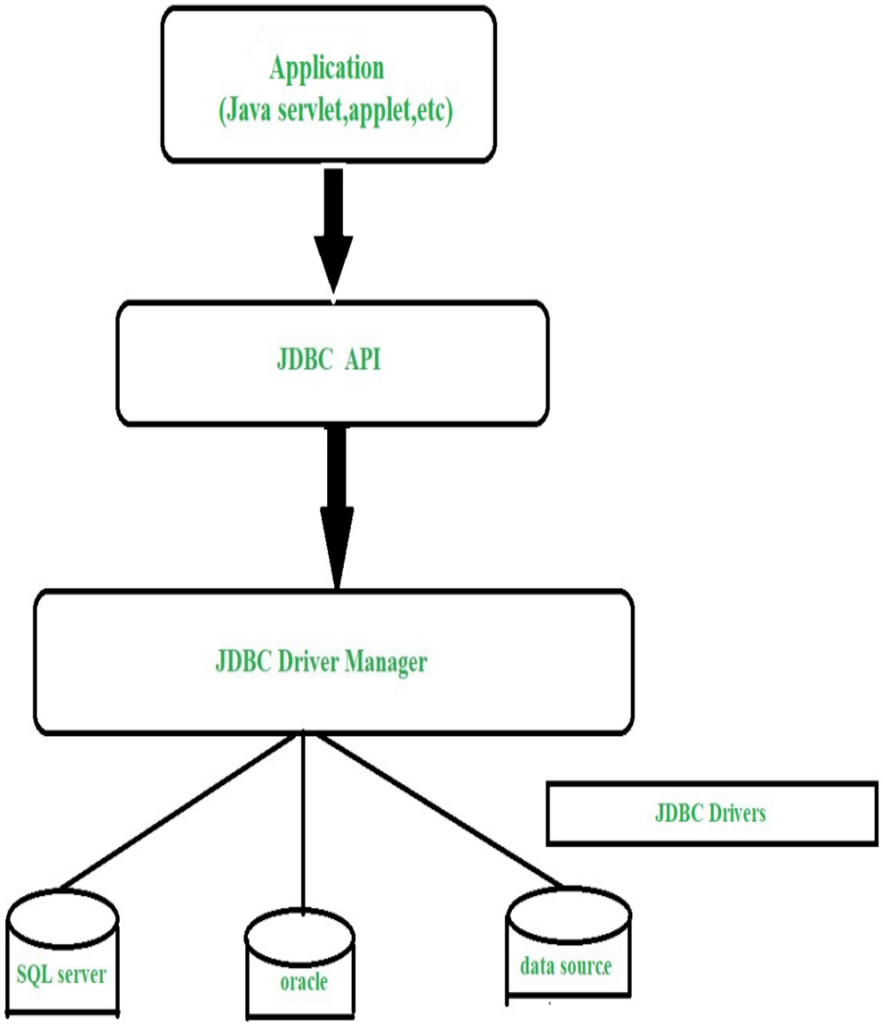
Unit 1 (JDBC)

Structure of JDBC

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets.



1. **Application:** It is a java applet or a servlet that communicates with a data source.
2. **The JDBC API:** The JDBC API allows Java programs to execute SQL statements and retrieve results. Some of the important interfaces defined in JDBC API are as follows: Driver interface , ResultSet Interface , RowSet Interface , PreparedStatement interface, Connection inteface, and Classes defined in JDBC API are as follows: DriverManager class, Types class, Blob class, clob class.
3. **DriverManager:** It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.
4. **JDBC drivers:** To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

JDBC drivers

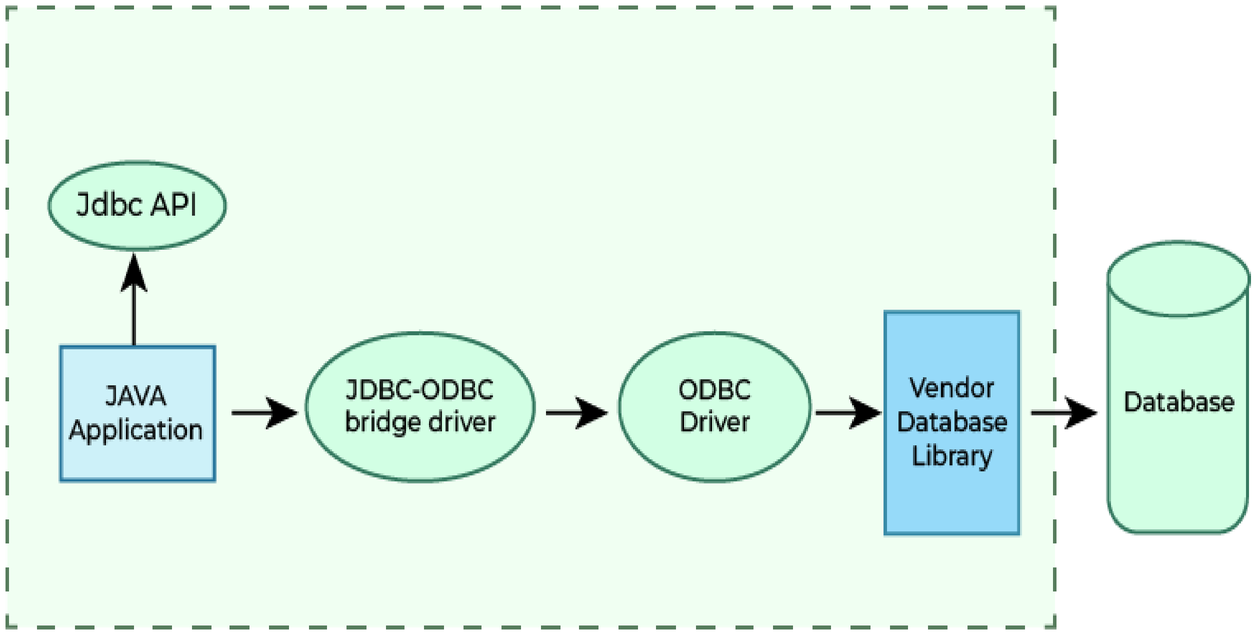
JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

JDBC drivers are the software components which implements interfaces in JDBC APIs to enable java application to interact with the database.

There are four types of JDBC drivers:

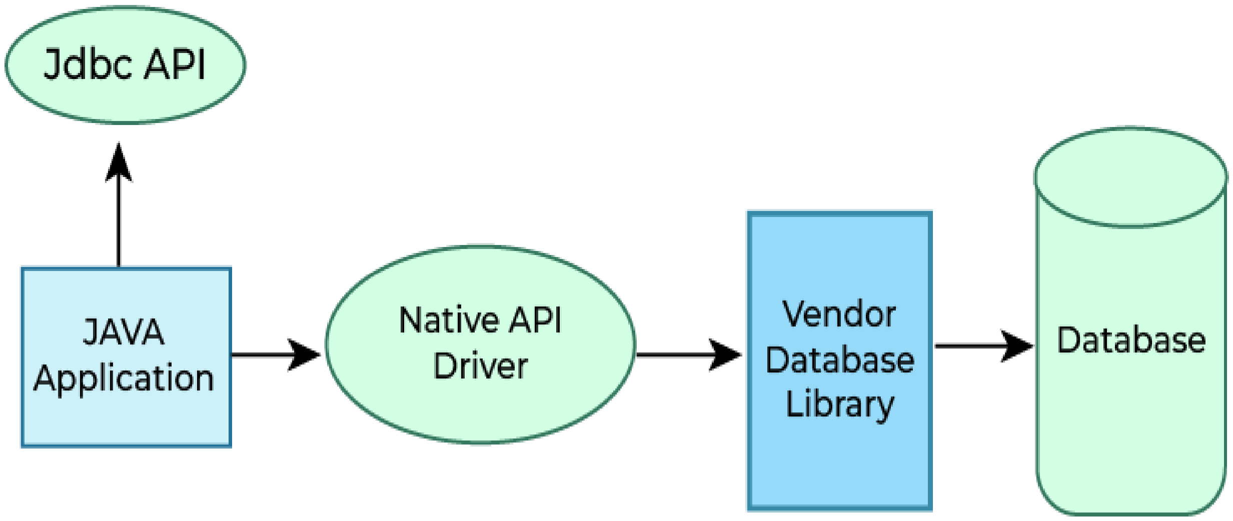
Type-1 driver:

This driver acts as a bridge between JDBC and ODBC (Open Database Connectivity). It converts JDBC method calls into ODBC calls and sends them to the ODBC driver.



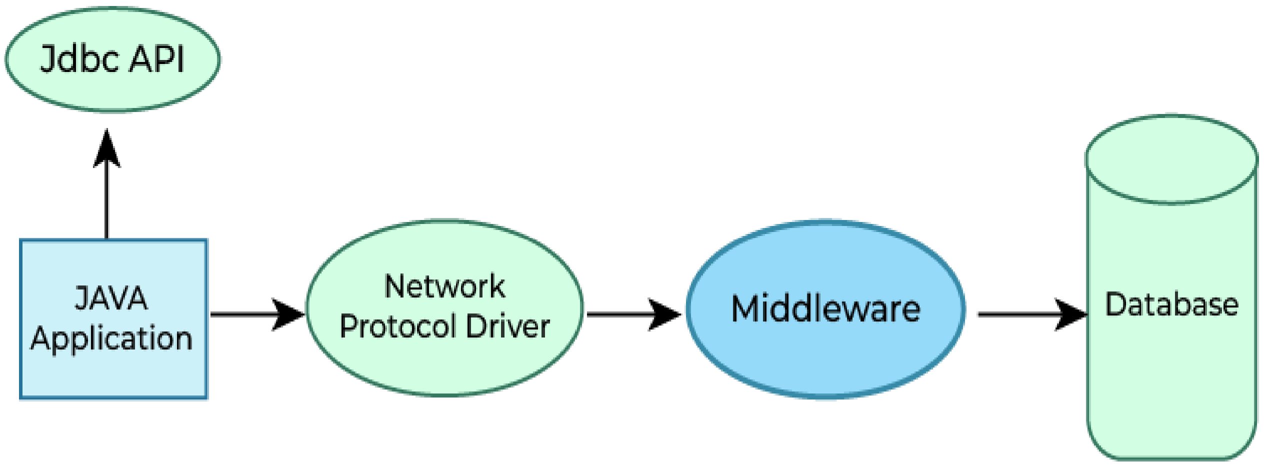
Type-2 driver:

This driver converts JDBC calls into database-specific calls using native libraries provided by the database vendor. It is also known as a partially Java driver because it relies on native code.



