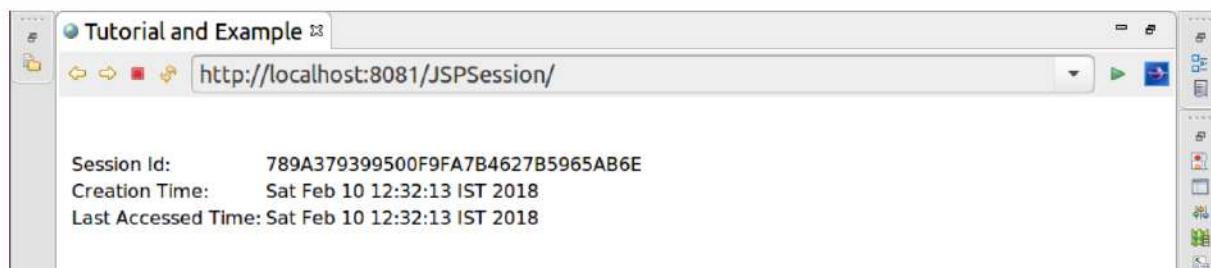
JSP session

In JSP, session object is used to establish a connection between a client and a server. It maintains the state between them so that server recognize the user easily.

The session object is an instance of servlet's HttpSession interface. This object contains the user's information and maintains the state till this session is active.

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<%
String id=session.getId();
Date d1=new Date(session.getCreationTime());
Date d2=new Date(session.getLastAccessedTime());
%>
<table>
<%="<tr><td>Session Id:</td><td>" +id+ "</td></tr>"%>
<br>
<%="<tr><td>Creation Time:</td><td>" +d1+ "</td></tr>"%>
<br>
<%="<tr><td>Last Accessed Time:</td><td>" +d2+ "</td></tr>"%>
</table>
</body>
</html>
```



JSP pageContext

The object pageContext is used to access all the information of JSP page. It is an instance of javax.servlet.jsp.PageContext.

The scope of pageContext can be defined explicitly. The object is valid up to the specified scope only. Several layers in which pageContext can be accessed are as follows:-

- SESSION_SCOPE
- REQUEST_SCOPE
- PAGE_SCOPE
- APPLICATION_SCOPE

Methods of pageContext

The various methods of pageContext are as follows:-

- void setAttribute(String attribute_name, Object attribute_value, int scope):- This method is used to set the attribute.
- Object getAttribute(String attribute_name, int scope):- This method return the attribute of object type.
- Object removeAttribute(String attribute_name, int scope):- This method is used to remove the attribute.
- Object findAttribute(String attribute_name):- This method search the attribute passed within its bracket.

Example of pageContext

This is a simple example that defines the way of set and get the various attributes.

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<h1 align="center">Tutorials:</h1>
<h2 align="center">
<%>
pageContext.setAttribute("Tutorial1", "HTML", PageContext.SESSION_SCOPE);
pageContext.setAttribute("Tutorial2", "Servlet", PageContext.REQUEST_SCOPE);
pageContext.setAttribute("Tutorial3", "JSP", PageContext.PAGE_SCOPE);
pageContext.setAttribute("Tutorial4", "JavaScript", PageContext.APPLICATION_SCOPE);
String name1=(String)pageContext.getAttribute("Tutorial1", PageContext.SESSION_SCOPE);
String name2=(String)pageContext.getAttribute("Tutorial2", PageContext.REQUEST_SCOPE);
String name3=(String)pageContext.getAttribute("Tutorial3", PageContext.PAGE_SCOPE);
String name4=(String)pageContext.getAttribute("Tutorial4", PageContext.APPLICATION_SCOPE);
out.println(name1);
out.println("<br>" + name2);
out.println("<br>" + name3);
out.println("<br>" + name4);
%>
</h2>
</body>
</html>
```



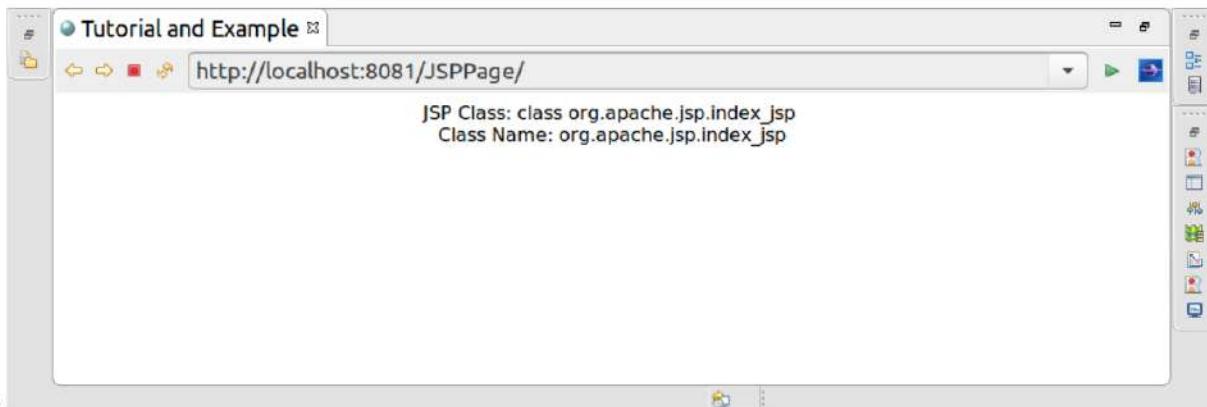
JSP page

In JSP, the purpose of page object is to create the servlet instance. The role of page object is similar to this keyword in Java.

The page object represents the entire JSP page. It is an instance of java.lang.Object class. In JSP, the utilization of page object is very rare.

