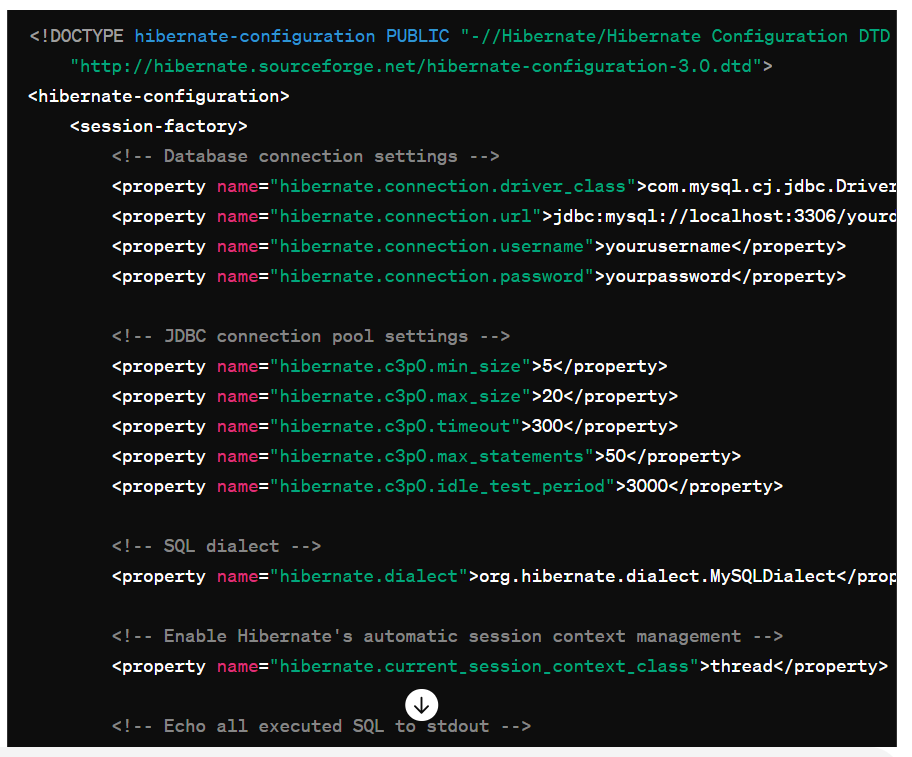
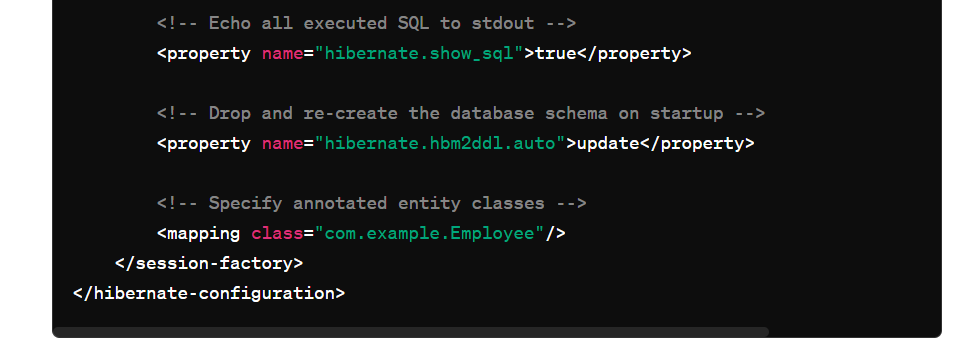
Certainly! Below is an example of a simple Hibernate application. This example includes setting up the configuration, mapping an entity to a database table, and performing basic CRUD operations.

