Digital Nurture 4.0

Deep Skilling - Java FSE

WEEK-3 HANDS ON

By Kaviya P

[SPRING](https://github.com/seshadrimr/Digital-Nurture-4.0-JavaFSE/tree/main/Java%20FSE/Deepskilling/Spring%20Data%20JPA%20with%20Hibernate) DATA WITH HIBERNATE

**Exercise 1: Employee Management System - Overview and Setup**

**Business Scenario:**

You are developing an employee management system that will manage employee data, departments, and their relationships.

**Instructions:**

1. **Creating a Spring Boot Project:**
   * Initialize a new Spring Boot project named **EmployeeManagementSystem**.
   * Add dependencies: **Spring Data JPA, H2 Database, Spring Web, Lombok**.
2. **Configuring Application Properties:**
   * Configure **application.properties** for H2 database connection.

*spring.datasource.url=jdbc:h2:mem:testdb*

*spring.datasource.driverClassName=org.h2.Driver*

*spring.datasource.username=sa*

*spring.datasource.password=password*

*spring.jpa.database-platform=org.hibernate.dialect.H2Dialect*

Setted Employee Management system

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

<relativePath/>

</parent>

<groupId>com.week3</groupId>

<artifactId>SpringAndMaven</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringAndMaven</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Data JPA -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version> <!-- or any recent version -->

<scope>runtime</scope>

</dependency>

<!-- If using Maven -->

<!-- This includes Logback + SLF4J bindings correctly -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

<!-- AspectJ (if needed explicitly) -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.21</version>

</dependency>

<!-- Spring Test -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**application.properties**

spring.application.name=SpringAndMaven

# Database

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

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

spring.datasource.username=root

spring.datasource.password=123

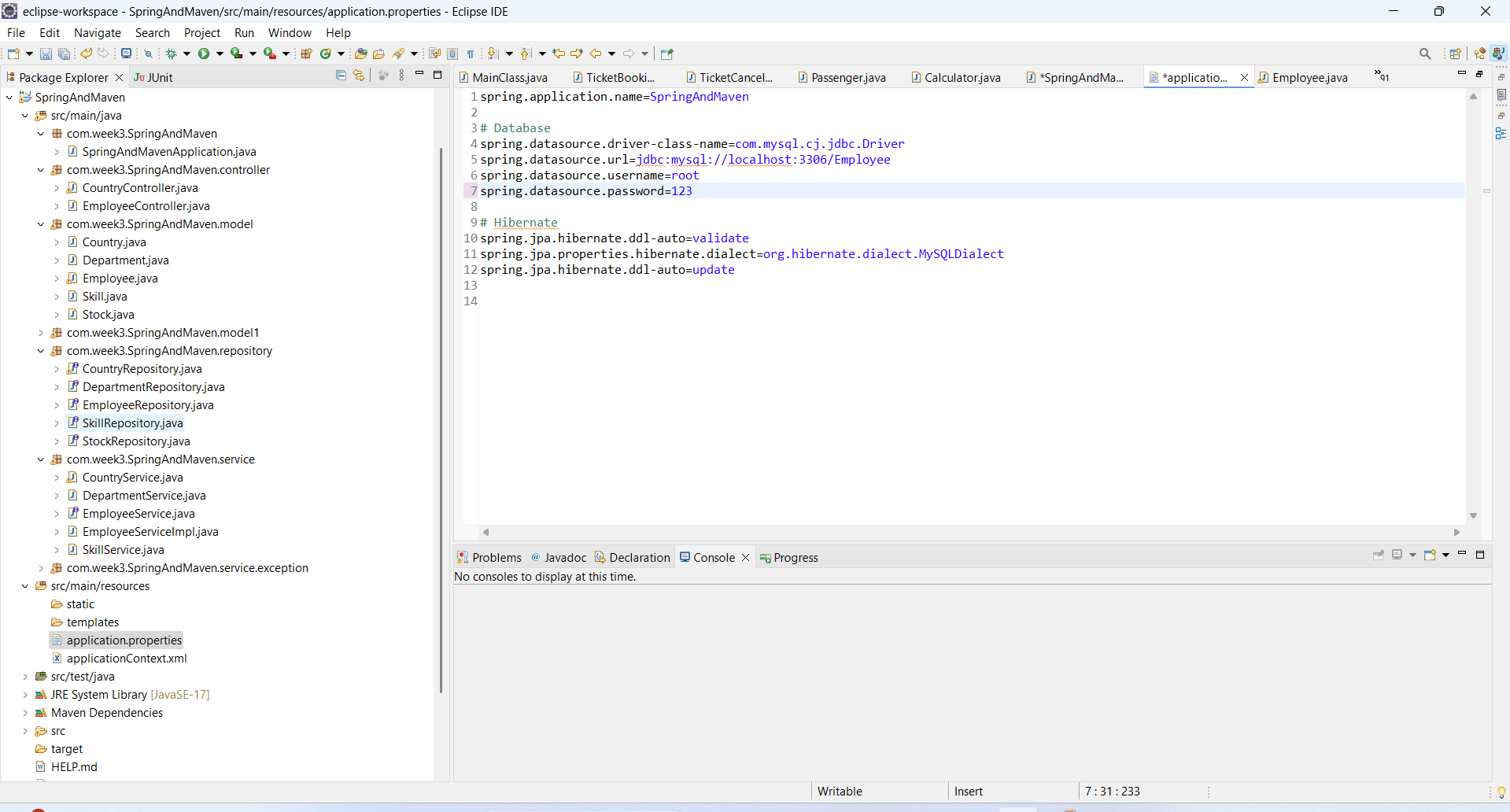
# Hibernate

spring.jpa.hibernate.ddl-auto=validate

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

spring.jpa.hibernate.ddl-auto=update

**OUTPUT**

****

**Exercise 2: Employee Management System - Creating Entities**

**Business Scenario:**

Define JPA entities for Employee and Department with appropriate relationships.

**Instructions:**

1. **Creating JPA Entities:**
   * Define **Employee** entity with fields: **id, name, email, department**.
   * Define **Department** entity with fields: **id, name**.
2. **Mapping Entities to Database Tables:**
   * Use annotations like **@Entity, @Table, @Id, @GeneratedValue**, etc.

Define one-to-many relationship between **Department** and **Employee**

**Employee.java**

**package** com.week3.SpringAndMaven.model;

**import** jakarta.persistence.\*;

@Entity

@Table(name = "employees")

**public** **class** Employee {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String name;

**private** String email;

// Many employees belong to one department

@ManyToOne

@JoinColumn(name = "department\_id")

**private** Department department;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** Department getDepartment() {

**return** department;

}

**public** **void** setDepartment(Department department) {

**this**.department = department;

}

}

**Department.java**

**package** com.week3.SpringAndMaven.model;

**import** java.util.\*;

**import** jakarta.persistence.\*;

@Entity

@Table(name = "departments")

**public** **class** Department {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String name;

// One department can have many employees

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

**private** List<Employee> employees;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** List<Employee> getEmployees() {