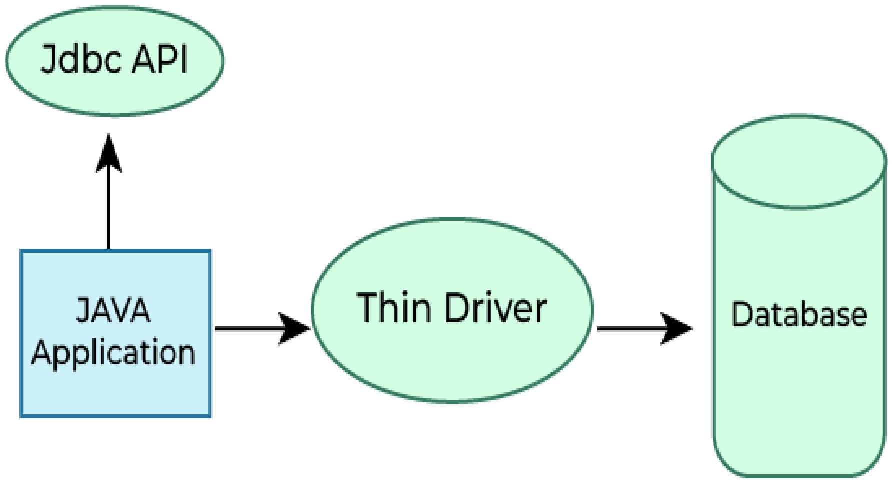
Type-3 driver:

This driver uses a middleware server that translates JDBC calls to the database-specific protocol. The client sends JDBC calls to the middleware server, which then communicates with the database.



Type-4 driver:

This driver converts JDBC calls directly into the database-specific protocol using Java. It is known as a pure Java driver because it does not require any native libraries or middleware.


Jdbc API
JAVA Application
Thin Driver
Database

Unit 2 (Servlet)

Servlet

1. A Servlet in Java is a small program that runs on a server and handles requests from web clients, typically through a web browser.
2. Servlets run on the server, not on the user's computer.
3. When a user interacts with a website (like clicking a button), their request is sent to the server, and a servlet processes that request.
4. Servlets are used to create dynamic web content.
5. For example, if you log into a website, a servlet might be responsible for checking your username and password against a database and then displaying your account information.
6. Servlets often work with the HTTP protocol, meaning they handle requests like GET (to retrieve data) and POST (to send data) from a web client.

Servlet life-cycle

1. The Servlet life-cycle is managed by the servlet container like Apache Tomcat.
2. The servlet container loads the servlet class when it is first requested by a client or during server startup if the servlet is configured to load at startup.
3. After loading the class, the servlet container creates an instance of the servlet by calling its no-argument constructor.
4. The init method is called once when the servlet instance is created. This method is used to perform any initialization tasks, such as setting up database connections or reading configuration settings.
5. After initialization, the servlet is ready to handle client requests. Each time a request is made to the servlet, the servlet container calls the service method.
6. When the servlet is no longer needed, or the server is shutting down, the servlet container calls the destroy method.
7. The destroy method is used to release any resources that were allocated during the servlet's lifecycle, such as closing database connections or freeing up memory.

Session

1. Session is used to save user information momentarily on the server.
2. It starts from the instance the user logs into the application and remains till the user logs out of the application or shuts down the machine.
3. In both cases, the session values are deleted automatically.
4. Hence, it functions as a temporary storage that can be accessed till the user is active in the application and can be accessed when the user requests a URI.
5. This session is stored in binary values, hence maintaining the security aspect, and can be decoded only at the server end.
6. Websites may use JavaScript or pixels as tracking mechanisms to monitor the user’s activity enhancing user experience and analytics.

Creating binding and deleting session

```
HttpSession session = request.getSession(true); // creates a new
session if one does not exist

session.setAttribute("user", userObject); // where "user" is the key
and userObject is the value being stored
session.setAttribute("cart", shoppingCart); // example of binding a
shopping cart object to the session

User user = (User) session.getAttribute("user"); // retrieving the
user object from the session
ShoppingCart cart = (ShoppingCart) session.getAttribute("cart"); //
retrieving the shopping cart object

session.removeAttribute("user"); // removes the "user" object from the
session
```

Session Tracking

1. Session tracking is the process of remembering and documenting customer conversations over time. Session management is another name for it. The HTTP

protocol is stateless, we require Session Tracking to make the client-server relationship stateful.

Session tracking is important for tracking conversions in online shopping, mailing applications, and E-Commerce applications.

There are four types of session tracking:

1. Cookies: Cookies are small pieces of data stored on the client's browser. The server sends a cookie to the client, and the client sends it back with each subsequent request.

2. URL rewriting: URL rewriting involves appending the session ID to the URL as a query parameter. This method is used when cookies are disabled.

3. Hidden Form Fields: This method involves embedding the session ID in hidden form fields within HTML forms.

4. HTTP Session Tracking (Server-Side) :This is the standard method used by most web frameworks and servlet containers. It involves storing session data on the server side and maintaining a session ID on the client side.

Communication of servlets in a web application

Java servlets can communicate with each other by passing a response object and user requests to another servlet. Here are some ways servlets can communicate:

1. RequestDispatcher:

The include() method calls a servlet using its URI and waits for its return before continuing. This method can be called multiple times within a servlet.

2. HttpServletResponse:

The sendRedirect method facilitates communication with other servlets.

3. ServletContext:

The setAttribute and getAttribute methods allow servlets to share information.

4. HttpSession:

Methods within this interface allow for maintaining state between transactions.

5. Sockets:

Applets and servlets can communicate interactively by sending messages back and forth using the same socket or multiple sockets. However, this method doesn't work for applets running behind firewalls.

Apache Tomcat

1. Apache Tomcat is a popular open-source web server and servlet container that implements Java Servlet, JavaServer Pages (JSP), and Java Expression Language (EL) technologies.
2. It’s widely used for hosting Java-based web applications.
3. Servlet containers like Tomcat provide a runtime environment for Java servlets, which extend web server functionality by generating dynamic content and handling web requests.
4. They manage the lifecycle of servlets, ensuring smooth operation by loading and unloading them as needed.
5. Java servlet containers may be combined with other web servers to provide a more complete runtime environment for deploying Java-based web applications.

6. Apache Tomcat also provides additional services such as security and resource management.
7. Tomcat is not a full-featured web server or application server, therefore it has limited enterprise-level features.

Unit 3 (JSP)

JSP

1. In Java, JSP stands for Jakarta Server Pages((JSP; formerly JavaServer Pages)).
2. It is a server-side technology that is used for creating web applications.
3. It is used to create dynamic web content.
4. JSP consists of both HTML tags and JSP tags.
5. In this, JSP tags are used to insert JAVA code into HTML pages.
6. It is an advanced version of Servlet Technology.
7. In this, Java code can be inserted in HTML/ XML pages or both.
8. JSP is first converted into a servlet by the JSP container before processing the client's request.
9. JSP has various features like JSP Expressions, JSP tags, JSP Expression Language, etc.

Uses of JSP

1. JSP allows for the creation of web pages that can display content dynamically based on user interactions, database queries, or other input.
2. JSP can interact with databases using Java Database Connectivity (JDBC) to retrieve, update, and display data.
3. JSP can handle user input from HTML forms, process it on the server-side, and generate appropriate responses. This can include login forms, registration forms and feedback forms.
4. JSP can be used to build content management systems where administrators can add, update, or delete content dynamically without having to edit the HTML code directly. This is beneficial for blogs and news website.
5. JSP integrates seamlessly with other Java technologies, making it a versatile choice for building complex web applications.
6. JSP pages can be used to define custom error pages that handle exceptions and display user-friendly error messages.

Implicit objects in JSP

In Java Server Pages (JSP), implicit objects are predefined variables that provide access to various elements of the web application environment without requiring explicit declaration.

These objects are automatically available in every JSP page, simplifying tasks like handling requests, responses, and session data.

Key implicit objects include:

- request: Represents the HttpServletRequest object, used to retrieve client data like form inputs and request parameters.
- response: Represents the HttpServletResponse object, used to send data back to the client.
- session: Represents the HttpSession object, used to manage user session data.
- application: Represents the ServletContext object, allowing access to application-wide parameters.
- out: Represents the JspWriter object, used to send output to the client.
- pageContext: Provides access to various scoped attributes.

JSP actions

JSP actions are special XML-like tags used in JavaServer Pages (JSP) to perform specific tasks such as including other resources, forwarding requests, and working with JavaBeans. These actions are processed on the server side and help in creating dynamic web content without writing complex Java code directly in the JSP file.

```
<jsp:include page="header.jsp" />
// Includes content from another resource (like a JSP or HTML file)
at runtime.
```

```
<jsp:getProperty name="user" property="username"/>

//It/ retrieves the value of a property from a JavaBean and inserts it
into the output.
```

```
<jsp:setProperty>
//It/ sets the value of a JavaBean property.
```

JSP lifecycle

1. Compilation of JSP page:

Here the generated java servlet file (*test.java/*) is compiled to a class file (test.class).

2. Classloading:
The classloader loads the Java class file into the memory. The loaded Java class can then be used to serve incoming requests for the JSP page.

3. Instantiation:
Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.

4. Initialization: jspInit() method is called only once during the life cycle immediately after the generation of the Servlet instance from JSP.

5. Request processing:
_jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.

6. JSP Cleanup: In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used.

Runtime Exception handling

1. The `try-catch` block is the primary way to handle runtime exceptions. If a runtime exception occurs within the `try` block, the `catch` block is executed.

