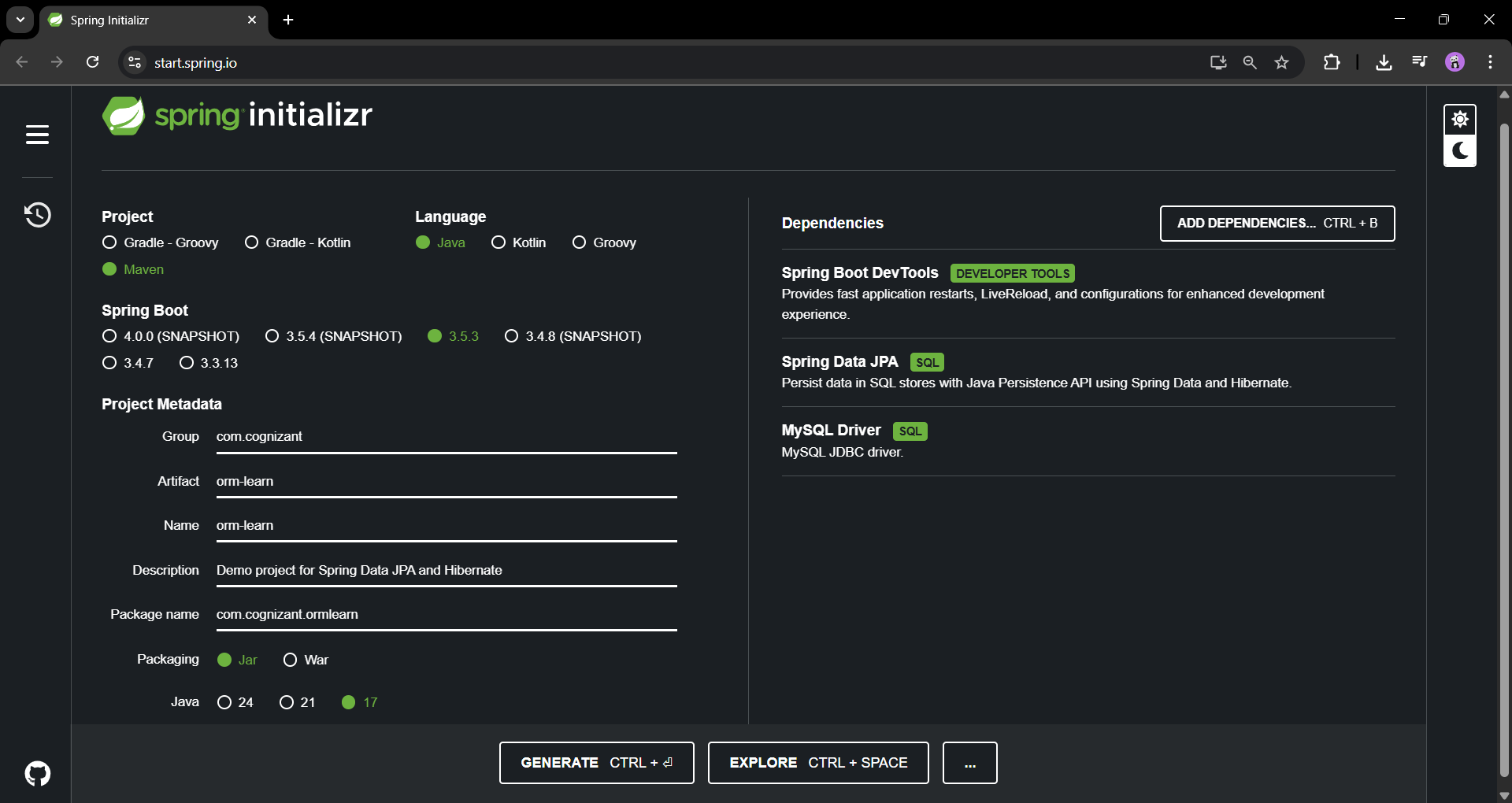