**return** employees;

}

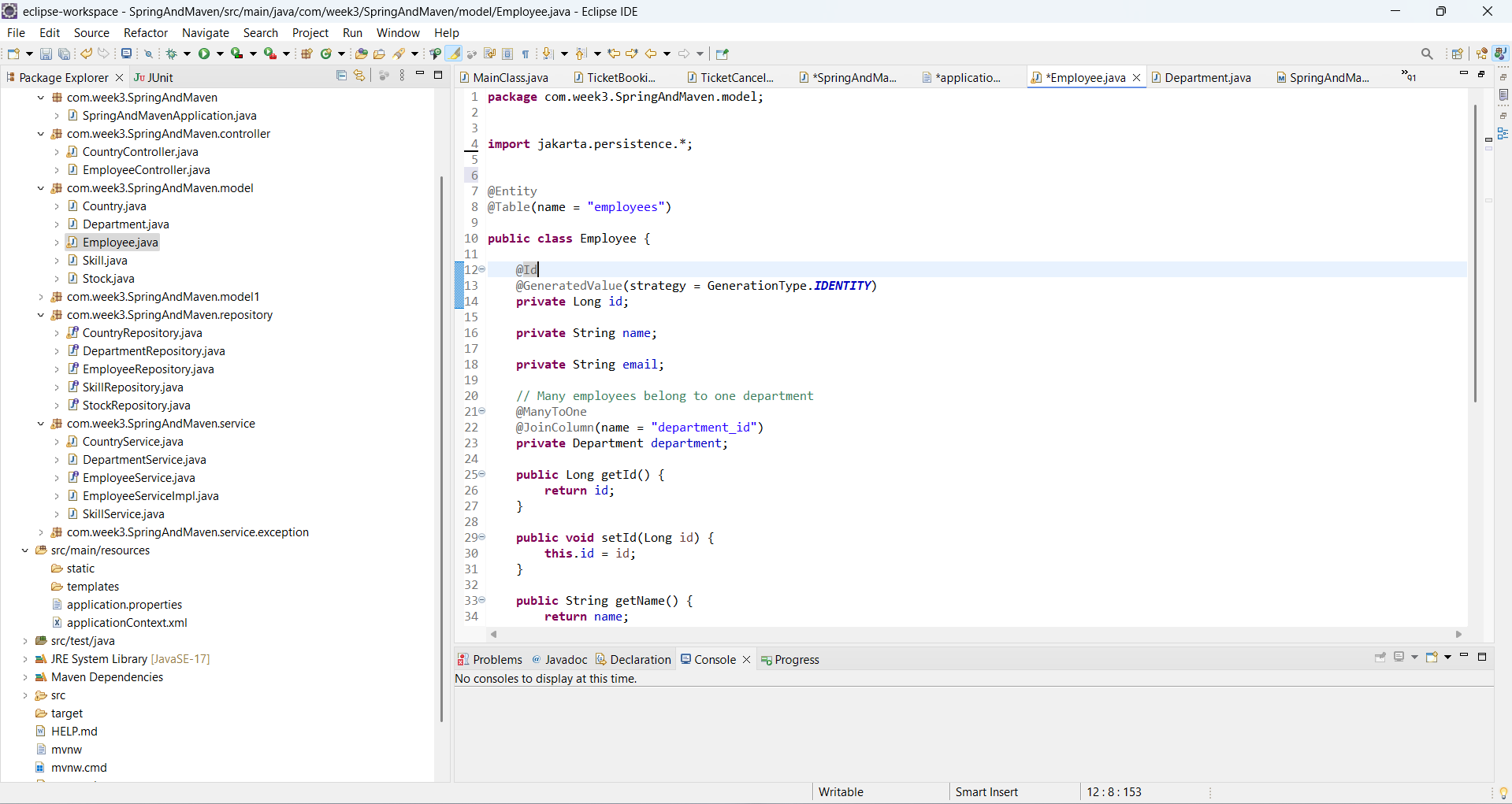
**public** **void** setEmployees(List<Employee> employees) {

**this**.employees = employees;

}

}

**OUTPUT**

****

**Exercise 3: Employee Management System - Creating Repositories**

**Business Scenario:**

Create repositories for Employee and Department entities to perform CRUD operations.

**Instructions:**

1. **Overview of Spring Data Repositories:**
   * Learn the benefits of using Spring Data repositories.
2. **Creating Repositories:**
   * Create **EmployeeRepository** and **DepartmentRepository** interfaces extending **JpaRepository**.
   * Define derived query methods in these repositories.

**DepartmentRepository.java**

**package** com.week3.SpringAndMaven.repository;

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

**import** com.week3.SpringAndMaven.model.Department;

**public** **interface** DepartmentRepository **extends** JpaRepository<Department, Long> {

// Derived query: find department by name

Department findByName(String name);

}

**EmployeeRepository.java**

**package** com.week3.SpringAndMaven.repository;

**import** java.util.\*;

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

**import** com.week3.SpringAndMaven.model.Employee;

**public** **interface** EmployeeRepository **extends** JpaRepository<Employee, Long> {

// Derived query method: find all employees by department name

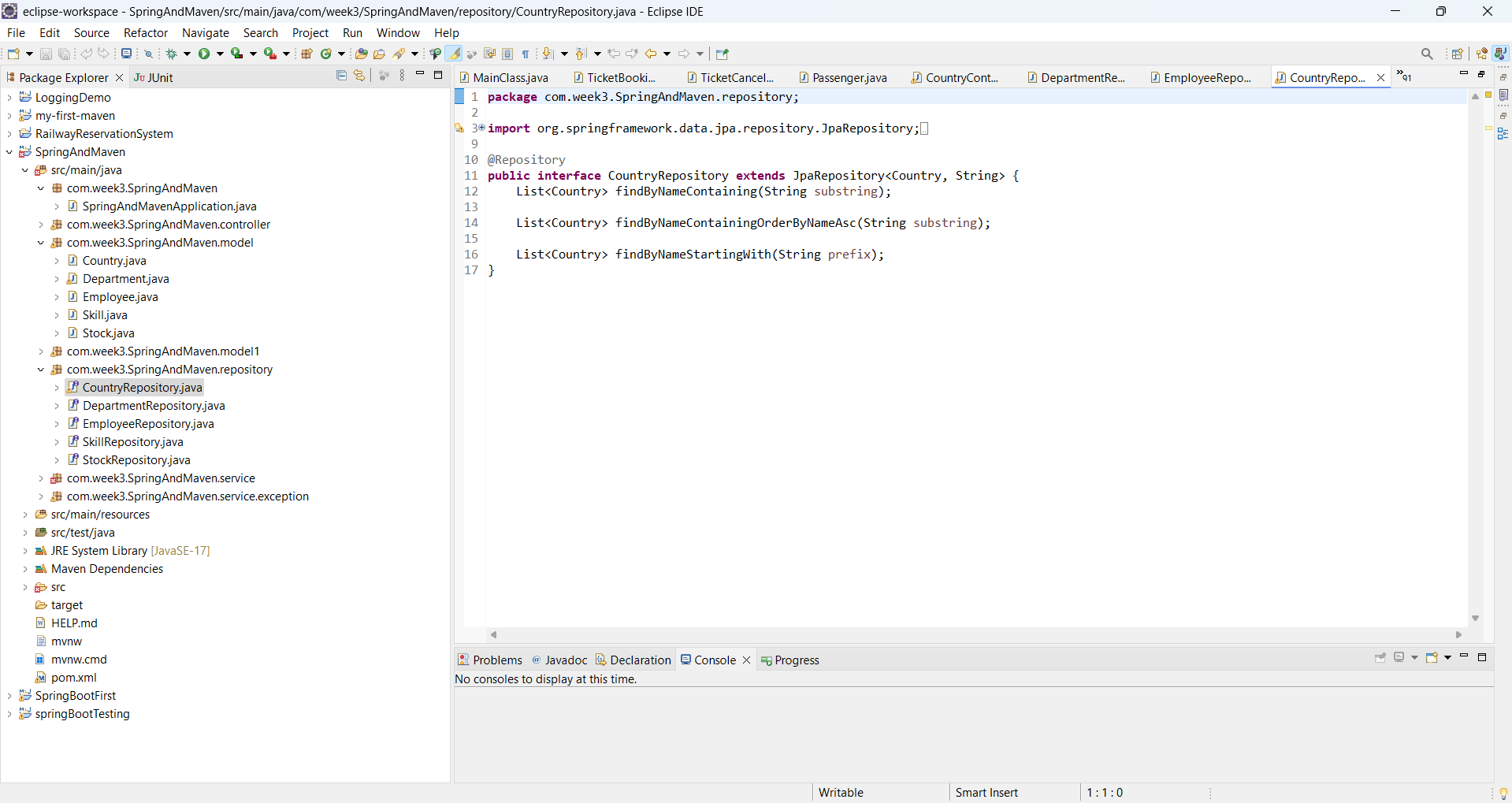
List<Employee> findByDepartmentName(String departmentName);

// Find employee by email

Employee findByEmail(String email);

}

**OUTPUT**



**Exercise 4: Employee Management System - Implementing CRUD Operations**

**Business Scenario:**

Implement CRUD operations for managing employees and departments.

**Instructions:**

1. **Basic CRUD Operations:**
   * Use **JpaRepository** methods to create, read, update, and delete employees and departments.
   * Implement RESTful endpoints for these operations using **EmployeeController** and **DepartmentController**.