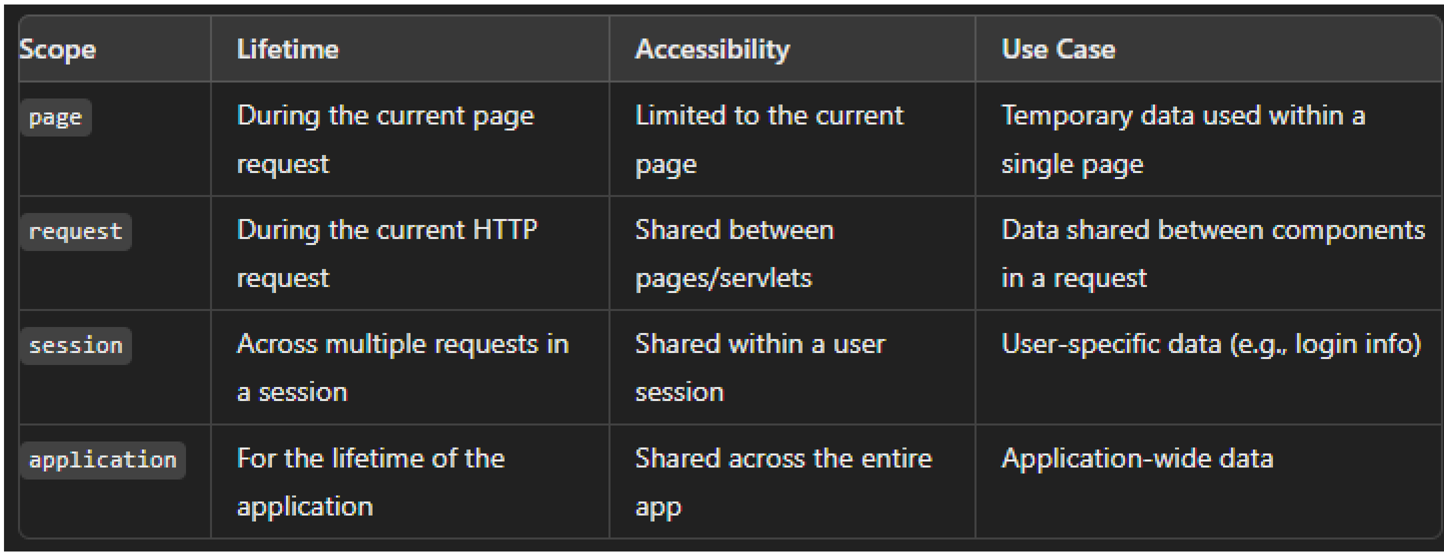
```
try {
    int result = 10 / 0;  // This will throw ArithmeticException
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero: " + e.getMessage());
}
```

2. The `finally` block is used in conjunction with `try-catch` to execute code regardless of whether an exception is thrown or not. It is typically used to release resources (like closing a file or database connection).

```
try {
    int[] numbers = {1, 2, 3};
    System.out.println(numbers[5]);  // ArrayIndexOutOfBoundsException
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Index out of bounds: " + e.getMessage());
} finally {
    System.out.println("This block is always executed.");
}
```
3. You can explicitly throw a runtime exception using the `throw` keyword. This is useful if you want to handle certain conditions manually.
```
public int divide(int a, int b) {
    if (b == 0) {
        throw new ArithmeticException("Cannot divide by zero");
    }
    return a / b;
}
```

Scope values for tag

In JSP, the `<jsp:useBean>` action tag is used to instantiate or reference a JavaBean. When working with JavaBeans in JSP, you can define the **scope** of the bean, which determines how long the bean is available and where it can be accessed.



Scope	Lifetime	Accessibility	Use Case
`page`	During the current page request	Limited to the current page	Temporary data used within a single page
`request`	During the current HTTP request	Shared between pages/servlets	Data shared between components in a request
`session`	Across multiple requests in a session	Shared within a user session	User-specific data (e.g., login info)
`application`	For the lifetime of the application	Shared across the entire app	Application-wide data

---

JSP page Directives

In JSP (JavaServer Pages), page directives are used to provide instructions or configure settings for the entire JSP page. A page directive applies to the entire JSP page, affecting how the JSP is processed by the JSP engine.

1. 'language'

Description: Specifies the programming language used in the JSP page. Typically, this is set to "java" since JSP uses Java as its scripting language.

Default Value: java

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

2. 'extends'
Description: Specifies the superclass of the servlet that will be generated from the JSP page. By default, the servlet generated from a JSP page extends the class `HttpServlet`.
Usage: Customizes the servlet class inheritance for advanced use cases.
```
<%@ page extends="com.example.MyCustomServlet" %>
```
3. '`import`'
Description: Specifies the Java packages or classes that are imported and available for use in the JSP page. It is equivalent to the `import` statement in Java.
```
<%@ page import="java.util.*, java.io/.*" %>
```

Client-side and server-side validation

Client-side and server-side validation are both ways to check if data entered into a web form is correct:

1. Client-side validation

This happens in the browser and provides immediate feedback to the user.

It's primarily a performance optimization feature, but it can be easily bypassed if not used with server-side validation.

2. Server-side validation

This happens on the web server and ensures that all data is validated correctly.

It's an important part of security and data handling.

For maximum security and usability, it's recommended to use both client-side and server-side validation.

For example, a user registration form might use client-side criteria to determine if a username is valid, but server-side validation to check if the username is unique.

Preventing Automatic creation of session in a JSP page

In JSP, a session is automatically created when a user accesses a page unless explicitly disabled. To prevent the automatic creation of a session in a JSP page, you can use the `session` attribute of the `<%@ page %>` directive and set it to `false`.

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

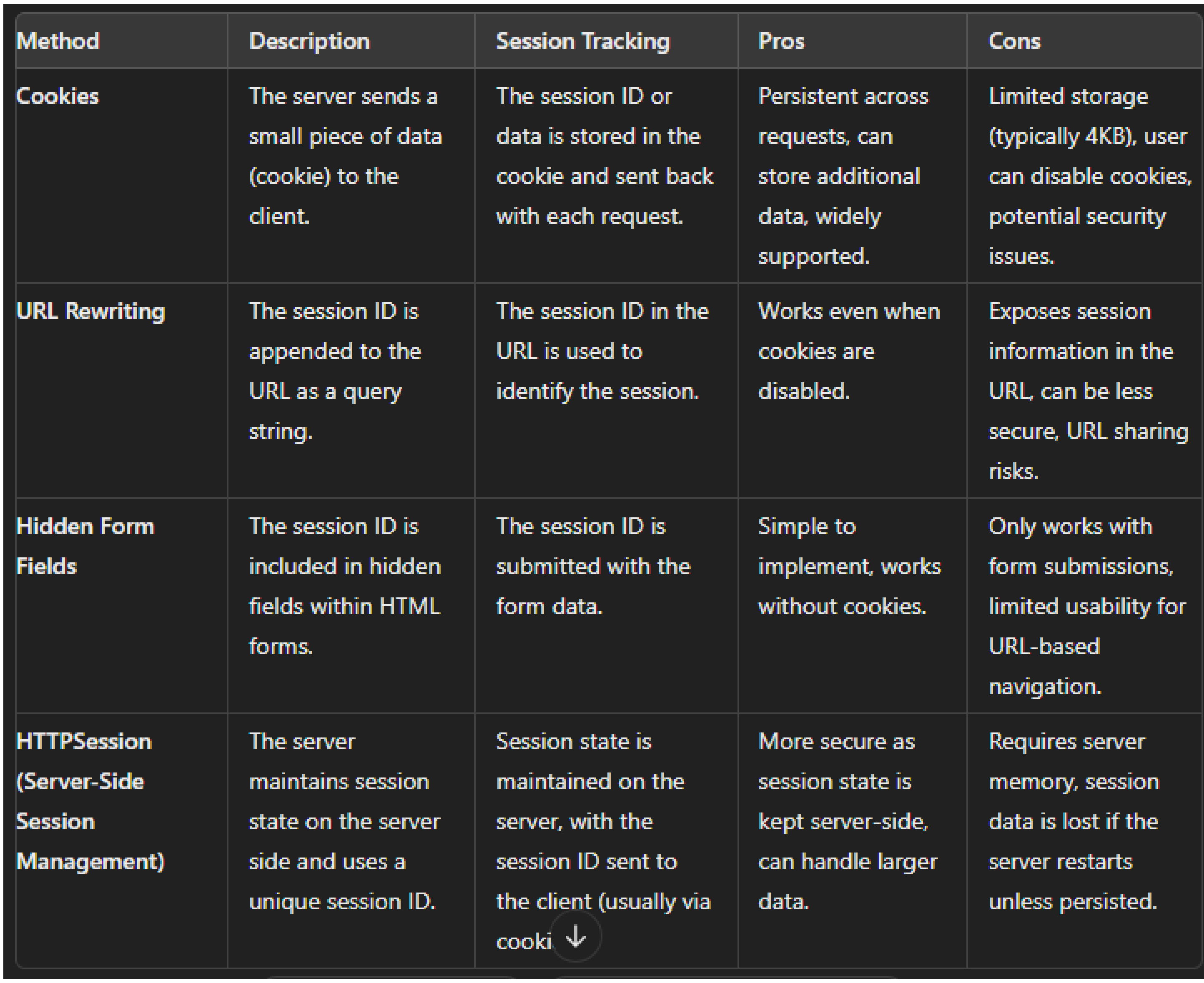
---

jspDestroy() method

1. The jspDestroy() method in JSP is part of the JSP lifecycle and is called when a JSP page is about to be destroyed or removed from memory by the JSP container.
2. This method is analogous to the destroy() method in a servlet and is used to perform cleanup activities before the JSP is unloaded.
3. It is the final method that gets called in the JSP lifecycle.
4. After jspDestroy() is executed, the JSP instance is removed from memory.
5. The jspDestroy() method is used for cleanup tasks.
6. For example, it can be used to close any open resources like database connections, file handles, network connections, or to release other allocated resources.
7. You do not have to explicitly call jspDestroy().
8. The JSP container automatically invokes it when the JSP page is about to be destroyed.

---

session maintaining between client and server



Method	Description	Session Tracking	Pros	Cons
Cookies	The server sends a small piece of data (cookie) to the client.	The session ID or data is stored in the cookie and sent back with each request.	Persistent across requests, can store additional data, widely supported.	Limited storage (typically 4KB), user can disable cookies, potential security issues.
URL Rewriting	The session ID is appended to the URL as a query string.	The session ID in the URL is used to identify the session.	Works even when cookies are disabled.	Exposes session information in the URL, can be less secure, URL sharing risks.
Hidden Form Fields	The session ID is included in hidden fields within HTML forms.	The session ID is submitted with the form data.	Simple to implement, works without cookies.	Only works with form submissions, limited usability for URL-based navigation.
HTTPSession (Server-Side Session Management)	The server maintains session state on the server side and uses a unique session ID.	Session state is maintained on the server, with the session ID sent to the client (usually via cooki	More secure as session state is kept server-side, can handle larger data.	Requires server memory, session data is lost if the server restarts unless persisted.

Exception handling methods in JSP.