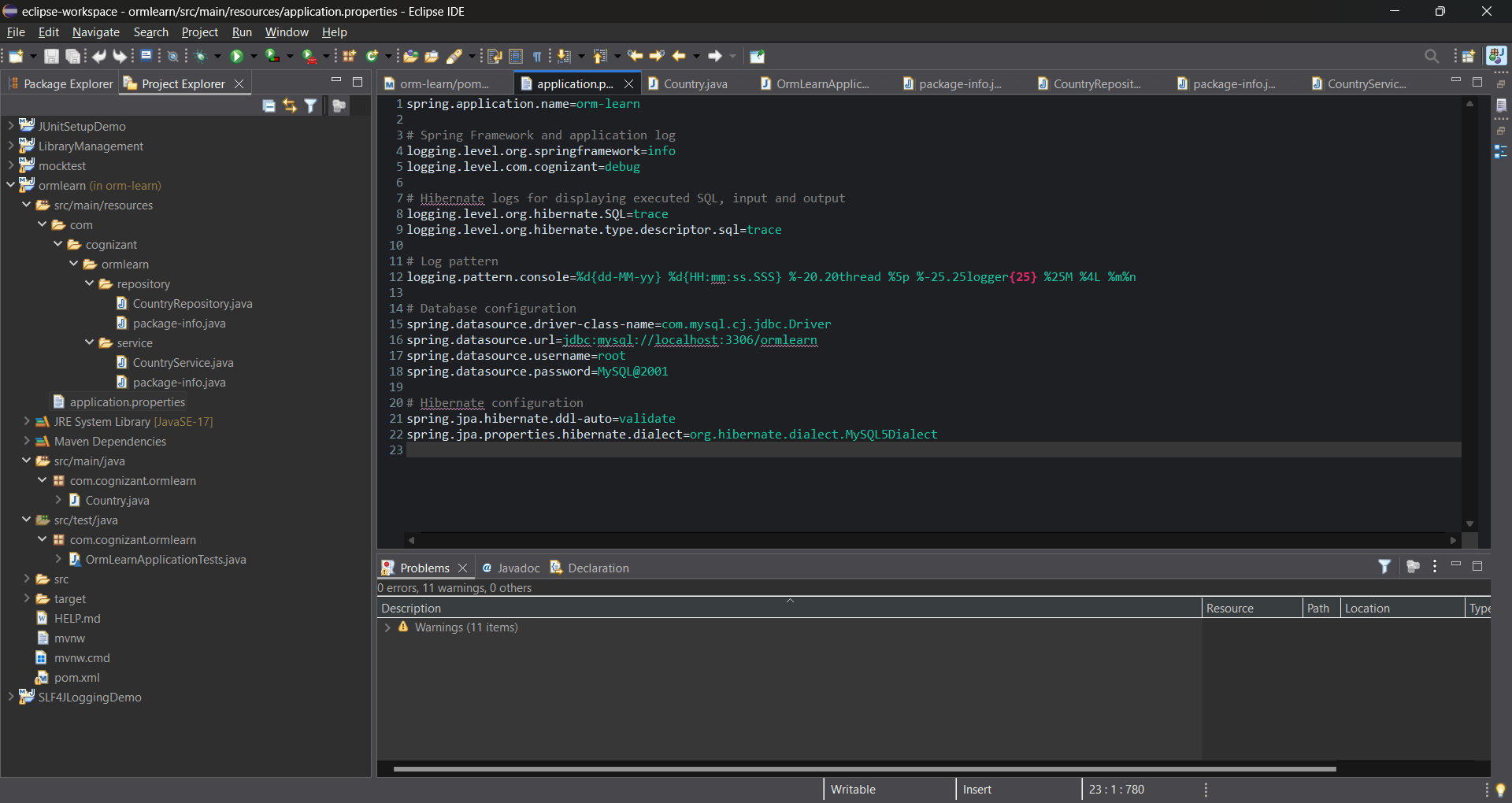