**DepartmentController.java**

**package** com.week3.SpringAndMaven.controller;

**import** java.util.List;

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

**import** org.springframework.web.bind.annotation.\*;

**import** com.week3.SpringAndMaven.model.Department;

**import** com.week3.SpringAndMaven.repository.DepartmentRepository;

@RestController

@RequestMapping("/departments")

**public** **class** DepartmentController {

@Autowired

**private** DepartmentRepository departmentRepository;

// Create department

@PostMapping

**public** Department createDepartment(@RequestBody Department department) {

**return** departmentRepository.save(department);

}

// Get all departments

@GetMapping

**public** List<Department> getAllDepartments() {

**return** departmentRepository.findAll();

}

// Get department by ID

@GetMapping("/{id}")

**public** Department getDepartmentById(@PathVariable Long id) {

**return** departmentRepository.findById(id).orElse(**null**);

}

// Update department

@PutMapping("/{id}")

**public** Department updateDepartment(@PathVariable Long id, @RequestBody Department updatedDepartment) {

**return** departmentRepository.findById(id).map(department -> {

department.setName(updatedDepartment.getName());

**return** departmentRepository.save(department);

}).orElse(**null**);

}

// Delete department

@DeleteMapping("/{id}")

**public** **void** deleteDepartment(@PathVariable Long id) {

departmentRepository.deleteById(id);

}

}

**EmployeeController.java**

**package** com.week3.SpringAndMaven.controller;

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

**import** org.springframework.web.bind.annotation.\*;

**import** com.week3.SpringAndMaven.model.Employee;

**import** com.week3.SpringAndMaven.repository.DepartmentRepository;

**import** com.week3.SpringAndMaven.repository.EmployeeRepository;

**import** com.week3.SpringAndMaven.service.EmployeeService;

**import** java.util.List;

@RestController

@RequestMapping("/employees")

**public** **class** EmployeeController {

@Autowired

**private** EmployeeRepository employeeRepository;

@Autowired

**private** DepartmentRepository departmentRepository;

// Create employee

@PostMapping

**public** Employee createEmployee(@RequestBody Employee employee) {

// Ensure department is loaded from DB if ID is present

**if** (employee.getDepartment() != **null** && employee.getDepartment().getId() != **null**) {

departmentRepository.findById(employee.getDepartment().getId())

.ifPresent(employee::setDepartment);

}

**return** employeeRepository.save(employee);

}

// Get all employees

@GetMapping

**public** List<Employee> getAllEmployees() {

**return** employeeRepository.findAll();

}

// Get employee by ID

@GetMapping("/{id}")

**public** Employee getEmployeeById(@PathVariable Long id) {

**return** employeeRepository.findById(id).orElse(**null**);

}

// Update employee

@PutMapping("/{id}")

**public** Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {

**return** employeeRepository.findById(id).map(employee -> {

employee.setName(updatedEmployee.getName());

employee.setEmail(updatedEmployee.getEmail());

**if** (updatedEmployee.getDepartment() != **null** && updatedEmployee.getDepartment().getId() != **null**) {

departmentRepository.findById(updatedEmployee.getDepartment().getId())

.ifPresent(employee::setDepartment);

}

**return** employeeRepository.save(employee);

}).orElse(**null**);

}

// Delete employee

@DeleteMapping("/{id}")

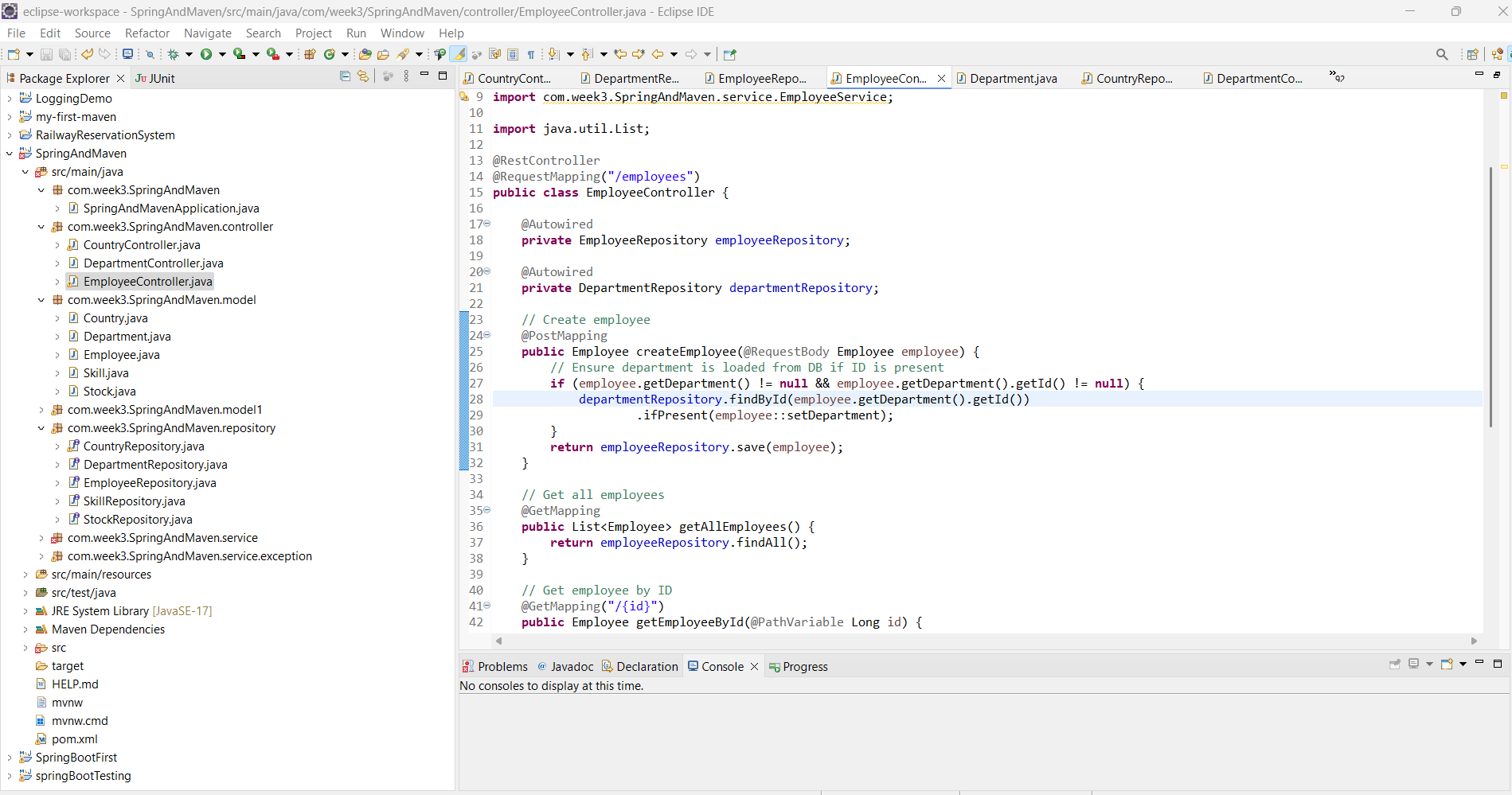
**public** **void** deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

**OUTPUT**

****

**Exercise 4: Employee Management System - Implementing CRUD Operations**

**Business Scenario:**

Implement CRUD operations for managing employees and departments.

**Instructions:**

1. **Basic CRUD Operations:**
   * Use **JpaRepository** methods to create, read, update, and delete employees and departments.
   * Implement RESTful endpoints for these operations using **EmployeeController** and **DepartmentController**.

**DepartmentController.java**

**package** com.week3.SpringAndMaven.controller;

**import** java.util.List;

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

**import** org.springframework.web.bind.annotation.\*;

**import** com.week3.SpringAndMaven.model.Department;

**import** com.week3.SpringAndMaven.repository.DepartmentRepository;

@RestController

@RequestMapping("/departments")

**public** **class** DepartmentController {

@Autowired

**private** DepartmentRepository departmentRepository;

// Create department

@PostMapping

**public** Department createDepartment(@RequestBody Department department) {

**return** departmentRepository.save(department);

}

// Get all departments

@GetMapping

**public** List<Department> getAllDepartments() {

**return** departmentRepository.findAll();

}

// Get department by ID

@GetMapping("/{id}")

**public** Department getDepartmentById(@PathVariable Long id) {

**return** departmentRepository.findById(id).orElse(**null**);

}

// Update department

@PutMapping("/{id}")

**public** Department updateDepartment(@PathVariable Long id, @RequestBody Department updatedDepartment) {

**return** departmentRepository.findById(id).map(department -> {

department.setName(updatedDepartment.getName());

**return** departmentRepository.save(department);

}).orElse(**null**);

}

// Delete department

@DeleteMapping("/{id}")

**public** **void** deleteDepartment(@PathVariable Long id) {

departmentRepository.deleteById(id);

}

}

**EmployeeController.java**

**package** com.week3.SpringAndMaven.controller;

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

**import** org.springframework.web.bind.annotation.\*;

**import** com.week3.SpringAndMaven.model.Employee;

**import** com.week3.SpringAndMaven.repository.DepartmentRepository;

**import** com.week3.SpringAndMaven.repository.EmployeeRepository;

**import** com.week3.SpringAndMaven.service.EmployeeService;

**import** java.util.List;

@RestController

@RequestMapping("/employees")

**public** **class** EmployeeController {

@Autowired

**private** EmployeeRepository employeeRepository;

@Autowired

**private** DepartmentRepository departmentRepository;

// Create employee

@PostMapping

**public** Employee createEmployee(@RequestBody Employee employee) {

// Ensure department is loaded from DB if ID is present

**if** (employee.getDepartment() != **null** && employee.getDepartment().getId() != **null**) {

departmentRepository.findById(employee.getDepartment().getId())