1. `try-catch` **block**: You can use the `try-catch` block to catch and handle exceptions within the JSP scriptlets.

```
<%
```

```
try {
    // Your code here
} catch (Exception e) {
    // Handle exception
}
%>
```

2. **`errorPage` attribute in `@page` directive**: Define an error page for the JSP using the `errorPage` attribute.

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

3. **`isErrorPage` attribute in `@page` directive**: On the error page, use the `isErrorPage` attribute to access the exception object.

```
<%@ page isErrorPage="true" %>
<%= exception.getMessage() %>
```

4. **Custom error handling using web.xml**: Define custom error handling in the `web.xml` file by specifying exception types or HTTP error codes.

```
<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/error.jsp</location>
</error-page>
```

5. **Using `Throwable` object**: On an error page, if `isErrorPage="true"`, you can use the implicit `exception` object, which is of type `Throwable`, to handle exceptions.

```
<%= exception.getClass().getName() %>
```

Unit 4 (Hibernate)

Hibernate

1. Hibernate in Java is an open-source Object-Relational Mapping (ORM) framework that simplifies the interaction between Java applications and relational databases.
2. Hibernate allows you to represent your database tables as Java classes and objects, enabling you to manipulate data using object-oriented concepts instead of writing complex SQL queries.
3. It hides the complexities of SQL and JDBC (Java Database Connectivity), making database operations easier to manage and maintain.
4. Hibernate takes care of storing and retrieving objects from the database, managing transactions, and ensuring data integrity.
5. It works with a wide range of databases like MySQL, PostgreSQL, Oracle, and others.
6. Writing less SQL code and working with objects simplifies development and speeds up the development process.
7. You can easily switch between different database systems without changing your application code significantly.
8. Hibernate provides caching mechanisms to improve application performance by reducing the number of database queries.

Hibernate Architecture

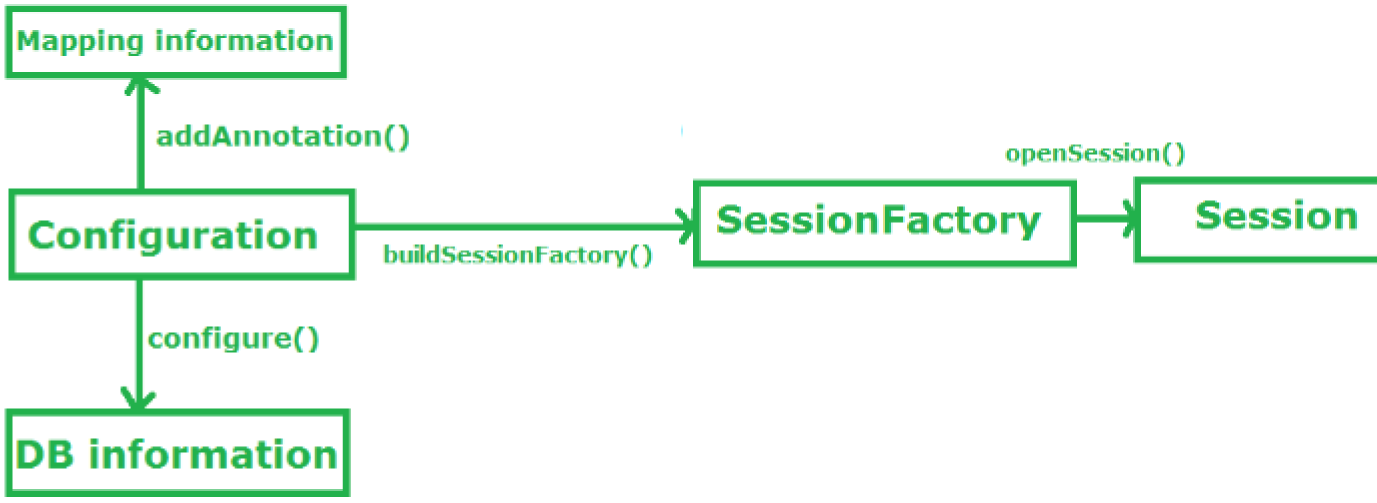


Fig: Hibernate Architecture

1. Configuration:

- Configuration is a class which activates Hibernate framework.
- It reads both configuration file and mapping files.
- It checks whether the config file is syntactically correct or not.

2. SessionFactory:

- SessionFactory is an Interface which is present in org.hibernate package and it is used to create Session Object.
- It is immutable and thread-safe in nature.

3. Session:

- Session is an interface which is present in org.hibernate package.
- Session object is created based upon SessionFactory.
- It opens the Connection/Session with Database software through Hibernate Framework.

4. openSession()

- openSession() is a method provided by the SessionFactory that creates and returns a new Session instance.
- This session is not bound to any transaction or context and is independent of any ongoing transactions in the application.

5. Transaction:

- Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.

6. Query:

- Query is an interface that present inside org.hibernate package.
- This interface exposes some extra functionality beyond that provided by Session.iterate() and Session.find():

7. Criteria:

- Criteria is a simplified API for retrieving entities by composing Criterion objects.

Hibernate Query Language (HQL) Clauses

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 queries are translated by Hibernate into conventional SQL queries, which in turns perform action on database.

1. FROM Clause: To load a whole persistent object into memory, the FROM clause is used.

```
String hib = "FROM Student";

Query query = session.createQuery(hib);
List results = query.list();
```

2. SELECT Clause: The SELECT clause is used when only a few attributes of an object are required rather than the entire object.

```
String hib = "SELECT S.roll FROM Student S";

Query query = session.createQuery(hib);
List results = query.list();
```

3. WHERE Clause: Filtering records is done with the WHERE clause. It's used to retrieve only the records that meet a set of criteria.

```
String hib = "FROM Student S WHERE S.id = 5";

Query query = session.createQuery(hib);
List results = query.list();
```

4. ORDER BY Clause: The ORDER BY clause is used to sort the results of an HQL query.

```
String hib = "FROM Student S WHERE S.id > 5 ORDER BY S.id DESC";

Query query = session.createQuery(hib);
List results = query.list();
```

Unit 5 (Spring Core)

Spring Framework

1. The Spring Framework is a powerful, feature-rich Java platform used to build enterprise-level applications.
2. This framework uses various new techniques such as Aspect-Oriented Programming (AOP), Plain Old Java Object (POJO), and dependency injection (DI), to develop enterprise applications.
3. Spring is an open source lightweight framework that allows Java EE 7 developers to build simple, reliable, and scalable enterprise applications.
4. This framework mainly focuses on providing various ways to help you manage your business objects.
5. It made the development of Web applications much easier as compared to classic Java frameworks and Application Programming Interfaces (APIs), such as Java database connectivity(JDBC), JavaServer Pages(JSP), and Java Servlet.
6. The Spring framework can be considered as a collection of sub-frameworks, also called layers, such as Spring AOP. Spring Object-Relational Mapping (Spring ORM). Spring Web Flow, and Spring Web MVC.

Inversion of Control

1. Inversion of Control (IoC) is a software design principle and a fundamental concept in the Spring framework that allows the container to control the creation of bean instances.
2. IoC transfers the control of objects or parts of a program to a container or framework.
3. This allows developers to write more loosely-coupled code, which makes the system more maintainable and extensible.
4. IoC can improve a program's pluggability, testability, usability, and loose coupling.
5. Dependency Injection (DI) is a pattern that can be used to implement IoC. In DI, the control being inverted is setting an object's dependencies.
6. A bean definition is a recipe for creating one or more objects.
7. The container uses the configuration metadata in the bean definition to create or acquire an object.

IoC Container

1. The Spring container is at the center of the Spring Framework.
2. It create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.
3. The Spring container uses Dependency injection to manage the components that make up an application.
4. These objects are called Spring Beans.
5. The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration meta-data provided.
6. The configuration meta-data can be represented either by XML, Java annotations, or Java code.
7. The following diagram represents a high-level view of how Spring works.
8. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.

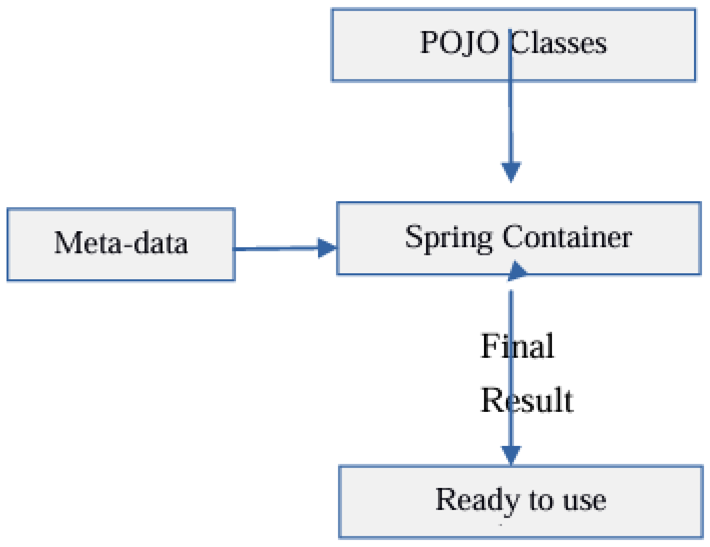


Fig. 5.3 Working of IOC Container

Beans

1. In the Spring Framework, a "bean" is essentially an object that is managed by the Spring IoC (Inversion of Control) container.
2. The container is responsible for instantiating, configuring, and assembling these objects, which are often referred to as beans.

3. The Spring container manages the lifecycle of beans, including their creation, initialization, and destruction.
4. You can hook into these lifecycle events using methods annotated with @PostConstruct and @PreDestroy, or by implementing InitializingBean and DisposableBean interfaces.
5. Beans can have their dependencies injected by the Spring container.
6. This can be done via constructor injection, setter injection, or field injection.
7. This is a core feature of Spring, enabling loose coupling between components.

Scope of beans

Beans can have different scopes, defining their lifecycle and visibility. Common scopes include:

- **Singleton**: A single instance of the bean is created for the entire Spring container.
- **Prototype**: A new instance of the bean is created each time it is requested.
- **Request**: A new instance of the bean is created for each HTTP request (relevant in web applications).
- **Session**: A new instance of the bean is created for each HTTP session (relevant in web applications).
- **Global Session**: A new instance of the bean is created for global HTTP sessions (used in portlet environments).

