JSP

JSP stands for - Java Server Pages.

- **Introduction :-**

It is server side technology that is used to create we applications. It is used to create dynamic web content. JSP consist of both HTML tag & JSP tags.

In this, JSP tags are used to insert java code into HTML pages.

It is an advance version of servlet technology.

- **Advantages :-**

1) easy to maintain.
2) No recompilation / redeployment required.
3) less coding.
4) Has access to entire API of java.

- **JSP Architecture :-**

JSP architecture is a ~~three~~ tier architecture, it has client, server, database.

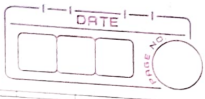
The client is web browser. web server uses a JSP engine.

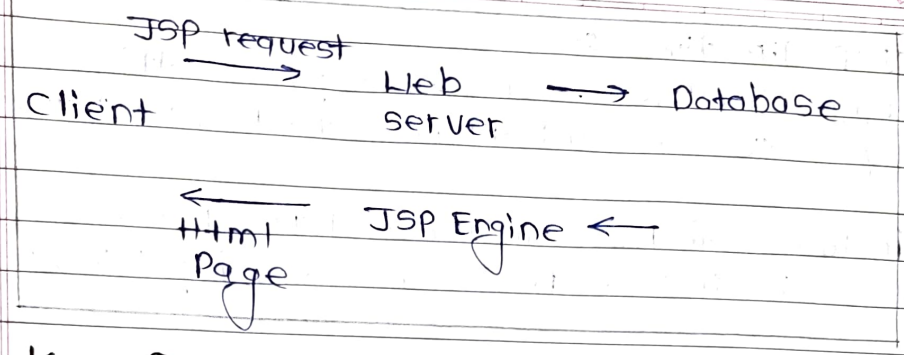
For ex :- Apache has a inbuilt JSP engine.

JSP engine interpret the request for JSP & provide runtime environment for understanding & processing the JSP files.

It reads, passes, build java servelet compile & execute java code & return the html page to client.

Webserver has access to database.





- Key Concepts :-

JSP syntax -

- Directives - control the overall processing of entire JSP pages.
e.g. :- <%@ Page contenttype = "text/html; charset = UTF-8" %>

- Declaration - Declare variable & method.
e.g. :- <% int count = 0; %>

- Scriplets - embed java code within html.
e.g. :- <% int count=0; %>

- Expression - output data to client.
e.g. :- <% = "Hi" %>

- JSP lifecycle :-

1) Translation :- JSP file converted to servlet.
2) Compilation :- Servelet get compiled.
3) Initialization :- Servelet initialized by web contain
4) Execution :- The service() method handles request.
5) Destruction :- The destory() method is called to cleanup.

1) JSP directives :-
1) Page directive - configures page setting.
e.g. :- <%@ page contentType = "text/html" %>

2) include directive - include static file.
e.g. :- <%@ include file = "header.jsp" %>

2) Form handling :-
Jsp handles data / form using :-
request.getparameter ("user");

3) Session Management :-
HttpSession - Stores data across multiple request.
e.g. :- HttpSession s = request.getSession();
s.setAttribute("user", "John");

4) JSP implicit object :-
Available predefined object like :-
request, response, session, application.

5) Error handling :-
Configure error pages using errorpage directive.
<%@ page errorPage = "error.jsp" %>

6) JSP Expression Language :-
EL simplifies accessing data from javaBeans, request, session. e.g. :- ${user.name}

7) Database Connectivity :-
JSP can connect to database using JDBC, through business logic should ideally be in servelet or bean.
e.g. :- Connection con = DriverManager.getConnection (url, user, pass);

JSTL

↳ JSTL :-

- Stands for Java Server pages standored tag library.
- It is collection of useful JSP tag which encapsulate core functionality common to many JSP applications.

JSTL has support for common, structural task such as iteration & conditional tags for manipulating XML, SQL tags.

↳ Classification of JSTL tags

1) Core tags
2) formatting tag
3) SQL tags
4) XML tags
5) JSTL functions.

