Java EE

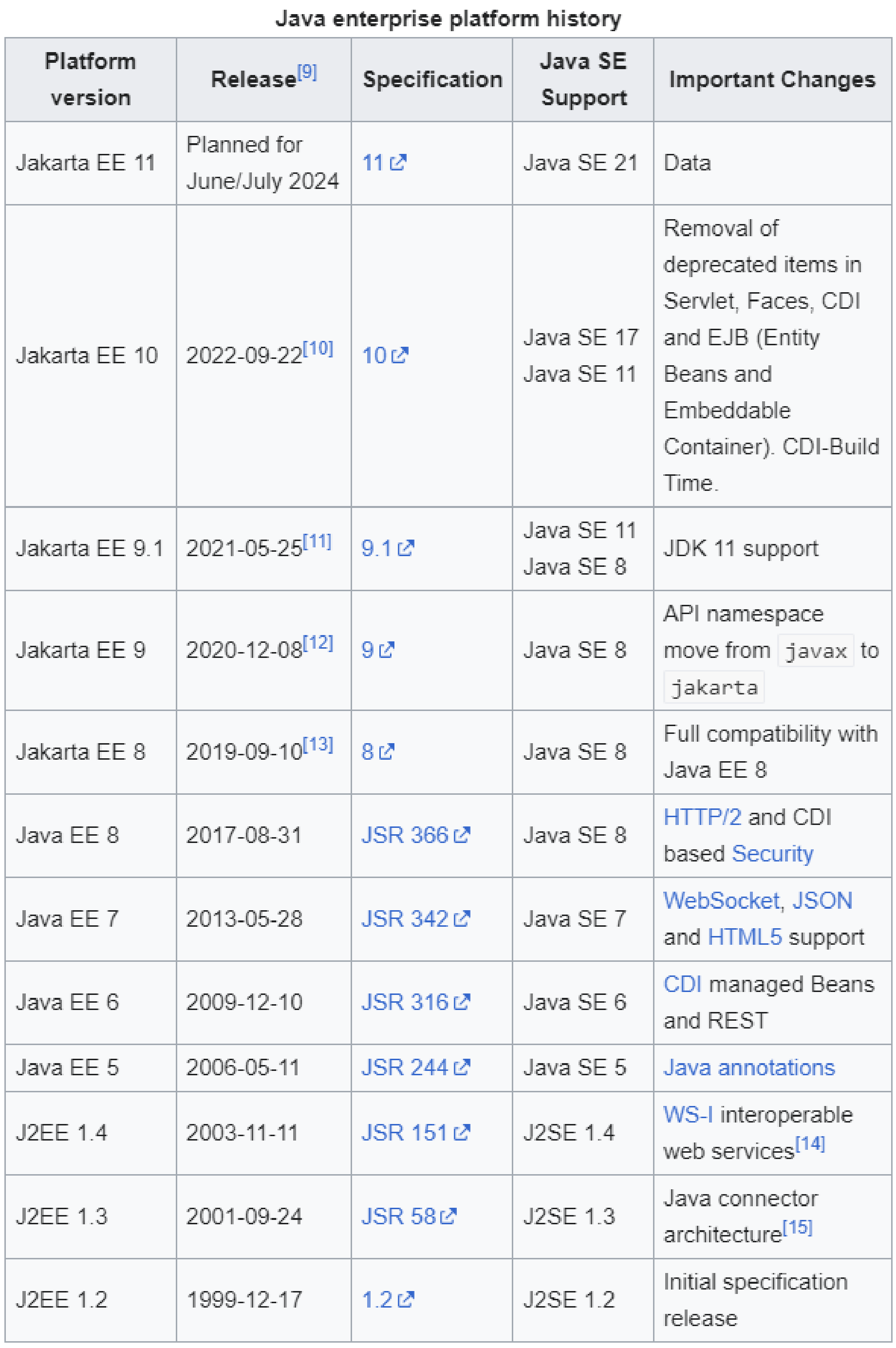
The **Java EE** stands for **Java Enterprise Edition**, which was earlier known as J2EE and is currently known as **Jakarta EE**. It is a set of specifications wrapping around Java SE (Standard Edition). The Java EE provides a platform for developers with enterprise features such as distributed computing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

Java SE vs Java EE

Java SE refers to standard edition and contains basic functionalities and packages required by a beginner or intermediate-level programmer. Java EE is an enhanced platform and a wrapper around Java SE. It has the edge over Java SE an also has a variety of aspects in which it outshines other features.

