**Exercise 1: Configuring a Basic Spring Application**

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

**applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bookRepository" class="com.library.repository.BookRepository" />

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void displayBooks() {

System.out.println("List of books...");

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

bookRepository.displayBooks();

}

}

**MainApp.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean("bookService", BookService.class);

service.showBooks();

}

}

**Exercise 2: Implementing Dependency Injection**

<property name="bookRepository" ref="bookRepository"/>

public void setBookRepository(BookRepository bookRepository) { ... }

**Exercise 3: Implementing Logging with Spring AOP**

**pom.xml**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.32</version>

</dependency>

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.\*;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

@Around("execution(\* com.library.service.\*.\*(..))")

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object result = joinPoint.proceed();

long end = System.currentTimeMillis();

System.out.println("Execution Time: " + (end - start) + " ms");

return result;

}

}

**applicationContext.xml**

<context:component-scan base-package="com.library" />

<aop:aspectj-autoproxy />

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="... http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd"

**Exercise 4: Creating and Configuring a Maven Project**

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.10.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

**Exercise 5: Configuring the Spring IoC Container**

**applicationContext.**

<beans xmlns="http://www.springframework.org/schema/beans"

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

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bookRepository" class="com.library.repository.BookRepository" />

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository" />

</bean>

</beans>

**2. Update BookService.java with Setter Injection**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void display() {

System.out.println("Service called:");

bookRepository.showBooks();

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void showBooks() {

System.out.println("Books are being fetched from the repository...");

}

}

**3. Create and Run Main Class**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = context.getBean("bookService", BookService.class);

bookService.display();

}

}

**Exercise 6: Configuring Beans with Annotations**

**applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

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

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="com.library" />

</beans>

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

@Service

public class BookService {

private BookRepository bookRepository;

@Autowired

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void display() {

System.out.println("Annotation-based Service:");

bookRepository.showBooks();

}

}

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void showBooks() {

System.out.println("Annotation-based Repository access...");

}

}

**Exercise 7: Constructor and Setter Injection**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void showBooks() {

System.out.println("Repository: Showing list of books...");

}

}

**2. BookService.java (Constructor + Setter)**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Constructor Injection called.");

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Setter Injection called.");

}

public void displayBooks() {

bookRepository.showBooks();

}

}

**3. applicationContext.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

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

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<bean id="bookService" class="com.library.service.BookService">

<constructor-arg ref="bookRepository"/>

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**4. LibraryManagementApplication.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = context.getBean("bookService", BookService.class);

bookService.displayBooks();

}

}

**Output:**

Constructor Injection called.

Setter Injection called.

Repository: Showing list of books...

**Exercise 8: Basic AOP with Logging**

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.\*;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

@Before("execution(\* com.library.service.\*.\*(..))")

public void logBefore(JoinPoint joinPoint) {

System.out.println("[AOP] Before: " + joinPoint.getSignature().getName());

}

@After("execution(\* com.library.service.\*.\*(..))")

public void logAfter(JoinPoint joinPoint) {

System.out.println("[AOP] After: " + joinPoint.getSignature().getName());

}

}

**2. Add Spring AOP Dependency in pom.xml**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.32</version>

</dependency>

**3. Update applicationContext.xml to enable AOP**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

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

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<context:component-scan base-package="com.library" />

<aop:aspectj-autoproxy/>

</beans>

**OUTPUT:**

Constructor Injection called.

Setter Injection called.

[AOP] Before: displayBooks

Repository: Showing list of books...

[AOP] After: displayBooks

**Exercise 9: Creating a Spring Boot Application**

pom.xml

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

Step 3: application.properties

spring.datasource.url=jdbc:h2:mem:librarydb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

Step 4: Define Entity and Repository

Book.java (Entity)

package com.library.model;

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getTitle() { return title; }

public void setTitle(String title) { this.title = title; }

public String getAuthor() { return author; }

public void setAuthor(String author) { this.author = author; }

}

**BookRepository.java**

package com.library.repository;

import com.library.model.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**Step 5: Create a REST Controller**

BookController.java

package com.library.controller;

import com.library.model.Book;

import com.library.repository.BookRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return bookRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {

Book book = bookRepository.findById(id).orElse(null);

if (book != null) {

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

return bookRepository.save(book);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

Step 6: Run the Application

LibraryManagementApplication.java

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

**Country.java**

package com.ormlearn.model;

import jakarta.persistence.\*;