1) Core tags :-

Syntax to include the JSTL core library in your JSP :-

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

1) tags & Description :-

1) <c:out> .... lik <%= ....> for expression.
2) <c:set> - set result of an expression evaluation in 'scope'.
3) <c:remove> - Remove scoped variable.
4) <c:catch> - catches any throwable that occurs in body & optionally exposes it.
5) <c:if> - simple conditional tag.
6) <c:choose> simple conditional tag named by <when> & <otherwise>.

<c:when>
<c:otherwise>
<c:import>
<c:foreach>
<c:forTokens>
<c:Param>
<c:redirect>
<c:url>

2) Formatting tags :-
Library -
<%@taglib prefix="fmt" url="http://java.sun.com/jsp/jsp/jstl/fmt" %>

1) <fmt:formatNumber>
2) <fmt:parseNumber>
3) <fmt:formatDate>
4) <fmt:parseDate>
5) <fmt:bundle>
6) <fmt:setLocal>
7) <fmt:setBundle>
8) <fmt:timeZone>
9) <fmt:setTimeZone>
10) <fmt:setmessage>

3) SQL tags :-
Library
<%@taglib prefix="s" url="http://java.sun.com/jsp/jsp/jstl/sql" %>
<sql:setDataSource>
<sql:query>
<sql:update>
<sql:param>
<sql:dateparam>

4) XML tags :-

Library :-

<%@ taglib prefix = "x" url = "http://java.sun.com/jstl/jstl/xml" %>

1) <x:out>
2) <x:parse>
3) <x:set>
4) <x:if>
5) <x:foreach>
6) <x:choose>
7) <x:when>
8) <x:otherwise>
9) <x:transform>
10) <x:param>

5) JSTL functions :-

Library -

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

1) fn:contains() :- check whether consist of substring or not.

2) fn:endwith() :- check string ends with specific suffix.

3) fn:indexof() :- returns position of first occurence of string.

4) fn:join() :- Join array element into string with delimeter.

5) fn:length() :- returns string length.

6) fn:replace() :- replace all occurences of substring with another.

7) fn: split() :- Split string in array.

8) fn: startwith() :- check string start with specific suffix.

9) fn: substring() :- extract substring.

10) fn: toLowerCase() :- Convert to lowercase.

11) fn: upperCase() :- Convert to uppercase.

12) fn: trim() :- remove whitespaces from begining & end of string.

Introduction to Spring

- **Introduction to Spring Framework :-**
Spring is a powerful and widely used framework for building enterprise-grade java application. It offers various modules to simplify the development of java application, particularly focusing on building web applications, REST APIs, microservice.

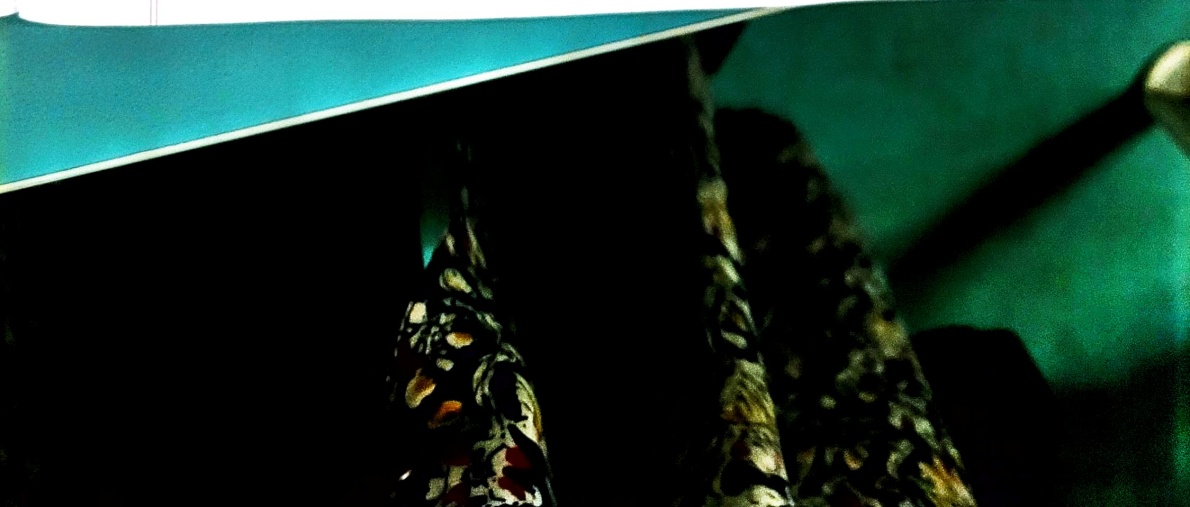
- **Key Concepts of Spring Framework :-**