Constructor injection

1. Constructor injection is a method of dependency injection in which dependencies are provided to a class through its constructor.
2. This is a core concept in the Spring Framework and is commonly used to promote immutability and enforce dependency requirements at the time of object creation.
3. When a class has dependencies that it requires to function, these dependencies are defined as constructor parameters.
4. The Spring container creates instances of the class and injects the required dependencies via the constructor.
5. This means that all dependencies are provided at the time the bean is instantiated.
6. Since dependencies are provided during object creation, they can be set as final, making the object immutable.
7. This is beneficial for thread safety and ensures that dependencies cannot be changed after the object is created.
8. Constructor injection makes it clear which dependencies are required for the class to function.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MyService {

    private final MyRepository myRepository;

    @Autowired
    public MyService(MyRepository myRepository) {
        this.myRepository = myRepository;
    }

    // other methods
}
```
In this example, `MyService` has a dependency on `MyRepository`, which is injected via the constructor. The `@Autowired` annotation tells Spring to use this constructor to inject `MyRepository`.

Setter injection

1. Setter injection is a method of dependency injection where dependencies are provided to a class via setter methods rather than through the constructor.
2. This approach is commonly used in the Spring Framework and offers flexibility for setting or changing dependencies after an object has been instantiated.
3. It allows for more flexibility in configuration. Dependencies can be set or changed at runtime, which can be useful for certain scenarios.
4. Setter injection can make unit testing easier because you can set dependencies manually or through test frameworks without needing to use a constructor.
5. Setter injection allows for mutable state because dependencies can be changed after the object is created.
6. This can be beneficial in some cases but may lead to unpredictable behavior if not managed carefully.

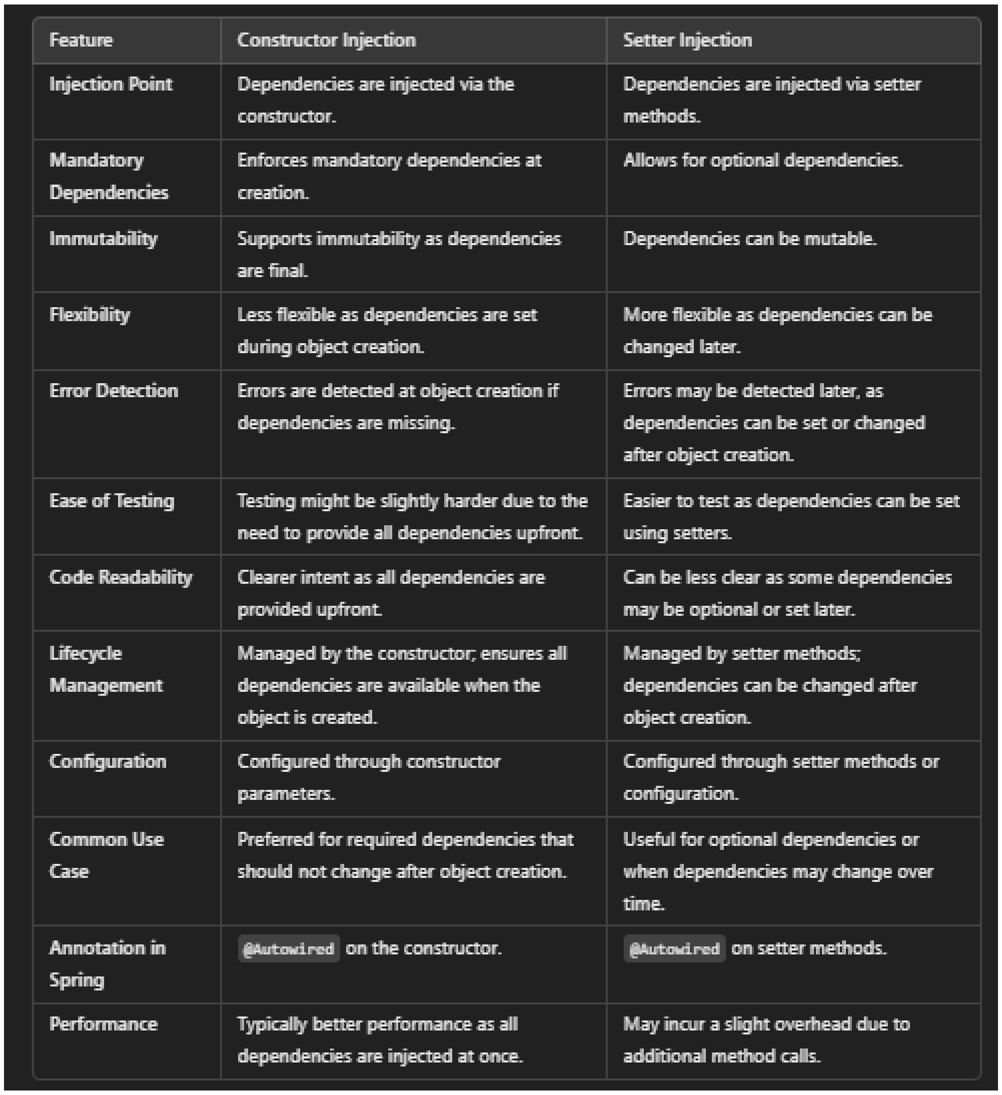
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MyService {

    private MyRepository myRepository;
    @Autowired
    public void setMyRepository(MyRepository myRepository) {
        this.myRepository = myRepository;
    }

    // other methods
}
```
In this example, the `setMyRepository` method is used to inject `MyRepository`. Spring will call this method after creating the `MyService` instance.

Difference in constructor injection and setter injection



Feature	Constructor Injection	Setter Injection
Injection Point	Dependencies are injected via the constructor.	Dependencies are injected via setter methods.
Mandatory Dependencies	Enforces mandatory dependencies at creation.	Allows for optional dependencies.
Immutability	Supports immutability as dependencies are final.	Dependencies can be mutable.
Flexibility	Less flexible as dependencies are set during object creation.	More flexible as dependencies can be changed later.
Error Detection	Errors are detected at object creation if dependencies are missing.	Errors may be detected later, as dependencies can be set or changed after object creation.
Ease of Testing	Testing might be slightly harder due to the need to provide all dependencies upfront.	Easier to test as dependencies can be set using setters.
Code Readability	Clearer intent as all dependencies are provided upfront.	Can be less clear as some dependencies may be optional or set later.
Lifecycle Management	Managed by the constructor; ensures all dependencies are available when the object is created.	Managed by setter methods; dependencies can be changed after object creation.
Configuration	Configured through constructor parameters.	Configured through setter methods or configuration.
Common Use Case	Preferred for required dependencies that should not change after object creation.	Useful for optional dependencies or when dependencies may change over time.
Annotation in Spring	`@Autowired` on the constructor.	`@Autowired` on setter methods.
Performance	Typically better performance as all dependencies are injected at once.	May incur a slight overhead due to additional method calls.

Unit 6 (Sprin MVC)

Spring Web MVC

1. Spring Web MVC is a framework in Spring used for building web applications.
2. It follows the Model-View-Controller (MVC) pattern, which helps separate an application's data (Model), user interface (View), and business logic (Controller).
3. In Spring Web MVC:

- **Model**: Represents the data and business logic.
- **View**: The user interface, such as JSP or Thymeleaf templates, that displays the data.
- **Controller**: Handles user requests, processes them using the model, and returns the view to display.

4. Spring Web MVC uses annotations to simplify configuration and mapping of HTTP requests to methods in controllers.
5. It supports flexible view rendering, form handling, and data binding, making it easier to develop scalable and maintainable web applications.

Working of MVC framework

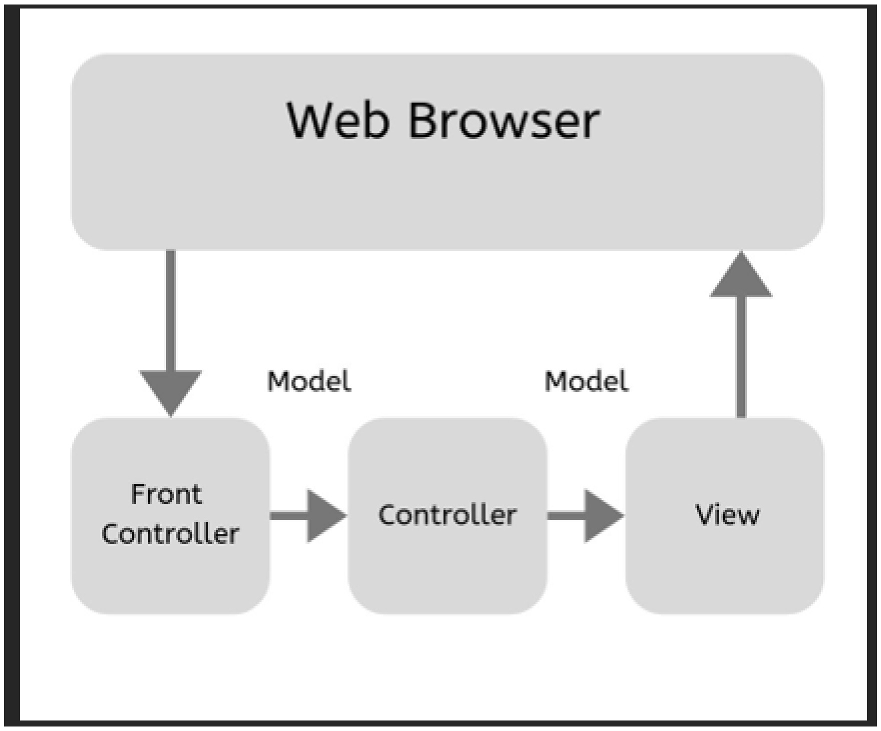
1. **Request Handling**: When a user makes an HTTP request, it is routed to a specific controller method based on URL patterns defined in the Spring configuration.
2. **Business Logic**: The controller processes the request, interacting with the model to retrieve or update data.
3. **View Rendering**: After processing, the controller selects a view template to render the response, passing any necessary model data to it.
4. **Response**: The view template generates HTML (or other formats) and sends it back to the user's browser as the HTTP response.