**1. Setup Hibernate Configuration**

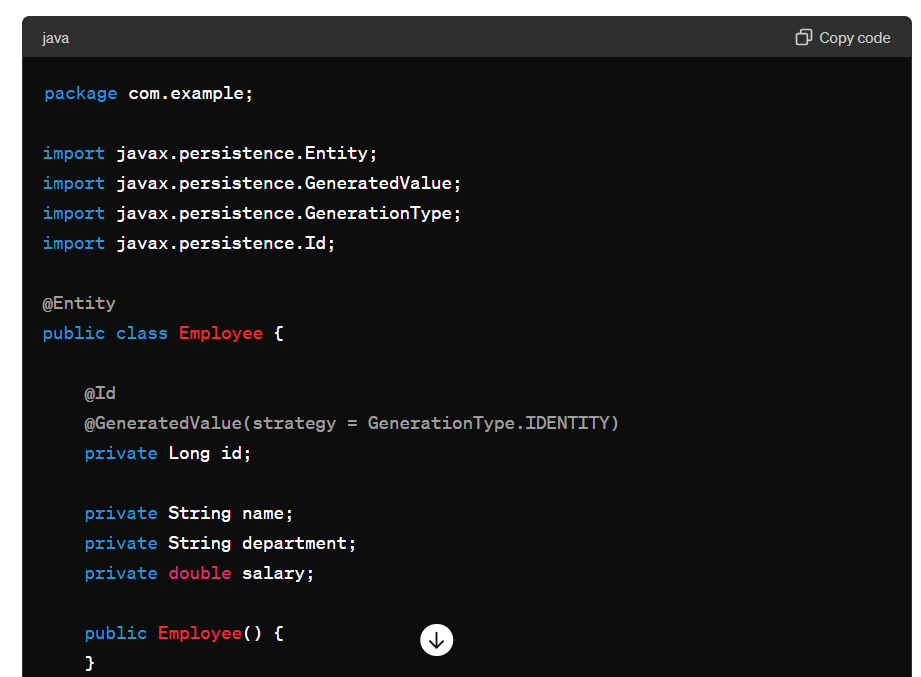
Create a configuration file named **hibernate.cfg.xml**:

