**JPA and Hibernate with Spring Boot**

The Java Persistence API provides Java developers with an api for mapping java objects to relational data.

**JPA (Java Persistence API) vs. Hibernate:**

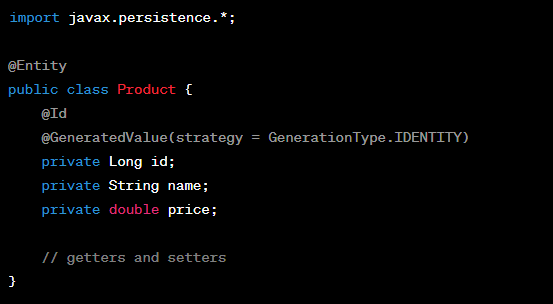
Hibernate is the most popular implementation of JPA. It was the most popular ORM framework option before JPA emerged and it provides additional features on top of JPA

JPA is a standard Java API for object-relational mapping (ORM), while Hibernate is a popular and widely used ORM framework that implements the JPA specification.

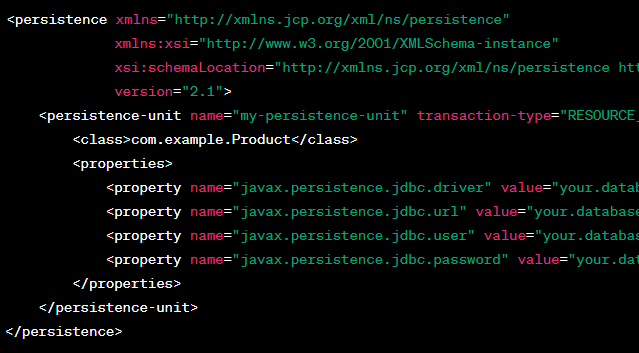
In short, JPA is a specification, and Hibernate is an implementation of that specification. You can use JPA with other implementations besides Hibernate, but Hibernate provides additional features and capabilities beyond the JPA standard.

**Below are some sample code snippets:**

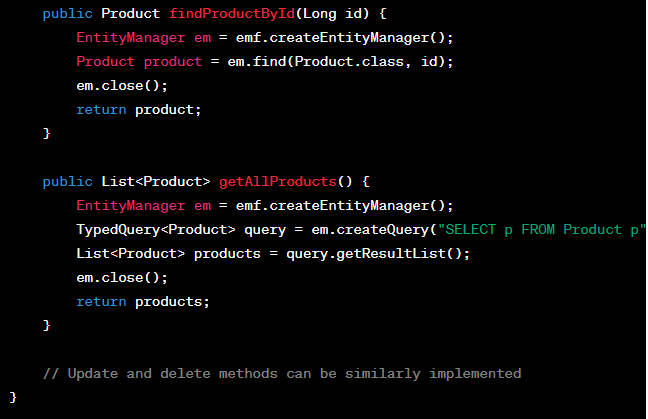
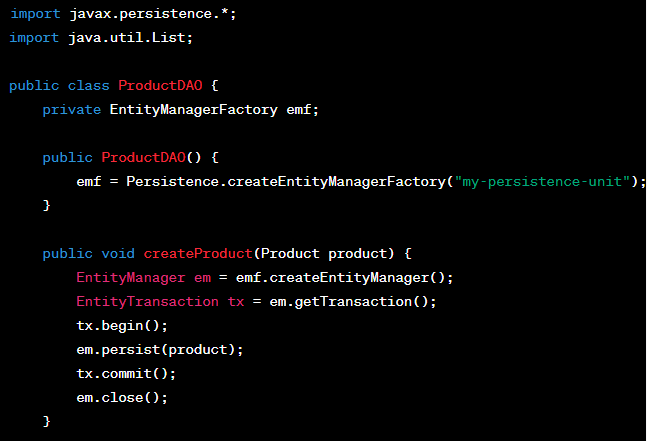
**1. Entity Class:** Define an entity class that maps to a database table. This class is annotated with JPA annotations.