Example of JSP Page

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tutorial and Example</title>
</head>
<body>
<center>
JSP Class: <%=page.getClass()%>
<br>
Class Name: <%=page.getClass().getName()%>
</center>
</body>
</html>
```



Output:

JSP exception

Like Java, exceptions can also be occurred in JSP. To handle these exceptions, JSP exception object is used.

Handling exceptions in JSP using exception object is much easier approach. This object is needed to be declared in a separate exception page. The exception page path is provided to the current page within page directive.

However, try catch block can also be used to handle these exceptions.

Example of exception

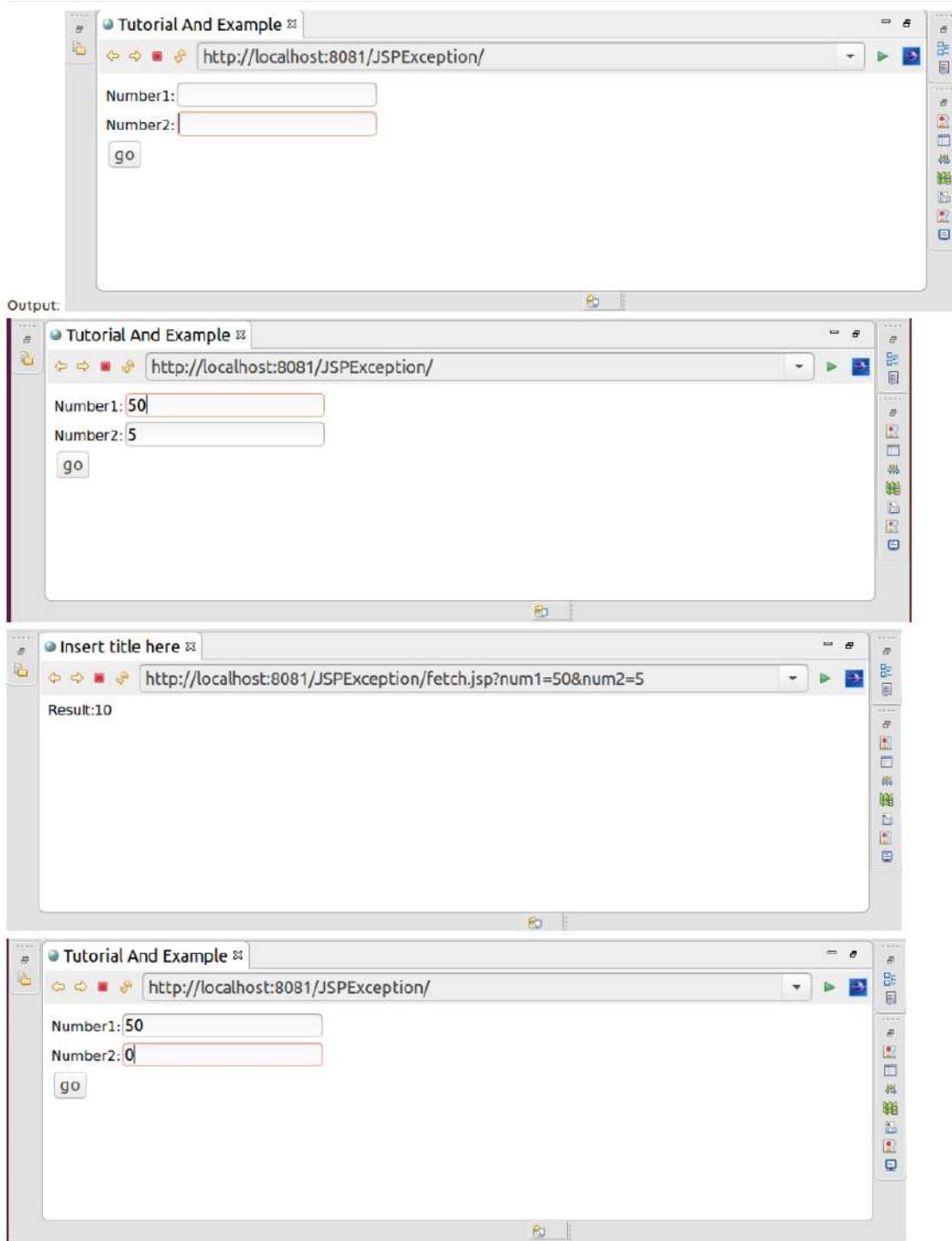
In this example, we use exception object to handle the exceptions. The result is generated on the basis of division of the provided numbers. If inappropriate number is given in the input then an exception occurs.

index.jsp

```
<form action="fetch.jsp">
Number1:<input type="text" name="num1"> <br>
Number2:<input type="text" name="num2"> <br>
<input type="submit" value="go">
</form>
```

fetch.jsp

```
<%@ page errorPage="Errorgen.jsp"%>
<%
String s1=request.getParameter("num1");
String s2=request.getParameter("num2");
int i1=Integer.parseInt(s1);
int i2=Integer.parseInt(s2);
int sum=i1/i2;
out.println("Result:"+sum);
%>
Errorgen.jsp
<%@ page isErrorPage="true"%>
<html>
<body>
Exception Occurs: <%=exception%>
</body>
</html>
```



JSP Action Tags

As the name implies, JSP action tags are used to perform various actions during the execution of JSP page. Here, each action performs some specific task. These tasks include:

- Forwarding the current page request to another page.
- Including external page to JSP page.
- Creating a JavaBean instance.

JSP provides additional functionality to JSP pages through these action tags. Unlike directive tags, action tag conveys the information to servlet container when request is being processed.

Syntax

```
<jsp:action_name attribute="value">
```

List of JSP Action Tags

These are the various action tags provided by JSP:-

- **jsp:forward** - This tag is used to forward the request of current JSP page to any another JSP, Servlet or HTML page.
- **jsp:include** - This tag includes the external resource to JSP page.
- **jsp:useBean** - This tag deals with the object of JavaBean.
- **jsp:setProperty** - This tag sets the property of JavaBean object.
- **jsp:getProperty** - This tag gets the property associated with JavaBean object.
- **jsp:text** - This tag is used to write template text in JSP page.
- **jsp:elements** - This tag is to define XML elements dynamically.
- **jsp:attribute** - This tag defines the attribute of dynamically generate XML elements.
- **jsp:body** - This tag defines the body of dynamically generate XML elements.
- **jsp:plugin** - This tag is used to integrate Java components with JSPPage.

JSP forward and include tag

In JSP, forward and include tags are the most frequently used action tags. Unlike Servlets, there is no need to create object of RequestDispatcher interface to use the functionality of forward and include. Thus, JSP provides direct tag for the same purpose.

JSP forward Tag

In JSP, forward tag terminates the execution of current JSP page and forward the control to new page which can be JSP, Servlet, HTML or any other. The response of new page is displayed to the user.

Syntax: -

```
<jsp:forward page="filename">
```

JSP include Tag

The role of include tag is to dispatch the external resource in the JSP page. The external resource can be JSP, Servlet HTML or any other page.

Syntax: -

```
<jsp:include page="filename">
```

Example of forward and include Tag

This example represents the functionality of both, forward and include tag.

index.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tutorial and Example</title>
</head>
<body>
<font size="5">
<form action="CheckPage.jsp">
Name:<input type="text" name="name"> <br>
Password:<input type="password" name="pass"> <br>
<input type="submit" value="submit">
</form>
</font>
</body>
</html>
```

CheckPage.jsp

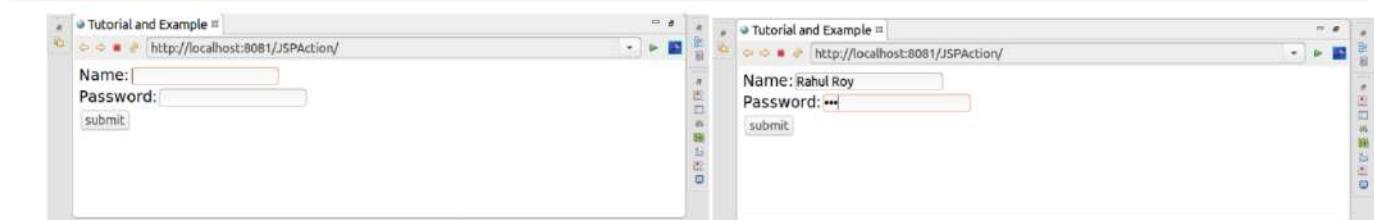
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tutorial and Example</title>
</head>
<body>
<font size="5">
<% String s=request.getParameter("pass");
if(s.length()>4)
{
%
<jsp:forward page="WelcomePage.jsp"></jsp:forward>
<%
}
}
else
{
%
<jsp:forward page="Error.jsp"></jsp:forward>
<%
}
%
</font>
</body>
</html>
```

WelcomePage.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tutorial and Example</title>
</head>
<body>
<font size="5">
<% String s=request.getParameter("name"); %>
<%="Welcome : "+s %>
</font>
</body>
</html>
```