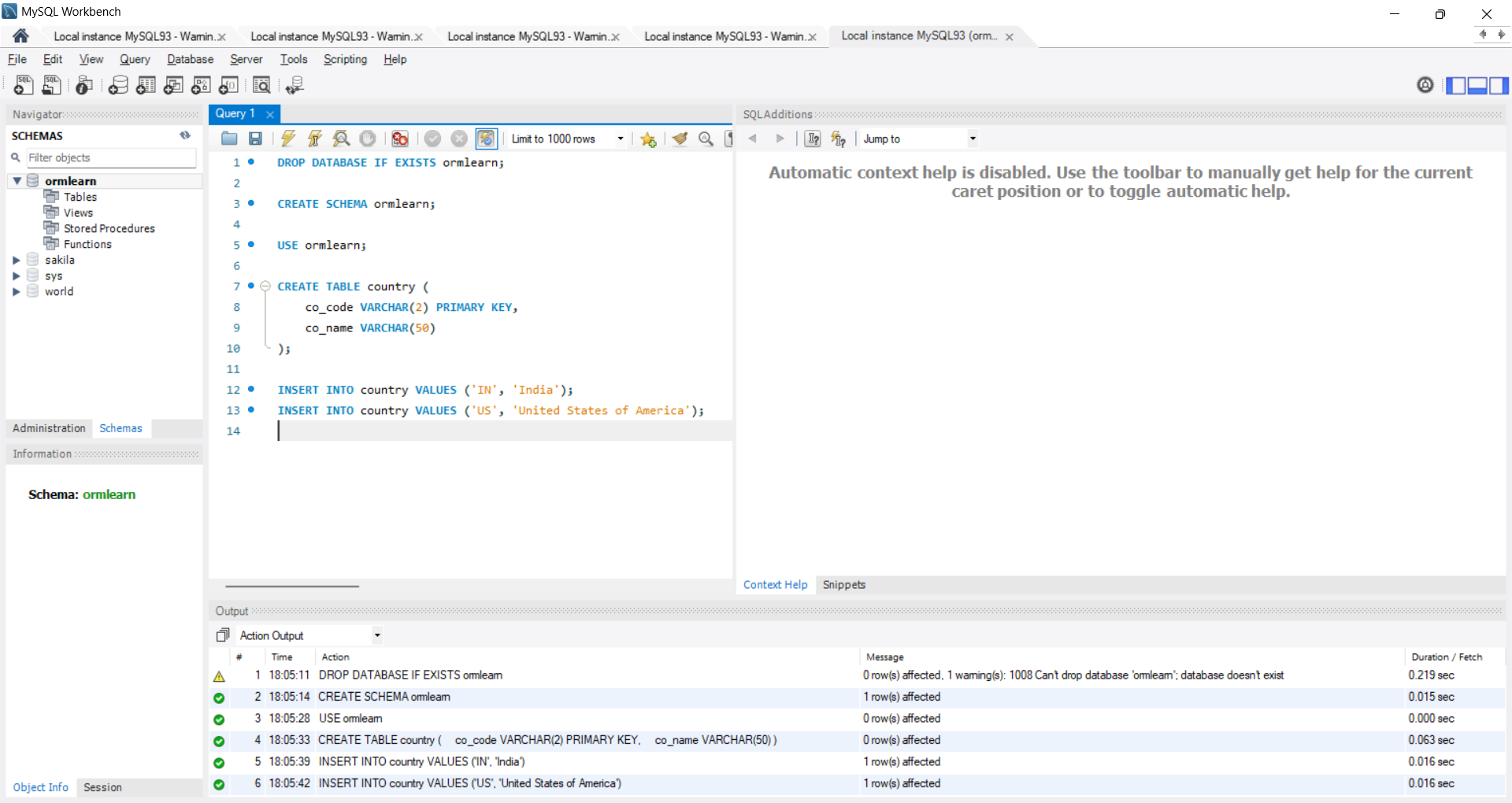
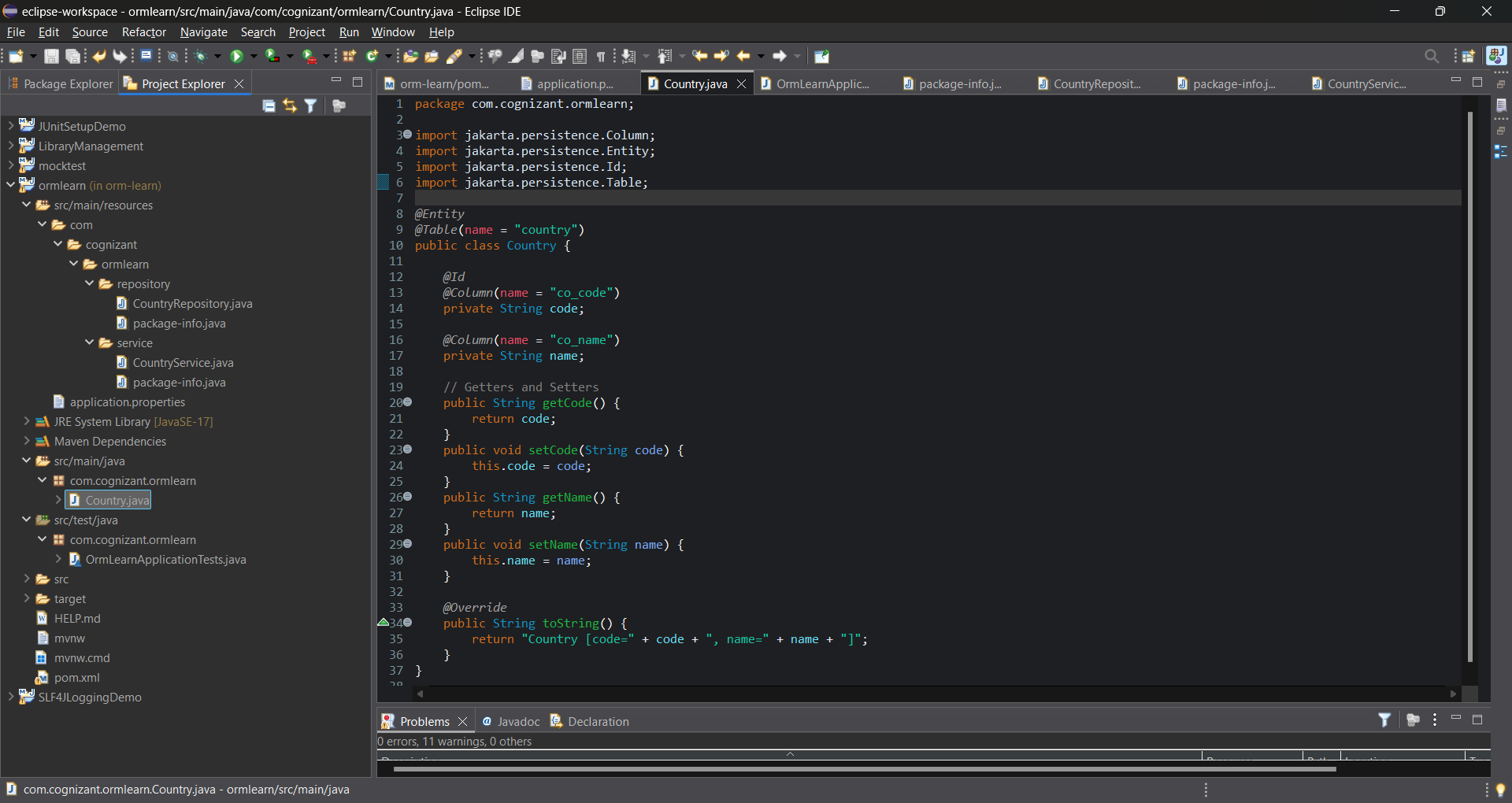