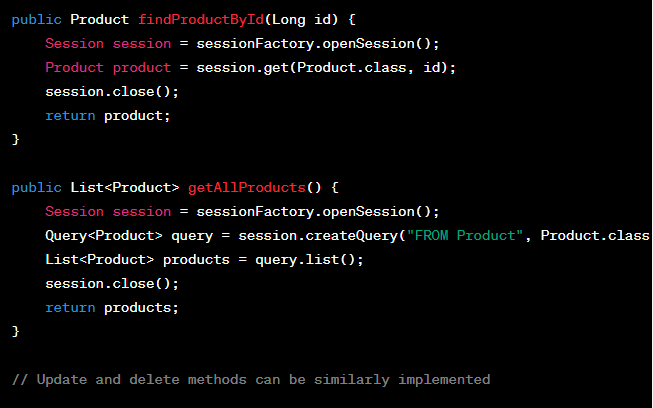
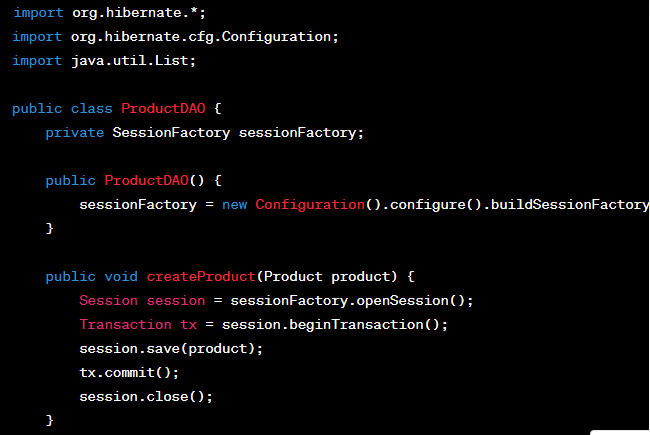
**2. Persistence Configuration:** Set up your **persistence.xml** file for JPA configuration.



**3. JPA Code:** Use JPA to interact with the database.



**4. Hibernate Code:** Using Hibernate is quite similar to JPA, but you don't need to create a **persistence.xml** file. Instead, you configure Hibernate using **hibernate.cfg.xml** or programmatically.



**Entity Class (Product.java):** In both JPA and Hibernate, you define an entity class to represent a table in your database. This class is annotated with JPA annotations, and it maps to a database table. In the example, we have an **Product** entity with fields for an **id**, **name,** and **price**. The **@Entity** annotation marks the class as an entity, and the **@Id** annotation specifies the primary key of the table.

**Persistence Configuration:** In JPA, you configure your persistence unit in a **persistence.xml** file. It specifies the entity classes, database connection details, and other properties.

In Hibernate, you can configure the **SessionFactory** in a **hibernate.cfg.xml** file or programmatically. It includes properties like database driver, URL, username, and password.