Error.jsp

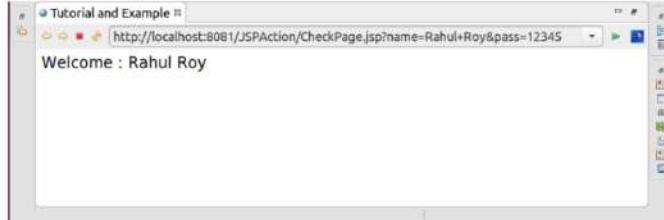
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tutorial and Example</title>
</head>
<body>
<font size="5">
Error: Password must be more than 4 digits <br>
<jsp:include page="index.jsp"></jsp:include>
</font>
</body>
</html>
```



The entered password must be more than 4 digits.



Enter a strong password (i.e. more than 4 digits) to see welcome page.



JSP useBean, getProperty and setProperty tags

Java bean is a special type of Java class that contains private variables and public setter and getter methods. Java bean class implements Serializable interface and provides default constructor.

JSP allows you to access the Java beans in JSP files. To invoke Java bean, useBean action tag is used and to access the properties of Java beans, JSP setter and getter method is used.

Example of useBean Tag

In this example, setProperty() and getProperty() method is used to set and get data of current Java class.

EmployeeBean.java

```
package com.tutorialandexample;
import java.io.Serializable;
public class EmployeeBean implements Serializable {
    private String name;
    private int id;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

index.jsp

```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<jsp:useBean id="emp" class="com.tutorialandexample.EmployeeBean">
<jsp:setProperty property="name" name="emp" value="Sushant Singh"/>
<jsp:setProperty property="id" name="emp" value="101"/>
<jsp:setProperty property="age" name="emp" value="23"/>
</jsp:useBean>
Employee ID : <jsp:getProperty property="id" name="emp"/> <br>
Employee Name : <jsp:getProperty property="name" name="emp"/> <br>
Employee Age : <jsp:getProperty property="age" name="emp"/>
</body>
</html>
```

JSP Cookie Handling

Cookie is a small piece of information sent by server to recognize the user. This information is stored at client side in the textual form.

In JSP, cookie is an object of javax.servlet.http.Cookie class. This object is used widely for session management purposes.

Methods of JSP Cookie

Some of the important method of cookie are as follows:-

- void setMaxAge(int expiry); - This method is used to set the expiry time of cookie. Here, time is represent in the form of seconds.
- void getMaxAge(); - This method returns the declared time of cookie after which it will expire.
- String getName(); - This method returns the name of the cookie.
- void setValue(String value); - This method is used to set associate the value with cookie.
- String getValue(); - This method returns the value associated with the cookie.

Example of JSP Cookie assessment

Index.jsp

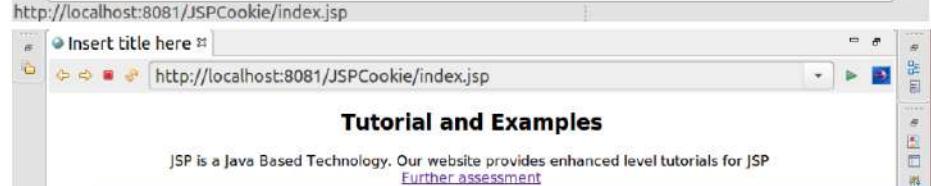
```
<html>
<body>
<%
String opt="Servlet";
Cookie[] ck=request.getCookies();
if(ck!=null)
{
    for(Cookie temp:ck)
    {
        if("technology".equals(temp.getName()))
        {
            opt=temp.getValue();
            break;
        }
    }
}
<br>
<center>
<h2>Tutorial and Examples</h2>
<%=opt+" is a Java Based Technology." %>
<%="Our website provides enhanced level tutorials for "+opt %>
<br>
<a href="selection.jsp">Further assessment</a>
</center>
</body>
</html>
```

Selection.jsp

```
<html>
<body>
<center>
Select the technology
<form action="cookiepage.jsp">
<select name="tech">
<option>Servlet</option>
<option>JSF</option>
<option>JavaScript</option>
</select>
<br>
<input type="submit" value="submit">
</form>
</center>
</body>
</html>
```

Cookiepage.jsp

```
<html>
<body>
<center>
<
String s=request.getParameter("tech");
Cookie c=new Cookie("technology",s);
c.setMaxAge(60);
response.addCookie(c);
</
<h2>thank you for selection</h2> <br>
<a href="index.jsp">Main page</a>
</center>
</body>
</html>
```



JSP Expression Language

JSP introduced the concept of expression language (EL) in JSP 2.0 version. The purpose of expression language is to access and manipulate data easily without using any type of scriptlet.

Why Expression language?

Before JSP 2.0, scriptlets are used to handle the business logic. But it is difficult task for web designers to understand and code the java part in scriptlets. Hence, now expression language is used to overcome this difficulty.

Syntax

This is the simple syntax of expression language:

```
$ {expression}
```

Expression Language Operator

JSP expression language provides an easy way to perform various arithmetic, relational, logical operations. These are some of the frequently used operators: -

- Arithmetic operators - It contains various mathematical operators such as addition (+), subtraction (-), division (/ or div), module (% or mod), multiplication (*).
- Logical operators - It contains operators like && (and), || (or).
- Relational operators - It contains operators like < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to).

Example of JSP Expression Language Operator

This is a simple example of JSP operators that perform various mathematical operations between two numbers (10 & 5) directly.

Index.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>Arithmetic Operators : 10&5 </h2>
Addition :
${10+5}
<br>
Subtraction :
${10-5}
<br>
Multiplication :
${10*5}
<br>
Division :
${10/5}
<br>
<h2>Relational Operators : 10&5 </h2>
Equality Check (==) :
${10==5}
<br>
Non-Equality Check (!=) :
${10!=5}
<br>
Less number check (<) :
${10<5}
<br>
Greater number check (>) :
${10>5}
</body>
```

JSP EL Implicit Objects

Apart from operators, JSP also provide expression language implicit objects. The role of these implicit objects is to get the attribute associated with various object. Here, is the list of expression language implicit objects with their purposes.

Implicit Object	Description
requestScope	Fetches the attribute associated with request object.
sessionScope	Fetches the attribute associated with session object.
pageScope	It provides the attribute associate within pageScope.
applicationScope	Fetches the attribute associated with application object.
param	It retrieves the single value associated with request parameter.
paramValues	It retrieves the array of values associated with request parameter.
header	It retrieves the single value associated with header parameter.
headerValues	It retrieves the array of values associated with header parameter.
cookie	It is used to get the value of cookies.
initParam	It is used to map an initialize parameter.
pageContext	The role of this object is similar to JSP pageContext implicit object.

Example of Expression Language Implicit Object

This example shows the way of using various expression language implicit objects.

index.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Tutorial and Example</title>
</head>
<body>
<%
request.setAttribute("Tutorial", "Java");
application.setAttribute("Tutorial", "Python");
request.getSession().setAttribute("Tutorial", "PHP");
%>
<center>
Host name:
${header["host"]} <br>
Path name:
${pageContext.request.contextPath} <br>
Tutorials: <br>
${requestScope.Tutorial} <br>
${applicationScope.Tutorial} <br>
${sessionScope.Tutorial} <br>
</center>
</body>
</html>
```

JSTL - Java Standard Tag Library

Java Standard tag library (JSTL) is a collection of various type of JSP tags. Each tag control the behavior of JSP page and perform specific task such as database access, conditional execution, internationalization etc.

JSTL delivers the functionality of programming methodology without using Java code. Thus, it provides ease to develop and maintain web applications.

Types of JSTL tag

In JSP, JSTL tags are divided into following parts:-

- JSTL Core
- JSTL Formatting
- JSTL SQL
- JSTL XML
- JSTL function

To implement JSTL tag in JSP page, it is required to add JSTL jar within lib directory of your project.