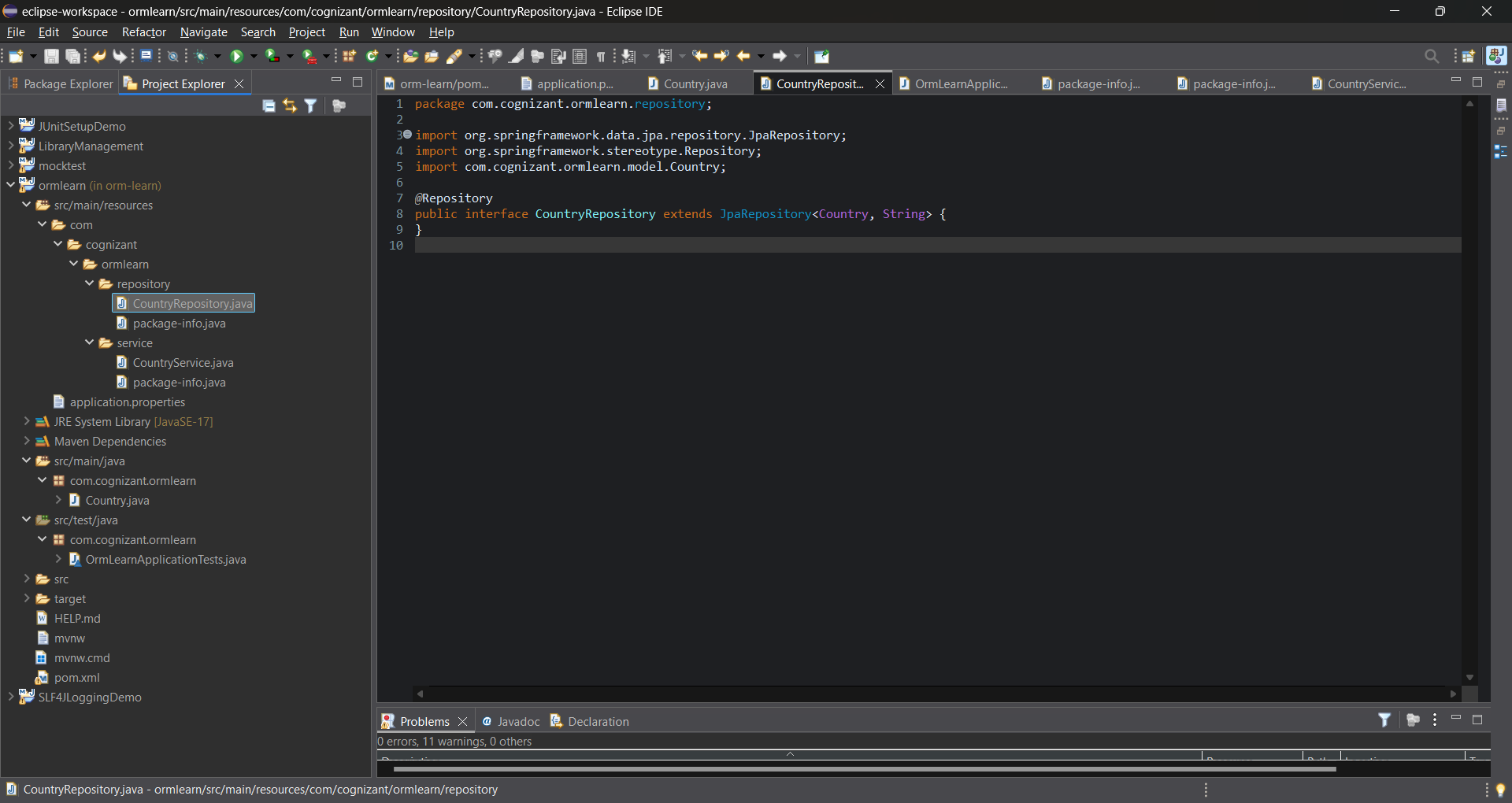
**Hands on 1: Spring Data JPA - Quick Example**