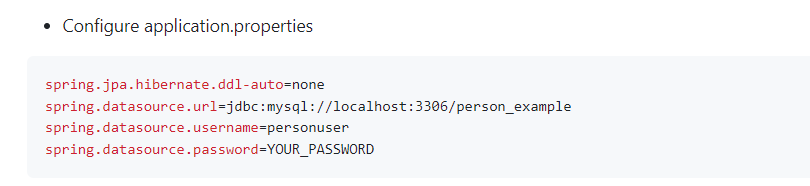
**JPA Code (ProductDAO.java):**

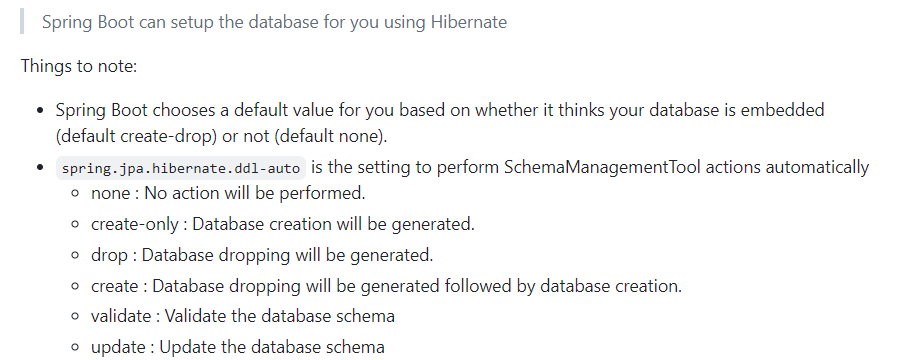
1. **EntityManagerFactory Initialization:** The **ProductDAO** class initializes the **EntityManagerFactory** in its constructor using the **Persistence.createEntityManagerFactory** method. It configures it with the name of the persistence unit defined in the **persistence.xml** file.
2. **Create a Product:** The **createProduct** method creates a new **EntityManager**, begins a transaction, and persists the **Product** object. It commits the transaction and closes the **EntityManager**.
3. **Find a Product by ID:** The **findProductById** method retrieves a product by its primary key using the **find** method of the **EntityManager**.
4. **Get All Products:** The **getAllProducts** method fetches all products from the database using a JPQL (Java Persistence Query Language) query and returns a list of products.

**Hibernate Code (ProductDAO.java):**

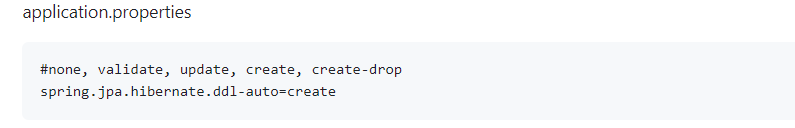
1. **SessionFactory Initialization:** The **ProductDAO** class initializes the **SessionFactory** in its constructor using Hibernate's **Configuration** class. It configures the **SessionFactory** with the **hibernate.cfg.xml** file.
2. **Create a Product:** The **createProduct** method creates a new **Session**, starts a transaction, and saves the **Product** object. It commits the transaction and closes the **Session**.
3. **Find a Product by ID:** The **findProductById** method retrieves a product by its primary key using the **get** method of the **Session**.
4. **Get All Products:** The **getAllProducts** method fetches all products from the database using HQL (Hibernate Query Language) and returns a list of products.

**Configure Steps:**





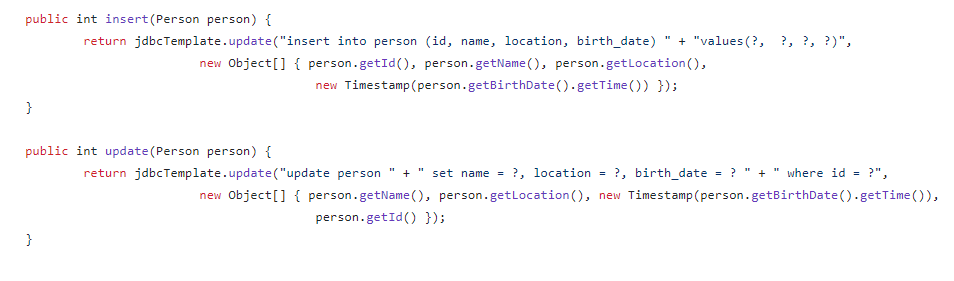
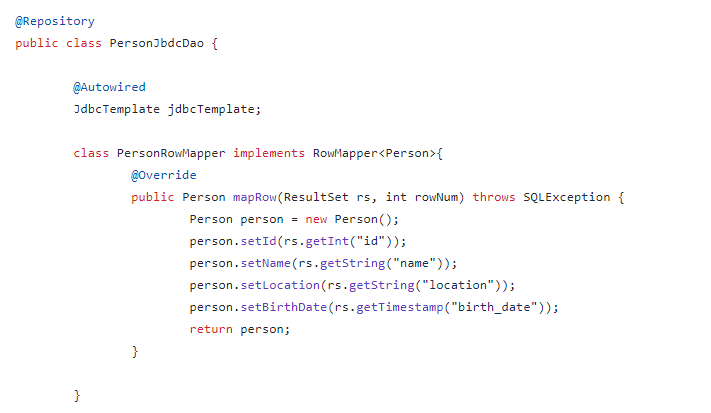
**Reference** : <https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#configurations-hbmddl>