@Entity

public class Country {

@Id

private String code;

private String name;

public String getCode() { return code; }

public void setCode(String code) { this.code = code; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

}

**CountryRepository.java**

package com.ormlearn.repository;

import com.ormlearn.model.Country;

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

import java.util.List;

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String name);

List<Country> findByNameContainingIgnoreCaseOrderByNameAsc(String name);

List<Country> findByNameStartingWithIgnoreCase(String prefix);

}

**OrmLearnApplication.java**

package com.ormlearn;

import com.ormlearn.model.Country;

import com.ormlearn.repository.CountryRepository;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private CountryRepository countryRepository;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

System.out.println("Countries containing 'ou':");

List<Country> results1 = countryRepository.findByNameContainingIgnoreCase("ou");

results1.forEach(c -> System.out.println(c.getCode() + " " + c.getName()));

System.out.println("\nCountries containing 'ou' sorted:");

List<Country> results2 = countryRepository.findByNameContainingIgnoreCaseOrderByNameAsc("ou");

results2.forEach(c -> System.out.println(c.getCode() + " " + c.getName()));

System.out.println("\nCountries starting with 'Z':");

List<Country> results3 = countryRepository.findByNameStartingWithIgnoreCase("Z");

results3.forEach(c -> System.out.println(c.getCode() + " " + c.getName()));

}

}

**Output:**

Countries containing 'ou':

BV Bouvet Island

DJ Djibouti

GP Guadeloupe

Countries containing 'ou' sorted:

BV Bouvet Island

DJ Djibouti

TF French Southern Territories

Countries starting with 'Z':

ZM Zambia

ZW Zimbabwe

**Stock.java**

package com.ormlearn.model;

import jakarta.persistence.\*;

import java.math.BigDecimal;

import java.util.Date;

@Entity

public class Stock {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int stId;

private String stCode;

@Temporal(TemporalType.DATE)

private Date stDate;

private BigDecimal stOpen;

private BigDecimal stClose;

private Long stVolume;

public int getStId() { return stId; }

public void setStId(int stId) { this.stId = stId; }

public String getStCode() { return stCode; }

public void setStCode(String stCode) { this.stCode = stCode; }

public Date getStDate() { return stDate; }

public void setStDate(Date stDate) { this.stDate = stDate; }

public BigDecimal getStOpen() { return stOpen; }

public void setStOpen(BigDecimal stOpen) { this.stOpen = stOpen; }

public BigDecimal getStClose() { return stClose; }

public void setStClose(BigDecimal stClose) { this.stClose = stClose; }

public Long getStVolume() { return stVolume; }

public void setStVolume(Long stVolume) { this.stVolume = stVolume; }

}

**Step 2: Create StockRepository with Query Methods**

**StockRepository.java**

package com.ormlearn.repository;

import com.ormlearn.model.Stock;

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

import java.math.BigDecimal;

import java.util.Date;

import java.util.List;

public interface StockRepository extends JpaRepository<Stock, Integer> {

List<Stock> findByStCodeAndStDateBetween(String stCode, Date start, Date end);

List<Stock> findByStCodeAndStCloseGreaterThan(String stCode, BigDecimal price);

List<Stock> findTop3ByOrderByStVolumeDesc();

List<Stock> findTop3ByStCodeOrderByStCloseAsc(String stCode);

}

**Step 3: Test the Queries in OrmLearnApplication**

**OrmLearnApplication.java**

package com.ormlearn;

import com.ormlearn.model.Stock;

import com.ormlearn.repository.StockRepository;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.math.BigDecimal;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private StockRepository stockRepository;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

Date start = sdf.parse("2019-09-01");

Date end = sdf.parse("2019-09-30");

List<Stock> fbStocks = stockRepository.findByStCodeAndStDateBetween("FB", start, end);

System.out.println("\nFacebook stocks - Sep 2019:");

fbStocks.forEach(this::printStock);

List<Stock> googStocks = stockRepository.findByStCodeAndStCloseGreaterThan("GOOGL", new BigDecimal("1250"));

System.out.println("\nGoogle stocks with close > 1250:");

googStocks.forEach(this::printStock);

List<Stock> topVolume = stockRepository.findTop3ByOrderByStVolumeDesc();

System.out.println("\nTop 3 highest volume days:");

topVolume.forEach(this::printStock);

List<Stock> lowestNetflix = stockRepository.findTop3ByStCodeOrderByStCloseAsc("NFLX");