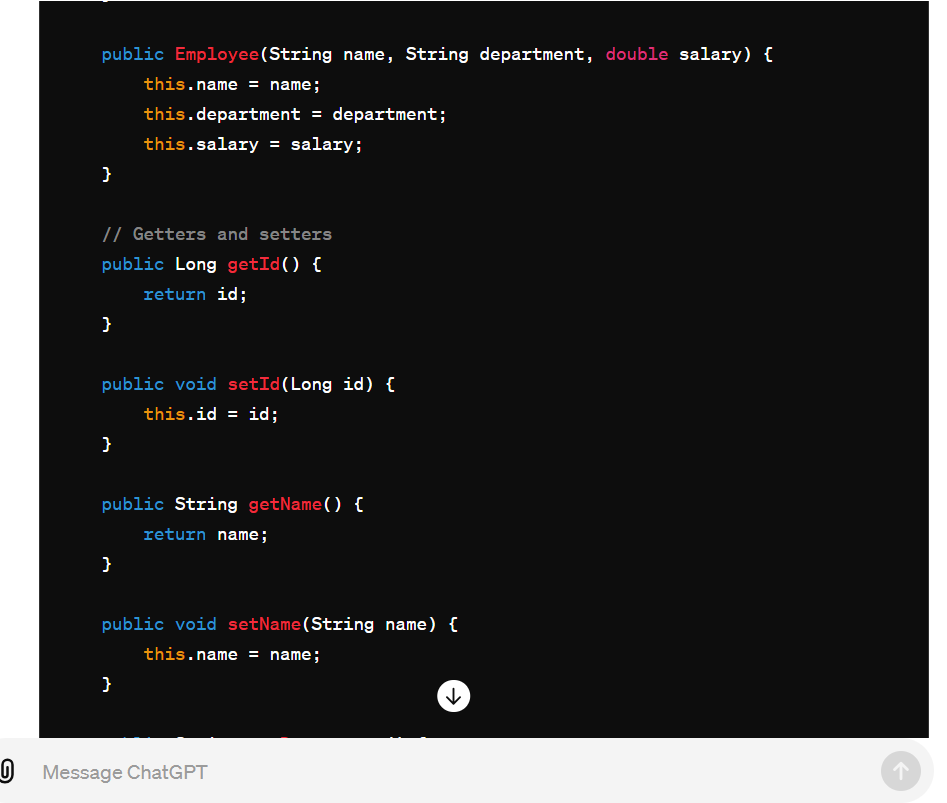




### 2. Create an Entity Class

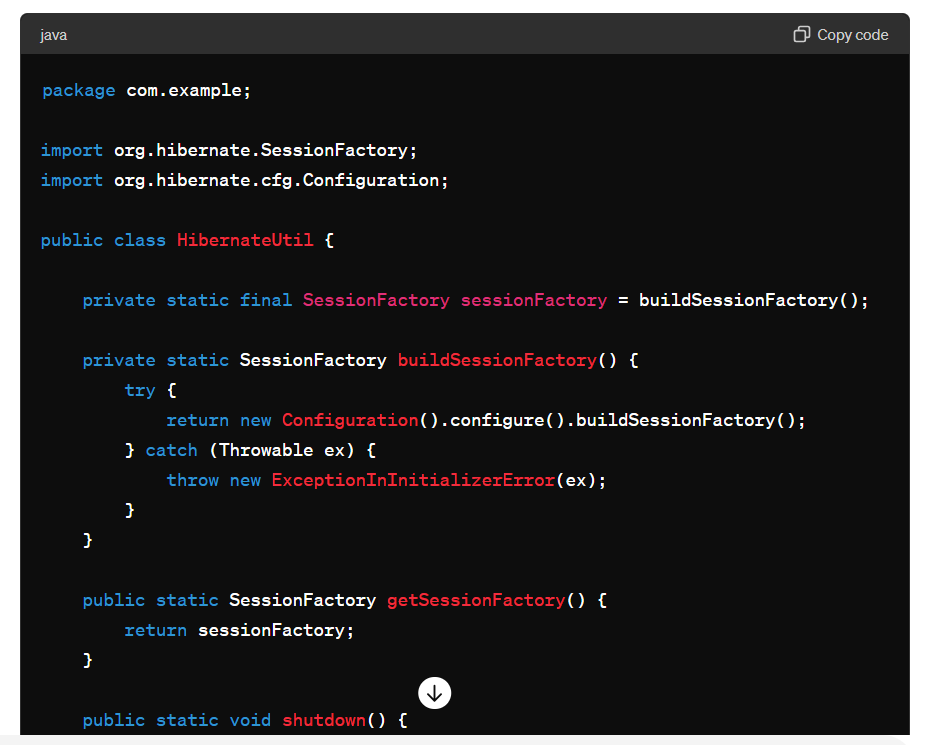
Create a class named **Employee.java**:

