JPA e Hibernate

**Introduzione a JPA e Hibernate**

* **JPA (Java Persistence API)**:
  + È una specifica definita da Java per la persistenza degli oggetti in database relazionali.
  + Fornisce un linguaggio comune per gestire la persistenza dei dati attraverso ORM (Object-Relational Mapping).
  + È agnostico rispetto all'implementazione.
  + Framework che implementano JPA: Hibernate, EclipseLink, OpenJPA, ecc.
* **Hibernate**:
  + È una delle implementazioni più popolari di JPA.
  + Offre funzionalità aggiuntive rispetto a JPA, come caching avanzato, gestione delle entità disconnesse, criteri dinamici.

**Concetti chiave di JPA**

**1. Entità**

* Una **entità** rappresenta una tabella in un database.
* Ogni riga della tabella è rappresentata da un'istanza della classe dell'entità.

Esempio di classe entità:

|  |
| --- |
| **import** jakarta.persistence.\*;  ​  @Entity  @Table(name = "film")  **public** **class** Film {     @Id     @GeneratedValue(strategy = GenerationType.IDENTITY)  **private** Long id;  ​     @Column(nullable = false)  **private** String titolo;  ​     @Column  **private** String regista;  ​     // Getter e Setter  }  ​ |

**2. Annotazioni principali di JPA**

* **@Entity**: Indica che la classe è un'entità.
* **@Table**: Specifica il nome della tabella nel database (opzionale).
* **@Id**: Indica il campo chiave primaria.
* **@GeneratedValue**: Configura la generazione automatica della chiave primaria. Strategie principali:
  + GenerationType.IDENTITY: Usa l'auto-incremento del database.
  + GenerationType.SEQUENCE: Usa una sequenza predefinita nel database.
* **@Column**: Personalizza i dettagli di una colonna (nome, nullable, lunghezza, ecc.).
* **@OneToOne**, **@OneToMany**, **@ManyToOne**, **@ManyToMany**: Definiscono le relazioni tra entità.

**3. Relazioni tra entità**

Le relazioni possono essere di diversi tipi:

* **One-to-One**:  
  Esempio: Una persona ha una carta d'identità.

|  |
| --- |
| @OneToOne  @JoinColumn(name = "carta\_id")  **private** CartaIdentita carta;  ​ |

* **One-to-Many**:  
  Esempio: Un autore ha più libri

|  |
| --- |
| @OneToMany(mappedBy = "autore", cascade = CascadeType.ALL)  **private** List<Libro> libri;  ​ |

* **Many-to-Many**:  
  Esempio: Più studenti possono frequentare più corsi.

|  |
| --- |
| @ManyToMany  @JoinTable(     name = "studente\_corso",     joinColumns = @JoinColumn(name = "studente\_id"),     inverseJoinColumns = @JoinColumn(name = "corso\_id")  )  **private** List<Corso> corsi;  ​ |

**4. Lifecycle delle Entità**

* **Transient**: Oggetto creato ma non associato al database.
* **Managed**: Oggetto gestito da Hibernate (entità persistente).
* **Detached**: Oggetto disconnesso dal contesto di persistenza.
* **Removed**: Oggetto in fase di rimozione dal database.

**5. Fetching (Lazy vs Eager)**

* **FetchType.LAZY** (default): I dati delle relazioni vengono caricati solo quando richiesti.
* **FetchType.EAGER**: I dati delle relazioni vengono caricati immediatamente.

Esempio:

|  |
| --- |
| @OneToMany(fetch = FetchType.LAZY)  **private** List<Libro> libri;  ​ |

Best Practice: Evitare l'EAGER fetching per collezioni di grandi dimensioni per migliorare le performance.

**6. Eventi e Callback**

Annotazioni per intercettare eventi nel ciclo di vita delle entità:

* @**PrePersist**, @**PostPersist**: Prima o dopo la persistenza.
* @**PreUpdate**, @**PostUpdate**: Prima o dopo un aggiornamento.
* @**PreRemove**, @**PostRemove**: Prima o dopo la rimozione.

Esempio:

|  |
| --- |
| @Entity  **public** **class** Film {     @PrePersist  **public** void prePersist() {         System.out.println("Entità in fase di persistenza");    }  }  ​ |

**Concetti chiave di Hibernate**

**1. Sessione e EntityManager**

* **EntityManager**: Interfaccia standard JPA per eseguire operazioni CRUD.
* **Sessione**: Specifica di Hibernate, equivalente a EntityManager ma con funzionalità aggiuntive.

Esempio di utilizzo di EntityManager:

|  |
| --- |
| EntityManagerFactory emf = Persistence.createEntityManagerFactory("mio\_persistence\_unit");  EntityManager em = emf.createEntityManager();  ​  em.getTransaction().begin();  Film film = **new** Film();  film.setTitolo("Inception");  em.persist(film);  em.getTransaction().commit();  ​  em.close();  ​ |

**2. Cache di Hibernate**

* **Cache di Primo Livello (L1)**:
  + Sempre attiva, per ogni sessione.
  + Contiene le entità già caricate o persistite.
* **Cache di Secondo Livello (L2)**:
  + Condivisa tra sessioni diverse.
  + Richiede configurazioni aggiuntive (esempio: EHCache, Infinispan).

**3. Query in Hibernate**

* **JPQL (Java Persistence Query Language)**: Linguaggio basato su oggetti per interagire con il database.  
  Esempio:

|  |
| --- |
| String jpql = "SELECT f FROM Film f WHERE f.titolo = :titolo";  TypedQuery<Film> query = em.createQuery(jpql, Film.**class**);  query.setParameter("titolo", "Inception");  List<Film> risultati = query.getResultList();  ​ |

**Native Query**: Permette di eseguire query SQL direttamente sul database.  
Esempio:

|  |
| --- |
| Query query = em.createNativeQuery("SELECT \* FROM film WHERE titolo = ?", Film.**class**);  query.setParameter(1, "Inception");  List<Film> risultati = query.getResultList();  ​ |

**Configurazione di JPA e Hibernate**

**1. Il file persistence.xml**

Esempio di configurazione base:

|  |
| --- |
| <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.1">     <persistence-unit name="mio\_persistence\_unit">         <**class**>com.example.Film</**class**>         <properties>             <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>             <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:testdb"/>             <property name="javax.persistence.jdbc.user" value="sa"/>             <property name="javax.persistence.jdbc.password" value=""/>             <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>             <property name="hibernate.hbm2ddl.auto" value="update"/>         </properties>     </persistence-unit>  </persistence>  ​ |

**2. Transazioni**

* JPA utilizza transazioni per garantire l'integrità dei dati.
* Le transazioni possono essere gestite manualmente o con **JTA** in ambienti enterprise.

Esempio di gestione manuale:

|  |
| --- |
| em.getTransaction().begin();  // Operazioni sul database  em.getTransaction().commit();  ​ |

**3. Tipi di mapping**

* **Mapping ORM**: Conversione tra classi Java e tabelle.
* **Mapping degli attributi**: Conversione tra campi Java e colonne.

**Best Practices**

1. Utilizzare **Named Queries** per query riutilizzabili:

|  |
| --- |
| @NamedQuery(name = "Film.findByTitolo", query = "SELECT f FROM Film f WHERE f.titolo = :titolo")  ​ |

1. Ottimizzare il caching per ridurre l'accesso al database.
2. Evitare di caricare troppi dati con **Eager Fetching** se non necessario.