**Cos'è Java EE / Jakarta EE?**

**Java EE** è una piattaforma estesa della programmazione Java progettata per supportare lo sviluppo di **applicazioni aziendali** che devono gestire elevate quantità di dati, traffico e transazioni. È particolarmente adatta per applicazioni **multi-tier** (con più livelli) e **distribuite** che spesso includono funzionalità come sicurezza, persistenza dei dati, gestione delle transazioni e altro.

Jakarta EE è lo standard moderno, gestito dalla **Eclipse Foundation** dopo che Oracle ha ceduto la gestione del progetto. Anche se il nome è cambiato, le tecnologie e i concetti di base sono rimasti simili.

**Architettura e Struttura delle Applicazioni Java EE**

Le applicazioni Java EE sono tipicamente divise in **più livelli (o layer)**:

1. **Client Tier (Livello del Client):** Il livello che interagisce direttamente con l'utente, può essere un'applicazione web, desktop o mobile.
2. **Web Tier (Livello Web):** Il livello che gestisce le richieste HTTP del client. Qui si trovano componenti come **servlet** e **JavaServer Pages (JSP)**.
3. **Business Tier (Livello di Business):** Contiene la logica di business dell'applicazione. Componenti come gli **Enterprise JavaBeans (EJB)** sono presenti qui.
4. **Data Tier (Livello dei Dati):** Interagisce con il database per memorizzare e recuperare dati. È possibile utilizzare **Java Persistence API (JPA)** per mappare oggetti Java su tabelle del database.

Questa struttura è progettata per separare le responsabilità, facilitando la manutenzione e la scalabilità dell'applicazione.

**Componenti Principali di Java EE**

**1. Servlet**

* Sono classi Java che elaborano le richieste HTTP e generano risposte (tipicamente sotto forma di pagine HTML).
* Possono gestire operazioni comuni come l'input dell'utente e la generazione dinamica di contenuti.

**2. JavaServer Pages (JSP)**

* Un'estensione delle servlet che permette di inserire codice Java direttamente all'interno di pagine HTML.
* Utile per creare interfacce web dinamiche e gestire la logica di presentazione (presentation logic).
* Utilizza **Expression Language (EL)** e **JSTL (JavaServer Pages Standard Tag Library)** per ridurre il codice Java nella pagina e semplificare il markup.

**3. Enterprise JavaBeans (EJB)**

* Componenti del **Business Tier** che incapsulano la logica di business.
* Offrono funzionalità avanzate come la **gestione delle transazioni**, **sicurezza**, e **gestione delle connessioni** al database.
* Tipi principali di EJB:
  + **Session Beans:** Per la logica di business. Possono essere **stateless** (senza stato) o **stateful** (con stato).
  + **Message-Driven Beans (MDB):** Gestiscono l'integrazione asincrona tramite la messaggistica (ad es., **Java Message Service** o JMS).

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

* Un framework ORM (Object-Relational Mapping) per gestire la persistenza dei dati.
* Consente di mappare le classi Java alle tabelle del database e semplifica le operazioni CRUD (Create, Read, Update, Delete).
* **Entity Beans** e **Entità JPA** rappresentano oggetti che possono essere salvati direttamente in un database relazionale.

**5. Java Message Service (JMS)**

* Fornisce la funzionalità di messaggistica asincrona, permettendo ai componenti di scambiarsi messaggi anche quando non sono direttamente connessi.
* Utile per applicazioni distribuite che richiedono interazioni asincrone e disaccoppiate.

**6. Java Transaction API (JTA)**

* Supporta la gestione delle transazioni distribuite, necessarie per garantire l'integrità dei dati in operazioni che coinvolgono più componenti e database.
* Assicura che operazioni multiple su diverse risorse (come database e sistemi di messaggistica) siano completate in modo atomico.

**7. Java Naming and Directory Interface (JNDI)**

* Permette di accedere a servizi remoti e risorse come database, file system, e altri oggetti aziendali registrati nel **naming service**.
* Utilizzato per cercare componenti come datasource o altre risorse di sistema in modo standardizzato.