.ifPresent(employee::setDepartment);

}

**return** employeeRepository.save(employee);

}

// Get all employees

@GetMapping

**public** List<Employee> getAllEmployees() {

**return** employeeRepository.findAll();

}

// Get employee by ID

@GetMapping("/{id}")

**public** Employee getEmployeeById(@PathVariable Long id) {

**return** employeeRepository.findById(id).orElse(**null**);

}

// Update employee

@PutMapping("/{id}")

**public** Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {

**return** employeeRepository.findById(id).map(employee -> {

employee.setName(updatedEmployee.getName());

employee.setEmail(updatedEmployee.getEmail());

**if** (updatedEmployee.getDepartment() != **null** && updatedEmployee.getDepartment().getId() != **null**) {

departmentRepository.findById(updatedEmployee.getDepartment().getId())

.ifPresent(employee::setDepartment);

}

**return** employeeRepository.save(employee);

}).orElse(**null**);

}

// Delete employee

@DeleteMapping("/{id}")

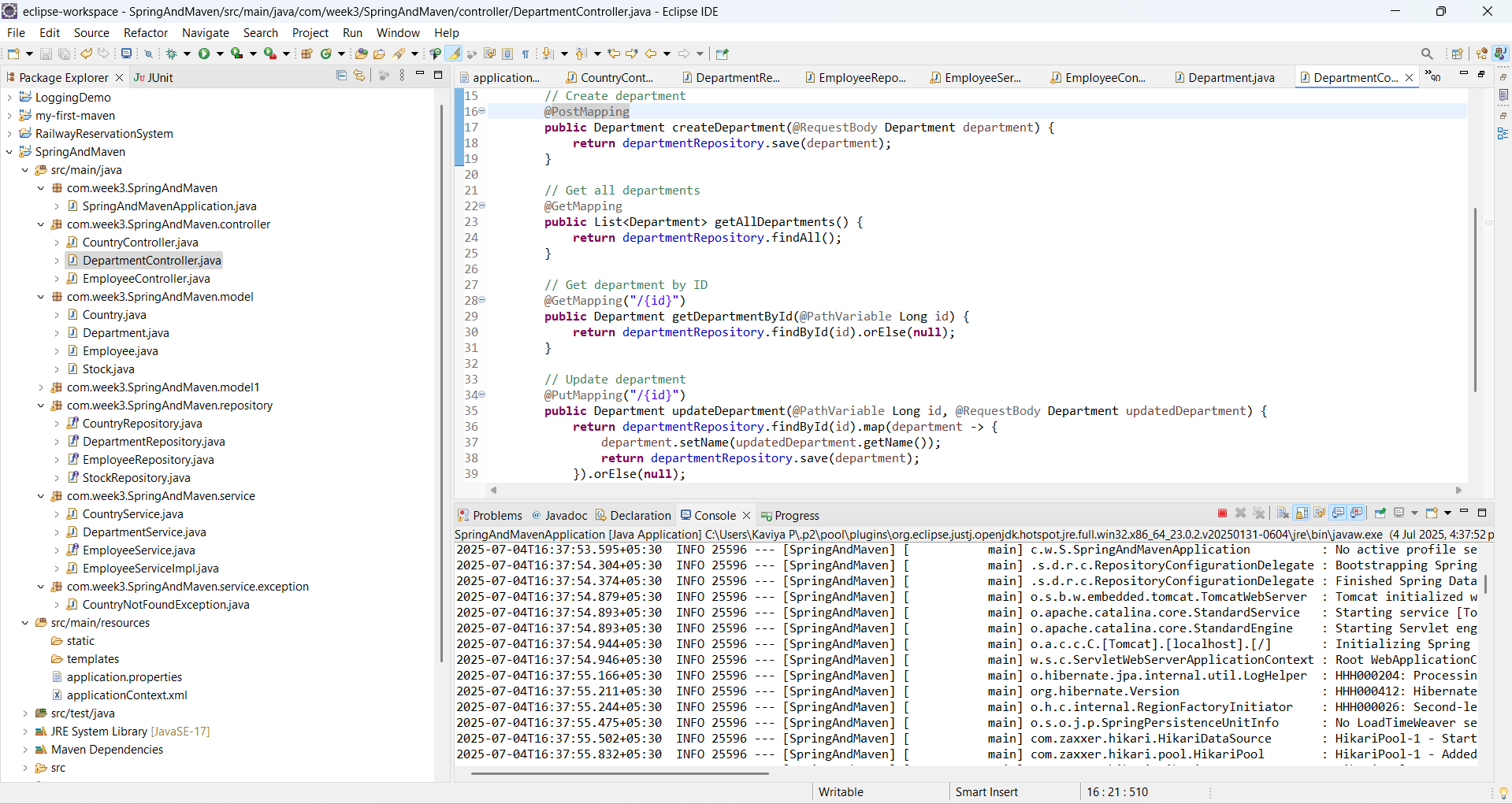
**public** **void** deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

**OUTPUT**

****

**Exercise 5: Employee Management System - Defining Query Methods**

**Business Scenario:**

Enhance your repository to support custom queries.

**Instructions:**

1. **Defining Query Methods:**
   * Use keywords in method names to create custom query methods.
   * Implement custom query methods using the **@Query** annotation.
2. **Named Queries:**
   * Define and execute named queries with **@NamedQuery** and **@NamedQueries**.

**EmployeeController.java**

**package** com.week3.SpringAndMaven.controller;

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

**import** org.springframework.web.bind.annotation.\*;

**import** com.week3.SpringAndMaven.model.Employee;

**import** com.week3.SpringAndMaven.repository.DepartmentRepository;

**import** com.week3.SpringAndMaven.repository.EmployeeRepository;

**import** com.week3.SpringAndMaven.service.EmployeeService;

**import** java.util.List;

@RestController

@RequestMapping("/employees")

**public** **class** EmployeeController {

@Autowired

**private** EmployeeRepository employeeRepository;

@Autowired

**private** DepartmentRepository departmentRepository;

// Create employee

@PostMapping

**public** Employee createEmployee(@RequestBody Employee employee) {

// Ensure department is loaded from DB if ID is present

**if** (employee.getDepartment() != **null** && employee.getDepartment().getId() != **null**) {

departmentRepository.findById(employee.getDepartment().getId())

.ifPresent(employee::setDepartment);

}

**return** employeeRepository.save(employee);

}

// Get all employees

@GetMapping

**public** List<Employee> getAllEmployees() {

**return** employeeRepository.findAll();

}

// Get employee by ID

@GetMapping("/{id}")

**public** Employee getEmployeeById(@PathVariable Long id) {

**return** employeeRepository.findById(id).orElse(**null**);

}

// Update employee

@PutMapping("/{id}")

**public** Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {

**return** employeeRepository.findById(id).map(employee -> {

employee.setName(updatedEmployee.getName());

**if** (updatedEmployee.getDepartment() != **null** && updatedEmployee.getDepartment().getId() != **null**) {

departmentRepository.findById(updatedEmployee.getDepartment().getId())

.ifPresent(employee::setDepartment);

}

**return** employeeRepository.save(employee);

}).orElse(**null**);

}

// Delete employee

@DeleteMapping("/{id}")

**public** **void** deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

**Employee.java**

**package** com.week3.SpringAndMaven.model;

**import** java.util.Date;

**import** jakarta.persistence.\*;

@Entity

@NamedQueries({

@NamedQuery(

name = "Employee.findByDepartmentName",

query = "SELECT e FROM Employee e WHERE e.department.name = :name"

),

@NamedQuery(

name = "Employee.findByNameStartsWith",

query = "SELECT e FROM Employee e WHERE e.name LIKE CONCAT(:prefix, '%')"

)

})

@Table(name = "employees")

**public** **class** Employee {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "em\_id")

**private** Long id;

@Column(name = "em\_name")

**private** String name;

@Column(name = "em\_salary")

**private** Double salary;

@Column(name = "em\_permanent")

**private** Boolean permanent;

@Column(name = "em\_date\_of\_birth")

**private** Date dateOfBirth;

@ManyToOne

@JoinColumn(name = "em\_dp\_id")

**private** Department department;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Double getSalary() {

**return** salary;

}

**public** **void** setSalary(Double salary) {

**this**.salary = salary;

}

**public** Boolean getPermanent() {

**return** permanent;

}

**public** **void** setPermanent(Boolean permanent) {

**this**.permanent = permanent;

}

**public** Date getDateOfBirth() {

**return** dateOfBirth;

}

**public** **void** setDateOfBirth(Date dateOfBirth) {

**this**.dateOfBirth = dateOfBirth;

}

**public** Department getDepartment() {

**return** department;

}

**public** **void** setDepartment(Department department) {

**this**.department = department;

}

}

**SpringAndMavenApplication.java**

**package** com.week3.SpringAndMaven;

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

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

@SpringBootApplication

**public** **class** SpringAndMavenApplication {

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

SpringApplication.*run*(SpringAndMavenApplication.**class**, args);