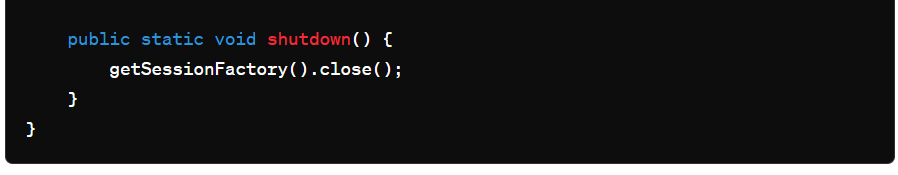




### 3. Hibernate Utility Class

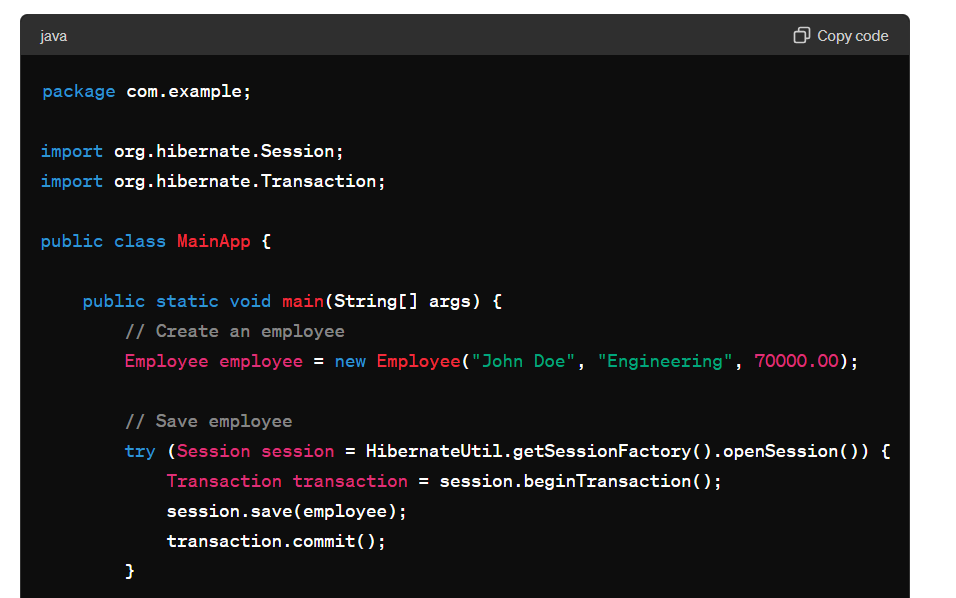
Create a utility class for Hibernate session factory management named **HibernateUtil.java**:





### 4. Main Application

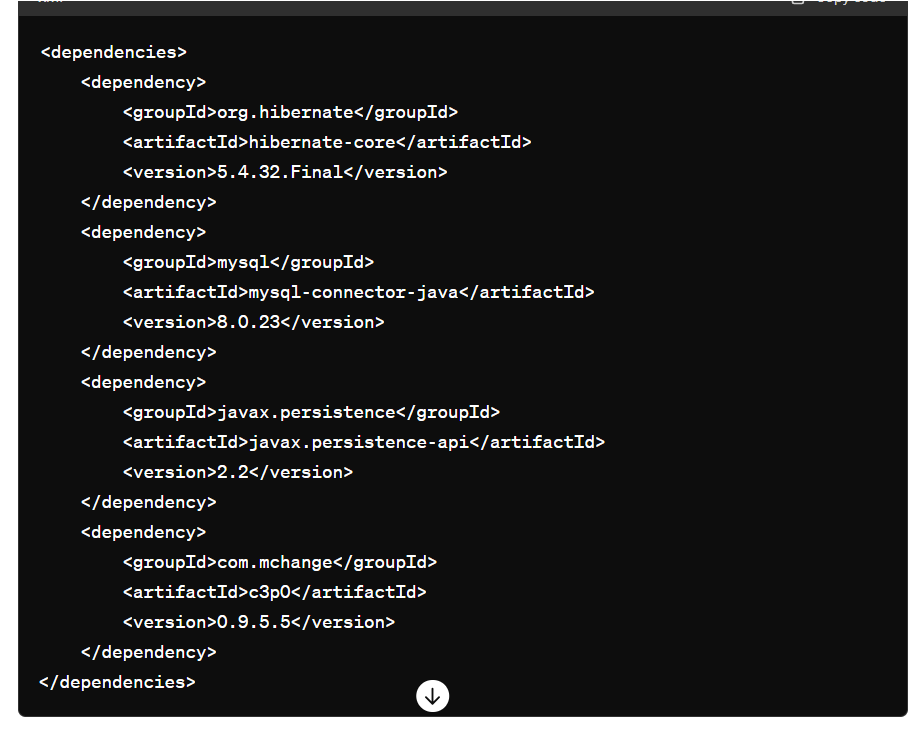
Create a main class to perform CRUD operations named **MainApp.java**:





### Maven Dependencies

Ensure you have the necessary dependencies in your **pom.xml**:



