**JDBC (Java Database Connectivity):** JDBC is the Java API for connecting and executing queries on a database. It provides a way for Java programs to interact with databases using SQL queries. JDBC requires writing SQL queries manually and handling database connections, statements, and result sets.

**Example Code: Ensure you have the MySQL JDBC driver (**mysql-connector-java**) in your classpath.**

|  |
| --- |
| **import java.sql.\*;**  **public class JdbcExample {**  **public static void main(String[] args) {**  **String url = "jdbc:mysql://localhost:3306/mydatabase";**  **String username = "root";**  **String password = "password";**  **try (Connection connection = DriverManager.getConnection(url, username, password);**  **Statement statement = connection.createStatement()) {**  **// Create table**  **statement.executeUpdate("CREATE TABLE IF NOT EXISTS users (id INT PRIMARY KEY, name VARCHAR(255))");**  **// Insert data**  **statement.executeUpdate("INSERT INTO users (id, name) VALUES (1, 'Alice')");**  **// Retrieve data**  **ResultSet resultSet = statement.executeQuery("SELECT \* FROM users");**  **while (resultSet.next()) {**  **System.out.println("ID: " + resultSet.getInt("id") + ", Name: " + resultSet.getString("name"));**  **}**  **} catch (SQLException e) {**  **e.printStackTrace();**  **}**  **}**  **}** |

**Spring JDBC:** Spring JDBC is a part of the Spring Framework that simplifies JDBC usage. It provides classes like JdbcTemplate and NamedParameterJdbcTemplate to reduce boilerplate code and handle common JDBC operations. Spring JDBC also manages database connections, statements, and result sets, making database interactions more efficient and less error-prone.

**Example Code:**

* Add the Spring JDBC dependency to your project (e.g., spring-jdbc).
* Configure a DataSource bean in your Spring configuration.
* Use JdbcTemplate to perform database operations.

|  |
| --- |
| **import org.springframework.jdbc.core.JdbcTemplate;**  **import javax.sql.DataSource;**  **public class SpringJdbcExample {**  **private JdbcTemplate jdbcTemplate;**  **public SpringJdbcExample(DataSource dataSource) {**  **this.jdbcTemplate = new JdbcTemplate(dataSource);**  **}**  **public void getUsers() {**  **jdbcTemplate.query("SELECT \* FROM users",**  **(rs, rowNum) -> "ID: " + rs.getInt("id") + ", Name: " + rs.getString("name"))**  **.forEach(System.out::println);**  **}**  **}** |

**Hibernate:** Hibernate is an Object-Relational Mapping (ORM) framework for Java. It simplifies database programming by mapping Java objects to database tables. Hibernate handles the translation of Java objects to SQL queries and vice versa, reducing the need for manual SQL queries. It also provides features like caching, lazy loading, and transaction management.

**Example Code:**

* Add the Hibernate dependencies to your project (e.g., hibernate-core, hibernate-entityManager).
* Configure Hibernate using hibernate.cfg.xml.
* Create an entity class (User) and map it to the users table.
* Use SessionFactory to obtain a Session and perform database operations.

|  |
| --- |
| import org.hibernate.Session; import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration; import java.util.List;  public class HibernateExample {  public static void main(String[] args) {  SessionFactory sessionFactory = new Configuration()  .configure("hibernate.cfg.xml")  .addAnnotatedClass(User.class)  .buildSessionFactory();  try (Session session = sessionFactory.openSession()) {  session.beginTransaction();  // Insert  User user = new User();  user.setId(2);  user.setName("Bob");  session.save(user);  // Retrieve  List<User> users = session.createQuery("FROM User").getResultList();  users.forEach(u -> System.out.println("ID: " + u.getId() + ", Name: " + u.getName()));  session.getTransaction().commit();  } catch (Exception e) {  e.printStackTrace();  } finally {  sessionFactory.close();  }  }  } |

**Spring Data JPA:**

* Add the Spring Data JPA dependency to your project (e.g., spring-data-jpa).
* Create a repository interface (UserRepository) that extends JpaRepository<User, Long>.
* Use the repository to perform CRUD operations.

|  |
| --- |
| import org.springframework.data.jpa.repository.JpaRepository;  public interface UserRepository extends JpaRepository<User, Long> {  } |