Introduction to Java EE Architecture

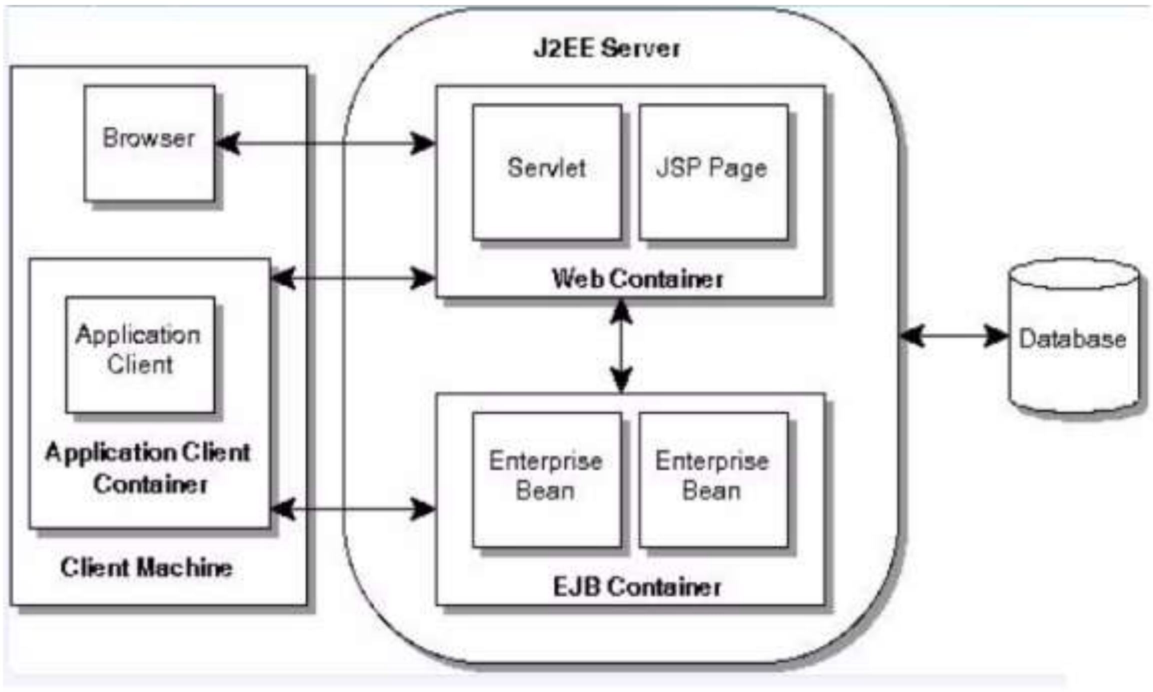
Java EE (Enterprise Edition) is a set of specifications that extend the Java SE (Standard Edition) with specifications for enterprise features such as distributed computing and web services. Java EE is designed to help developers build large-scale, multi-tiered, scalable, and secure applications.