**Tecnologie di Supporto in Java EE**

**WildFly e JBoss (Application Servers)**

**WildFly** e **JBoss** sono **application server** compatibili con Java EE che eseguono le applicazioni. Gestiscono la distribuzione e l'esecuzione dei componenti come servlet, JSP, EJB, e altri. **WildFly** è la versione community, mentre **JBoss EAP** è la versione commerciale supportata da Red Hat.

**Maven (Build Tool)**

**Maven** è uno strumento di build che gestisce le dipendenze, la compilazione, il packaging e la distribuzione delle applicazioni Java EE. Ogni progetto Java EE ha tipicamente un file pom.xml che dichiara tutte le dipendenze e le configurazioni di build.

**MyBatis (Persistenza Personalizzata)**

**MyBatis** è una libreria che facilita l'accesso al database, ma, a differenza di JPA, permette una gestione più diretta e personalizzata delle query SQL. È utile quando si richiede un controllo preciso delle operazioni sul database, ad esempio per ottimizzare query SQL complesse.

**Ciclo di Vita delle Applicazioni Java EE**

Il ciclo di vita delle applicazioni Java EE in un application server come WildFly include i seguenti passaggi:

1. **Sviluppo:** Scrivi il codice dell’applicazione utilizzando componenti Java EE come servlet, EJB, e JPA.
2. **Build:** Con Maven o un altro strumento, compili e generi un file .war o .ear per distribuire l’applicazione.
3. **Distribuzione:** Carichi il file .war o .ear nel server applicativo.
4. **Esecuzione:** Il server applicativo gestisce il ciclo di vita dei componenti, inclusi il caricamento, l'inizializzazione, e la distruzione.
5. **Gestione e Monitoraggio:** Durante l'esecuzione, puoi monitorare le performance dell’applicazione, configurare la gestione della sicurezza e bilanciare il carico su più server, se necessario.

**Esempio di Flusso con Java EE**

Immaginiamo un’applicazione Java EE per una **biblioteca online** che consente agli utenti di prenotare libri.

1. **Livello di Presentazione (Client e Web Tier):**
   * L’utente accede alla pagina web della biblioteca.
   * Le richieste vengono gestite da servlet che interagiscono con pagine JSP per la visualizzazione dinamica dei contenuti (ad esempio, la lista dei libri disponibili).
2. **Livello di Business (Business Tier):**
   * Un EJB gestisce la logica di prenotazione dei libri. Controlla la disponibilità e aggiorna lo stato di prenotazione dei libri.
   * L’EJB si occupa anche della gestione delle transazioni, garantendo che tutte le operazioni legate a una prenotazione vengano eseguite correttamente.
3. **Livello dei Dati (Data Tier):**
   * Un modulo JPA gestisce la persistenza dei dati della biblioteca nel database.
   * Ogni volta che un utente prenota un libro, i dati vengono salvati nel database tramite JPA.
4. **Messaggistica Asincrona (opzionale):**
   * Se la biblioteca offre notifiche di prenotazione via email, si potrebbe usare JMS per inviare un messaggio a un sistema di invio notifiche in modo asincrono.

**In sintesi**

Java EE / Jakarta EE è una piattaforma potente che semplifica la gestione di applicazioni aziendali complesse grazie a una serie di componenti e API. L'architettura a più livelli e il supporto integrato per aspetti essenziali come la gestione delle transazioni, la sicurezza, e la persistenza dei dati rendono Java EE particolarmente adatta per ambienti di produzione.

**1. Cos'è MySQL e a cosa serve?**

**MySQL** è un sistema di gestione di database (DBMS - Database Management System).

Un DBMS serve a memorizzare, gestire e recuperare grandi quantità di dati in modo organizzato.

In MySQL, i dati sono organizzati in **tabelle** (come dei fogli Excel), e tu puoi interagire con questi dati usando **query SQL** per fare operazioni come aggiungere, modificare, eliminare o leggere i dati.