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.ui.Model;

@Controller
public class GreetingController {

    @GetMapping("/greet")
    public String greet(@RequestParam(name="name", required=false,
defaultValue="World") String name, Model model) {
        model.addAttribute("name", name);
        return "greet"; // Refers to greet.jsp or greet.html
    }
}
```
In this example, the controller method handles GET requests to `/greet`, processes the request, adds data to the model, and returns the view name to be rendered.

Spring MVC architecture



- **Model**– It contains the data of the application. A data can be asingle object or a collection of objects.
- **Controller**–It contains the business logic of an application. Here, the@Controller annotation is used to mark the class as the controller.
- **View**–It represents the provided information in a particularformat. Generally, JSP+JSTL is used to create a view page. Although spring alsosupports other view technologies such as Apache Velocity, Thymeleaf andFreeMarker.

- **Front Controller –** In Spring Web MVC, theDispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

Features of the Spring MVC framework

1. ***Powerful framework :*** The Spring Web MVC framework provides straightforward and powerful configuration of the framework as well as of application classes such as JavaBeans.
2. ***Easier testing:*** Most of the Spring classes are designed as JavaBeans, which allows us to inject the test data using the setter method of these JavaBeans classes.
3. ***Separation of roles:*** Each component of a Spring MVC framework does a different role during request handling. A request is handled by components such as the Controller, Validator, Model Object, View Resolver and HandlerMapping interfaces.
4. ***No need for the duplication of code:*** In the Spring MVC framework, we can use the existing business code in any component of the Spring MVC application. Therefore, no duplication of the code uprises in a Spring MVC application.
5. ***Specific validation and binding:*** Validation errors are displayed when any unmatched data is entered in a form.

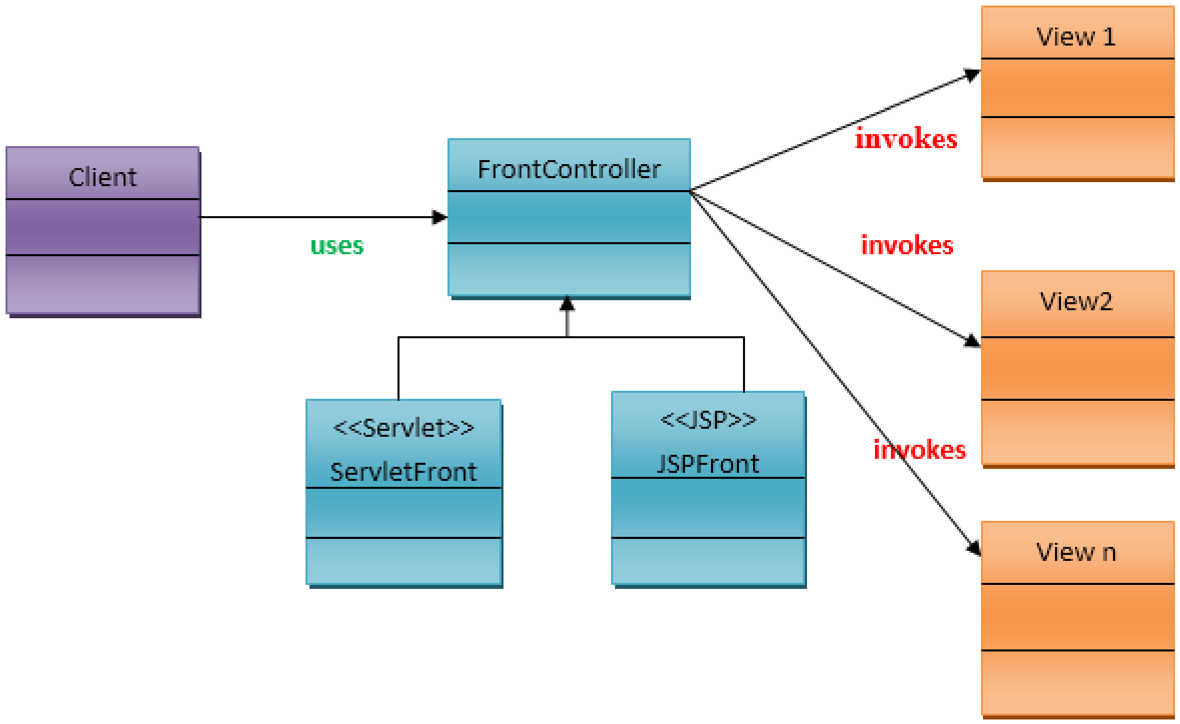
Front Controller Design Pattern

The front controller design pattern means that all requests that come for a resource in an application will be handled by a single handler and then dispatched to the appropriate handler for that type of request. The front controller may use other helpers to achieve the dispatching mechanism.

Design components:

- **Controller :** The controller is the initial contact point for handling all requests in the system. The controller may delegate to a helper to complete authentication and authorization of a user or to initiate contact retrieval.
- **View:** A view represents and displays information to the client. The view retrieves information from a model. Helpers support views by encapsulating and adapting the underlying data model for use in the display.

- **Dispatcher:** A dispatcher is responsible for view management and navigation, managing the choice of the next view to present to the user, and providing the mechanism for vectoring control to this resource.
- **Helper :** A helper is responsible for helping a view or controller complete its processing. Thus, helpers have numerous responsibilities, including gathering data required by the view and storing this intermediate model, in which case the helper is sometimes referred to as a value bean.



DispatcherServlet

1. DispatcherServlet plays a crucial role in handling HTTP requests and responses in a Spring-based web application.
2. When the web application starts, the DispatcherServlet is initialized.
3. Its configuration is typically defined in the web.xml file or, more commonly in modern applications, in a Java configuration class using @WebServlet.
4. The servlet routes incoming requests to the appropriate controllers based on URL mappings defined in the application.
5. The servlet then invokes the controller methods, which handle the business logic.
6. Once the controller has processed the request and returned a model and view name, the DispatcherServlet uses the view resolver to render the view.
7. After the view is rendered, DispatcherServlet sends the final response back to the client.

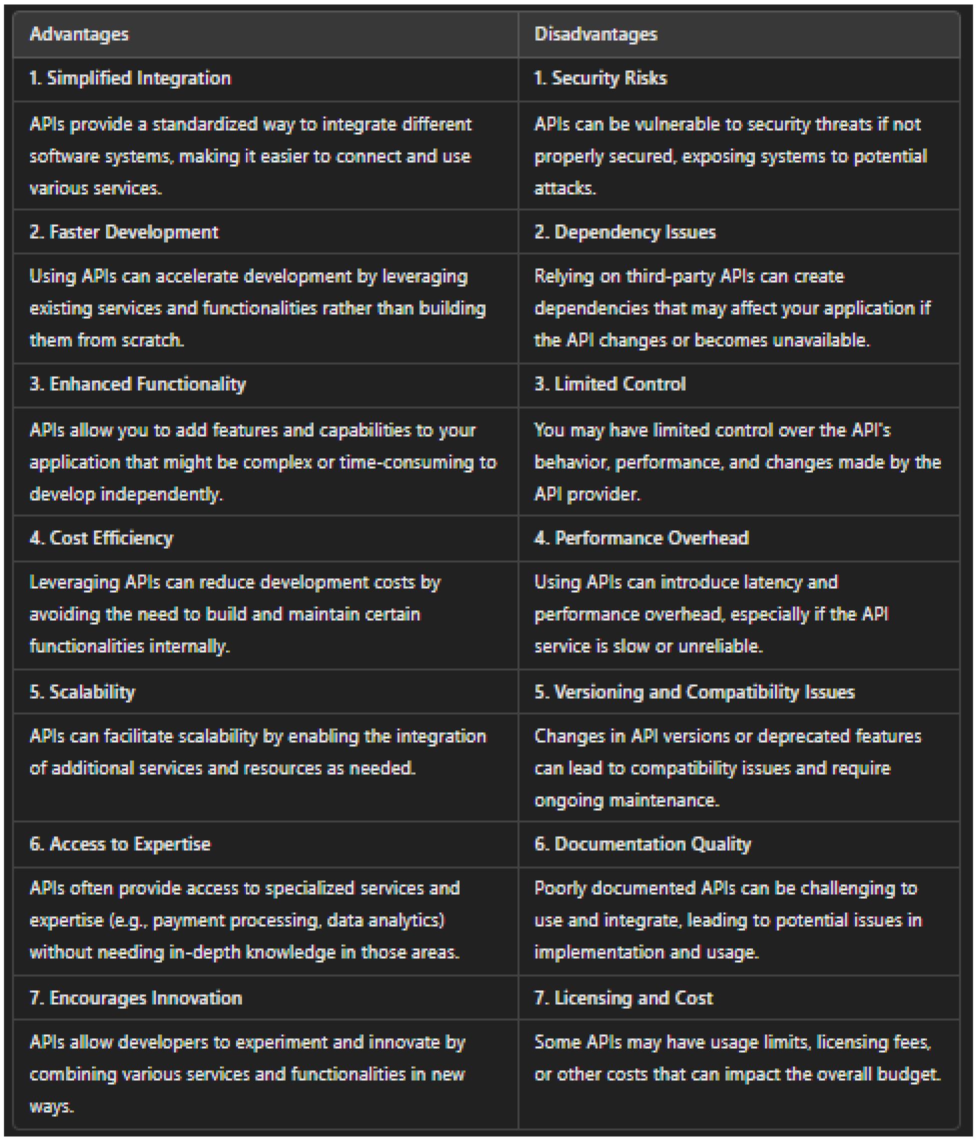
Unit 7 (Java Mail)

Application Programming Interface(API)

1. APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols.
2. API architecture is usually explained in terms of client and server.
3. The application sending the request is called the client, and the application sending the response is called the server.
4. There are four different ways that APIs can work depending on when and why they were created.

- **SOAP APIs:** These APIs use Simple Object Access Protocol. Client and server exchange messages using XML. This is a less flexible API that was more popular in the past.
- **RPC APIs:** These APIs are called Remote Procedure Calls. The client completes a function (or procedure) on the server, and the server sends the output back to the client.
- **Websocket API:** It uses JSON objects to pass data. A WebSocket API supports two-way communication between client apps and the server.
- **REST APIs:** These are the most popular and flexible APIs found on the web today. The client sends requests to the server as data. The server uses this client input to start internal functions and returns output data back to the client.

Advantages and Disadvantages of API



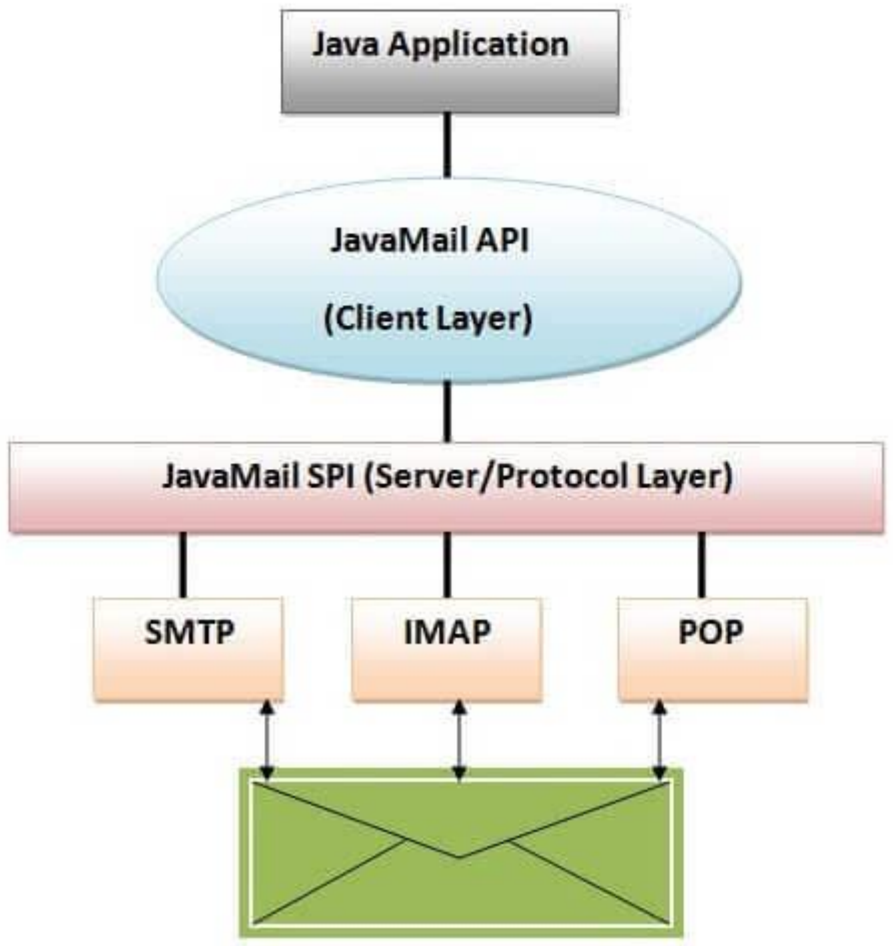
Advantages	Disadvantages
1. Simplified Integration	1. Security Risks
APIs provide a standardized way to integrate different software systems, making it easier to connect and use various services.	APIs can be vulnerable to security threats if not properly secured, exposing systems to potential attacks.
2. Faster Development	2. Dependency Issues
Using APIs can accelerate development by leveraging existing services and functionalities rather than building them from scratch.	Relying on third-party APIs can create dependencies that may affect your application if the API changes or becomes unavailable.
3. Enhanced Functionality	3. Limited Control
APIs allow you to add features and capabilities to your application that might be complex or time-consuming to develop independently.	You may have limited control over the API's behavior, performance, and changes made by the API provider.
4. Cost Efficiency	4. Performance Overhead
Leveraging APIs can reduce development costs by avoiding the need to build and maintain certain functionalities internally.	Using APIs can introduce latency and performance overhead, especially if the API service is slow or unreliable.
5. Scalability	5. Versioning and Compatibility Issues
APIs can facilitate scalability by enabling the integration of additional services and resources as needed.	Changes in API versions or deprecated features can lead to compatibility issues and require ongoing maintenance.
6. Access to Expertise	6. Documentation Quality
APIs often provide access to specialized services and expertise (e.g., payment processing, data analytics) without needing in-depth knowledge in those areas.	Poorly documented APIs can be challenging to use and integrate, leading to potential issues in implementation and usage.
7. Encourages Innovation	7. Licensing and Cost
APIs allow developers to experiment and innovate by combining various services and functionalities in new ways.	Some APIs may have usage limits, licensing fees, or other costs that can impact the overall budget.

JavaMail API

1. The JavaMail API is a Java library that provides a platform-independent and protocol-independent framework for sending and receiving emails.

2. It supports various email protocols, such as SMTP (Simple Mail Transfer Protocol) for sending emails, and IMAP (Internet Message Access Protocol) and POP3 (Post Office Protocol) for receiving them.
3. JavaMail allows developers to create, read, and manage email messages in their Java applications.
4. It offers classes for constructing email messages, including attachments and HTML content, and for managing email sessions and authentication.
5. By abstracting the complexities of email protocols, JavaMail simplifies email operations and integrates seamlessly with Java applications.
6. It is widely used in enterprise applications and server-side solutions for automated email communication.

JavaMail Architecture



Java Message Service (JMS)

1. Java Message Service (JMS) is a Java API that allows applications to create, send, receive, and read messages in a messaging system.
2. It provides a way for Java applications to communicate with other Java applications or with systems that support messaging in a loosely coupled manner.
3. In JMS, there are typically a Sender and a Receiver.
4. The Sender transmits a message, while the Receiver receives it.
5. A Message-Oriented Middleware (MOM) broker acts as an intermediary between the Sender and the Receiver.
6. This broker handles the Sender's message, passing it through the network to the Receiver. MOM is essentially a message queue (MQ) application.
7. The Sender and Receiver are loosely coupled, meaning they do not need to know about each other’s implementation details.
8. For instance, the Sender can be a .NET or mainframe-based application, while the Receiver might be a Java or Spring-based application.
9. The Receiver can also send messages back to the Sender, enabling two-way communication. This setup maintains the loosely coupled nature of the interaction.

Using Java Mail API to send email using java codes