JSTL Core Tags

In JSTL, core tags are most frequently used tags that provides variable support, manages URL and control the flow of JSP page. To use JSTL it is required to associate the below URL in your JSP page:-

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

It is not mandatory to use 'c' in the prefix of JSTL core tag. You can also use any other character or word. But as a convention, it is better to use it.

Core Tags

Here, is the list of JSTL core tags:-

- <c:out> tag: - This tag is used to print statements or results. It works as an alternative of <%=expression %> tag.
- <c:set> tag: - This tag is used to declare the variable with its value and scope.
- <c:remove> tag: - This tag is used to remove the attributes.
- <c:if> tag: - This tag works as a conditional statement and used for testing conditions.
- <c:choose> tag: - The role of this tag is similar to switch statement.
- <c:when> tag: - The role of this tag is similar to case in switch statement. Thus, it test the conditions.
- <c:otherwise> tag: - This tag behaves similar to default in switch statement. Thus, it executes when no condition for <c:when tag> matches.
- <c:catch> tag: - This tag is used to handle the exceptions.
- <c:import> tag: - This tag includes another files of any type such as JSP, HTML, XML etc. to current JSP file.
- <c:forEach> tag: - This is an iteration tag used to traverse the elements.
- <c:param> tag: - This tag is used to add the parameters.
- <c:url> tag: -

Example of JSTL core Tag

In this example, the functionality of certain JSTL core tags are shown in an easy way.

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="marks" scope="session" value="${35}">
<c:if test="${marks>30}">
<p>Congrats!! You <c:out value="${'passed')}"> the exam </p>
</c:if>
<c:choose>
<c:when test="${marks>80}">
<c:out value="${'Good Performance')"/>
</c:when>
<c:when test="${marks<80||marks>30}">
<c:out value="${'Average Performance')"/>
</c:when>
<c:otherwise>
<c:out value="${'Bad Performance')"/>
</c:otherwise>
</c:choose>
</body>
</html>
```

JSTL Formatting Tags

In JSTL, the role of formatting tag is to specify the type of format that represents date and time. The following syntax represent the URL of formatting tag: -

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/fmt" prefix = "fmt" %>
```

Formatting tags

These are some commonly used formatting tags: -

- <fmt:parseNumber>: - This tag is used to parse the string on the basis of attribute associate with it.
- <fmt:parseDate>: - This tag is used to parse the string of a date and time.
- <fmt:formatDate>: - This tag handles the date in the various format.
- <fmt:message>: - This tag is used to display the internationalized message.
- <fmt:setTimeZone>: - This tag specifies the type of time zone.

Example of JSTL Format Tag

This is a simple example of JSTL format tag.

index.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html>
  <head>
    <title>Tutorial and Example</title>
  </head>
  <body>
    <h2>Format Tag example</h2>
    <c:set var="number" value="99.40" />
      <fmt:parseNumber var="n" integerOnly="true" type="number" value="${number}" />
      Parse Number is :: <c:out value="${n}" /> <br>
    <c:set var="date" value="02-04-2018" />
    <fmt:parseDate value="${date}" var="pd" pattern="dd-MM-yyyy" />
    Parse Date is :: <c:out value="${pd}" />
    <br>
    <c:set var="Date" value="<%=new java.util.Date()%>" />
    Format Date is :: <fmt:formatDate type="time" value="${Date}" />
  </body>
</html>
```

JSTL SQL Tags

In JSTL, SQL tags are used to access and manipulate various operations of database. It is responsible to interact the JSP page with any type of Database such as MySQL, Oracle, SQLite, etc. The below syntax is used to include SQL library within JSP:-

```
<%@ taglib prefix = "sql" uri = "http://java.sun.com/jsp/jstl/sql" %>
```

SQL Tags

These are some important JSTL SQL tags:-

- <sql:setDataSource> - This tag creates a DataSource to communicate JSP with database server.
- <sql:query> - This tag executes the SQL query that is specified with it.
- <sql:transaction> - This tag performs various database action with single connection.
- <sql:param> - This tag provides parameter to SQL statement.
- <sql:dateParam> - This tag provides date in the form of parameter to SQL statement.
- <sql:update> - This tag is used to update the value in database.

Example of JSTL SQL Tag

This is a simple example of JSTL SQL Tag that fetches data from MYSQL.

index.jsp

```
<%@ page import="java.io.* , java.util.* , java.sql.*" %>
<%@ page import="javax.servlet.http.* , javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
<title>sql:query Tag</title>
</head>
<body>
<sql:setDataSource var="db" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/jtp_gym"
    user="root" password="" />
<sql:query dataSource="${db}" var="rs">
SELECT * from admin;
</sql:query>
<table border="2" width="100%">
<tr>
<th>Student ID</th>
<th>Password</th>
<th>U Id</th>
</tr>
<c:forEach var="table" items="${rs.rows}">
<tr>
<td><c:out value="${table.id}" /></td>
<td><c:out value="${table.password}" /></td>
<td><c:out value="${table.user_id}" /></td>
</tr>
</c:forEach>
</table>
</body>
</html>
```

JSTL XML Tags

In JSTL, XML tags are used for creating and manipulating xml documents. The following syntax represent the URL of XML tag:-

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>
```

To utilize the functionality of XML tags, it is required to download and add two jar files within the lib directory of your project. Following are the required jar files:-

- XercesImpl.jar
- xalan.jar

XML Tags

- <x:out>: - This role of this tag is similar to <%=expression%> tag but for XPath expression.
- <x:set>: - This tag associate the value with variable for XPath expression.
- <x:if>: - This tag treats as a conditional statement for XPath expression.
- <x:choose>: - This tag function as a switch statement for XPath expression.
- <x:when>: - This tag is used to test the conditions just same as Case in switch statement.
- <x:otherwise>: - This tag invokes if none of the <x:when> tag condition satisfies. Thus, it treats as default of switch statement.

Example of JSTL XML Tag

This example shows the functionality of various tags of JSTL XML.

index.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
<h2>JSTL XML Tag Demo </h2>
<c:set var="website">
<website>
    <name>Tutorial and Example</name>
    <tutorial>Java Server Pages</tutorial>
</website>
</c:set>
<x:parse xml="${website}" var="output"/>
<b>Website :: </b>
<x:out select="$output/website[1]/name" /><br>
<b>Tutorial ::</b>
<x:out select="$output/website[1]/tutorial" />
</body>
```

JSTL Function Tags

In JSTL, the purpose of function tags is similar to the various string methods in Java. Hence, these tag performs common string manipulation operations.

Functional tag is represented by following URL:-

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

Function Tags

These are some frequently used function tags: -

- fn:length(): - This tag returns the length of the string i.e. number of characters present in the string.
- fn:trim(): - This tag removes the spaces from both ends of the string.
- fn:replace(): - This tag is used to replace one string with another.
- fn:substring(): - This tag provides the substring (i.e. part of string) the string.
- fn:join(): - This tag is used to concatenates separate strings of an array
- fn:split(): - This tag breaks the string into array of substrings.
- fn:contains(): - This tag checks whether the input string contains the specified substring.
- fn:toUpperCase(): - This tag converts the lower case elements of a string into upper case.
- fn:toLowerCase(): - This tag converts the upper case elements of a string into lower case.

Example of JSTL Function Tag

This is an example of JSTL Function tag that shows various JSTL function tags functionality.

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>Function Tag</title>
</head>
<body>
<c:set var="web" value="Tutorial and Example"/>
Length of String :: ${fn:length(web)} <br>
Substring of String :: ${fn:substring(web,0,8)} <br>
Is it contains word "Tutorial" :: ${fn:contains(web,'Tut')}<br>
In UpperCase :: ${fn:toUpperCase(web)}
</body>
</html>
```

JSP Custom Tag

JSP custom tag provides flexibility to programmers, to declare their own tag. Thus, a user-defined tag can be created through custom tags to perform various operations.