}

}

**EmployeeRepository.java**

**package** com.week3.SpringAndMaven.repository;

**import** java.util.\*;

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

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

**import** org.springframework.data.repository.query.Param;

**import** com.week3.SpringAndMaven.model.Employee;

**public** **interface** EmployeeRepository **extends** JpaRepository<Employee, Long> {

// 1. Derived Query Methods

List<Employee> findByName(String name);

List<Employee> findByDepartmentName(String deptName);

List<Employee> findBySalaryGreaterThan(**double** salary);

List<Employee> findByNameStartingWith(String prefix);

// 2. Custom JPQL using @Query

@Query("SELECT e FROM Employee e WHERE e.salary > :salary")

List<Employee> getEmployeesWithSalaryMoreThan(@Param("salary") **double** salary);

@Query("SELECT e FROM Employee e WHERE e.department.name = :deptName")

List<Employee> getEmployeesByDepartmentName(@Param("deptName") String name);

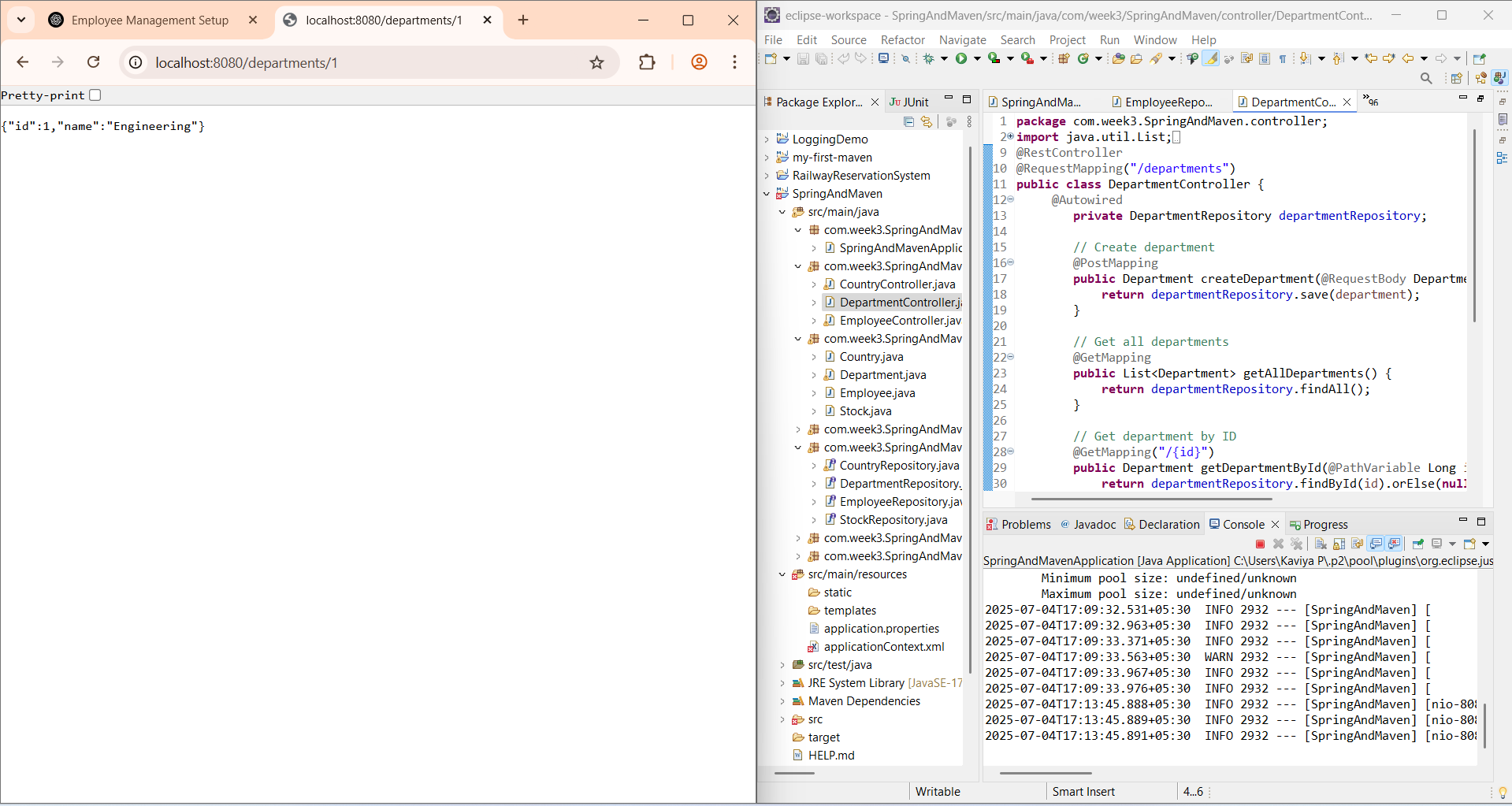
// 3. Native Query Example

@Query(value = "SELECT \* FROM employee WHERE name LIKE %:key%", nativeQuery = **true**)

List<Employee> searchByName(@Param("key") String keyword);

}

**OUTPUT**



**Exercise 6: Employee Management System - Implementing Pagination and Sorting**

**Business Scenario:**

Add pagination and sorting capabilities to your employee search functionality.

**Instructions:**

1. **Pagination:**
   * Implement pagination for the employee list using **Page** and **Pageable**.
2. **Sorting:**
   * Add sorting functionality to your queries.
   * Combine pagination and sorting in your search endpoint.

**Employee.java**

**package** com.week3.SpringAndMaven.model;

**import** java.util.Date;

**import** jakarta.persistence.\*;

@Entity

@NamedQueries({

@NamedQuery(

name = "Employee.findByDepartmentName",

query = "SELECT e FROM Employee e WHERE e.department.name = :name"

),

@NamedQuery(

name = "Employee.findByNameStartsWith",

query = "SELECT e FROM Employee e WHERE e.name LIKE CONCAT(:prefix, '%')"

)

})

@Table(name = "employees")

**public** **class** Employee {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "em\_id")

**private** Long id;

@Column(name = "em\_name")

**private** String name;

@Column(name = "em\_salary")

**private** Double salary;

@Column(name = "em\_permanent")

**private** Boolean permanent;

@Column(name = "em\_date\_of\_birth")

**private** Date dateOfBirth;

@ManyToOne

@JoinColumn(name = "em\_dp\_id")

**private** Department department;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Double getSalary() {

**return** salary;

}

**public** **void** setSalary(Double salary) {

**this**.salary = salary;

}

**public** Boolean getPermanent() {

**return** permanent;

}

**public** **void** setPermanent(Boolean permanent) {

**this**.permanent = permanent;

}

**public** Date getDateOfBirth() {

**return** dateOfBirth;

}

**public** **void** setDateOfBirth(Date dateOfBirth) {

**this**.dateOfBirth = dateOfBirth;

}

**public** Department getDepartment() {

**return** department;

}

**public** **void** setDepartment(Department department) {

**this**.department = department;

}

}

**EmployeeRepository.java**

**package** com.week3.SpringAndMaven.controller;

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

**import** org.springframework.web.bind.annotation.\*;

**import** com.week3.SpringAndMaven.model.Employee;

**import** com.week3.SpringAndMaven.repository.DepartmentRepository;

**import** com.week3.SpringAndMaven.repository.EmployeeRepository;

**import** com.week3.SpringAndMaven.service.EmployeeService;

**import** org.springframework.data.domain.Page;

**import** org.springframework.data.domain.Pageable;

**import** org.springframework.data.domain.PageRequest;

**import** org.springframework.data.domain.Sort;

**import** java.util.List;

@RestController

@RequestMapping("/employees")

**public** **class** EmployeeController {

@Autowired

**private** EmployeeRepository employeeRepository;

@Autowired

**private** DepartmentRepository departmentRepository;

// Create employee

@PostMapping

**public** Employee createEmployee(@RequestBody Employee employee) {

// Ensure department is loaded from DB if ID is present

**if** (employee.getDepartment() != **null** && employee.getDepartment().getId() != **null**) {

departmentRepository.findById(employee.getDepartment().getId())

.ifPresent(employee::setDepartment);

}

**return** employeeRepository.save(employee);

}

@GetMapping("/paginated/list")

**public** List<Employee> getPaginatedSortedList(

@RequestParam(defaultValue = "0") **int** page,

@RequestParam(defaultValue = "5") **int** size,

@RequestParam(defaultValue = "id") String sortBy,

@RequestParam(defaultValue = "asc") String direction) {

Sort sort = direction.equalsIgnoreCase("desc") ? Sort.*by*(sortBy).descending() : Sort.*by*(sortBy).ascending();