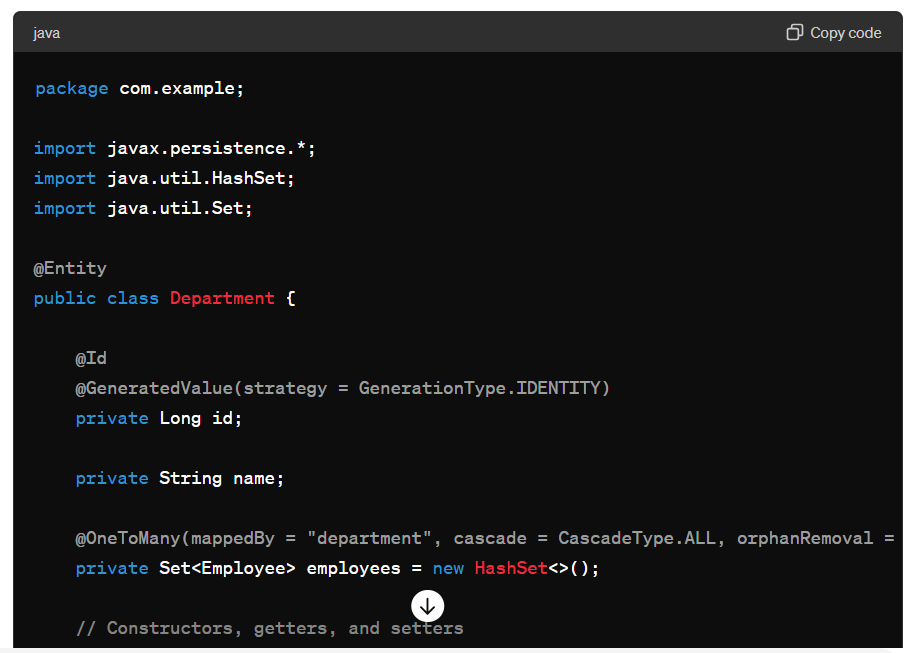
This is a simple example to demonstrate the basic setup and usage of Hibernate for CRUD operations.

**Types :**

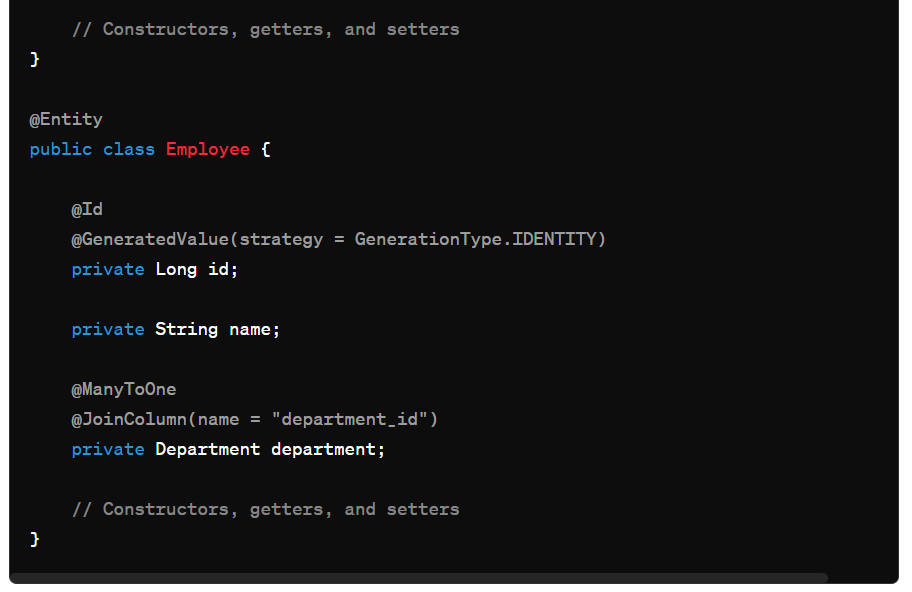
### 1. One-to-Many Relationship

**Entities: Department and Employee**

In a one-to-many relationship, one entity (e.g., **Department**) is related to multiple instances of another entity (e.g., **Employee**). This is often represented with a collection in the entity class.