**2. Cos'è DBeaver e perché lo usiamo?**

**DBeaver** è un programma che ti permette di connetterti a un database MySQL (o altri DBMS) e interagire con esso senza dover scrivere solo codice SQL. È come un'interfaccia grafica che facilita la gestione dei database, dove puoi visualizzare tabelle, eseguire query e modificare dati senza complicazioni. Per esempio, puoi vedere il contenuto di una tabella o creare nuove tabelle direttamente dall’interfaccia.

**3. Cos'è Eclipse e a cosa serve nel collegamento con il database?**

**Eclipse** è un ambiente di sviluppo integrato (IDE - Integrated Development Environment) usato per scrivere codice in vari linguaggi di programmazione, come Java. Con Eclipse, puoi creare applicazioni Java che interagiscono con MySQL, per esempio per mostrare dati in una finestra o gestirli direttamente tramite il programma.

**4. Come collegare MySQL con Eclipse usando DBeaver?**

Ora che sai a cosa servono ciascuno di questi strumenti, vediamo come farli lavorare insieme!

**Passaggi principali:**

1. **Installa e configura MySQL**:
   * Assicurati che MySQL sia installato e in esecuzione sul tuo computer.
   * Crea un database (puoi farlo da DBeaver, come vedremo dopo) e imposta un utente con le autorizzazioni necessarie per accedervi.
2. **Collegati a MySQL da DBeaver**:
   * Apri DBeaver e scegli **Database → Nuova connessione**.
   * Seleziona **MySQL** come tipo di database.
   * Inserisci le credenziali richieste: nome utente, password, e se hai creato un database specifico, inserisci il nome del database. Dopo aver inserito queste informazioni, clicca su **Finish**.
   * A questo punto, DBeaver si connetterà a MySQL e potrai vedere il tuo database nell’interfaccia. Qui puoi creare tabelle, inserire dati e fare tutte le operazioni che vuoi per configurare il database.
3. **Configura Eclipse per connetterti a MySQL**:
   * Prima di tutto, in Eclipse, avrai bisogno di un **driver JDBC (Java Database Connectivity)** per MySQL. Questo è un file .jar che permette alla tua applicazione Java di comunicare con il database.
   * Scarica il driver JDBC per MySQL (solitamente chiamato mysql-connector-java.jar) dal sito ufficiale MySQL.
   * Aggiungi il file .jar al progetto in Eclipse. Per farlo, fai clic col tasto destro sul progetto → **Properties** → **Java Build Path** → **Libraries** → **Add External JARs** e seleziona il file del driver.
4. **Scrivi codice Java per connetterti al database**:
   * Ora puoi scrivere del codice Java in Eclipse per connetterti a MySQL. Qui un esempio di base per collegarti:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

​

public class DatabaseConnection {

  public static void main(String[] args) {

      String url = "jdbc:mysql://localhost:3306/nome\_del\_tuo\_database";

      String user = "il\_tuo\_username";

      String password = "la\_tua\_password";

​

      try {

          Connection connection = DriverManager.getConnection(url, user, password);

          System.out.println("Connessione riuscita!");

          connection.close();

       } catch (SQLException e) {

          e.printStackTrace();

       }

   }

}

​

* + Qui, localhost:3306 è l’indirizzo locale del tuo server MySQL e nome\_del\_tuo\_database è il nome del database che hai creato. Cambia il\_tuo\_username e la\_tua\_password con quelli che usi per accedere a MySQL.

1. **Esegui il programma e verifica la connessione**:
   * Esegui il programma Java in Eclipse. Se tutto è configurato correttamente, dovresti vedere il messaggio **"Connessione riuscita!"** nella console di Eclipse, il che significa che il tuo programma Java ha stabilito una connessione con MySQL.

**Riassunto**

1. **MySQL** gestisce i dati.
2. **DBeaver** ti permette di vedere e gestire i dati facilmente.
3. **Eclipse** è l’ambiente dove scrivi il codice Java per interagire con il database.

