**2)**

**//Country.java**

**package** com.cognizant.orm\_learn.model;

**import** jakarta.persistence.Column;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.Table;

@Entity

@Table(name = "country")

**public** **class** Country {

@Id

@Column(name = "co\_code") // expecting column co\_code

**private** String code;

@Column(name = "co\_name") // expecting column co\_name

**private** String name;

@Override

**public** String toString() {

**return** "Country [code=" + code + ", name=" + name + "]";

}

}

**//CountryService.java**

**package** com.cognizant.orm\_learn.service;

**import** java.util.List;

**import** org.springframework.transaction.annotation.Transactional;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.cognizant.orm\_learn.model.Country;

**import** com.cognizant.orm\_learn.repository.CountryRepository;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

}

**//application.properties**

spring.application.name=orm-learn

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger**{25}** %25M %4L %m%n

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=Nila@2004

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**//CountryRepository**

**package** com.cognizant.orm\_learn.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.stereotype.Repository;

**import** com.cognizant.orm\_learn.model.Country;

@Repository

**public** **interface** CountryRepository **extends** JpaRepository<Country, String> {

}

**//OrmLearnApplication**

**package** com.cognizant.orm\_learn;

**import** java.util.List;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ApplicationContext;

**import** com.cognizant.orm\_learn.model.Country;

**import** com.cognizant.orm\_learn.service.CountryService;

@SpringBootApplication

**public** **class** OrmLearnApplication {

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(OrmLearnApplication.**class**);

**private** **static** CountryService *countryService*;

**public** **static** **void** main(String[] args) {

ApplicationContext context = SpringApplication.*run*(OrmLearnApplication.**class**, args);

***LOGGER***.info("Inside main");

*countryService* = context.getBean(CountryService.**class**);

*testGetAllCountries*();

}

**private** **static** **void** testGetAllCountries() {

***LOGGER***.info("Start");

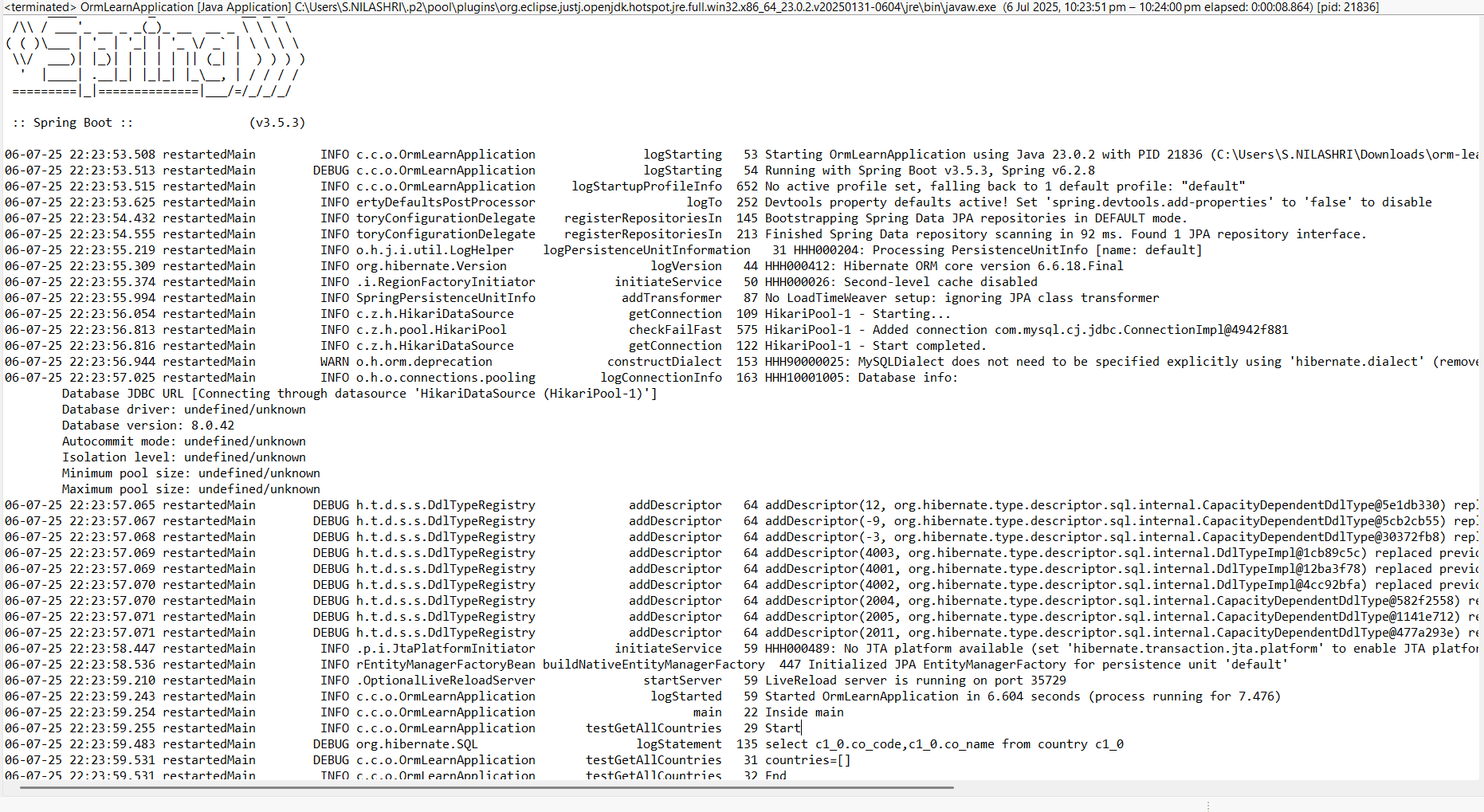
List<Country> countries = *countryService*.getAllCountries();