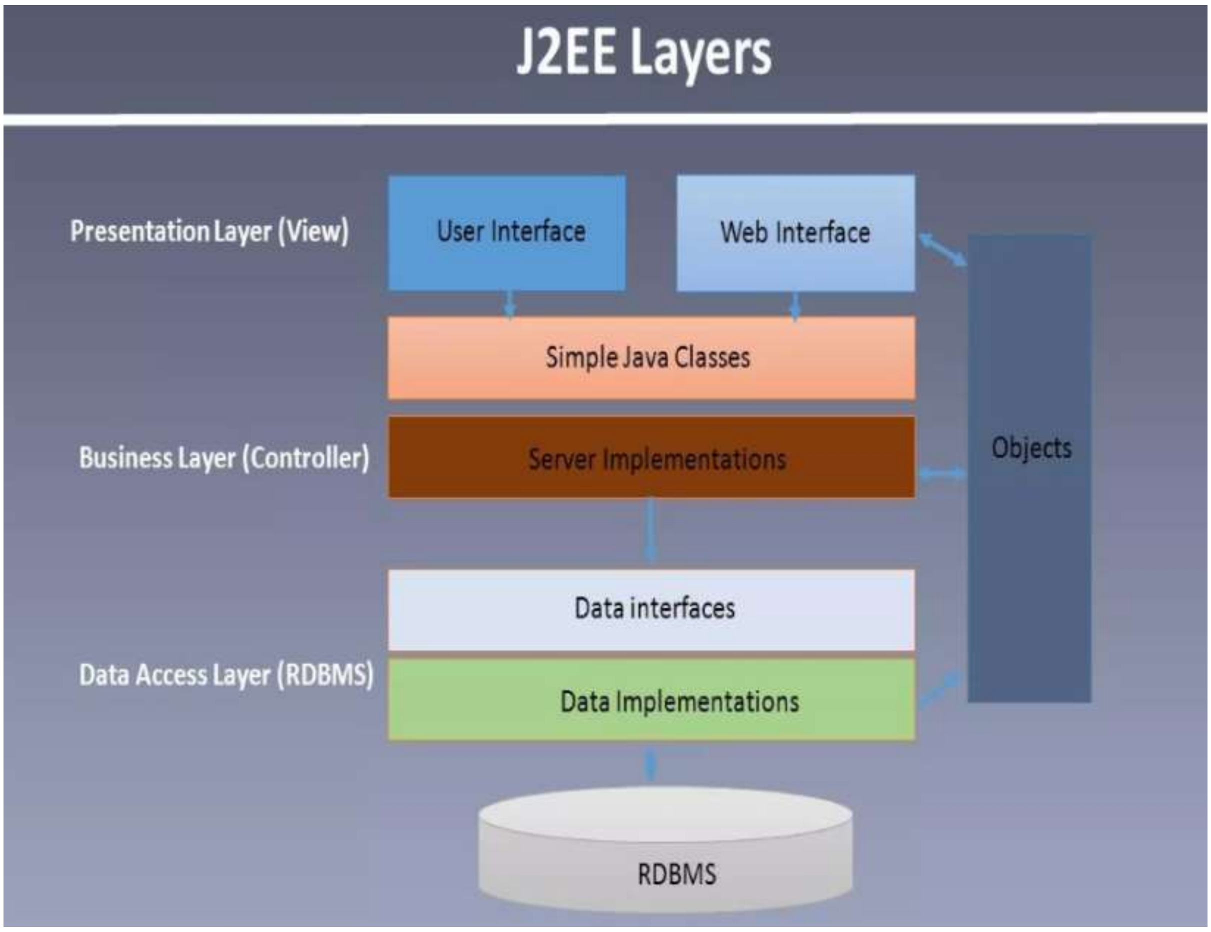


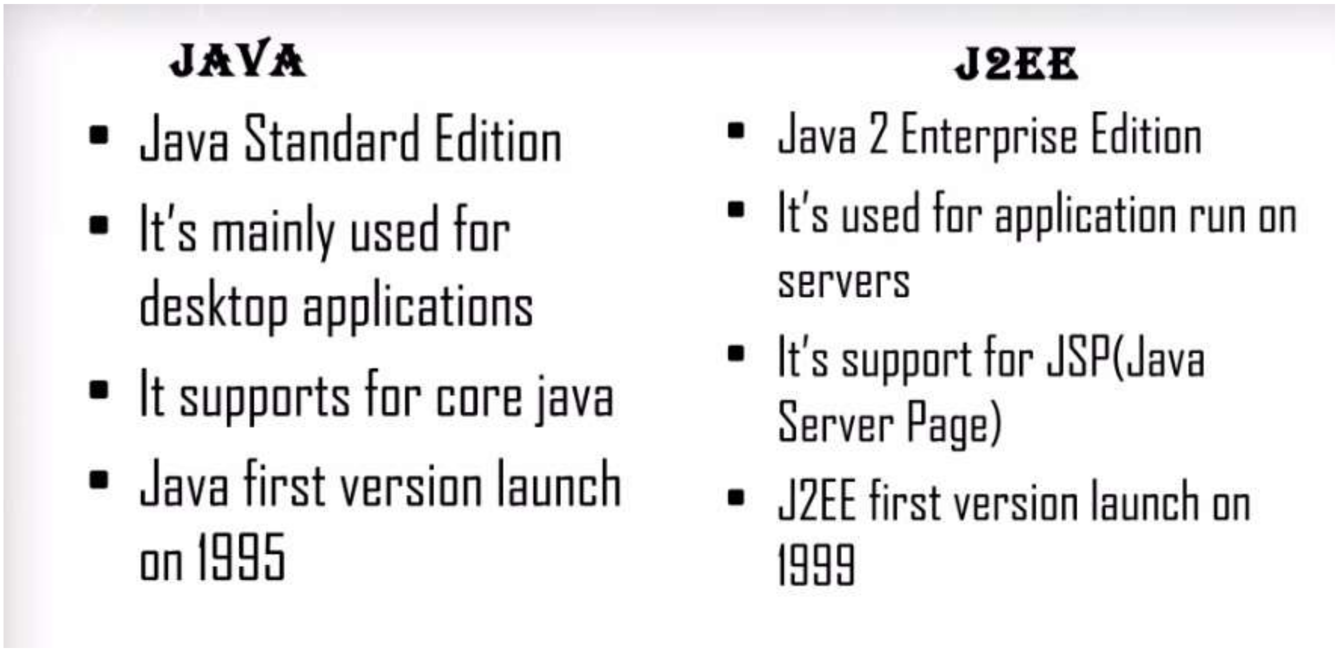
Java enterprise platform history

Platform version	Release[9]	Specification	Java SE Support	Important Changes
Jakarta EE 11	Planned for June/July 2024	11	Java SE 21	Data
Jakarta EE 10	2022-09-22[10]	10	Java SE 17 Java SE 11	Removal of deprecated items in Servlet, Faces, CDI and EJB (Entity Beans and Embeddable Container). CDI-Build Time.
Jakarta EE 9.1	2021-05-25[11]	9.1	Java SE 11 Java SE 8	JDK 11 support
Jakarta EE 9	2020-12-08[12]	9	Java SE 8	API namespace move from `javax` to `jakarta`
Jakarta EE 8	2019-09-10[13]	8	Java SE 8	Full compatibility with Java EE 8
Java EE 8	2017-08-31	JSR 366	Java SE 8	HTTP/2 and CDI based Security
Java EE 7	2013-05-28	JSR 342	Java SE 7	WebSocket, JSON and HTML5 support
Java EE 6	2009-12-10	JSR 316	Java SE 6	CDI managed Beans and REST
Java EE 5	2006-05-11	JSR 244	Java SE 5	Java annotations
J2EE 1.4	2003-11-11	JSR 151	J2SE 1.4	WS-I interoperable web services[14]
J2EE 1.3	2001-09-24	JSR 58	J2SE 1.3	Java connector architecture[15]
J2EE 1.2	1999-12-17	1.2	J2SE 1.2	Initial specification release

Key Components of Java EE Architecture:







JAVA	J2EE
Java Standard Edition	Java 2 Enterprise Edition
It's mainly used for desktop applications	It's used for application run on servers
It supports for core java	It's support for JSP(Java Server Page)
Java first version launch on 1995	J2EE first version launch on 1999

1. **Enterprise JavaBeans (EJB):**

   - **Session Beans:** Handle business logic. Can be stateful or stateless.

- **Message-Driven Beans:** Process asynchronous messages.
- **Entity Beans:** Represent persistent data (though this is less common now, with JPA taking over).

2. Java Persistence API (JPA):

- Provides a way to manage relational data in Java applications. It handles data persistence by mapping Java objects to database tables.

3. JavaServer Faces (JSF):