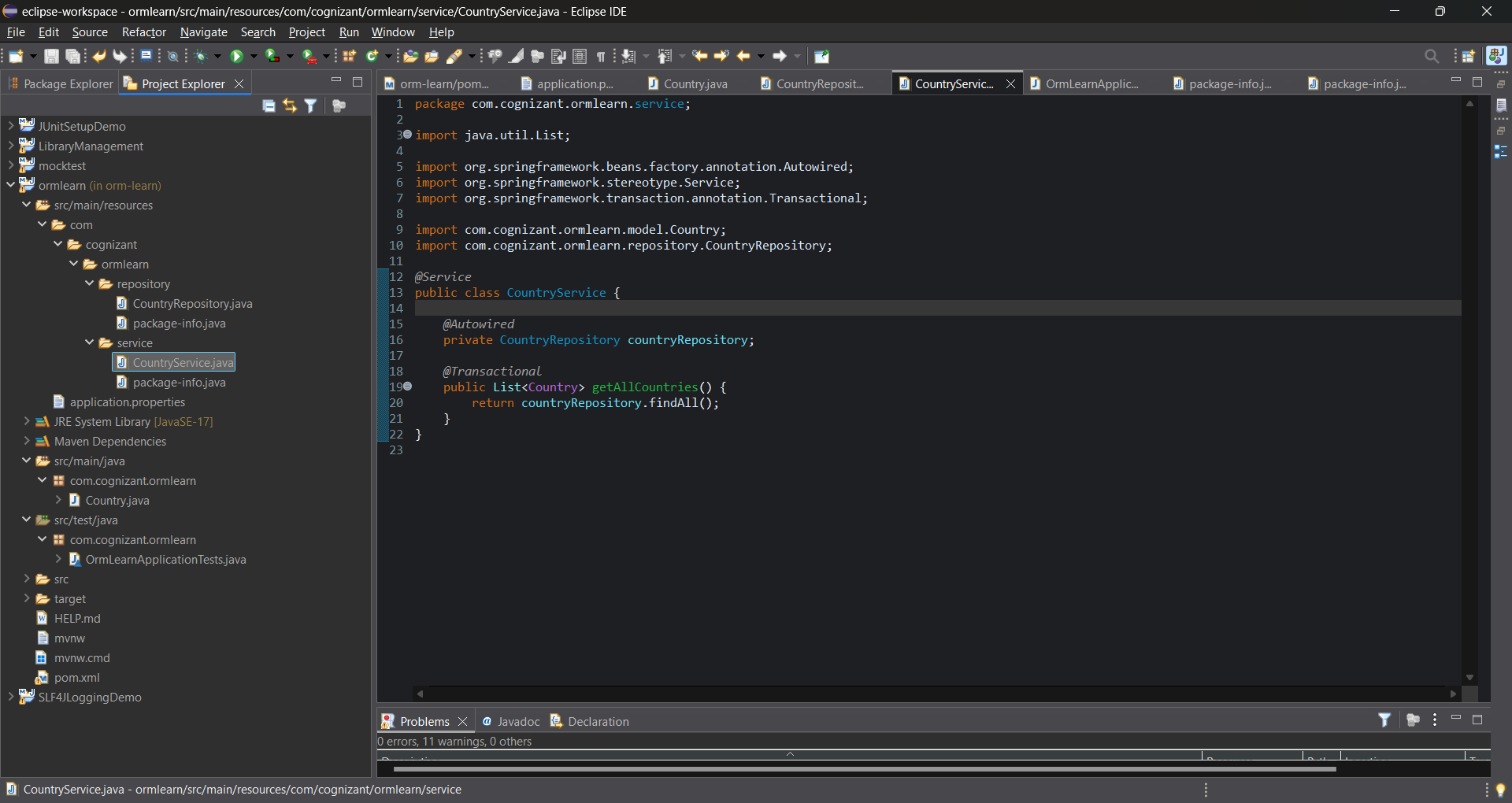


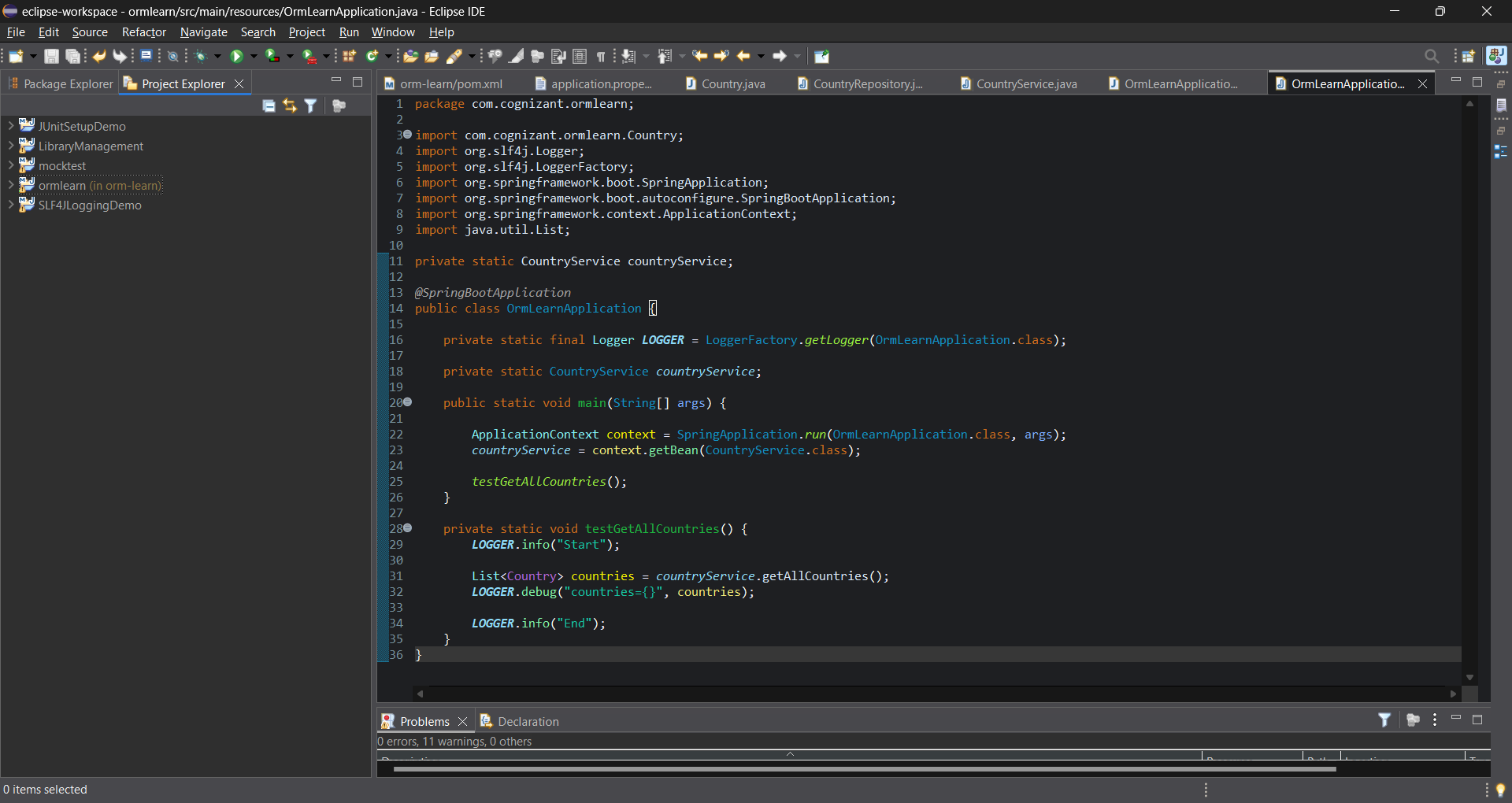








****



# ****Hands-On 4: Difference Between JPA, Hibernate, and Spring Data JPA****

## ****Introduction****

When developing Java applications that interact with databases, developers often use frameworks and tools to simplify data persistence. Three commonly used terms in this context are **JPA**, **Hibernate**, and **Spring Data JPA**. Although they are related, they serve different purposes.

## ****1. Java Persistence API (JPA)****

JPA stands for **Java Persistence API**. It is a **specification** (JSR 338) that provides a set of guidelines for mapping Java objects to database tables. In simple terms, JPA defines **how to store, retrieve, and manage data from Java objects**, but it does not provide any actual working code or implementation.

* JPA is like a **contract** or **standard**, not a tool or framework itself.
* It ensures that different tools or frameworks can interact with databases in a consistent way.
* JPA alone cannot interact with the database; it requires an implementation like Hibernate.

## ****2. Hibernate****

Hibernate is an **Object Relational Mapping (ORM) tool** that provides a concrete implementation of the JPA specification. It simplifies database operations by allowing developers to work with Java objects instead of writing complex SQL queries.

* Hibernate is one of the most popular implementations of JPA.
* It follows JPA guidelines but also provides additional features beyond the JPA specification.

**Example of Hibernate Code:**

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

In this example, the developer is responsible for:

* Opening a session
* Starting a transaction
* Saving the object
* Handling exceptions
* Closing the session