```
import javax.mail.*;
import javax.mail.internet.*;
import java.util.Properties;

public class SendEmail {

    public static void main(String[] args) {
        // Email configuration
        String host = "smtp.example.com"; // SMTP server address
        final String user = "your-email@example.com"; // Your email ID
        final String password = "your-password"; // Your email
password
```

```
        // Set up properties
        Properties props = new Properties();
        props.put("mail.smtp.host", host);
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.port", "587"); // Port for TLS
        props.put("mail.smtp.starttls.enable", "true"); // Enable TLS

        // Create session with authenticator
        Session session = Session.getInstance(props, new
Authenticator() {
            protected PasswordAuthentication
getPasswordAuthentication() {
                return new PasswordAuthentication(user, password);
            }
        });

        try {
            // Create and send email
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(user));
            message.setRecipients(Message.RecipientType.TO,
InternetAddress.parse("recipient@example.com"));
            message.setSubject("Test Email");
            message.setText("Hello, this is a test email sent from
Java!");

            Transport.send(message);

            System.out.println("Email sent successfully!");
        } catch (MessagingException e) {
            e.printStackTrace(); // Print error details
        }
    }
}
```

Unit 8 (Java with JSON)

JSON

1. JSON (JavaScript Object Notation) is a lightweight data interchange format that's easy for humans to read and write, and easy for machines to parse and generate.
2. It is primarily used to transmit data between a server and web applications, but it's also commonly used in various programming environments for storing and exchanging data.
3. JSON can represent two types of structured data: objects and arrays.
4. Objects are unordered collections of name/value pairs, and are indicated by curly brackets.
5. JSON is often used to send data between a server and a web application.
6. It's also a common API output in many applications.

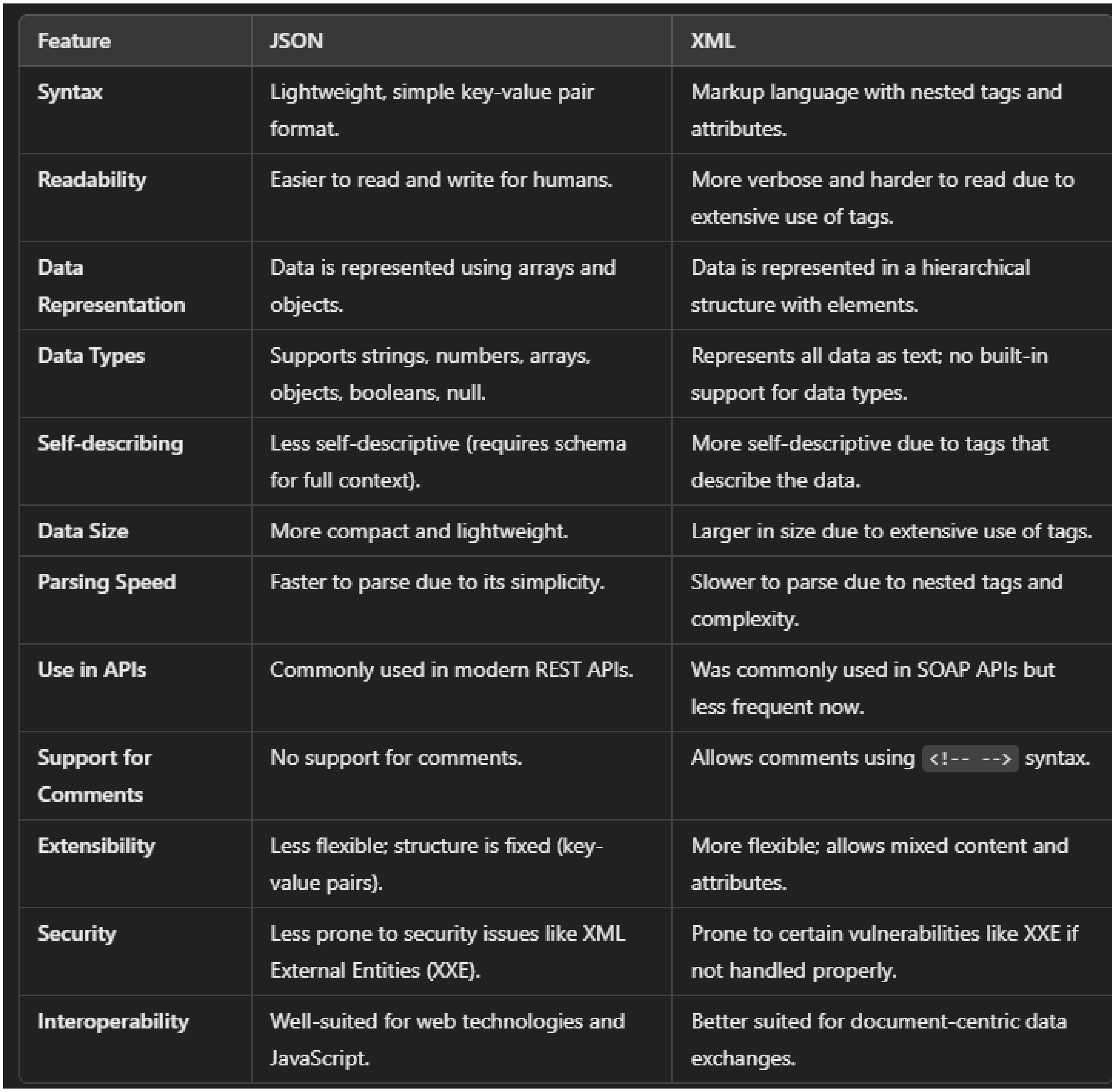
Applications of JSON

- **Data transfer:** JSON is a lightweight alternative to XML for transferring data between systems and programming languages.

  It's commonly used in web applications and APIs.

- **Storing data:** JSON can be used to store temporary data and the state of applications.
- **Configuring data:** JSON can be used for configuration files, such as storing credentials and log file paths for applications.
- **Simplifying data models:** JSON can simplify complex documents into human-readable files.
- **Semi-structured data:** JSON is a widely used format for semi-structured data because it doesn't require a schema.
- **Sparse datasets:** JSON objects (key/value pairs) can be useful for storing sparse datasets.

Difference in JSON and XML formats



Feature	JSON	XML
Syntax	Lightweight, simple key-value pair format.	Markup language with nested tags and attributes.
Readability	Easier to read and write for humans.	More verbose and harder to read due to extensive use of tags.
Data Representation	Data is represented using arrays and objects.	Data is represented in a hierarchical structure with elements.
Data Types	Supports strings, numbers, arrays, objects, booleans, null.	Represents all data as text; no built-in support for data types.
Self-describing	Less self-descriptive (requires schema for full context).	More self-descriptive due to tags that describe the data.
Data Size	More compact and lightweight.	Larger in size due to extensive use of tags.
Parsing Speed	Faster to parse due to its simplicity.	Slower to parse due to nested tags and complexity.
Use in APIs	Commonly used in modern REST APIs.	Was commonly used in SOAP APIs but less frequent now.
Support for Comments	No support for comments.	Allows comments using `<!-- -->` syntax.
Extensibility	Less flexible; structure is fixed (key-value pairs).	More flexible; allows mixed content and attributes.
Security	Less prone to security issues like XML External Entities (XXE).	Prone to certain vulnerabilities like XXE if not handled properly.
Interoperability	Well-suited for web technologies and JavaScript.	Better suited for document-centric data exchanges.

Different data types in JSON

JSON supports the following data types:

1. **String**: A sequence of characters, enclosed in double quotes.
   - Example: `"name": "Alice"`
2. **Number**: Numeric values, including integers and floating-point numbers.
   - Example: `"age": 25` or `"height": 5.7`
3. **Boolean**: Represents logical values `true` or `false`.
   - Example: `"isStudent": true`
4. **Array**: An ordered list of values, which can contain any of the other data types, including objects.
   - Example: `"hobbies": ["reading", "swimming", "coding"]`
5. **Object**: A collection of key-value pairs, where the keys are strings, and the values can be any JSON data type (including other objects).
   - Example:

