Java EE (Enterprise Edition), ora noto come **Jakarta EE**  
  
è una piattaforma per lo sviluppo di applicazioni aziendali robuste, scalabili e sicure. Ti spiegherò in maniera semplice e schematica i suoi aspetti principali, con un focus sui modi per collegarsi a database e browser, dal più vecchio al più moderno, includendo i principali framework.

**1. Cos’è Java EE / Jakarta EE?**

È un insieme di **specifiche** e **API** standardizzate per costruire applicazioni aziendali. Fornisce servizi già pronti per la gestione di:

* **Connessione ai database**.
* **Sicurezza**.
* **Web services**.
* **Componenti distribuiti** (EJB, ecc.).
* **Applicazioni web**.

Java EE è basato su **container** che gestiscono risorse e ciclo di vita delle applicazioni, eliminando molte complessità per lo sviluppatore.

**2. Connessione al Database**

Ecco i modi principali per collegarsi a un database in Java EE, dal più vecchio al più moderno:

**a. JDBC (Java Database Connectivity)**

* **Cos'è:** API di basso livello per connettersi a un database usando query SQL dirette.
* **Come si usa:**
  1. Caricare il driver JDBC.
  2. Aprire una connessione (DriverManager.getConnection()).
  3. Creare uno Statement o PreparedStatement per eseguire query.
  4. Gestire i risultati (ResultSet).
  5. Chiudere risorse manualmente.
* **Pro:** Molto semplice.
* **Contro:** Richiede molto codice boilerplate e non è scalabile per progetti grandi.

**b. JPA (Java Persistence API)**

* **Cos'è:** API che permette la gestione degli oggetti (modello orientato agli oggetti) e li mappa direttamente alle tabelle del database (ORM - Object Relational Mapping).
* **Implementazioni comuni:** Hibernate, EclipseLink.
* **Come si usa:**
  1. Definire entità con annotazioni come @Entity e @Table.
  2. Utilizzare un EntityManager per gestire operazioni CRUD.
* **Pro:** Scrittura di query più semplice e meno codice boilerplate.
* **Contro:** Curva di apprendimento iniziale.

**c. EJB (Enterprise Java Beans) con JPA**

* **Cos'è:** Utilizzo di **Session Beans** (EJB) per logica aziendale e gestione transazionale automatica.
* **Come si usa:**
  1. Annotare una classe come @Stateless o @Stateful.
  2. Usare JPA per la persistenza.
  3. Container gestisce in automatico transazioni e pooling di risorse.
* **Pro:** Gestione automatica di transazioni.
* **Contro:** Complessità aggiuntiva per progetti piccoli.

***d. Spring Data JPA (Estensione di Spring Framework)***

* ***Cos'è:*** *Framework che semplifica ulteriormente JPA, eliminando quasi tutto il codice boilerplate.*
* ***Come si usa:***
  1. *Creare repository usando interfacce come JpaRepository.*
  2. *Usare metodi predefiniti o definire query con nomi di metodo (findByNome).*
* ***Pro:*** *Super intuitivo e veloce da configurare.*
* ***Contro:*** *Legato al framework Spring.*

***e. Micronaut, Quarkus, e Helidon***

* ***Cos'è:*** *Framework moderni per microservizi con supporto a JPA e JDBC ottimizzati.*
* ***Pro:*** *Ottimizzati per avvio rapido e ambiente cloud.*
* ***Contro:*** *Nuovi e meno documentati rispetto a Spring.*

**3. Connessione al Browser**

Dal più vecchio al più moderno:

**a. Servlets**

* **Cos'è:** API per creare applicazioni web ricevendo e rispondendo a richieste HTTP.
* **Come si usa:**
  1. Creare una classe che estende HttpServlet.
  2. Implementare i metodi doGet e doPost.
  3. Configurare il servlet in web.xml o con annotazioni (@WebServlet).
* **Pro:** Semplice per iniziare.
* **Contro:** Richiede molto codice e non è pratico per progetti grandi.

**b. JSP (JavaServer Pages)**

* **Cos'è:** Tecnologia che combina HTML e Java (scriptlet).
* **Come si usa:** Scrivere HTML con tag speciali <% %> per integrare codice Java.
* **Pro:** Semplice per creare pagine dinamiche.
* **Contro:** Mescolare logica di business e presentazione non è consigliato.

**c. JSF (JavaServer Faces)**

* **Cos'è:** Framework component-based per costruire interfacce utente web.
* **Come si usa:** Utilizzare file XML o XHTML con tag JSF per definire interfaccia e logica.
* **Pro:** Componente standard di Java EE.
* **Contro:** Non più molto popolare.

**d. Framework moderni**

1. **Spring MVC**
   * Permette di creare controller (@Controller) per gestire richieste HTTP.
   * Supporta template engine moderni come Thymeleaf o integrazione con frontend JS.
2. **Vaadin**
   * Framework che consente di costruire interfacce utente direttamente in Java senza HTML.
3. **RESTful API con JAX-RS**
   * **Cos'è:** Specifica per creare servizi RESTful.
   * **Come si usa:** Annotare metodi con @GET, @POST, @Path per rispondere alle richieste.
   * **Pro:** Facilmente combinabile con frontend moderni (Angular, React).
4. **Micronaut, Quarkus, e Helidon**
   * Semplici per creare microservizi con endpoint REST.

**4. Framework principali**

Ecco un elenco di framework popolari per lavorare con Java EE / Jakarta EE:

| **Framework** | **Uso principale** | **Pro** | **Contro** |
| --- | --- | --- | --- |
| Hibernate | ORM con JPA | Meno boilerplate | Può essere complesso |
| Spring | Tutto (Web, ORM, REST) | Ecosistema ampio, comunità attiva | Dipendenza da Spring |
| Vaadin | UI (senza HTML) | Full-stack Java per web app | Poco usato rispetto ai moderni |
| JSF | Componenti UI | Standard Java EE | Non più molto diffuso |
| Micronaut | Microservizi | Ottimizzato per cloud-native | Nuovo |
| Quarkus | Microservizi | Avvio rapido, supporto nativo | Relativamente giovane |

**5. Come scegliere?**

* **Se sei agli inizi:** Parti con JDBC per capire la base. Poi esplora JPA.
* **Se lavori a progetti grandi:** Usa Spring Boot (con Spring Data JPA).
* **Per microservizi moderni:** Valuta Micronaut o Quarkus.
* **Se lavori su legacy:** Potresti trovare JSF o EJB.

**6. Esempio Pratico**

Ecco uno schema di flusso per una moderna applicazione con Java EE:

1. **Database**: JPA con Hibernate.
2. **Backend**: Servizi REST con JAX-RS o Spring MVC.
3. **Frontend**: Angular, React, o Thymeleaf per HTML dinamico.
4. **Deployment**: Server come WildFly, Payara, o container Docker.