1) **Inversion of control (IoC) / Dependency Injection**
Spring's core feature is the IoC container, which is responsible for managing object creation and wiring dependencies. Instead of creating objects manually, Spring handles it via DI, improving testability and maintenance.

2) **Aspect-Oriented programming (AOP) :-**
AoP helps separate cross-cutting concerns like logging, security and transaction management from the business logic.

3) **Spring Modules :-**

1) Spring Core :- Provides DI and IoC container.
2) Spring MVC :- For building web applications with a model-view-controller architecture.
3) Spring Boot :- simplifies Spring application development by providing pre-configured settings and embedded servers.
4) Spring Data :- Provides a consistent and easy way to handle database operations.
5) Spring security :- For security in web application.
6) Spring cloud :- To build scalable and resilient microservices.



- **Steps to implement Spring framework :-**

1) Step 1 : Install Required Tools
   - JDK :- Download and install the Java Development Kit (JDK)
   - IDE :- Install an IDE such as Spring Tool Suite (STS) or Eclipse.
   - Maven / Gradle :- Spring projects typically use Maven or Gradle for dependency management.

2) Step 2 : Create a Spring project.
   - open your IDE
   - Create a new Spring Starter Project or use Spring Initializer to bootstrap a Spring project with Maven or Gradle.

Example using Spring Initalizer :-
1) Go to spring Initializer
2) Select your project type, java version, dependencies.
3) Generate the project, download the .zip file, and extract it.
4) open the project in your IDE.

3) Step 3 : Add dependencies
The dependencies are mangged in the pom.xml file

4) step 4 : Define a Configuration Class
In spring, you can use java-based configuration instead of XML configuration. The @configuration annotation indicates that this class contains spring configuration settings.

5) Step 5 : Create a Controller
A controller handles the user's requests and returns appropriate views or responses. Use @Controller or @RestController annotations to create a controller.

6) Step 6 : Create a View
For spring MVC applications, create view templates using Thymeleaf or JSP.

7) Step 7 : Configure Application Properties
Spring Boot uses application.properties or application.yml files for configuration, such as database settings, server port etc.

8) Step 8 : Running the Spring Application.
The spring Boot application is typically started using the @SpringBootApplication annotation, which serves as a combination of @Configuration, @EnableAutoConfiguration and @ComponentScan.

9) Build and Run Application
- In the IDE : Right-click on the project and choose Run as → Spring Boot App.
- Command Line : Navigate to project directory and run

10) Access Your Application
- open your browser and go to http://localhost:8080 to view the spring MVC web page.
- For REST endpoints, access API URLs like http://localhost:8080/api/greeting.

SpringBoot :-

Bean Lifecycle :- Bean lifecycle is mangged by the spring container.

- When we run the program then first of all, the spring container gets started.
- After that, the container creates the instance of a bean as per the request and then dependencies are injected.
- And finally, the bean is destroyed when the spring container is closed.
- Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom init() method and the destroy() method.