These custom tags behave as operations on tag handler when JSP page is translated into servlet. These operations are invoked by web container when servlet of JSP page is executed.

How to create Custom Tag?

These are the steps that you need to follow to create custom tag: -

- Tag Handler class – Create a tag handler class with .java extension. This class is used to perform operations of custom tag.
- JSP file – Create a JSP file and include a tag library in it.
- TLD file – It is required to create tag library file (i.e. TLD file) in WEB-INF directory of your project. This file is represented by .tld extension.

It is required to add both external jars i.e. jsp-api.jar and servlet-api.jar in classpath of your project before deploying the application.

Example of JSP Custom Tag

In this example,

customTagHandler.java

```
package com.tutorialandexample;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class CustomTagHandler extends SimpleTagSupport {
public void doTag() throws JspException, IOException
{
JspWriter jw=getJspContext().getOut();
jw.println("<h2>JSP Custom Tag</h2>");
}
}
```

Here, javax.servlet.jsp.tagext.* package contains all the classes that support custom tags and the role of SimpleTagSupport is to provide methods through which JSPWriter object can be accessed.

index.jsp

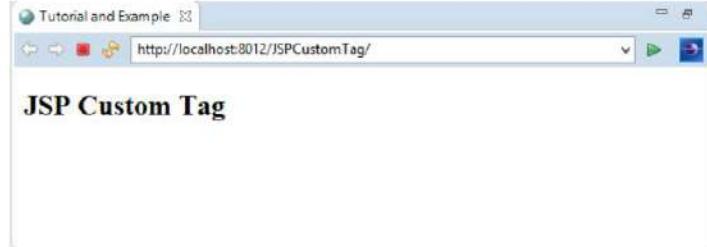
```
<html>
<head>
<title>Tutorial and Example</title>
</head>
<body>
    <%@ taglib uri="WEB-INF/xyz.tld" prefix="m" %>
<m:message/>
</body>
</html>
```

xyz.tld

```
<taglib>
<tlib-version>1.0</tlib-version>
<jsp-version>2.0</jsp-version>
<short-name>MyCustom</short-name>
<tag>
<name>message</name>
<tag-class>com.tutorialandexample.CustomTagHandler</tag-class>
<body-content>empty</body-content>
</tag>
</taglib>
```

Here, the name of tag (message) is given within the `<name>` tag and the full name of class is given within the `<tag-class>` tag. The value "empty" given within `<body-content>` means it doesn't contain any body.

Output



Note - We can also provide body and attribute within custom tags.

Explore More

Subscription : Premium CDAC NOTES & MATERIAL



Contact to Join
Premium Group



Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

codewitharrays.in freelance project available to buy contact on 8007592194

SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySQL
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySQL
3	Tour and Travel management System	React+Springboot+MySQL
4	Election commition of India (online Voting System)	React+Springboot+MySQL
5	HomeRental Booking System	React+Springboot+MySQL
6	Event Management System	React+Springboot+MySQL
7	Hotel Management System	React+Springboot+MySQL
8	Agriculture web Project	React+Springboot+MySQL
9	AirLine Reservation System / Flight booking System	React+Springboot+MySQL
10	E-commerce web Project	React+Springboot+MySQL
11	Hospital Management System	React+Springboot+MySQL
12	E-RTO Driving licence portal	React+Springboot+MySQL
13	Transpotation Services portal	React+Springboot+MySQL
14	Courier Services Portal / Courier Management System	React+Springboot+MySQL
15	Online Food Delivery Portal	React+Springboot+MySQL
16	Municipal Corporation Management	React+Springboot+MySQL
17	Gym Management System	React+Springboot+MySQL
18	Bike/Car ental System Portal	React+Springboot+MySQL
19	CharityDonation web project	React+Springboot+MySQL
20	Movie Booking System	React+Springboot+MySQL

freelance_Project available to buy contact on 8007592194

21	Job Portal web project	React+Springboot+MySQL
22	LIC Insurance Portal	React+Springboot+MySQL
23	Employee Management System	React+Springboot+MySQL
24	Payroll Management System	React+Springboot+MySQL
25	RealEstate Property Project	React+Springboot+MySQL
26	Marriage Hall Booking Project	React+Springboot+MySQL
27	Online Student Management portal	React+Springboot+MySQL
28	Resturant management System	React+Springboot+MySQL
29	Solar Management Project	React+Springboot+MySQL
30	OneStepService LinkLabourContractor	React+Springboot+MySQL

31	Vehical Service Center Portal	React+Springboot+MySQL
32	E-wallet Banking Project	React+Springboot+MySQL
33	Blogg Application Project	React+Springboot+MySQL
34	Car Parking booking Project	React+Springboot+MySQL
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySQL
36	Society management Portal	React+Springboot+MySQL
37	E-College Portal	React+Springboot+MySQL
38	FoodWaste Management Donate System	React+Springboot+MySQL
39	Sports Ground Booking	React+Springboot+MySQL
40	BloodBank mangement System	React+Springboot+MySQL
41	Bus Tickit Booking Project	React+Springboot+MySQL
42	Fruite Delivery Project	React+Springboot+MySQL
43	Woodworks Bed Shop	React+Springboot+MySQL
44	Online Dairy Product sell Project	React+Springboot+MySQL
45	Online E-Pharma medicine sell Project	React+Springboot+MySQL
46	FarmerMarketplace Web Project	React+Springboot+MySQL
47	Online Cloth Store Project	React+Springboot+MySQL
48	Train Ticket Booking Project	React+Springboot+MySQL
49	Quizz Application Project	JSP+Springboot+MySQL
50	Hotel Room Booking Project	React+Springboot+MySQL
51	Online Crime Reporting Portal Project	React+Springboot+MySQL
52	Online Child Adoption Portal Project	React+Springboot+MySQL
53	online Pizza Delivery System Project	React+Springboot+MySQL
54	Online Social Complaint Portal Project	React+Springboot+MySQL
55	Electric Vehical management system Project	React+Springboot+MySQL
56	Online mess / Tiffin management System Project	React+Springboot+MySQL
57	Online Examination Portal Project	React+Springboot+MySQL
58	Lawyer / Advocate Appointment Booking System	React+Springboot+MySQL
59	Café Management System	React+Springboot+MySQL
60	Agriculture Product Rent system Portal	React+Springboot+MySQL

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/vXz0kZQi5to?si=IiOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/lynLouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyl4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKz

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrk7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizlyJlh
15	Employee management system project	https://youtu.be/ID1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWA OzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vn0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Sr.No	Project Name	YouTube Link
21	Online Bus ticket booking system Project	https://youtu.be/FJ0RUZfMdv8?si=auHjmNgHMrpaNzvY
22	Online Mess / Tiffin Booking System Project	https://youtu.be/NTVmHFDowyl?si=yrvClbE6fdJ0B7dQ
23		
24		
25		

TAP ON THE ICONS TO JOIN!