Pageable pageable = PageRequest.*of*(page, size, sort);

**return** employeeRepository.findAll(pageable).getContent();

}

// Get all employees

@GetMapping

**public** List<Employee> getAllEmployees() {

**return** employeeRepository.findAll();

}

// Get employee by ID

@GetMapping("/{id}")

**public** Employee getEmployeeById(@PathVariable Long id) {

**return** employeeRepository.findById(id).orElse(**null**);

}

// Update employee

@PutMapping("/{id}")

**public** Employee updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmployee) {

**return** employeeRepository.findById(id).map(employee -> {

employee.setName(updatedEmployee.getName());

**if** (updatedEmployee.getDepartment() != **null** && updatedEmployee.getDepartment().getId() != **null**) {

departmentRepository.findById(updatedEmployee.getDepartment().getId())

.ifPresent(employee::setDepartment);

}

**return** employeeRepository.save(employee);

}).orElse(**null**);

}

// Delete employee

@DeleteMapping("/{id}")

**public** **void** deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

**EmployeeRepository.java**

**package** com.week3.SpringAndMaven.repository;

**import** java.util.\*;

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

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

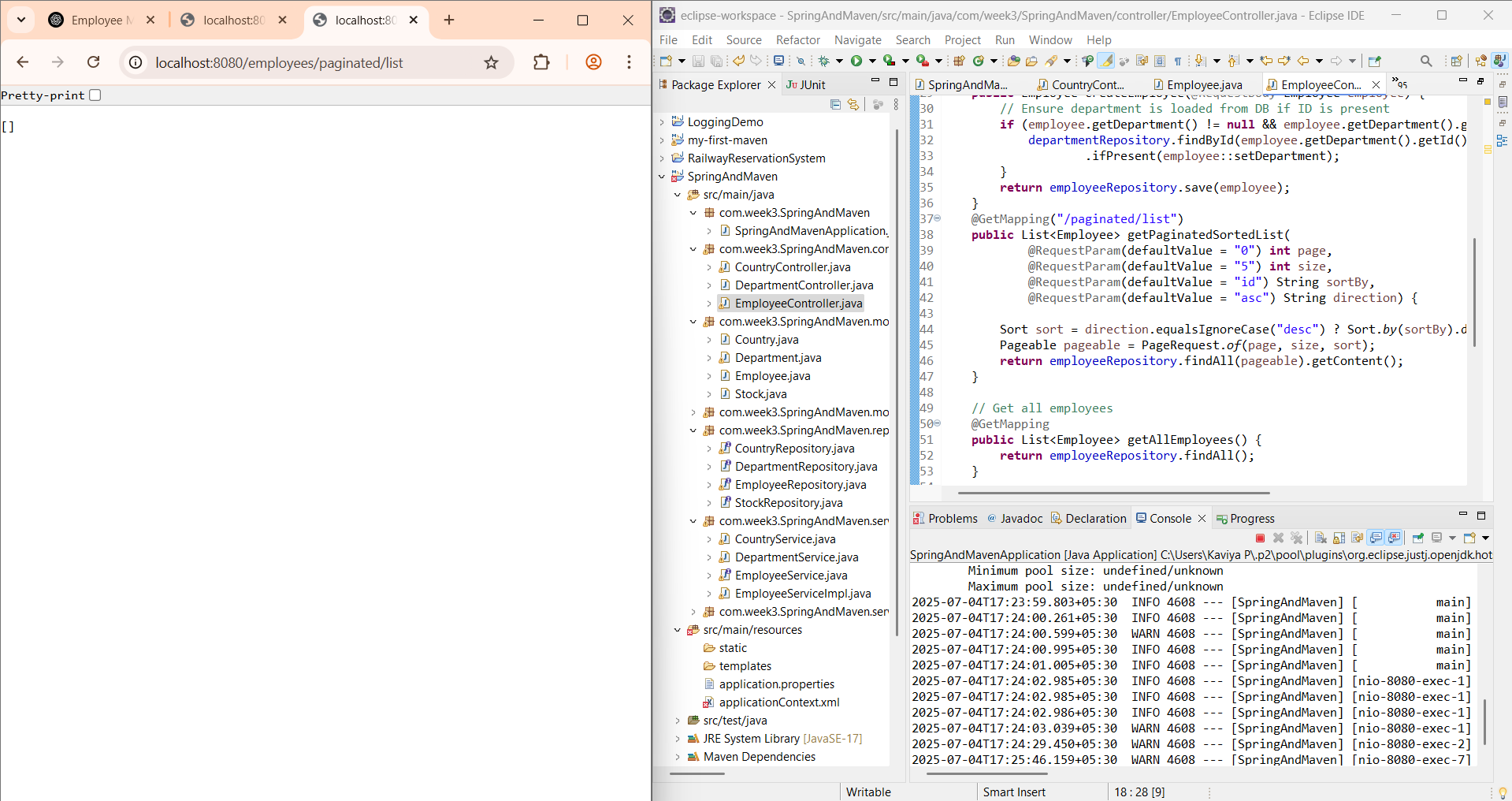
**import** org.springframework.data.repository.query.Param;

**import** com.week3.SpringAndMaven.model.Employee;

**public** **interface** EmployeeRepository **extends** JpaRepository<Employee, Long> {

}

**OUTPUT**

****

**Exercise 7: Employee Management System - Enabling Entity Auditing**

**Business Scenario:**

Implement auditing to track the creation and modification of employees and departments.

**Instructions:**

1. **Entity Auditing:**
   * Enable auditing in your application by configuring auditing properties.
   * Use annotations like **@CreatedBy, @LastModifiedBy, @CreatedDate**, and **@LastModifiedDate**.

**AuditConfig.java**

**package** com.week3.SpringAndMaven;

**import** java.util.Optional;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.data.domain.AuditorAware;

@Configuration

**public** **class** AuditConfig {

@Bean

**public** AuditorAware<String> auditorAware() {

// This can be replaced with logged-in user from Spring Security

**return** () -> Optional.*of*("system\_user");

}

}

**SpringAndMavenApplication.java**

**package** com.week3.SpringAndMaven;

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

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

**import** org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication

@EnableJpaAuditing(auditorAwareRef = "auditorAware")

**public** **class** SpringAndMavenApplication {

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

SpringApplication.*run*(SpringAndMavenApplication.**class**, args);

}

}

**Auditable.java**

**package** com.week3.SpringAndMaven.model;

**import** java.time.LocalDateTime;

**import** org.springframework.data.annotation.\*;

**import** org.springframework.data.jpa.domain.support.AuditingEntityListener;

**import** jakarta.persistence.Column;

**import** jakarta.persistence.EntityListeners;

**import** jakarta.persistence.MappedSuperclass;

@MappedSuperclass

@EntityListeners(AuditingEntityListener.**class**)

**public** **class** Auditable {

@CreatedBy

@Column(updatable = **false**)

**private** String createdBy;

@CreatedDate

@Column(updatable = **false**)

**private** LocalDateTime createdDate;

@LastModifiedBy

**private** String lastModifiedBy;

@LastModifiedDate

**private** LocalDateTime lastModifiedDate;

}

**Department.java**

**package** com.week3.SpringAndMaven.model;

**import** java.util.List;

**import** jakarta.persistence.\*;

@Entity

**public** **class** Department **extends** Auditable{

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "dp\_id")

**private** Long id;

@Column(name = "dp\_name")

**private** String name;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**Employee.java**

**package** com.week3.SpringAndMaven.model;

**import** java.util.Date;

**import** jakarta.persistence.\*;

@Entity

@Table(name = "employees")

**public** **class** Employee **extends** Auditable{

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "em\_id")

**private** Long id;

@Column(name = "em\_name")

**private** String name;

@Column(name = "em\_salary")

**private** Double salary;

@Column(name = "em\_permanent")

**private** Boolean permanent;

@Column(name = "em\_date\_of\_birth")

**private** Date dateOfBirth;

@ManyToOne

@JoinColumn(name = "em\_dp\_id")