@OneToMany(mappedBy = "department", cascade = CascadeType.ALL, orphanRemoval = true)



#### Explanation:

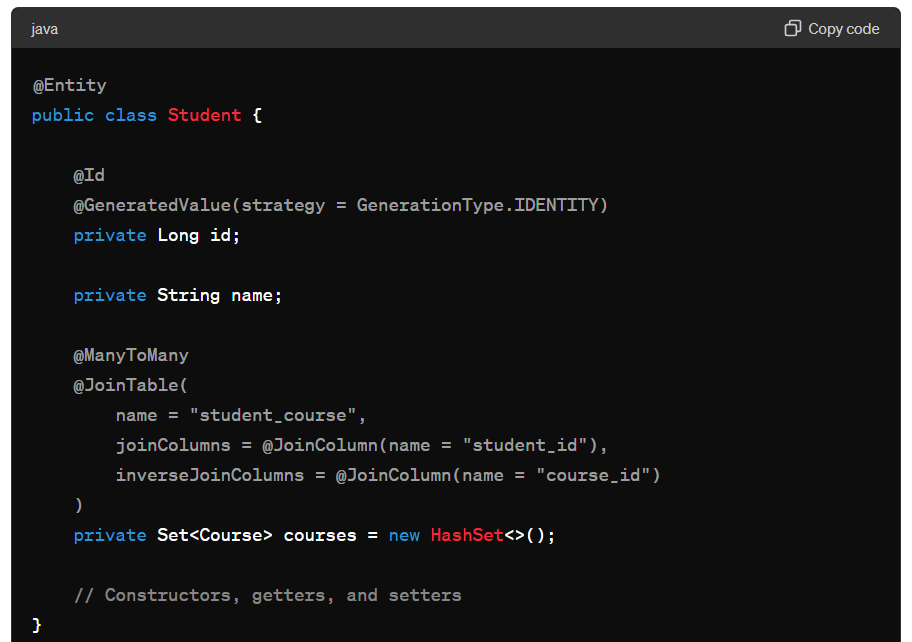
* **Department Entity**: Contains a set of **Employee** objects. The **@OneToMany** annotation defines the relationship, and **mappedBy = "department"** specifies that the **department** field in **Employee** is the owner of the relationship.
* **Employee Entity**: Each **Employee** has a reference to a **Department** with the **@ManyToOne** annotation and a foreign key column **department\_id**.

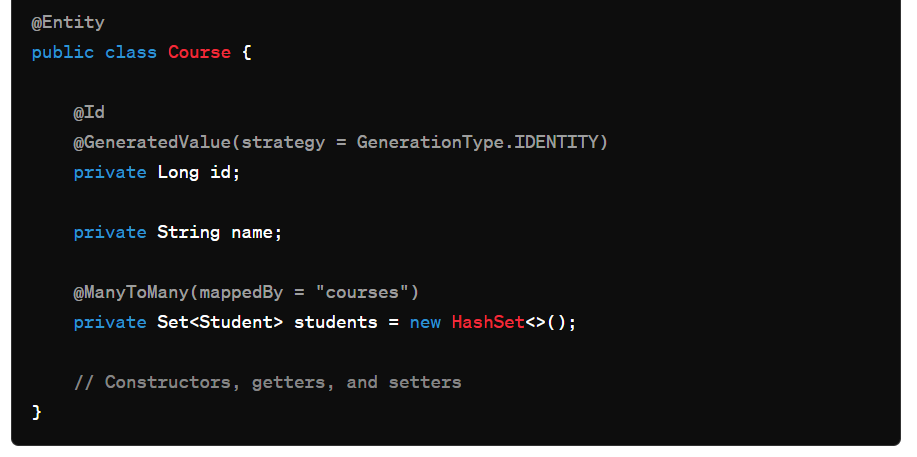
### 2. Many-to-Many Relationship

#### Explanation:

In a many-to-many relationship, multiple instances of one entity (e.g., **Student**) can be associated with multiple instances of another entity (e.g., **Course**). This relationship usually requires a join table to manage the associations.