**Hibernate & JPA code Snippets:**

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

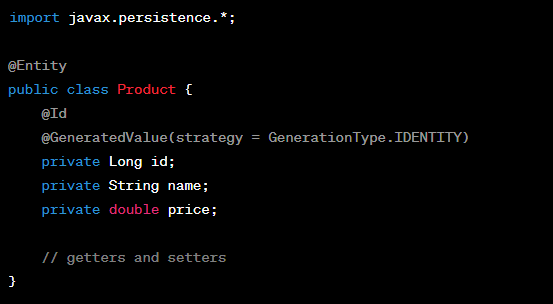
**2. Persistence Configuration:** Set up your persistence.xml file for JPA configuration.

**3. JPA Code:**

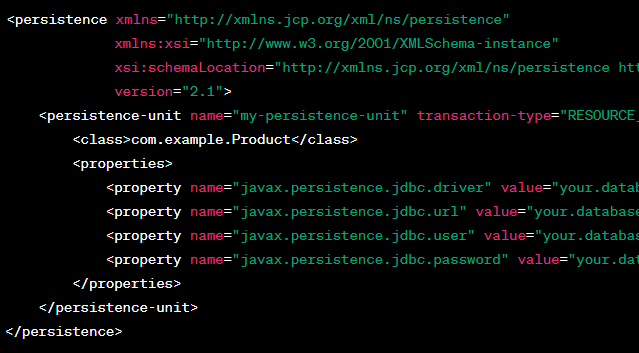
Use JPA to interact with the database.

**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.

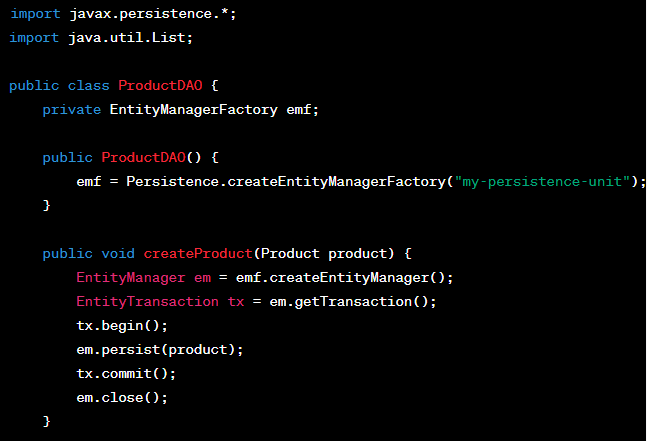
**1. Entity Class:**

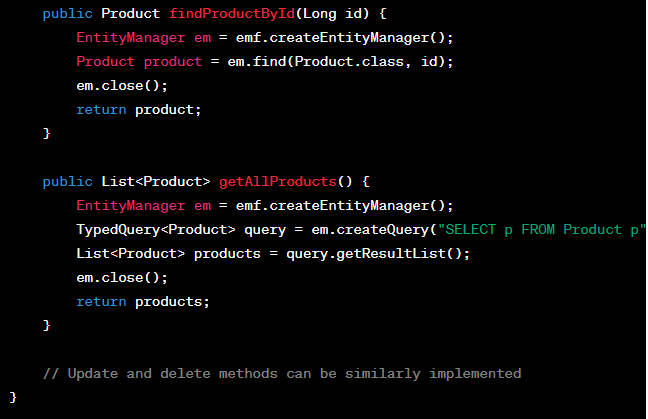


Persistence Configuration:

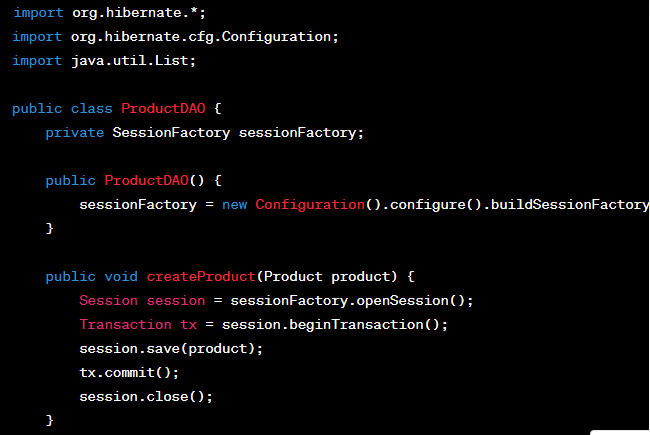


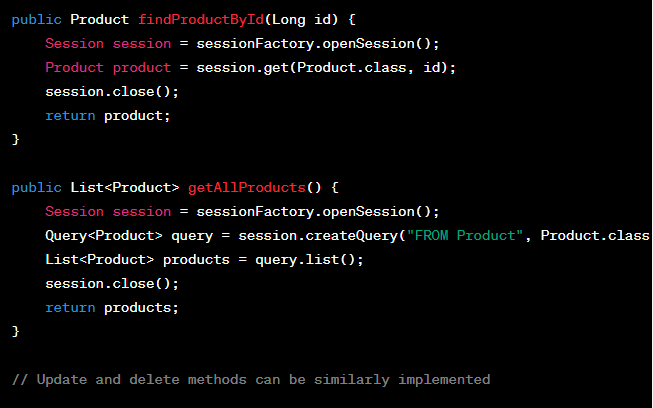
**JPA Code:**



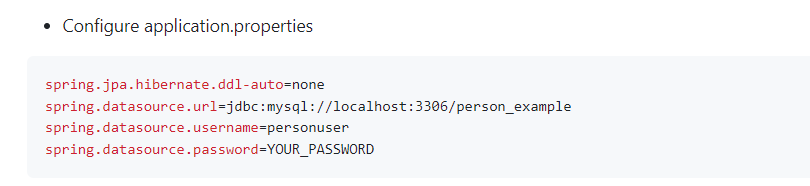


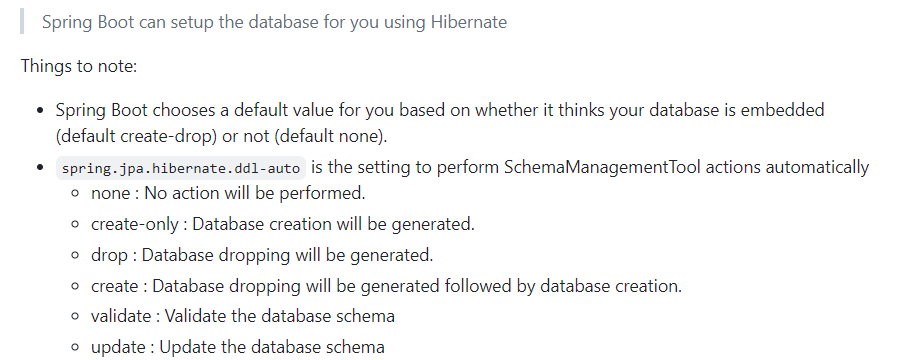
**Hibernate Code:**

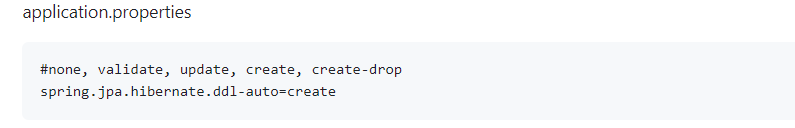




**Configure Steps:**







**SessionFactory (Hibernate):**

The SessionFactory is part of the Hibernate ORM framework.

It is used to create Session instances, which represent a single unit of work with the database.

The Session is used to perform database operations and manage the persistence lifecycle of entities.

The SessionFactory is typically created once for the entire application and is thread-safe, so it can be shared among multiple threads.

**EntityManagerFactory (JPA):**

The EntityManagerFactory is part of the JPA (Java Persistence API) specification.

It is used to create EntityManager instances, which are used to interact with the database in a JPA-based application.

The EntityManager is similar to the Session in Hibernate and is used for database operations and entity management.

The EntityManagerFactory is typically created once for the entire application and is thread-safe, so it can be shared among multiple threads.