TOP 50 SPRING BOOT ANNOTATIONS FOR INTERVIEW

1. @SpringBootApplication

1. Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
2. Marks the main class as the entry point of a Spring Boot application.
3. Enables auto-configuration for configuring Spring beans.
4. Scans components within the base package of the annotated class.
5. Simplifies Spring application setup and bootstrapping.

Example:

```
java
Code :
@SpringBootApplication
public class MyApp {
    public static void main(String[] args) { SpringApplication.run(MyApp.class,
        args);
}
}
```

2. @RestController

1. Combines @Controller and @ResponseBody.
2. Defines a controller for REST APIs.
3. Automatically serializes returned objects into HTTP responses.
4. Simplifies API creation by eliminating explicit @ResponseBody
5. Works well with @GetMapping, @PostMapping, etc.

Example:

```
java  
Code :  
@RestController  
  
public class UserController { @  
    GetMapping("/user") public  
    String getUser() { return  
        "User data";  
    }  
}
```

3. @GetMapping

1. Maps HTTP GET requests to handler methods.
2. A shortcut for @RequestMapping(method = RequestMethod.GET).
3. Used for fetching or querying data.
4. Supports dynamic path variables and query parameters.
5. Helps build RESTful GET APIs.

Example:

```
java  
Code :  
@GetMapping("/user/{id}")  
  
public String getUserById(@PathVariable int id) { return
```

```
"User with ID: " + id;  
}
```

4. @PostMapping

1. Maps HTTP POST requests to methods.
2. A shortcut for @RequestMapping(method = RequestMethod.POST).
3. Handles data creation or input submission.
4. Often paired with @RequestBody for accepting JSON data.
5. Simplifies form or API POST handling.

Example:

java
Code :

```
@PostMapping("/user")  
  
public String createUser(@RequestBody String user) {  
    return "User created: " + user;  
}
```

5. @PutMapping

1. Maps HTTP PUT requests to methods.
2. Used for updating resources.
3. Suitable for idempotent update operations.
4. Works well with @RequestBody for accepting input.
5. Often used for updating existing records.

Example:

java
Code :

```
@PutMapping("/user/{id}")  
  
public String updateUser(@PathVariable int id, @RequestBody String user) {
```

```
        return "User " + id + " updated with data: " + user;  
    }  
  
}
```

6. @DeleteMapping

1. Maps HTTP DELETE requests to method
2. Handles resource deletion on the server.
3. Simplifies handling of HTTP DELETE requests.
4. Typically includes @PathVariable to identify the resource.
5. Useful for RESTful DELETE APIs.

Example:

```
java  
Code :  
@DeleteMapping("/user/{id}")  
  
public String deleteUser(@PathVariable int id) {  
    return "User " + id + " deleted";  
}
```

7. @RequestMapping

1. General-purpose annotation for mapping HTTP requests.
2. Can handle all HTTP methods (GET, POST, etc.).
3. Used at both class and method levels.
4. Offers flexibility in request handling.
5. Replaced by @GetMapping, @PostMapping, etc., for specific methods.

Example:

java :

```
@RequestMapping("/api")
public class ApiController {
    @RequestMapping(value = "/users", method = RequestMethod.GET) public
    String getUsers() {
        return "All users";
    }
}
```



8. @PathVariable

1. Binds method parameters to URI variables.
2. Extracts values from the URL path.
3. Works with @GetMapping and @DeleteMapping.
4. Supports type conversion for values.
5. Useful for dynamic endpoints.

Example:

java

Code :

```
@GetMapping("/user/{id}")  
public String getUser(@PathVariable int id) { return  
    "User ID: " + id;  
}
```

9. @RequestParam

1. Binds query parameters to method arguments.
2. Supports default values for missing parameters.
3. Helps handle HTTP GET query parameters.
4. Works well with form submission.
5. Provides optional and required parameter settings.

Example:

java

Code :

```
@GetMapping("/search")  
public String search(@RequestParam String keyword) { return  
    "Search keyword: " + keyword;  
}
```

10. @RequestBody

1. Maps HTTP request body to method parameters.
2. Supports JSON, XML, and other formats.
3. Used in POST and PUT methods.
4. Requires a compatible deserialization library.
5. Simplifies handling of input data.

Example:

java

Code :

```
@PostMapping("/add")
public String addUser(@RequestBody User user) { return
    "User added: " + user.getName();
}
```

11. @ResponseBody

1. Indicates that a method's return value should be serialized into the HTTP response body.
2. Converts objects into JSON or XML for RESTful responses.
3. Automatically included in @RestController.
4. Can be used on individual controller methods.
5. Useful for creating non-view-based responses.

Example:

java

Code :

```
@GetMapping("/status") @
ResponseBody
public String getStatus() {
    return "Application is running";
}
```

12. @Controller

1. Marks a class as a Spring MVC controller.
2. Used to define traditional web controllers that return views (e.g., HTML).
3. Often paired with @RequestMapping for route handling.
4. Returns a ModelAndView or a logical view name.
5. Unlike @RestController, it does not include @ResponseBody by default.

Example:

java

Code :

@Controller

```
public class HomeController { @  
    GetMapping("/home") public  
    String home() {  
        return "home"; // Refers to home.html in templates  
    }  
}
```

13. @Service

1. Marks a class as a service layer component.
2. Indicates that it holds business logic.
3. Automatically detected and registered as a Spring bean.
4. Promotes separation of concerns between layers.
5. Works well with dependency injection.

Example:

java

Code :

@Service

```
public class UserService {  
    public String getUser() {  
        return "User service called";  
    }  
}
```

14. @Repository

1. Marks a class as a DAO (Data Access Object).
2. Indicates that it interacts with the database.
3. Automatically detected and registered as a Spring bean.
4. Provides exception translation for persistence-related errors.
5. Promotes separation of concerns in the data layer.

Example:

java

Code :

@Repository

```
public class UserRepository {
```

```
public String findUserById(int id) { return  
    "User found with ID: " + id;  
}  
}
```

15. @Component

1. Marks a class as a Spring-managed bean.
2. Acts as a generic stereotype for any Spring component.
3. Automatically detected during component scanning.
4. Can be used as a parent annotation for custom stereotype
5. Works across all layers of the application.

Example:

java

Code :

```
@Component public  
class Utility {  
    public String formatText(String text) {  
        return text.toUpperCase();  
    }  
}
```

16. @Autowired

1. Injects dependencies into Spring-managed beans.
2. Can be applied to constructors, setters, or fields.
3. Automatically resolves and injects a matching bean.
4. Reduces boilerplate code compared to manual bean wiring.
5. Requires a matching bean defined in the Spring context.

Example:

java

Code :

```
@Service
```

```
public class UserService { @  
    Autowired  
    private UserRepository userRepository;  
}
```

17. @Qualifier

1. Used with @Autowired to specify which bean to inject when multiple beans match.
2. Helps resolve ambiguity in dependency injection.
3. Works with bean names or custom qualifiers.
4. Ensures precise bean injection.
5. Useful for applications with multiple implementations of an interface.

Example:

java
Code :
@Service

```
public class UserService { @  
    Autowired  
    @Qualifier("adminRepository")  
    private UserRepository userRepository;  
}
```

18. @Primary

1. Marks a bean as the primary candidate for autowiring.
2. Used when multiple beans of the same type are present.
3. Eliminates the need for @Qualifier in some cases.