System.out.println("\nNetflix lowest 3 closing prices:");

lowestNetflix.forEach(this::printStock);

}

private void printStock(Stock s) {

System.out.println(s.getStCode() + " | " + s.getStDate() + " | Open: " + s.getStOpen() +

" | Close: " + s.getStClose() + " | Volume: " + s.getStVolume());

}

}

**Hands-On: Implementing Queries**

**Step 1: Sample Entity - Country.java**

@Entity

public class Country {

@Id

private String code;

private String name;

}

**Step 2: Create CountryRepository with HQL/JPQL**

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

@Query("SELECT c FROM Country c")

List<Country> findAllCountries();

@Query("SELECT c FROM Country c WHERE c.name LIKE %:keyword%")

List<Country> searchByKeyword(@Param("keyword") String keyword);

@Query("SELECT COUNT(c) FROM Country c")

long countAllCountries();

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

List<Country> findNativeByStartingLetter(@Param("prefix") String prefix);

}

**Step 3: Run and Test in OrmLearnApplication**

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private CountryRepository countryRepository;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) {

System.out.println("JPQL: All countries");

countryRepository.findAllCountries().forEach(c -> System.out.println(c.getCode() + ": " + c.getName()))

System.out.println("\nJPQL: Search by keyword 'land'");

countryRepository.searchByKeyword("land").forEach(c -> System.out.println(c.getName()));

System.out.println("\nJPQL: Count of countries: " + countryRepository.countAllCountries());

System.out.println("\nNative SQL: Starting with 'Z'");

countryRepository.findNativeByStartingLetter("Z").forEach(c -> System.out.println(c.getName()));

}

}

**Hands-On: Criteria Query Example**

**CountryCriteriaService.java**

@Service

public class CountryCriteriaService {

@PersistenceContext

private EntityManager entityManager;

public List<Country> getCountriesByPrefix(String prefix) {

CriteriaBuilder cb = entityManager.getCriteriaBuilder();

CriteriaQuery<Country> cq = cb.createQuery(Country.class);

Root<Country> countryRoot = cq.from(Country.class);

cq.select(countryRoot)

.where(cb.like(countryRoot.get("name"), prefix + "%"));

TypedQuery<Country> query = entityManager.createQuery(cq);

return query.getResultList();

}

}

**OrmLearnApplication.java (Add)**

@Autowired

private CountryCriteriaService criteriaService;

System.out.println("\nCriteria: Countries starting with 'A'");

criteriaService.getCountriesByPrefix("A").forEach(c -> System.out.println(c.getName()));

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

**application.properties**

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

spring.h2.console.enabled=true

spring.jpa.hibernate.ddl-auto=update

**Exercise 2: Creating Entities**

**Department.java**

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private tring name;

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

private List<Employee> employees = new ArrayList<>();

}

**Employee.java**

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

**Exercise 3: Creating Repositories**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByDepartmentId(Long departmentId);

}

**DepartmentRepository.java**

public interface DepartmentRepository extends JpaRepository<Department, Long> {

Department findByName(String name);

}

**Exercise 4: Implementing CRUD with Controllers**

**EmployeeController.java**

@RestController

@RequestMapping("/employees")

@RequiredArgsConstructor

public class EmployeeController {

private final EmployeeRepository employeeRepo;

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepo.findAll();

}

@GetMapping("/{id}")

public ResponseEntity<Employee> getEmployee(@PathVariable Long id) {

return employeeRepo.findById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepo.save(employee);

}

@PutMapping("/{id}")

public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee updatedEmp) {

return employeeRepo.findById(id)

.map(emp -> {

emp.setName(updatedEmp.getName());

emp.setEmail(updatedEmp.getEmail());

emp.setDepartment(updatedEmp.getDepartment());

return ResponseEntity.ok(employeeRepo.save(emp));

}).orElse(ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {

employeeRepo.deleteById(id);

return ResponseEntity.noContent().build();

}

}

**DepartmentController.java**

@RestController

@RequestMapping("/departments")

@RequiredArgsConstructor

public class DepartmentController {

private final DepartmentRepository deptRepo;

@GetMapping

public List<Department> getAllDepartments() {

return deptRepo.findAll();

}

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return deptRepo.save(department);

}

@GetMapping("/{id}")

public ResponseEntity<Department> getDepartment(@PathVariable Long id) {

return deptRepo.findById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

}

**Exercise 8: Creating Projections**

**1. Interface-Based Projection**

public interface EmployeeNameEmailProjection {

String getName();