- A component-based web framework for building user interfaces for web applications. It simplifies the development of server-side user interfaces.

4. Java Servlet API:

- Defines how servlets interact with clients, including handling HTTP requests and responses.

5. Java Message Service (JMS):

- Provides a way to create, send, receive, and read messages in a distributed system, supporting asynchronous communication.

6. Context and Dependency Injection (CDI):

- Manages the lifecycle and interactions of stateful objects, enhancing modularity and testability.

7. Java Transaction API (JTA):

- Manages transactions across multiple resources, ensuring data consistency and integrity.

8. **JavaMail API:**

- Facilitates the sending and receiving of email messages from Java applications.

9. **Java API for RESTful Web Services (JAX-RS):**

- Allows developers to create RESTful web services in Java.

Different Types of Servers

1. **Web Server:**

- **Purpose:** Handles HTTP requests and responses. It serves static content like HTML, CSS, and JavaScript files.
- **Examples:** Apache HTTP Server, Nginx.

2. **Application Server:**

- **Purpose:** Provides a runtime environment for applications and includes web server capabilities along with additional services like transaction management, security, and business logic processing.
- **Examples:** Apache Tomcat (often used as a web server but also supports servlets and JSP), JBoss/WildFly, IBM WebSphere, Oracle WebLogic.

Deployment Tools:

There are various tools available for deploying **J2EE** applications. Popular ones include **Apache Tomcat, JBoss, WebLogic, and**. These tools provide the necessary infrastructure to deploy and manage **J2EE** applications in different environments.

3. **Database Server:**

    - **Purpose:** Manages and provides access to databases. It handles database queries and transactions.
    - **Examples:** MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.