***LOGGER***.debug("countries={}", countries);

***LOGGER***.info("End");

}

}



**4)**

Understanding the distinction between **JPA**, **Hibernate**, and **Spring Data JPA** is essential for Java developers working with databases. Each plays a different role in Java-based persistence, with increasing levels of abstraction and ease of use.

**Java Persistence API (JPA)**

* JPA is a **specification** (JSR 338) for managing relational data in Java applications.
* It provides a **standard API** for object-relational mapping (ORM) but **does not include any implementation**.
* JPA defines:
  + Annotations (@Entity, @Id, etc.)
  + Entity lifecycle management
  + Query language (JPQL)
* It needs an **implementation provider** (like Hibernate or EclipseLink) to work.

**Hibernate**

* Hibernate is a **popular ORM framework** and the **most widely used implementation of JPA**.
* It provides:
  + Full support for JPA annotations and APIs.
  + Advanced features like lazy loading, caching, and HQL (Hibernate Query Language).
* Can be used standalone or with Spring Framework.
* Developers manually manage:
  + Sessions
  + Transactions
  + Exception handling

**Spring Data JPA**

* Spring Data JPA is **not an implementation of JPA**, but rather a **high-level abstraction layer** built on top of JPA (usually Hibernate).
* It reduces boilerplate code by:
  + Automatically implementing repository interfaces
  + Providing ready-to-use CRUD operations
* Handles:
  + Transactions
  + Query generation (from method names or @Query)
  + Integration with Spring’s dependency injection and transaction management
* Developers only need to define interfaces; Spring handles the rest at runtime.

**Code Comparison**

**Hibernate (Manual 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;

}

* Manual session and transaction handling
* More boilerplate and error-prone code

**Spring Data JPA (Simplified)**

**EmployeeRepository.java**

java

CopyEdit

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

java

CopyEdit

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

* No need to manage sessions or transactions explicitly
* save() method is inherited from JpaRepository

**Conclusion**

* **JPA** defines the "what" of ORM in Java.
* **Hibernate** provides a concrete "how" by implementing JPA.
* **Spring Data JPA** goes a step further by automating and simplifying database access through abstraction and integration with Spring.

For enterprise-level Java applications, using **Spring Data JPA with Hibernate** as the JPA provider is the most efficient and scalable approach.