**private** Department department;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Double getSalary() {

**return** salary;

}

**public** **void** setSalary(Double salary) {

**this**.salary = salary;

}

**public** Boolean getPermanent() {

**return** permanent;

}

**public** **void** setPermanent(Boolean permanent) {

**this**.permanent = permanent;

}

**public** Date getDateOfBirth() {

**return** dateOfBirth;

}

**public** **void** setDateOfBirth(Date dateOfBirth) {

**this**.dateOfBirth = dateOfBirth;

}

**public** Department getDepartment() {

**return** department;

}

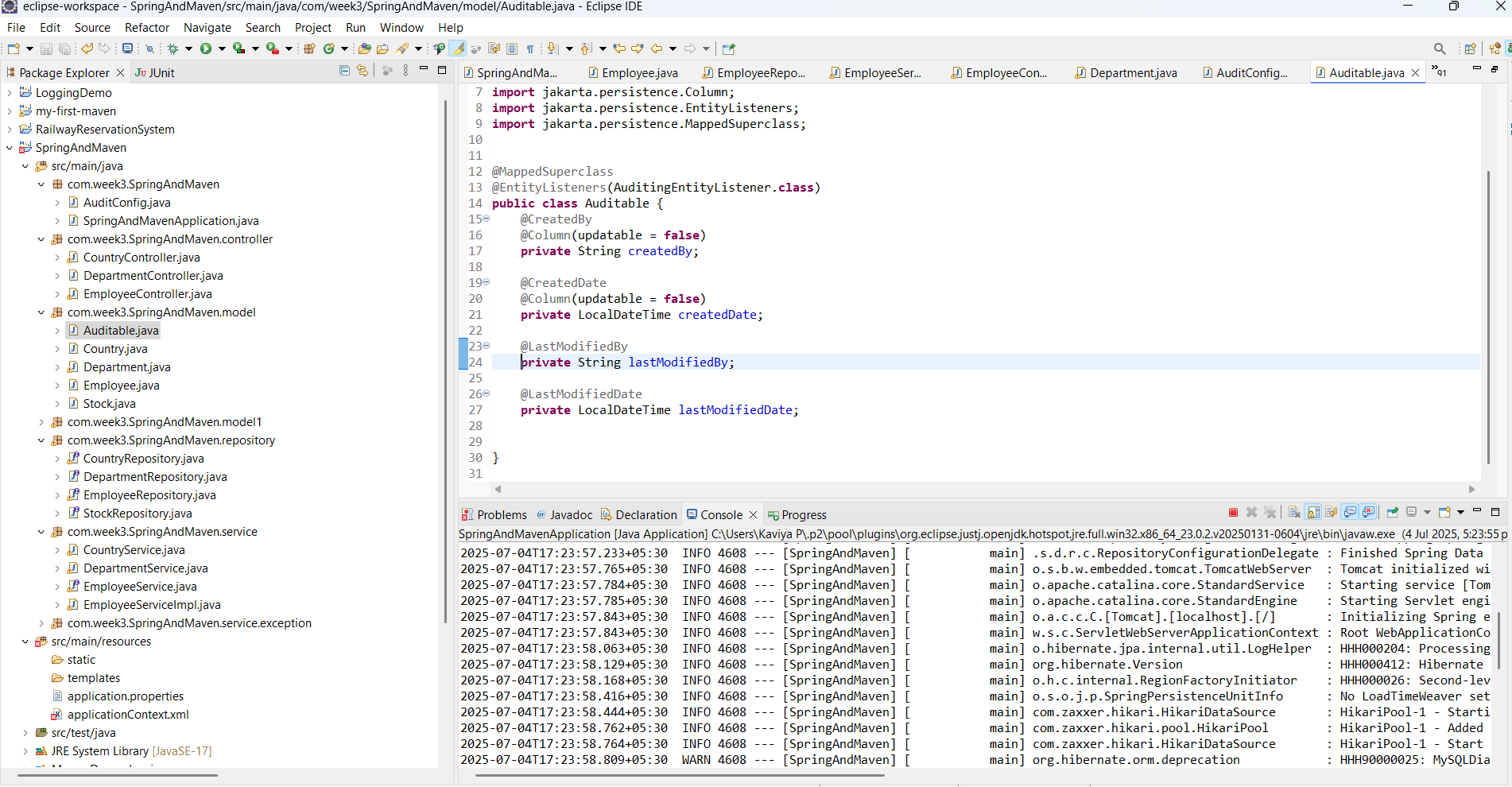
**public** **void** setDepartment(Department department) {

**this**.department = department;

}

}

**OUTPUT:**



**Exercise 8: Employee Management System - Creating Projections**

**Business Scenario:**

Create projections to fetch specific data subsets from the employee and department entities.

**Instructions:**

1. **Projections:**
   * Define interface-based and class-based projections.
   * Use **@Value** and constructor expressions to control the fetched data.

**EmployeeDTO.java**

**package** com.week3.SpringAndMaven;

**public** **class** EmployeeDTO {

**private** String name;

**private** String departmentName;

**public** EmployeeDTO(String name, String departmentName) {

**this**.name = name;

**this**.departmentName = departmentName;

}

// Getters and setters

}

**EmployeeNameSalaryProjection.java**

**package** com.week3.SpringAndMaven;

**public** **interface** EmployeeNameSalaryProjection {

String getName();

Double getSalary();

}

**EmployeeRepository.java**

**package** com.week3.SpringAndMaven.repository;

**import** java.util.\*;

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

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

**import** org.springframework.data.repository.query.Param;

**import** com.week3.SpringAndMaven.model.Employee;

**public** **interface** EmployeeRepository **extends** JpaRepository<Employee, Long> {

@Query("SELECT e FROM Employee e")

List<EmployeeNameSalaryProjection> findEmployeeNameAndSalary();

}

**Employee.java**

**package** com.week3.SpringAndMaven;

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

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

**import** org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication

@EnableJpaAuditing(auditorAwareRef = "auditorAware")

**public** **class** SpringAndMavenApplication {

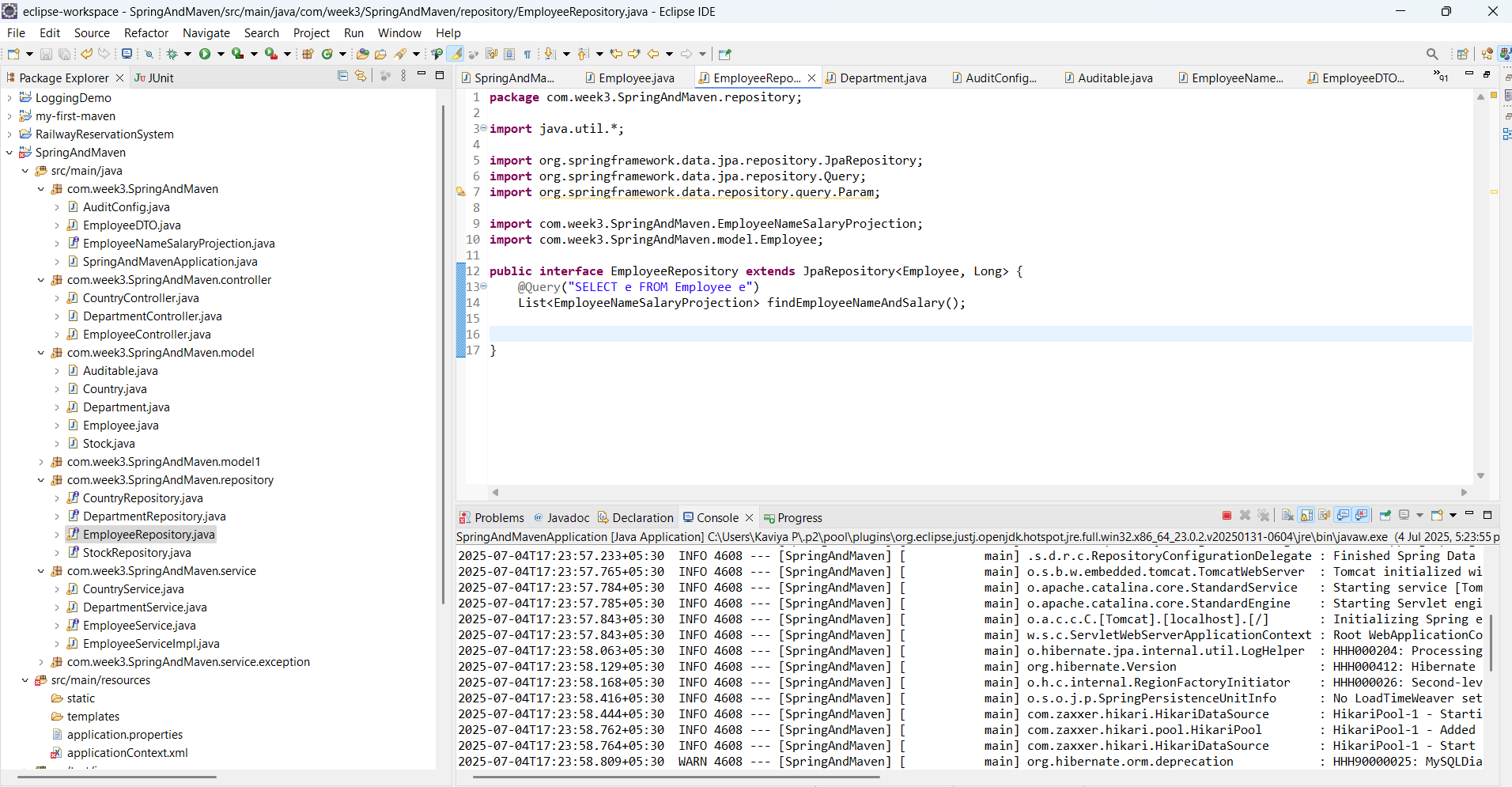
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringAndMavenApplication.**class**, args);

}

}

**OUTPUT**



**Exercise 9: Employee Management System - Customizing Data Source Configuration**

**Business Scenario:**

Customize your data source configuration and manage multiple data sources.