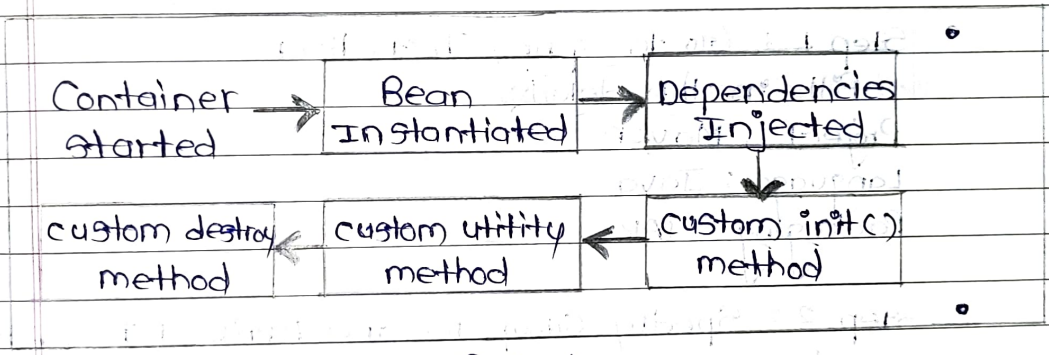
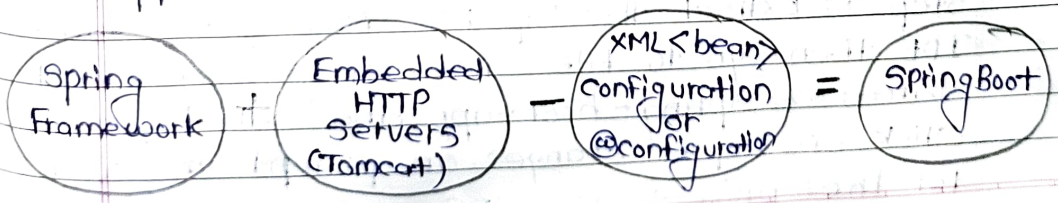
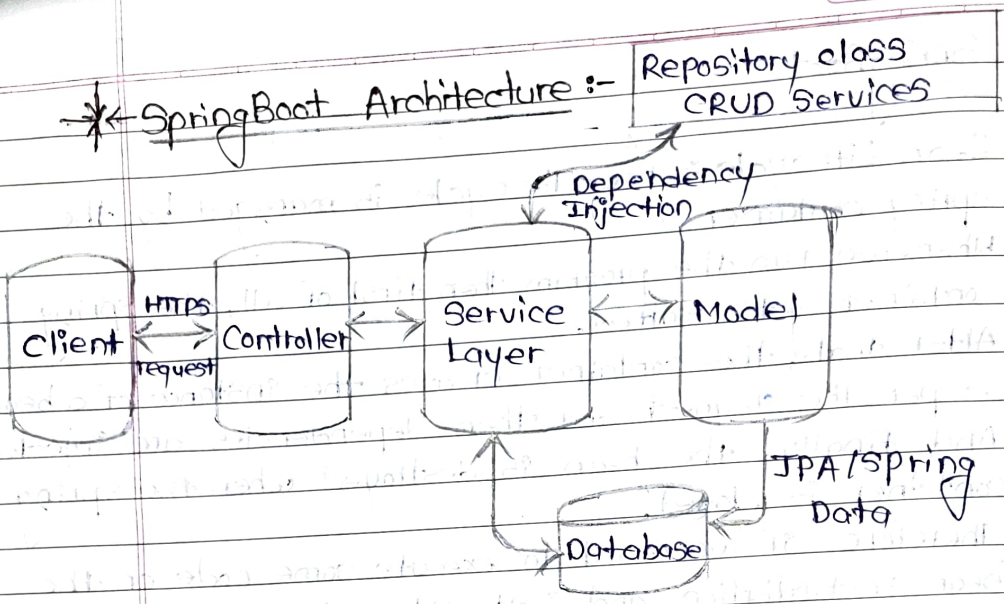


fig: Bean LifeCycle

SpringBoot :- SpringBoot is the most popular java framework that is used for developing RESTful we application.



✳ SpringBoot Architecture :-



✳ Steps to create SpringBoot Application.

- Step 1 : Go to Spring Initializer
  Fill in the details
  Project : Maven
  Language : Java
  Packaging : JAR

- Step 2 : Specify Group Id and Artifact Id. Here we have specified Group name as "com.gfg" and the Artifact Id as "SpringBootApp".

