J2EE architecture, Servlet API and Overview

1. J2EE Architecture

Overview:
Java 2 Platform, Enterprise Edition (J2EE) is designed for building large-scale, multi-tiered, distributed applications. It offers a standard architecture and a set of APIs for enterprise applications.

Core Components:

- **Client Tier**: The client layer where users interact with the system. It can be a web browser, thick client, or mobile application.
- **Web Tier**: Composed of Servlets and JavaServer Pages (JSP), handling HTTP requests and generating dynamic web content.
- **Business Tier**: Contains the business logic of the application. This tier often uses Enterprise JavaBeans (EJB) to encapsulate business logic.
- **Enterprise Information System (EIS) Tier**: Manages data persistence, typically through JDBC or messaging systems like JMS.

Containers:

- **Web Container**: Manages the lifecycle of Servlets and JSPs (e.g., Tomcat).
- **EJB Container**: Manages Enterprise JavaBeans, handling transactions, security, and life cycle.
- **Application Client Container**: Manages stand-alone applications.

J2EE Services:

- **JMS (Java Message Service)**: For messaging.
- **JDBC (Java Database Connectivity)**: For database connectivity.
- **JavaMail**: For sending and receiving emails.

---

2. Enterprise Application Concepts

Overview:
Enterprise applications are large-scale applications used in business environments to support business processes, like managing customer data, inventory, or transactions.

Characteristics:

- **Scalability**: Designed to handle large volumes of transactions and users.
- **Reliability**: High availability and fault tolerance are critical.

- **Security**: Protect sensitive data and ensure proper authentication and authorization.
- **Maintainability**: Must be easily updated or expanded without affecting overall system performance.

Enterprise Application Architecture:

- **Component-Based Development**: Uses reusable components such as EJB, Servlets, and JSP to break the application into manageable units.
- **Service-Oriented Architecture (SOA)**: Enterprise applications often use SOA principles to integrate diverse services.

Technologies Used:

- **EJB (Enterprise JavaBeans)**: For transaction management, security, and business logic.
- **JMS (Java Message Service)**: For asynchronous communication.
- **JDBC**: For connecting to databases.
- **JNDI (Java Naming and Directory Interface)**: For accessing naming services.

---

3. N-Tier Application Concepts

Overview:
N-Tier architecture divides an application into logical layers or tiers, each responsible for specific tasks. It separates concerns, improving modularity, scalability, and maintainability.

Tiers in N-Tier Architecture:

1. **Presentation Layer**: The client-side interface (e.g., web browsers, mobile apps).
2. **Business Logic Layer**: Where the core application logic resides, often implemented with EJB or other Java classes.
3. **Data Access Layer**: Handles interaction with databases, typically using JDBC or ORM frameworks like Hibernate.
4. **Database Layer**: The back-end database, such as MySQL or Oracle.

Advantages:

- **Separation of Concerns**: Each layer can evolve independently.
- **Scalability**: Each layer can scale independently by adding servers to the layers that are under heavy load.
- **Maintainability**: Easier to update or maintain specific layers without affecting the entire system.

---

4. J2EE Platform

Overview:
J2EE provides a standard platform for building and deploying enterprise-level applications. It extends the capabilities of the standard Java Development Kit (JDK) with additional APIs and services to create robust, scalable applications.

Key Features:

- **Cross-Platform**: J2EE applications can run on any operating system that supports the Java Virtual Machine (JVM).
- **Distributed Computing**: J2EE provides support for distributed systems with technologies like RMI (Remote Method Invocation) and EJB.
- **Component-Based**: Encourages the use of reusable components such as Servlets, EJBs, and JSP.
- **Security**: Built-in mechanisms for authentication, authorization, and data encryption.

Main J2EE Technologies:

- **Servlets**: Used to process HTTP requests.
- **JSP**: Used to generate dynamic web content.
- **EJB**: Manages business logic in a distributed environment.
- **JMS**: Provides messaging capabilities for loosely coupled systems.
- **JDBC**: Allows interaction with databases.

---

5. Servlet Model Overview

Overview:
A **Servlet** is a server-side Java program that processes client requests and generates dynamic web content. It runs in a servlet container, such as Tomcat, which manages the lifecycle and requests.

Servlets vs JSP:

- **Servlets**: Primarily used for processing logic and handling HTTP requests.
- **JSP**: Focuses on presentation and can be seen as an abstraction over Servlets.

Key Concepts:

- **Request and Response**: Servlets receive requests (HTTP) from clients and send back responses (usually HTML).
- **HttpServlet**: The most commonly used servlet class for handling HTTP requests.

Advantages:

- **Efficiency**: Servlets run in the server's memory, improving performance over traditional CGI scripts.
- **Extensibility**: Can be extended to create customized behavior.

---

6. Servlet Life Cycle

Overview:
The servlet life cycle is managed by the servlet container and consists of four key phases: loading, initialization, request handling, and destruction.

Stages:

1. **Loading and Instantiation**: The servlet class is loaded into memory by the container.
2. **Initialization**: The servlet's `init()` method is invoked to initialize resources or settings.
3. **Request Handling**: For each incoming request, the container calls the servlet's `service()` method to process the request and generate a response.
4. **Destruction**: When the servlet is no longer needed, the container calls the `destroy()` method to release resources and perform cleanup.

Important Methods:

- **`init()`**: Called once, when the servlet is loaded.
- **`service()`**: Called for each request.
- **`destroy()`**: Called when the servlet is destroyed.

---

7. HTTP Methods

Overview:
HTTP methods define the type of action the client wants to perform on the resource identified by the URL. The most common HTTP methods are used to manage CRUD (Create, Read, Update, Delete) operations.

Common HTTP Methods:

- **GET**: Retrieves data from the server. Used for fetching resources.
- **POST**: Sends data to the server to create or update a resource (e.g., submitting a form).
- **PUT**: Updates a resource or creates it if it does not exist.
- **DELETE**: Removes a resource from the server.
- **HEAD**: Similar to GET, but only retrieves the headers without the body content.
- **OPTIONS**: Returns the allowed HTTP methods for a given resource.
- **PATCH**: Partially updates a resource, unlike PUT, which replaces the entire resource.

Usage in Servlets:

In the servlet `service()` method, different HTTP methods are handled by `doGet()`, `doPost()`, `doPut()`, and so on, each corresponding to a specific HTTP method.