4. Ensures a default preference for bean injection.
5. Simplifies dependency management.

Example:

java

Code :

```
@Configuration
```

```
public class AppConfig { @  
    Bean  
    @Primary
```



```
public UserRepository userRepository() { return  
    new UserRepository();  
}  
}
```

19. @Bean

1. Marks a method as a bean definition in Java-based configuration.
2. Defines Spring beans manually in @Configuration classes.
3. Used to configure third-party libraries or non-Spring classes.
4. Provides fine-grained control over bean creation.
5. Supports dependency injection in method parameters.

Example:

```
java  
Code :  
@Configuration  
public class AppConfig { @  
    Bean  
    public Utility utility() {  
        return new Utility();  
    }  
}
```

20. @Configuration

1. Marks a class as a source of Spring bean definitions.
2. Typically used for Java-based configuration.
3. Replaces XML configuration files.
4. Works well with @Bean for explicit bean creation.
5. Scanned automatically if in the component scan path.

Example:

java

Code :

```
@Configuration  
public class AppConfig { @  
    Bean  
    public UserRepository userRepository() { return  
        new UserRepository();  
    }  
}
```

21. @Scope

1. Defines the scope of a Spring bean (e.g., singleton, prototype).
2. Works with @ Component, @ Service, and other bean-defining annotations.
3. Defaults to singleton scope.
4. Useful for creating new instances per request in prototype scope.
5. Enhances bean lifecycle management.

Example:

```
java  
Code :  
@Component  
@Scope("prototype")  
public class PrototypeBean {  
    public PrototypeBean() {  
        System.out.println("Prototype instance created");  
    }  
}
```

22. @Lazy

1. Indicates that a bean should be lazily initialized.
2. Defers bean creation until it is first requested.
3. Useful for optimizing application startup time.
4. Works with @Component, @Service, and @Bean.
5. Particularly effective in large, complex applications.

Example:

```
java
Code :
@Component @
Lazy
public class LazyBean { public
    LazyBean() {
        System.out.println("Lazy bean initialized");
    }
}
```

23. @Value

1. Injects values into fields, methods, or constructor parameters.
2. Supports property placeholders (e.g., \${property.name}).
3. Allows default values using : syntax (e.g., \${property.defaultValue}).
4. Can inject primitive types, strings, or arrays.
5. Often used to read values from application.properties.

Example:

```
java
Code :
@Component public
class Config {
```

@Value("\${app.name:DefaultApp}")



```
private String appName;

public String getAppName() {
    return appName;
}

}
```

24. @PropertySource

1. Loads properties files into the Spring environment.
2. Can load files from the classpath or file system.
3. Works with @Value for property injection.
4. Supports multiple property files.
5. Simplifies externalized configuration.

Example:

```
java
Code :
@Configuration
@PropertySource("classpath:application.properties")
public class AppConfig {
```

25. @EnableConfigurationProperties

1. Enables the use of configuration properties in a Spring Boot app.
2. Typically used with @ConfigurationProperties.
3. Simplifies binding of properties to Java objects.
4. Automatically registers annotated classes as beans.
5. Enhances type-safe configuration.

Example:

```
java
```

Code :

```
@EnableConfigurationProperties(AppProperties.class) @  
Configuration  
public class Config {  
}
```

26. @ConfigurationProperties

1. Binds external configuration properties to a Java bean.
2. Supports nested properties and data structures.
3. Works with @ EnableConfigurationProperties.
4. Enables type-safe access to configuration values.
5. Useful for managing complex configurations.

Example:

java
Code :

```
@ConfigurationProperties(prefix = "app")  
public class AppProperties {  
    private String name;  
    private String version;  
  
    // Getters and setters  
}
```

27. @Conditional

1. Enables conditional bean registration based on custom logic.
2. Can be combined with @ Bean, @ Component, etc.
3. Useful for creating beans only when certain conditions are met.
4. Supports both built-in and custom conditions.
5. Commonly used for environment-specific configurations.

Example:

```
java
Code :
@Configuration
public class AppConfig { @
    Bean
    @Conditional(MyCondition.class) public
    MyBean myBean() {
        return new MyBean();
    }
}
```

28. @Profile

1. Activates beans only in specific environments or profiles.
2. Works with @Component, @Service, and @Configuration.
3. Commonly used for dev, test, and prod configurations.
4. Profiles are activated using spring.profiles.active.
5. Simplifies environment-specific bean management.

Example:

```
java
Code :
@Component
@Profile("dev")
public class DevBean { public
    DevBean() {
        System.out.println("DevBean initialized");
    }
}
```

29. @EventListener

1. Registers a method to listen for application events.
2. Simplifies event-driven programming.
3. Supports both custom and predefined events.
4. Works with asynchronous event handling.
5. Replaces traditional ApplicationListener implementation.

Example:

java

Code :

```
@Component
```

```
public class MyEventListener { @  
    EventListener  
    public void handleEvent(ApplicationReadyEvent event) {  
        System.out.println("Application is ready!");  
    }  
}
```

30. @EnableAsync

1. Enables asynchronous method execution.
2. Works with @ Async annotation for methods.
3. Requires a task executor bean configuration.
4. Improves performance for non-blocking operations.
5. Often used for background tasks.

Example:

java

Code :

```
@Configuration  
@EnableAsync
```

```
public class AppConfig {  
}  
  
@Service  
public class AsyncService { @  
    Async  
    public void asyncMethod() {  
        System.out.println("Running asynchronously!");  
    }  
}
```

31. @Async

1. Marks a method for asynchronous execution.
2. Requires `@EnableAsync` in the configuration.
3. Methods annotated with `@Async` will execute in a separate thread.
4. Used for background or time-consuming tasks.
5. Supports custom task executors for fine control.

Example:

```
java  
Code :  
@Service  
  
public class AsyncService { @  
    Async  
    public void executeAsyncTask() {  
        System.out.println("Executing in a separate thread");  
    }  
}
```

32. @EnableScheduling

1. Enables Spring's scheduled task execution capability.
2. Works with @Scheduled annotation.
3. Often used in service-level components.
4. Requires no additional configuration for basic scheduling.
5. Simplifies implementation of periodic or delayed tasks.

Example:

```
java
Code :
@Configuration
@EnableScheduling
public class SchedulerConfig {
}

@Component
public class TaskScheduler {
    @Scheduled(fixedRate = 5000)
    public void scheduledTask() {
        System.out.println("Task executed every 5 seconds");
    }
}
```

33. @Scheduled

1. Configures scheduled tasks for periodic execution.
2. Supports fixed rate, fixed delay, and cron expressions.
3. Requires @EnableScheduling in the configuration.
4. Can run tasks in the background at defined intervals.
5. Simplifies periodic task implementation.

Example:

```
java
Code :
@Component
public class MyTask {
    @Scheduled(cron = "0 0 * * * ?")
    public void runTask() {
        System.out.println("Task runs at the start of every hour");
    }
}
```

34. **@EnableTransactionManagement**

1. Enables annotation-driven transaction management.
2. Works with `@Transactional` for declarative transactions.
3. Automatically manages commit, rollback, and propagation.
4. Often used in data-access layers.
5. Reduces boilerplate code for transaction handling.

Example:

```
java
Code :
@Configuration
@EnableTransactionManagement
public class AppConfig {
```

35. **@Transactional**

1. Manages database transactions at the method or class level.
2. Supports rollback for exceptions by default.
3. Configures transaction propagation and isolation levels.