- Step 3 : Click on generate which will download the starter project.

- Step 4 : Extract the zip file. Now open a suitable IDE then go to File → New → Project from existing Sources → Spring-boot-app and select pom.xml Click on import changes on prompt and wait for the project to sync.

- Step 5 : Go to src → main → java → com.gfg.SpringBoot.app, Create a java class with name as controller and add the annotation @RestController. Now create Get API

- Step 6 : This application is now ready to run. Run the SpringBootAppApplication class and wait for tomcat server to start.

- Step 7 : Now go to the browser and enter the URL localhost:8080. observe the output and now do the same for localhost:8080/gfg

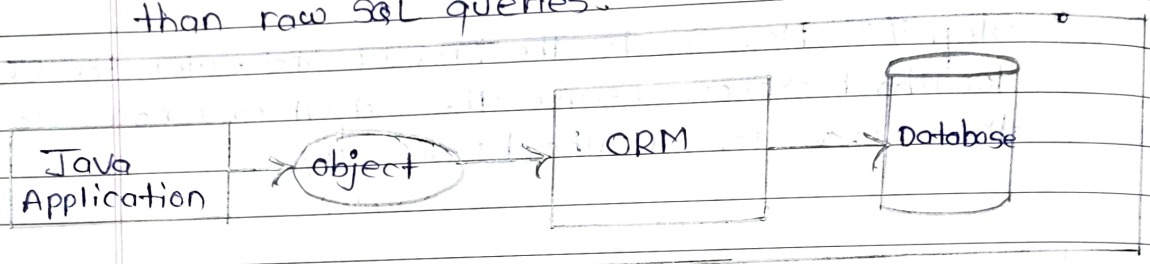
Advantages :-

- It creates stand-alone Spring applications that can be started using java -jar.
- It tests web applications easily with the help of different embedded HTTP servers such as Tomcat.
- It provides opinionated 'starter' POMs to simplify our "Maven" configuration.
- There is no requirement for XML configuration.

Autowiring :- Autowiring in the spring framework can inject dependencies automatically. The spring container detects those dependencies specified in the configuration file and the relationship bet the beans.
Modes of Autowiring :- No, byName, byType, constructor, Autodetect.

→ Hibernate :- Hibernate is an object-relational mapping (ORM) framework for java, which simplifies the interaction between an application and its database by mapping java objects to database tables.

- ORM (Object-Relational Mapping) :- Hiberate maps java classes to database tables, making it easier to work with data by using objects rather than raw SQL queries.



- Java Persistence API (JPA) :-
The Java Persistence API is a specification for managing relational data in Java applications. It simplifies the process of persisting, retrieving and managing data from a relational database using java objects.

→ Steps to build a Hibernate Application

1) Setup Development Environment
   - Install JDK
   - Eclipse IDE
   - MYSQL or any database relational

2) Create a New Maven Project
   - Open your IDE and create a new Maven project
   - In the pom.xml file, add dependencies for Hibernate, MySQL.

3) Create the Hibernate Configuration file
- The Hibernate configuration file is where you specify database connection details and other Hibernate-specific properties.
- Create a hibernate.cfg.xml file in the src/main/resources directory.

4) Create the Model Class (Entity)
- Create a simple model class that will map to a database table.

5) Create the Hibernate Utility class (SessionFactory)
- To simplify getting a sessionfactory, you can create a utility class that initializes hibernate and provides a session factory.

6) Perform CRUD operations
- Now, you can write code to perform CRUD (Create, Read, Update, Delete) operations on the database using Hibernate.

Generator classes in Hibernate.

- The <generator> class is a sub-element of id. It is used to generate the unique identifier for the objects of persistent class.
- All the generator classes implements the org.hibernate.id.IdentifierGenerator Interface.

Hibernate Annotations :-

1) @Entity :- Marks a class as a Hibernate entity.

2) @Table :- Maps a class to a specific database table.

3) @Id :- Marks the primary key of the entity.

4) @GeneratedValue :- Configures the auto-generation of primary key values.

5) @Column :- Maps a class field to a table column.