Questi strumenti insieme ti permettono di creare applicazioni Java che possono aggiungere, modificare, eliminare e leggere dati da un database, aprendoti molte possibilità per creare applicazioni più interattive e potenti!

Ecco un riassunto delle tecnologie che stai utilizzando per Java Enterprise, con una breve descrizione della funzione di ciascuna:

**1. MySQL - Database**

* **A cosa serve**: MySQL è un **sistema di gestione di database** (DBMS) che organizza e memorizza i dati in tabelle. Le tue applicazioni possono utilizzare MySQL per archiviare e recuperare informazioni strutturate, come dati di utenti, ordini, cataloghi, ecc.

**2. DBeaver - Interfaccia per Database**

* **A cosa serve**: DBeaver è un **client grafico per database**. Ti permette di connetterti a MySQL (o altri DBMS), esplorare tabelle, eseguire query SQL e gestire i dati senza dover scrivere tutto il codice SQL manualmente. È utile per configurare e controllare il contenuto del database.

**3. Eclipse - Ambiente di Sviluppo (IDE)**

* **A cosa serve**: Eclipse è un **IDE (Integrated Development Environment)** dove scrivi il tuo codice Java. È qui che sviluppi e testi il tuo programma, configurando anche le connessioni al database e al server. Con Eclipse, puoi eseguire e fare debug del codice per assicurarti che funzioni come previsto.

**4. WildFly - Application Server**

* **A cosa serve**: WildFly è un **server per applicazioni Java Enterprise**. Gestisce ed esegue applicazioni Java basate su Java EE (o Jakarta EE), permettendo di gestire web application complesse, con funzionalità avanzate come gestione di sessioni utente, transazioni, sicurezza e connessioni ai database.

**5. Red Hat (JBoss EAP) - Versione aziendale di WildFly**

* **A cosa serve**: Red Hat offre **JBoss EAP (Enterprise Application Platform)**, la versione commerciale e certificata di WildFly, con supporto tecnico e stabilità a lungo termine per ambienti di produzione aziendali. WildFly è open source e flessibile per i test e sviluppo; JBoss EAP è usato in ambienti dove è richiesto supporto costante e sicurezza aziendale.

**Schema generale di utilizzo**

* **Eclipse**: Scrivi il codice.
* **WildFly/JBoss EAP**: Esegui l'applicazione, che gestisce le richieste degli utenti e si connette al database.
* **DBeaver/MySQL**: Gestisci i dati del database, a cui l’applicazione accede tramite il server per archiviare e recuperare informazioni.

Questo set di strumenti ti permette di sviluppare, gestire ed eseguire applicazioni Java enterprise complete e affidabili!

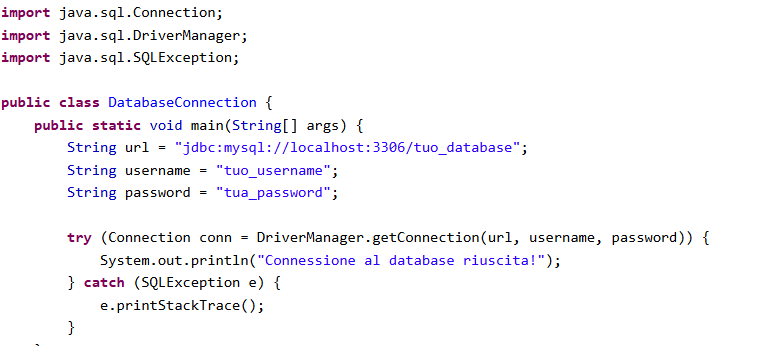
Esistono diversi modi per connettere eclipse a un database?

Sì, ci sono diversi modi per connettere **Eclipse** a un database! Eclipse, come ambiente di sviluppo integrato (IDE), supporta varie modalità per lavorare con database in progetti Java, specialmente in contesti Java EE o Jakarta EE. Ecco le principali opzioni:

**1. Connessione Tramite JDBC (Java Database Connectivity)**

JDBC è l'API standard per connettere applicazioni Java a database relazionali come MySQL, PostgreSQL, Oracle, ecc.