4. **Mail Server:**

    - **Purpose:** Handles sending and receiving email.
    - **Examples:** Microsoft Exchange Server, Postfix, Sendmail.

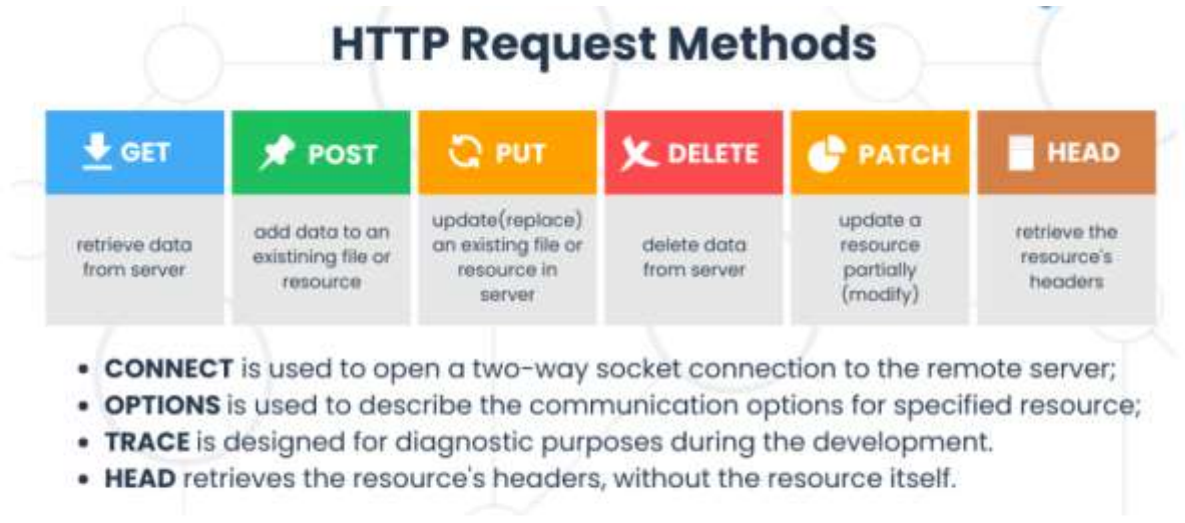
5. **Proxy Server:**

   - **Purpose:** Acts as an intermediary between clients and servers, often used for load balancing, security, or caching.
   - **Examples:** HAProxy, Squid.

HTTP Stateless Protocol and its Methods

HTTP (Hypertext Transfer Protocol) is a stateless protocol, meaning that each request from a client to a server is independent; the server does not retain any information about previous requests. This characteristic simplifies server design but requires additional mechanisms (like cookies, sessions, or tokens) for maintaining state across multiple requests.

Common HTTP Methods:



HTTP Request Methods

GET	POST	PUT	DELETE	PATCH	HEAD
retrieve data from server	add data to an existining file or resource	update(replace) an existing file or resource in server	delete data from server	update a resource partially (modify)	retrieve the resource's headers

- **CONNECT** is used to open a two-way socket connection to the remote server;
- **OPTIONS** is used to describe the communication options for specified resource;
- **TRACE** is designed for diagnostic purposes during the development.
- **HEAD** retrieves the resource's headers, without the resource itself.

1. **GET:**

   - **Purpose:** Retrieve data from the server. The request should not change the state of the server.
   - **Use Case:** Requesting a webpage or resource.

2. **POST:**

   - **Purpose:** Send data to the server to create or update a resource. The request may change the state of the server.
   - **Use Case:** Submitting form data or uploading a file.

3. **PUT:**
    - **Purpose:** Update or replace a resource on the server with the data provided in the request.
    - **Use Case:** Updating a resource, like modifying user details.

4. **DELETE:**
    - **Purpose:** Remove a resource from the server.
    - **Use Case:** Deleting a user or record.

5. **HEAD:**
    - **Purpose:** Retrieve the headers of a resource without the body. Used to check metadata.
    - **Use Case:** Checking if a resource has been modified.

6. **OPTIONS:**
    - **Purpose:** Request information about the communication options available for a resource, like supported methods.
    - **Use Case:** Discovering allowed methods and features of a resource.

7. **PATCH:**
    - **Purpose:** Apply partial modifications to a resource.
    - **Use Case:** Updating part of a resource, such as changing only specific fields of a user profile.

These components and methods are foundational in web and enterprise application development, helping developers build robust and scalable solutions.