6) @OneToMany, @ManyToOne, @ManyToMany :- Define relationships between entities.

Hibernate Query Language (HQL) :-

- HQL (Hibernate Query Language) is an object-oriented query language similar to SQL but operates on objects rather than tables.
- It works on the entity model.

RESTful WebServices :-

- **Introduction to RESTful Web Services :-**

- REST (Representational State Transfer) is an architectural style for designing networked applications.

- RESTful Services expose resources in a standardized way using URIs (Uniform Resource Identifiers). These services are stateless and rely on HTTP as the protocol for communication.

- **Key Concepts of REST :-**

1) Resources :- Everything in REST is considered a resource. Resources can be any object, data or service that can be accessed over the network.

- Each resource is identified by a unique URI (Uniform Resource Identifier)

2) HTTP Methods :-
RESTful services use standard HTTP methods to perform operations on resources:
1) GET :- Retrieve a resource or collection of resources.
2) POST :- Create a new resource.
3) PUT :- Update an existing resource.
4) DELETE :- Remove a resource.

3) Statelessness :- Each request from a client to a server must contain all the information needed to understand and process the request.

- The server does not store any session information about the client.

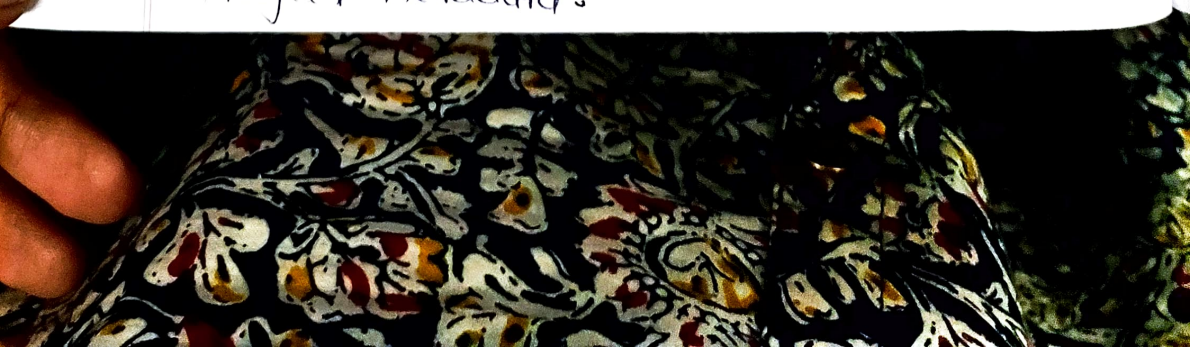
4) Representation :- Resources can have multiple representations (e.g. JSON, XML). The client specifies the desired representation via the 'Accept' header in the HTTP request.

5) Client-Server Architecture :- The client and server are seperate entities. The client handles the user interface, while the server manages data and business logic.

* Implementing RESTful web services using SpringBoot.

1) Step 1 : Set up your development environment
   - Install Java : Ensure you have JDK 8.
   - Install Maven : Download and install Apach Maven.
   - IDE : Use an IDE like Eclipse, Spring Tool Suite.

2) Step 2 : Create a SpringBoot project
   - Go to [Spring Initializer] (https://start.spring.io)
   - choose your project metadata :
     1) Project : Maven project
     2) Language : Java
     3) Project Metadata :



- Group : 'com.example'
- Artifact : 'demo'
- Name : 'demo'
- Project Name : 'com.example.demo'

- Add Dependencies :
- Spring Web
- Spring Data JPA

3) Step 3 : Open the project in Your IDE
- open your IDE and import the Maven project.
- Wait for dependencies to download.

4) Step 4 : Create the Model Class :
create a model class that represents the resource.

5) Step 5 : Create the Repository Interface.
Create a repository interface to interact with the database.

6) Step 6 : Create the Service Layer
create a service class to handle business logic.

7) Step 7 : create the Controller
create a controller to handle HTTP requests.

8) Step 8 : Configure Application Properties.
Set up database in 'src/main/resources/application.properties'.

9) Step 9 : Run Your Application.