**Instructions:**

1. **Spring Boot Auto-Configuration:**
   * Leverage Spring Boot auto-configuration for data sources.
2. **Externalizing Configuration:**
   * Externalize configuration with application.properties.
   * Manage multiple data sources within your application.

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

<relativePath/>

</parent>

<groupId>com.week3</groupId>

<artifactId>SpringAndMaven</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringAndMaven</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Data JPA -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version> <!-- or any recent version -->

<scope>runtime</scope>

</dependency>

<!-- If using Maven -->

<!-- This includes Logback + SLF4J bindings correctly -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

<!-- AspectJ (if needed explicitly) -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.21</version>

</dependency>

<!-- Spring Test -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**applications.property**

spring.application.name=SpringAndMaven

# Database

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

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

spring.datasource.username=root

spring.datasource.password=123

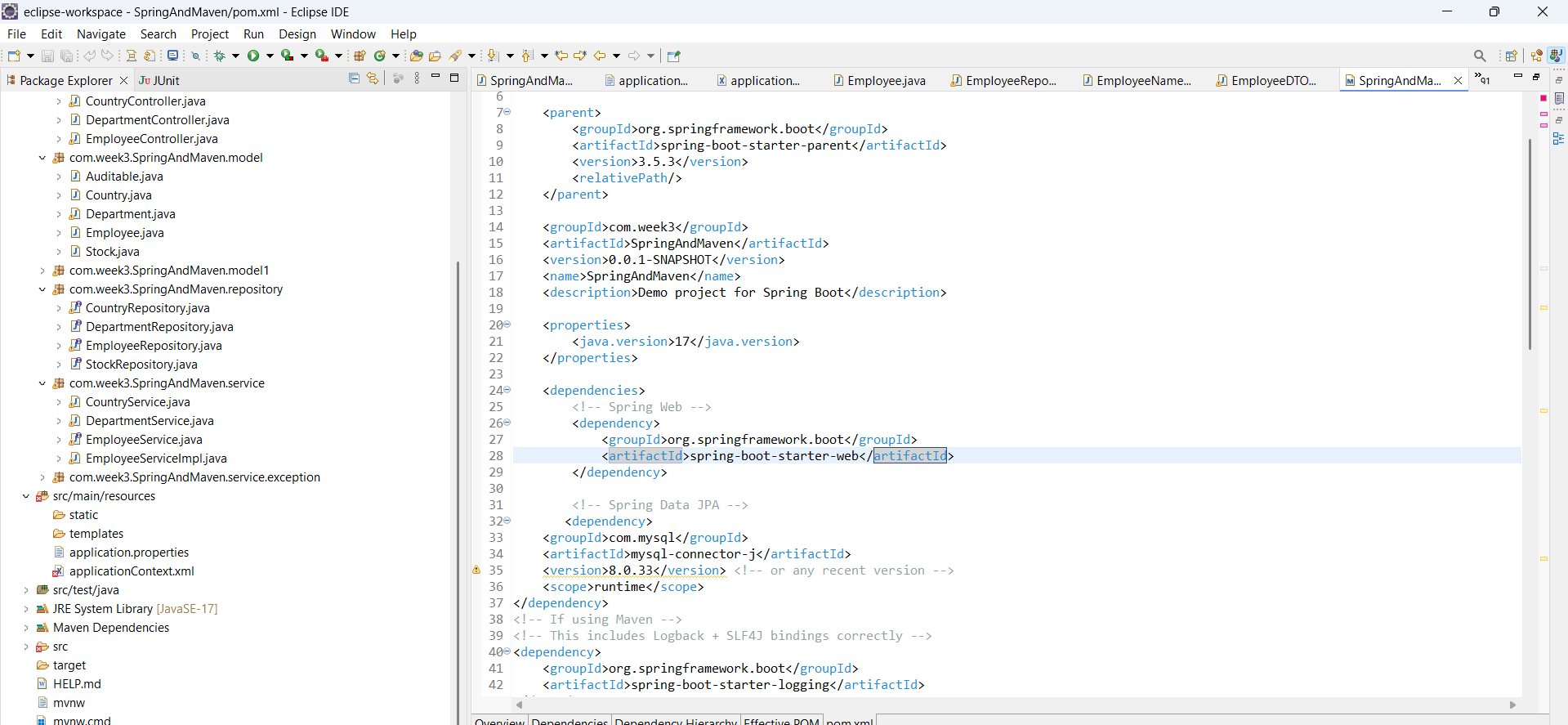
# Hibernate

spring.jpa.hibernate.ddl-auto=validate

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

spring.jpa.hibernate.ddl-auto=update

**OUTPUT**



**Exercise 10: Employee Management System - Hibernate-Specific Features**

**Business Scenario:**

Leverage Hibernate-specific features to enhance your application's performance and capabilities.

**Instructions:**

1. **Hibernate-Specific Annotations:**
   * Use Hibernate-specific annotations to customize entity mappings.
2. **Configuring Hibernate Dialect and Properties:**
   * Configure Hibernate dialect and properties for optimal performance.
3. **Batch Processing:**
   * Implement batch processing with Hibernate for bulk operations.

**Employee.java**

**package** com.week3.SpringAndMaven.model;

**import** java.util.Date;

**import** org.hibernate.annotations.DynamicInsert;

**import** org.hibernate.annotations.DynamicUpdate;

**import** jakarta.persistence.\*;

@Entity

@Table(name = "employees")

@DynamicInsert

@DynamicUpdate

**public** **class** Employee **extends** Auditable{

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "em\_id")

**private** Long id;

@Column(name = "em\_name")

**private** String name;

@Column(name = "em\_salary")

**private** Double salary;

@Column(name = "em\_permanent")

**private** Boolean permanent;

@Column(name = "em\_date\_of\_birth")

**private** Date dateOfBirth;

@ManyToOne

@JoinColumn(name = "em\_dp\_id")

**private** Department department;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Double getSalary() {

**return** salary;

}

**public** **void** setSalary(Double salary) {

**this**.salary = salary;

}

**public** Boolean getPermanent() {

**return** permanent;

}

**public** **void** setPermanent(Boolean permanent) {

**this**.permanent = permanent;

}

**public** Date getDateOfBirth() {

**return** dateOfBirth;

}

**public** **void** setDateOfBirth(Date dateOfBirth) {

**this**.dateOfBirth = dateOfBirth;

}

**public** Department getDepartment() {

**return** department;

}

**public** **void** setDepartment(Department department) {

**this**.department = department;

}

}

**application.properties**

# Hibernate

spring.jpa.hibernate.ddl-auto=validate

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

spring.jpa.hibernate.ddl-auto=update

# Use JDBC batch processing

spring.jpa.properties.hibernate.jdbc.batch\_size=20

spring.jpa.properties.hibernate.order\_inserts=true

spring.jpa.properties.hibernate.order\_updates=true

# Enable second-level cache (optional)

spring.jpa.properties.hibernate.cache.use\_second\_level\_cache=true

spring.jpa.properties.hibernate.cache.region.factory\_class=org.hibernate.cache.jcache.JCacheRegionFactory

**SpringAndMavenApplication.java**

**package** com.week3.SpringAndMaven;

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

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

**import** org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication

@EnableJpaAuditing(auditorAwareRef = "auditorAware")

**public** **class** SpringAndMavenApplication {

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

SpringApplication.*run*(SpringAndMavenApplication.**class**, args);

}

}

**EmployeeService.java**

**package** com.week3.SpringAndMaven.service;

**import** com.week3.SpringAndMaven.model.Employee;

**import** jakarta.persistence.EntityManager;

**import** jakarta.persistence.PersistenceContext;

**import** jakarta.transaction.Transactional;

**import** java.util.List;

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

@Service

**public** **class** EmployeeService {

@PersistenceContext

**private** EntityManager entityManager;

@Transactional

**public** **void** saveInBatch(List<Employee> employees) {

**for** (**int** i = 0; i < employees.size(); i++) {

entityManager.persist(employees.get(i));

**if** (i % 20 == 0) {

entityManager.flush();

entityManager.clear();

}

}

}

}

**OUTPUT**