4. Often used in repository or service layers.
5. Simplifies error handling in database operations.

Example:

java

Code :

```
@Service
```

```
public class TransactionService { @  
    Transactional  
    public void performTransactionalTask() {  
        // Database operations  
    }  
}
```

36. **@RestControllerAdvice**

1. Handles exceptions for REST APIs globally.
2. Combines `@ControllerAdvice` and `@ResponseBody`.
3. Provides centralized error handling for REST controllers.
4. Can define custom exception-handling logic.
5. Simplifies the management of API error responses.

Example:

java

Code :

```
@RestControllerAdvice
```

```
public class GlobalExceptionHandler {  
  
    @ExceptionHandler(RuntimeException.class)  
    public String handleRuntimeException(RuntimeException ex) { return  
        "Error: " + ex.getMessage();  
    }  
}
```

}

37. @SessionAttributes

1. Specifies attributes to store in the session scope.
2. Used in Spring MVC controllers.
3. Helps share attributes between handler methods.
4. Works with @ModelAttribute.
5. Useful for managing user sessions or temporary data.

Example:

java

Code :

```
@Controller
```

```
@SessionAttributes("user") public  
class SessionController {  
    @ModelAttribute("user")  
    public User user() {  
        return new User();  
    }  
}
```

38. @RequestAttribute

1. Binds a request attribute to a method parameter.
2. Useful for passing data across filters and controllers.
3. Simplifies accessing request attributes.
4. Reduces boilerplate code compared to manual extraction.
5. Supports type conversion for parameters.

Example:

java

Code :

```
@Controller
```

```
public class MyController { @  
    GetMapping("/greet")  
    public String greet(@ RequestAttribute("name") String name) { return  
        "Hello, " + name;  
    }  
}
```

39. @EnableJpaRepositories

1. Enables JPA repositories in a Spring Boot application.
2. Automatically detects interfaces extending JpaRepository.
3. Simplifies database interaction with JPA.
4. Configures the base package for scanning repository interfaces.
5. Reduces boilerplate for data access layers.

Example:

```
java  
Code :  
@Configuration  
@ EnableJpaRepositories(basePackages = "com.example.repositories") public class  
JpaConfig {  
}
```

40. @MappedSuperclass

1. Marks a class as a JPA mapped superclass.
2. Provides a base class for JPA entities.
3. Fields in the superclass are mapped to database columns.
4. Cannot be directly instantiated or queried.
5. Simplifies code reuse in JPA entities.

Example:

```
java
Code :
@MappedSuperclass

public abstract class BaseEntity { @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private LocalDateTime createdAt;

}
```

41. @Embedded

1. Marks an attribute in an entity class as an embeddable type.
2. Used for embedding reusable value objects in entities.
3. Simplifies modeling of composite fields like addresses or measurements.
4. Works with the @ Embeddable annotation in the reusable class.
5. Reduces redundancy by reusing components in multiple entities.

Example:

```
java
Code :
@Entity

public class Employee { @
    Id
    @GeneratedValue
    private Long id;
    @Embedded
    private Address address;
```

```
}
```

```
@Embeddable  
public class Address { private  
    String street; private String  
    city; private String zip;  
}
```

42. @Embeddable

1. Marks a class as embeddable for JPA entities.
2. Used with the @Embedded annotation in the parent entity.
3. Fields in the class are mapped as part of the containing entity.
4. Makes code modular and reusable.
5. Useful for representing composite attributes.

Example:

```
java  
Code :  
@Embeddable  
public class Address { private  
    String street; private String  
    city; private String zip;  
}
```

43. @ElementCollection

1. Maps a collection of basic or embeddable types to a database table.
2. Used for lists, sets, or maps of values in entities.

3. Requires no additional table relationships.
4. Automatically maps collections to a join table.
5. Simplifies modeling of multi-valued attributes.

Example:

```
java  
Code :  
@Entity  
  
public class Employee { @  
    Id  
    @GeneratedValue  
    private Long id;  
    @ElementCollection  
    private List<String> skills;  
}
```

44. @Enumerated

1. Specifies how an enumeration should be mapped to the database.
2. Supports `EnumType.ORDINAL` and `EnumType.STRING`.
3. Ensures type-safe handling of enums in JPA.
4. Used with entity fields representing enums.
5. Avoids errors by explicitly defining mapping.

Example:

```
java  
Code :  
@Entity  
  
public class Task { @  
    Id  
    @GeneratedValue
```

```
private Long id;  
@Enumerated(EnumType.STRING) private  
TaskStatus status;  
}
```

```
public enum TaskStatus {  
    PENDING, COMPLETED  
}
```

45. @Query

1. Defines custom JPQL or SQL queries for JPA repositories.
2. Enhances flexibility for complex queries.
3. Used with repository interface methods.
4. Supports parameterized queries with @Param.
5. Simplifies custom data access logic.

Example:

```
java  
Code :  
@Repository  
  
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
    @Query("SELECT e FROM Employee e WHERE e.name = :name") List<Employee>  
    findByName(@Param("name") String name);  
}
```

46. @Modifying

1. Indicates a repository method that performs a modifying query.
2. Works with @Query for update or delete operations.
3. Requires transactional support with @Transactional.

4. Enhances data manipulation capabilities in repositories.
5. Ensures proper handling of non-select queries.

Example:

java

Code :

```
@Repository
```

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
    @Modifying  
    @Query("UPDATE Employee e SET e.salary = :salary WHERE e.id = :id") void  
    updateSalary(@Param("id") Long id, @Param("salary") double  
    salary);  
}
```

47. @GeneratedValue

1. Specifies the generation strategy for primary keys.
2. Supports strategies like AUTO, IDENTITY, SEQUENCE, and TABLE.
3. Works with the @Id annotation in entities.
4. Automatically assigns unique identifiers to entities.
5. Simplifies key generation for database tables.

Example:

java

Code :

```
@Entity
```

```
public class Employee { @  
    Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
}
```

48. @Lob

1. Maps a field to a large object (LOB) in the database.
2. Supports storing large data like text or binary files.
3. Can be used for CLOB (Character LOB) or BLOB (Binary LOB).
4. Works with @Column for additional configurations.
5. Simplifies handling of large data in entities.

Example:

```
java
Code :
@Entity
public class Document { @
  Id
  @GeneratedValue
  private Long id;
  @Lob
  private byte[] content;
}
```

49. @Temporal

1. Maps date/time fields to appropriate SQL types.
2. Supports DATE, TIME, and TIMESTAMP.
3. Works with java.util.Date or java.util.Calendar.
4. Ensures accurate handling of date/time data.
5. Avoids incorrect type mapping in database schema.

Example:

```
java
Code : @
Entity
```

```
public class Event { @  
    Id  
    @GeneratedValue  
    private Long id;  
    @Temporal(TemporalType.TIMESTAMP) private  
    Date eventDate;  
}
```

50. @JsonIgnore

1. Prevents serialization or deserialization of a field in JSON.
2. Used in classes processed by Jackson.
3. Helps exclude sensitive or unnecessary fields from JSON responses.
4. Can be used with bidirectional relationships to avoid infinite loops.
5. Simplifies control over JSON output.

Example:

```
java  
Code :  
@Entity  
public class User {  
    @Id  
    @GeneratedValue  
    private Long id;  
    private String username;  
  
    @JsonIgnore  
    private String password;}
```



<https://www.youtube.com/@codewitharrays>



<https://www.instagram.com/codewitharrays/>



<https://t.me/codewitharrays> Group Link: <https://t.me/ccee2025notes>



[+91 8007592194 +91 9284926333](#)



codewitharrays@gmail.com



<https://codewitharrays.in/project>