* **Passaggi per usare JDBC in Eclipse:**
  1. Aggiungi il **driver JDBC** del database al tuo progetto in Eclipse. Puoi fare questo scaricando il driver (un file .jar) e includendolo nella build del progetto (attraverso il Build Path di Eclipse).
  2. Scrivi il codice Java per creare una connessione al database. Ad esempio:



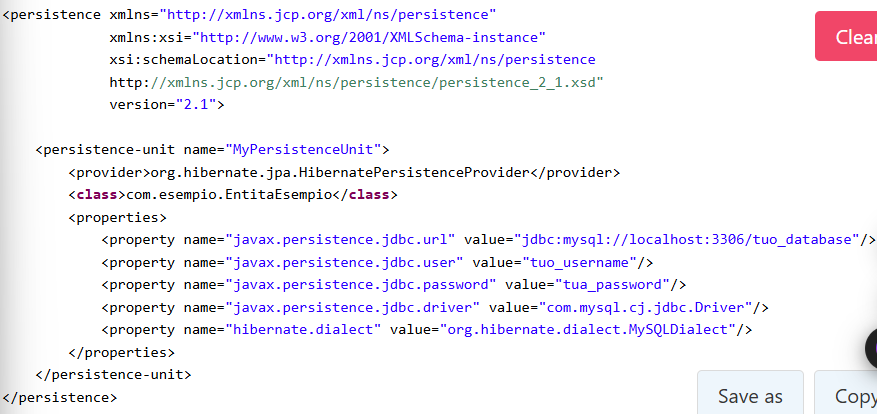
* 1. Configura le impostazioni di connessione nel tuo codice: URL, username e password.

Questa modalità è semplice ma richiede di gestire manualmente la connessione e le query SQL. È adatta per progetti di dimensioni ridotte o per chi inizia a esplorare JDBC.

**2. Connessione con JPA (Java Persistence API)**

JPA è uno standard Java per la mappatura degli oggetti Java su database relazionali (ORM). In Eclipse, puoi configurare JPA per connetterti a un database senza dover scrivere query SQL manuali per ogni operazione, poiché JPA gestisce queste operazioni con annotazioni e query orientate agli oggetti.

* **Passaggi per usare JPA in Eclipse:**
  1. Assicurati di avere **JPA e un provider ORM** (come **Hibernate** o **EclipseLink**). Eclipse supporta entrambi.
  2. Aggiungi le dipendenze (ad es., tramite Maven o scaricando i file .jar necessari).
  3. Configura un file persistence.xml che specifica le impostazioni di connessione e il provider ORM. Esempio:



* 1. Definisci le entità JPA con annotazioni come @Entity, @Id, @Column, ecc., per mappare le tabelle del database.

Questa opzione è ideale per applicazioni aziendali Java EE e progetti più grandi dove è necessaria una gestione automatizzata della persistenza.

**3. Uso di MyBatis per la Mappatura Manuale delle Query SQL**

**MyBatis** è una libreria di mappatura SQL per Java che permette di mappare le query SQL a oggetti Java in modo più controllato rispetto a JPA, mantenendo però l’astrazione delle query.

* **Passaggi per usare MyBatis in Eclipse:**
  1. Aggiungi MyBatis come dipendenza al progetto.
  2. Configura il file di configurazione mybatis-config.xml per definire le impostazioni di connessione.
  3. Definisci i **mapper XML** o le **annotazioni** per creare query SQL personalizzate e collegarle a metodi Java.

MyBatis è utile per applicazioni in cui sono richieste query SQL personalizzate e dettagliate e un controllo più diretto sulle operazioni del database.

**4. Connessione con Eclipse Data Tools Platform (DTP)**

Eclipse include il plugin **DTP** per gestire e visualizzare i database direttamente dall’ambiente Eclipse. È comodo per esplorare il database e visualizzare i dati.