#### Code Example:





#### Explanation:

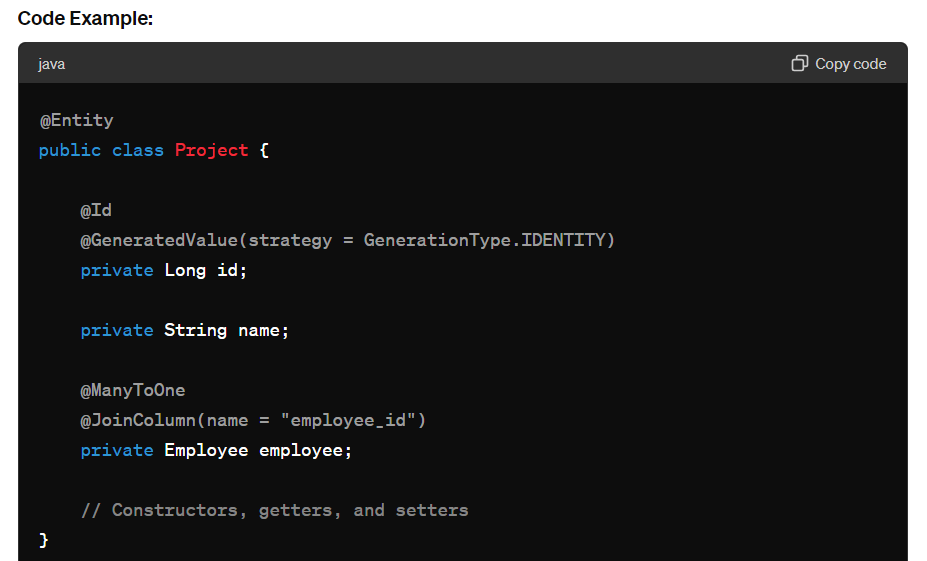
* **Student Entity**: Contains a set of **Course** objects. The **@ManyToMany** annotation along with **@JoinTable** specifies the join table **student\_course** that includes **student\_id** and **course\_id** as join columns.
* **Course Entity**: Each **Course** also contains a set of **Student** objects, and **mappedBy = "courses"** indicates that the **courses** field in **Student** owns the relationship.

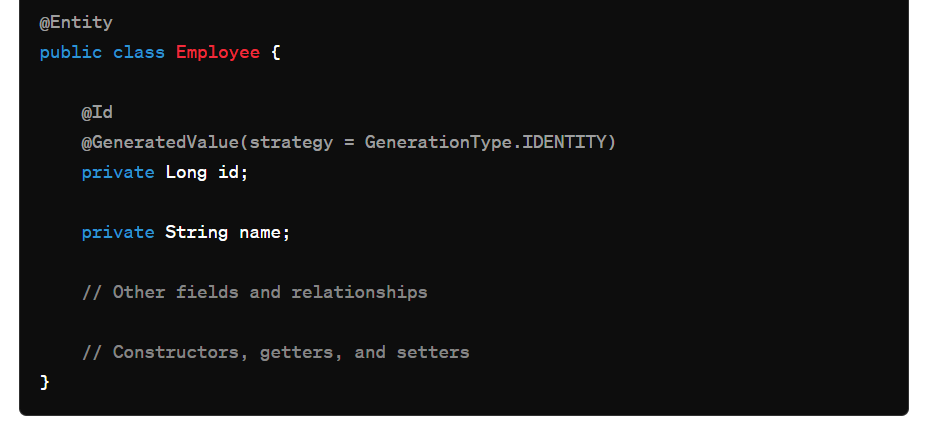
### 3. Many-to-One Relationship

#### Explanation:

A many-to-one relationship is the inverse of a one-to-many relationship. Multiple instances of one entity (e.g., **Project**) are associated with a single instance of another entity (e.g., **Employee**).