String getEmail();

}

EmployeeRepository.java:

List<EmployeeNameEmailProjection> findByDepartmentId(Long deptId);

**2. Class-Based Projection**

public class EmployeeDTO {

private String name;

private String email;

public EmployeeDTO(String name, String email) {

this.name = name;

this.email = email;

}

}

EmployeeRepository.java:

@Query("SELECT new com.example.dto.EmployeeDTO(e.name, e.email) FROM Employee e WHERE e.department.id = :deptId")

List<EmployeeDTO> findEmployeeDTOsByDeptId(@Param("deptId") Long deptId);

**Exercise 9: Customizing Data Source Configuration**

**1. Spring Boot Auto Configuration**

application.properties

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

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

**2. Multiple Data Sources (Optional)**

Define two separate configs like:

**application.properties**

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

spring.datasource.username=root

spring.datasource.password=root

app.secondary.datasource.url=jdbc:mysql://localhost:3306/logs

app.secondary.datasource.username=root

app.secondary.datasource.password=root

**PrimaryDataSourceConfig.java**

@Configuration

@EnableJpaRepositories(basePackages = "com.example.repo.primary",

entityManagerFactoryRef = "primaryEntityManagerFactory",

transactionManagerRef = "primaryTransactionManager")

public class PrimaryDataSourceConfig {

}

**Exercise 10: Hibernate-Specific Features**

**1. Hibernate Annotations**

@DynamicUpdate

@SelectBeforeUpdate

@BatchSize(size = 10)

@Entity

public class Employee {

...

}

**2. Custom Dialect and Properties**

In application.properties:

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.properties.hibernate.format\_sql=true

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

spring.jpa.properties.hibernate.jdbc.batch\_size=30

**3. Batch Processing Example**

@Autowired

private EntityManager em;

@Transactional

public void saveEmployeesInBatch(List<Employee> employees) {

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

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

if (i % 30 == 0) {

em.flush();

em.clear();

}

}

}

### application.properties

Create or update the src/main/resources/application.properties file:

properties

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

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

## ****Exercise 2: Creating Entities****

package com.example.employeemanagementsystem.entity;

import jakarta.persistence.\*;

import lombok.\*;

import java.util.List;

@Entity@Table(name = "departments")@Data@NoArgsConstructor@AllArgsConstructorpublic class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees;

}

### **2. Create the Employee Entity**

package com.example.employeemanagementsystem.entity;

import jakarta.persistence.\*;

import lombok.\*;

@Entity@Table(name = "employees")@Data@NoArgsConstructor@AllArgsConstructorpublic class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

## ****Exercise 3: Creating Repositories****

#### **EmployeeRepository.java**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByNameContaining(String keyword);

List<Employee> findByDepartmentId(Long departmentId);

}

#### **DepartmentRepository.java**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Department;import org.springframework.data.jpa.repository.JpaRepository;

public interface DepartmentRepository extends JpaRepository<Department, Long> {

Department findByName(String name);

}

## ****Exercise 4: Implementing CRUD Operations****

### **EmployeeController.java**

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.entity.Employee;import com.example.employeemanagementsystem.repository.EmployeeRepository;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.web.bind.annotation.\*;

import java.util.List;import java.util.Optional;

@RestController@RequestMapping("/employees")public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@GetMapping("/{id}")

public Optional<Employee> getEmployeeById(@PathVariable Long id) {

return employeeRepository.findById(id);

}

@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());

employee.setDepartment(updatedEmployee.getDepartment());

return employeeRepository.save(employee);

}).orElseThrow(() -> new RuntimeException("Employee not found"));

}

@DeleteMapping("/{id}")

public void deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

### **DepartmentController.java**

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.entity.Department;import com.example.employeemanagementsystem.repository.DepartmentRepository;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.web.bind.annotation.\*;

import java.util.List;import java.util.Optional;

@RestController@RequestMapping("/departments")public class DepartmentController {

@Autowired

private DepartmentRepository departmentRepository;

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

@GetMapping("/{id}")

public Optional<Department> getDepartmentById(@PathVariable Long id) {

return departmentRepository.findById(id);

}

@PutMapping("/{id}")

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

return departmentRepository.findById(id).map(dept -> {

dept.setName(updatedDept.getName());

return departmentRepository.save(dept);

}).orElseThrow(() -> new RuntimeException("Department not found"));