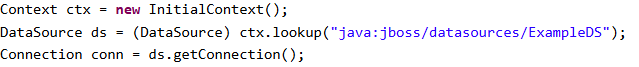
* **Passaggi per usare DTP:**
  1. Apri **Database Perspective** in Eclipse.
  2. Configura una **connessione al database** con un driver JDBC.
  3. Usa l’interfaccia grafica di Eclipse per eseguire query, navigare tra tabelle e visualizzare i dati.

DTP non è un’opzione di connessione per l'applicazione stessa, ma è utile per testare le connessioni e ottenere una visione d'insieme del database.

**5. Connessione con DataSource e JNDI (Java Naming and Directory Interface)**

Questa modalità è tipicamente usata negli **application server** come WildFly o Tomcat. Un DataSource consente di gestire le connessioni al database in modo centralizzato e configurabile. Si usa in progetti Java EE distribuiti dove il database viene configurato esternamente al codice.

* **Passaggi per usare DataSource con JNDI:**
  1. Configura il datasource nel tuo server applicativo (es., in WildFly o Tomcat).
  2. Accedi al datasource tramite un **lookup JNDI** all'interno del codice. Esempio:



Questo approccio è preferito per applicazioni Java EE che girano su server, poiché permette di gestire le connessioni in modo centralizzato e sicuro.

**Quale Metodo Scegliere?**

La scelta dipende dal tipo di progetto:

* **Progetti di piccola-media dimensione**: JDBC può essere sufficiente.
* **Progetti aziendali di media-grande dimensione**: JPA con Hibernate o EclipseLink è consigliato per la gestione automatizzata delle operazioni di database.
* **Progetti complessi con richieste SQL specifiche**: MyBatis offre flessibilità nelle query.
* **Contesti con server applicativi**: DataSource e JNDI sono ideali per gestire connessioni centralizzate.

quindi quale sarebbe un flusso di lavoro per creare un progetto biblioteca ad esempio?

Un flusso di lavoro per creare un progetto **biblioteca** utilizzando Java EE può essere suddiviso in diverse fasi, ognuna delle quali comprende strumenti e tecnologie Java Enterprise. Questo esempio seguirà un approccio che sfrutta JPA per la gestione dei dati, un server applicativo (come WildFly) e Maven per la gestione delle dipendenze.

Ecco come impostare il progetto.

**1. Pianificazione del Progetto**

* **Obiettivi principali**: Identificare le funzionalità che vuoi realizzare. Ad esempio:
  + Gestire libri e utenti (aggiungere, modificare, eliminare).
  + Consentire agli utenti di prendere in prestito e restituire libri.
  + Visualizzare informazioni come disponibilità, genere e autore di ciascun libro.
* **Requisiti tecnici**:
  + Database per archiviare dati relativi a libri, utenti, e prestiti.
  + Java EE per la gestione delle componenti enterprise (servlet, JPA, ecc.).
  + Un’interfaccia utente, possibilmente una pagina JSP o un framework JavaScript se vuoi un front-end più avanzato.

**2. Setup di Eclipse e Maven**

* Crea un progetto **Maven** in Eclipse (Maven facilita la gestione delle dipendenze e delle configurazioni).
* Configura il file pom.xml di Maven per includere le dipendenze necessarie:
  + **JPA** (con provider come Hibernate).
  + **Servlet API** per creare servlet che rispondano a richieste HTTP.
  + **MySQL JDBC driver** (o il driver adatto al database scelto).

|  |
| --- |
| <dependencies>     <!-- JPA con Hibernate -->     <dependency>         <groupId>org.hibernate</groupId>         <artifactId>hibernate-core</artifactId>         <version>5.4.32.Final</version>     </dependency>       <!-- Java Servlet API -->     <dependency>         <groupId>javax.servlet</groupId>         <artifactId>javax.servlet-api</artifactId>         <version>4.0.1</version>         <scope>provided</scope>     </dependency>       <!-- JDBC driver (ad esempio, MySQL) -->     <dependency>         <groupId>mysql</groupId>         <artifactId>mysql-connector-java</artifactId>         <version>8.0.26</version>     </dependency>  </dependencies>  ​ |