#### Code Example:





#### Explanation:

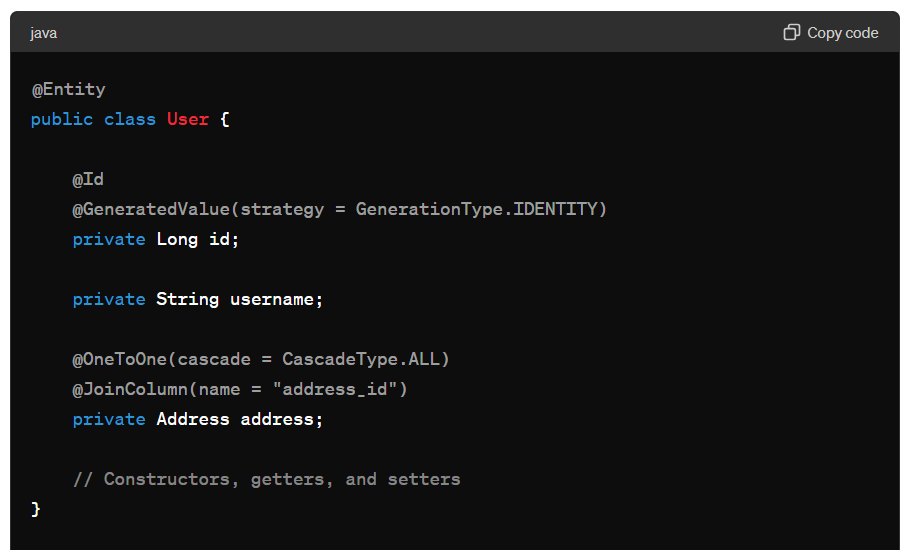
* **Project Entity**: Each **Project** has a reference to an **Employee** with the **@ManyToOne** annotation and a foreign key column **employee\_id**.
* **Employee Entity**: Does not need to explicitly reference the **Project** entity, but could include a **@OneToMany** annotation if needed.

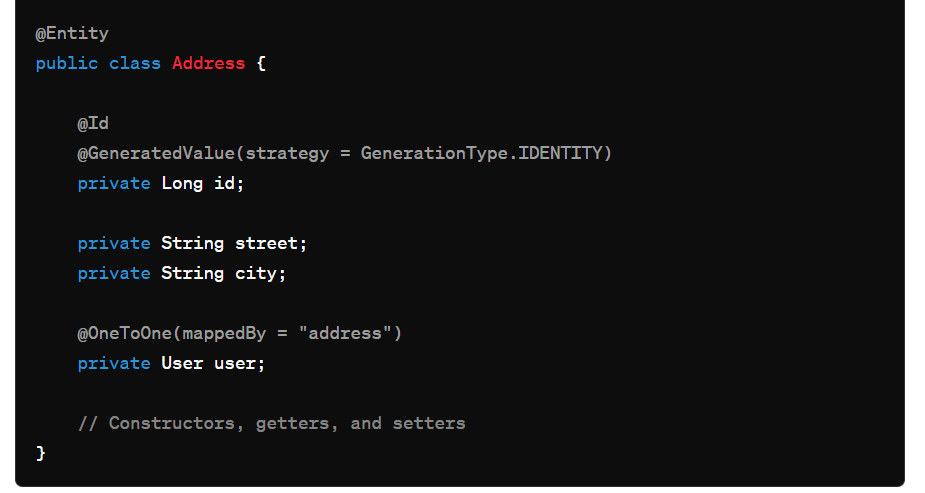
### 4. One-to-One Relationship

#### Explanation:

In a one-to-one relationship, one entity (e.g., **User**) is associated with exactly one instance of another entity (e.g., **Address**).

#### Code Example:





#### Explanation:

* **User Entity**: Contains a reference to **Address** with the **@OneToOne** annotation and a foreign key column **address\_id**. The **cascade = CascadeType.ALL** option ensures that operations such as persist and delete are cascaded to the **Address**.
* **Address Entity**: Contains a reference back to **User** with the **@OneToOne** annotation and **mappedBy = "address"** indicating that the **address** field in **User** owns the relationship.