## ****3. Spring Data JPA****

**Spring Data JPA** is a part of the Spring ecosystem that builds on top of JPA and ORM providers like Hibernate. It provides an additional layer of abstraction and reduces the amount of boilerplate code required for data access operations.

* It does **not** provide a JPA implementation itself.
* It relies on an existing JPA provider (commonly Hibernate).
* It simplifies data access by generating most of the common queries automatically.
* It also manages transactions automatically in many cases.

**Example of Spring Data JPA Code:**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

With Spring Data JPA:

* No need to open sessions manually
* No need to explicitly handle transactions (Spring does it)
* CRUD operations like save, findById, delete are ready to use
* Less boilerplate code, more focus on business logic

## ****4. Summary of Differences****

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (Set of rules) | JPA Implementation (ORM Tool) | Abstraction layer over JPA (uses Hibernate) |
| Provides Implementation | No | Yes | No (Relies on JPA implementation) |
| Reduces Boilerplate Code | Minimal | Some, still manual | Yes, greatly simplified |
| Manages Transactions | No | Yes, but manual control | Yes, handled by Spring |
| Most Common Usage Together | Defines Rules | Implements JPA | Simplifies usage of Hibernate and JPA |

## ****5. Conclusion****

In summary:

* **JPA** defines how to persist data but does not provide working code.
* **Hibernate** implements JPA and allows Java objects to be stored in the database.
* **Spring Data JPA** simplifies data access further by reducing boilerplate code and automating common operations.

Together, these tools help developers build robust, scalable, and maintainable Java applications that interact with relational databases efficiently.