**3. Configurazione del Database**

* **Crea un database** per la biblioteca (ad esempio, biblioteca\_db) in MySQL.
* **Definisci le tabelle principali**: libri, utenti, prestiti, ecc.
* Configura le credenziali e l'URL di connessione nel file persistence.xml, all'interno di src/main/resources/META-INF.

|  |
| --- |
| <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"              xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence              http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_1.xsd"              version="2.1">  ​     <persistence-unit name="BibliotecaPU">         <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>         <**class**>com.esempio.Biblioteca.Libro</**class**>         <**class**>com.esempio.Biblioteca.Utente</**class**>         <properties>             <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/biblioteca\_db"/>             <property name="javax.persistence.jdbc.user" value="username"/>             <property name="javax.persistence.jdbc.password" value="password"/>             <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>             <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>         </properties>     </persistence-unit>  </persistence>  ​ |

**4. Creazione delle Entità JPA**

* **Definisci le entità** per il dominio della biblioteca, come Libro, Utente, e Prestito, usando annotazioni JPA (@Entity, @Id, @OneToMany, ecc.).
* Esempio di entità Libro:

|  |
| --- |
| **import** javax.persistence.\*;  ​  @Entity  **public** **class** Libro {     @Id     @GeneratedValue(strategy = GenerationType.IDENTITY)  **private** Long id;  ​  **private** String titolo;  **private** String autore;  **private** String genere;  **private** boolean disponibile;  ​     // Getter e setter  }  ​ |

**5. Creazione di Servlet e JSP**

* **Servlet per la gestione delle richieste HTTP**: Crea servlet che rispondano a varie richieste (come aggiungere libri, prendere in prestito, restituire libri, ecc.). Ogni servlet può corrispondere a un’azione specifica.

Esempio di servlet:

|  |
| --- |
| **import** javax.servlet.\*;  **import** javax.servlet.http.\*;  **import** java.io.IOException;  ​  **public** **class** AggiungiLibroServlet **extends** HttpServlet {  **protected** void doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {         // Logica per aggiungere un libro nel database    }  }  ​ |

**JSP per l'interfaccia utente**: Utilizza pagine JSP per mostrare la lista dei libri, il modulo di registrazione, ecc. Puoi inserire codice Java nelle pagine JSP per mostrare i dati provenienti dalle servlet.

**6. Utilizzo di WildFly come Application Server**

* **Configura WildFly** per il deploy dell'applicazione. Aggiungi il progetto a WildFly in Eclipse e configura il server per puntare alla versione corretta di JDK e JRE.
* **Deploy dell'applicazione**: Verifica che l’app sia accessibile all’indirizzo configurato per WildFly (ad es., http://localhost:8080/nomeProgetto).

**7. Configurazione di Maven e MyBatis (opzionale)**

* Puoi configurare **MyBatis** se vuoi avere più controllo sulle query SQL, soprattutto per query complesse. In alternativa, puoi usare direttamente **JPA**.
* Con Maven, puoi facilmente aggiungere altre librerie o configurazioni, se necessario.

**8. Testing e Debugging**

* **Testa l'applicazione** per verificare che le operazioni CRUD funzionino come previsto. Assicurati che la gestione del prestito di libri e le transazioni siano coerenti (ad esempio, un libro non può essere prestato due volte).
* Utilizza il debugger di Eclipse per risolvere eventuali errori e assicurarti che i dati vengano gestiti correttamente.

**Riassunto del Flusso di Lavoro**

1. **Definisci requisiti e database**.
2. **Crea il progetto Maven in Eclipse** e configura le dipendenze.
3. **Definisci il modello dati** con entità JPA.
4. **Crea servlet** per gestire le operazioni di aggiunta, visualizzazione, prestito, restituzione di libri, ecc.
5. **Utilizza WildFly come server** e configura il deploy.
6. **Testa e ottimizza** per garantire un funzionamento corretto.