**In summary, the main difference is that SessionFactory is specific to Hibernate and creates Session instances, while EntityManagerFactory is part of the JPA specification and creates EntityManager instances. Both serve as factories for creating objects that manage database interactions in their respective frameworks.**

**Note:**

When you define a repository interface that extends JpaRepository (or one of its subinterfaces) and annotate it with @Repository, Spring Data JPA automatically generates the implementation for you. This implementation internally uses an EntityManager obtained from the EntityManagerFactory to perform database operations.

So, even though you don't explicitly use EntityManager in your code, it's still essential behind the scenes for Spring Data JPA to manage entity persistence and perform database operations on your behalf.

**References:**

**hibernate.cfg.xml:**

|  |
| --- |
| <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <!-- Database connection settings -->  <session-factory>  <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydatabase</property>  <property name="hibernate.connection.username">root</property>  <property name="hibernate.connection.password">password</property>    <!-- Specify dialect -->  <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>    <!-- Enable Hibernate's automatic session context management -->  <property name="hibernate.current\_session\_context\_class">thread</property>    <!-- Disable the second-level cache -->  <property name="hibernate.cache.provider\_class">org.hibernate.cache.NoCacheProvider</property>    <!-- Echo all executed SQL to stdout -->  <property name="hibernate.show\_sql">true</property>    <!-- Drop and re-create the database schema on startup -->  <property name="hibernate.hbm2ddl.auto">create-drop</property>    <!-- Mapping files -->  <!-- <mapping resource="com/example/YourEntity.hbm.xml"/> -->  </session-factory>  </hibernate-configuration> |

|  |
| --- |
| import org.hibernate.SessionFactory;  import org.hibernate.boot.MetadataSources;  import org.hibernate.boot.registry.StandardServiceRegistry;  import org.hibernate.boot.registry.StandardServiceRegistryBuilder;  public class HibernateUtil {  private static SessionFactory sessionFactory;  public static SessionFactory getSessionFactory() {  if (sessionFactory == null) {  // Configure Hibernate using Java code  StandardServiceRegistry registry = new StandardServiceRegistryBuilder()  .configure() // By default, it looks for hibernate.cfg.xml  .build();  try {  MetadataSources sources = new MetadataSources(registry);  sessionFactory = sources.buildMetadata().buildSessionFactory();  } catch (Exception e) {  StandardServiceRegistryBuilder.destroy(registry);  e.printStackTrace();  }  }  return sessionFactory;  }  public static void shutdown() {  if (sessionFactory != null) {  sessionFactory.close();  }  }  } |

If you want to configure Hibernate programmatically without using an XML configuration file, you can do so by directly setting properties on the **StandardServiceRegistryBuilder** and **MetadataSources**

|  |
| --- |
| import org.hibernate.SessionFactory;  import org.hibernate.boot.Metadata;  import org.hibernate.boot.MetadataSources;  import org.hibernate.boot.registry.StandardServiceRegistry;  import org.hibernate.boot.registry.StandardServiceRegistryBuilder;  import org.hibernate.cfg.Environment;  import org.hibernate.service.ServiceRegistry;  import java.util.Properties;  public class HibernateUtil {  private static SessionFactory sessionFactory;  public static SessionFactory getSessionFactory() {  if (sessionFactory == null) {  try {  StandardServiceRegistry standardRegistry = new StandardServiceRegistryBuilder()  .applySetting(Environment.DRIVER, "com.mysql.jdbc.Driver")  .applySetting(Environment.URL, "jdbc:mysql://localhost:3306/mydatabase")  .applySetting(Environment.USER, "root")  .applySetting(Environment.PASS, "password")  .applySetting(Environment.DIALECT, "org.hibernate.dialect.MySQL5Dialect")  .applySetting(Environment.SHOW\_SQL, "true")  .applySetting(Environment.HBM2DDL\_AUTO, "update")  .build();  Metadata metadata = new MetadataSources(standardRegistry)  .addAnnotatedClass(User.class)  .getMetadataBuilder()  .build();  sessionFactory = metadata.getSessionFactoryBuilder().build();  } catch (Exception e) {  e.printStackTrace();  if (standardRegistry != null) {  StandardServiceRegistryBuilder.destroy(standardRegistry);  }  }  }  return sessionFactory;  }  public static void shutdown() {  if (sessionFactory != null) {  sessionFactory.close();  }  }  } |