```
"address":
{
"city": "New York",
"zip": "10001"
}
```

6. Null: Represents an empty or non-existent value.
Example: `"middleName": null`

Jackson library

1. The Jackson library is a popular high-performance JSON processing library for Java.
2. It allows developers to easily convert Java objects to JSON and vice versa (this process is called serialization and deserialization).
3. ObjectMapper is the core class used to map between JSON and Java objects. It handles serialization (Java object to JSON) and deserialization (JSON to Java object).
4. Jackson is known for its flexibility, ease of use, and powerful features, making it a common choice for working with JSON in Java applications.
5. Jackson supports two forms of data binding: Simple Data Binding: Converting simple JSON to/from basic Java types such as Strings, integers, arrays, etc. and Full Data Binding: Converting JSON to/from complex custom objects and data structures.

GSON

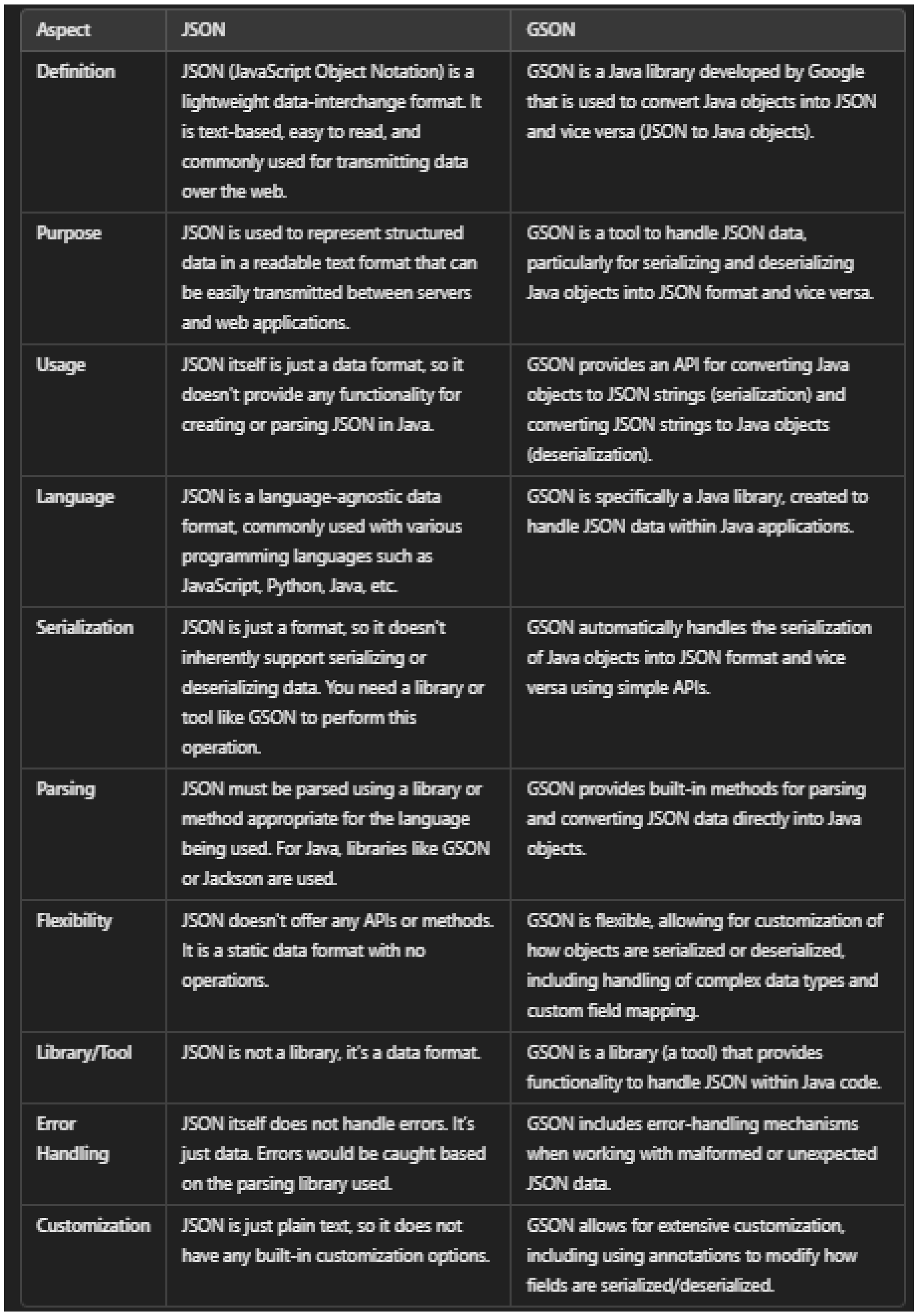
1. GSON (Google's JSON) is an open-source Java library developed by Google for serializing Java objects to JSON and deserializing JSON back into Java objects.
2. It is widely used for working with JSON data in Java applications and provides simple APIs for converting Java objects into JSON representations and vice versa.
3. Unlike some libraries (like Jackson), GSON can work without requiring annotations in most cases.
4. It relies on Java's reflection mechanism to map JSON fields to Java fields automatically.
5. GSON can handle complex nested data structures, such as lists, arrays, maps, and custom objects.
6. GSON provides custom type adapters for advanced serialization and deserialization logic, allowing developers to handle special cases.

---

process of Publishing a Service using JSON in JSP

1. Ensure a Java web project with a web server like Apache Tomcat is ready.
2. Properly configure the project to handle JSP files and map requests using web.xml or servlet annotations.
3. Add a library like Jackson or GSON to the project for JSON serialization.
4. If using Maven, include the necessary dependencies for these libraries.
5. Define Java classes representing the data (e.g., user details, product information) that you wish to send as JSON responses.
6. Set the content type of the JSP page to application/json so the response is recognized as JSON.
7. Use a JSON library (like GSON) to convert Java objects into JSON format.
8. In the JSP page, serialize the Java objects into a JSON string.
9. Output the JSON string as the response using the JSP’s output stream.

Difference in JSON and GSON



Aspect	JSON	GSON
Definition	JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is text-based, easy to read, and commonly used for transmitting data over the web.	GSON is a Java library developed by Google that is used to convert Java objects into JSON and vice versa (JSON to Java objects).
Purpose	JSON is used to represent structured data in a readable text format that can be easily transmitted between servers and web applications.	GSON is a tool to handle JSON data, particularly for serializing and deserializing Java objects into JSON format and vice versa.
Usage	JSON itself is just a data format, so it doesn't provide any functionality for creating or parsing JSON in Java.	GSON provides an API for converting Java objects to JSON strings (serialization) and converting JSON strings to Java objects (deserialization).
Language	JSON is a language-agnostic data format, commonly used with various programming languages such as JavaScript, Python, Java, etc.	GSON is specifically a Java library, created to handle JSON data within Java applications.
Serialization	JSON is just a format, so it doesn't inherently support serializing or deserializing data. You need a library or tool like GSON to perform this operation.	GSON automatically handles the serialization of Java objects into JSON format and vice versa using simple APIs.
Parsing	JSON must be parsed using a library or method appropriate for the language being used. For Java, libraries like GSON or Jackson are used.	GSON provides built-in methods for parsing and converting JSON data directly into Java objects.
Flexibility	JSON doesn't offer any APIs or methods. It is a static data format with no operations.	GSON is flexible, allowing for customization of how objects are serialized or deserialized, including handling of complex data types and custom field mapping.
Library/Tool	JSON is not a library, it's a data format.	GSON is a library (a tool) that provides functionality to handle JSON within Java code.
Error Handling	JSON itself does not handle errors. It's just data. Errors would be caught based on the parsing library used.	GSON includes error-handling mechanisms when working with malformed or unexpected JSON data.
Customization	JSON is just plain text, so it does not have any built-in customization options.	GSON allows for extensive customization, including using annotations to modify how fields are serialized/deserialized.