JDBC:



**JDBC Template:**



Spring ORM Frameworks

* ORM, Spring Data JPA and Hibernate
* Perform CRUD operations against a database with two simple steps
* Configure auto generated IDs for the Primary Key fields
* Realize the power of Spring Data Finder methods
* Load data from database without implementing any code or SQL
* Learn and use JPQL - Java Persistence Query Language
* Execute native Sql queries from your Spring Data Application
* Use Paging and Sorting
* Hibernate Associations and implement all the four types of associations

**1. ORM Basics:**

* **ORM (Object Relational Mapping)**
* **What is JPA?**
* **What and why Spring Data?**

**2. Simple CRUD Operations:**

Project - **Productdata**

Use case:

Product: Product – Table Name

DB

Product - Entity Class

Product Repository – Repository Interface

Configure the Data Source

Step 1: Create the database and tables.

Product.sql

Step 2: Create the Entity class.

Step 3: Create the Repository.

Step 4: Configure the Data Source.

Step 5: Test the Application.

Create/Read/Update/Delete/Exists

3. ID Generators - Project – Id Generator

Product product = new Product();

product.setId(123);

product.setName("Iphone");

repository.save(product);

GenerationType.AUTO

GenerationType.IDENTITY

GenerationType.SEQUENCE

GenerationType.TABLE

@Id

@GeneratedValue(Strategy = GenerationType.AUTO)

int id;

**Persistence provider** refers to the specific JPA implementation used in our application to persist objects to the database

Database

**GenerationType.AUTO**

**Persistence provider**

GenerationType.IDENTITY

GenerationType.SEQUENCE

|  |  |  |
| --- | --- | --- |
| Id |  |  |
|  |  |  |
|  |  |  |

GenerationType.TABLE

**GenerationType.IDENTITY**

Database

**Persistence provider**

|  |  |  |
| --- | --- | --- |
| Id |  |  |
|  |  |  |
|  |  |  |

PRIMARY KEY – Auto Increment

**GenerationType.TABLE**

Database

**Persistence provider**

|  |  |  |
| --- | --- | --- |
| Id |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| MYID |  |  |
|  |  |  |
|  |  |  |

Special Table ----🡪

Create the Database Schema:

ID Gen Employee.sql

4. Spring Data Finder Methods:

Finder Methods

product

id

name

desc

price

ProductRepository

List<product> findByName(String name);

List<product> findByNameAndDesc(String name, String desc);

List<product> findByDescLike(String desc);

List<product> findByPriceGreaterThan(Double price);

Spring data jpa documentation :

**https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repository-query-keywords**

Paging and Sorting:

JPQL

**Java Persistence Query Language**

Employee

Int id;

String firstName;

String lastName;

Object based Queries:

**employeetab**

|  |  |  |
| --- | --- | --- |
| Empid | Fname | lname |
|  |  |  |
|  |  |  |

Select \* from Employee;

Select firstName,lastName from Employee;

**Relationships In Hibernate:**

**Association Mapping: (Unidirectional & bidirectional)**

**order**

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Productname | Description | shippingaddress |
|  |  |  |  |
|  |  |  |  |

**Order**

|  |  |  |
| --- | --- | --- |
|  | Address\_Id | Product\_id |
|  |  |  |
|  |  |  |

**product**

|  |  |  |
| --- | --- | --- |
| id |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Id | Street | city |
|  |  |  |
|  |  |  |

**Address**

One to One:

**Person**

Long id

License license

**License**

Long id

Person person

Many to Many:

Order

Long id

Set<Product> product

Product

Long id

String name

Set<Order> orders

One to Many & Many to One:

Customer

Long Id

Set<PhoneNumber> numbers

PhoneNumber

Long id

String number

@OneToOne

@OneTo Many

@ManyToOne

@ManyToMany