}

@DeleteMapping("/{id}")

public void deleteDepartment(@PathVariable Long id) {

departmentRepository.deleteById(id);

}

}

## **Exercise 5: Defining Query Methods**

EmployeeRepository.java

List<Employee> findByNameContainingIgnoreCase(String name);

List<Employee> findByDepartment\_Name(String deptName); // Nested property

List<Employee> findByEmailEndingWith(String domain);

### **Custom Queries Using** @Query

@Query("SELECT e FROM Employee e WHERE e.name LIKE %:name%")

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

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

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

### **Named Queries**

Update Employee.java:

@Entity@NamedQueries({

@NamedQuery(

name = "Employee.findByDepartmentId",

query = "SELECT e FROM Employee e WHERE e.department.id = :deptId"

),

@NamedQuery(

name = "Employee.findByEmailDomain",

query = "SELECT e FROM Employee e WHERE e.email LIKE %:domain"

)

})public class Employee {

}

@Query(name = "Employee.findByDepartmentId")

List<Employee> findByDepartmentIdNamed(@Param("deptId") Long deptId);

@Query(name = "Employee.findByEmailDomain")

List<Employee> findByEmailDomain(@Param("domain") String domain);

## ****Exercise 6: Implementing Pagination and Sorting****

**EmployeeRepository.java**

### ****Pagination****

Page<Employee> findAll(Pageable pageable);

### **Sorting**

Sorting is built into Pageable, no need for separate methods.  
EmployeeController.java

@GetMapping("/paginated")public Page<Employee> getEmployeesPaginated(

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

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

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

@RequestParam(defaultValue = "asc") String direction

) {

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

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

return employeeRepository.findAll(pageable);

}

## ****Exercise 7: Enabling Entity Auditing****

### ****Step 1: Enable JPA Auditing in Your Application****

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

@SpringBootApplication@EnableJpaAuditingpublic class EmployeeManagementSystemApplication {

public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

### ****Step 2: Create an AuditorAware Implementation****

AuditorAwareImpl.java

package com.example.employeemanagementsystem.config;

import org.springframework.data.domain.AuditorAware;import org.springframework.stereotype.Component;

import java.util.Optional;

@Componentpublic class AuditorAwareImpl implements AuditorAware<String> {

@Override

public Optional<String> getCurrentAuditor() {

return Optional.of("System"); // Placeholder for this example

}

}

### 3. **Step 3: Add Auditing Fields to Entities**

Employee.java

@CreatedDate@Column(updatable = false)private LocalDateTime createdDate;

@LastModifiedDateprivate LocalDateTime modifiedDate;

@CreatedBy@Column(updatable = false)private String createdBy;

@LastModifiedByprivate String modifiedBy;

@Entity@EntityListeners(AuditingEntityListener.class)public class Employee {

}

### **4.Test Auditing**

Try POST and PUT requests to /employees and /departments and check if audit fields are populated automatically in the H2 console.

## **Exercise 8: Creating Projections**

### **1. Interface-Based Projection**

**EmployeeNameEmailProjection.java**

package com.example.employeemanagementsystem.projection;

public interface EmployeeNameEmailProjection {

String getName();

String getEmail();

}

**In EmployeeRepository:**

List<EmployeeNameEmailProjection> findByDepartmentId(Long departmentId);

### **2. Class-Based Projection with @Value Constructor Expression**

EmployeeSummaryDTO.java

package com.example.employeemanagementsystem.dto;

public class EmployeeSummaryDTO {

private String name;

private String departmentName;

public EmployeeSummaryDTO(String name, String departmentName) {

this.name = name;

this.departmentName = departmentName;

}

}

**EmployeeRepository:**

@Query("SELECT new com.example.employeemanagementsystem.dto.EmployeeSummaryDTO(e.name, e.department.name) FROM Employee e")

List<EmployeeSummaryDTO> fetchEmployeeSummaries();

**Example Controller Endpoint**

@GetMapping("/summaries")public List<EmployeeSummaryDTO> getEmployeeSummaries() {

return employeeRepository.fetchEmployeeSummaries();

}

## **Exercise 9: Customizing Data Source Configuration**

### **1. Spring Boot Auto-Configuration for Primary Data Source**

### application.properties:

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

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

### Externalize and Add a **Secondary Data Source**

### application.properties (continued):

app.datasource.secondary.url=jdbc:h2:mem:secondarydb

app.datasource.secondary.driver-class-name=org.h2.Driver

app.datasource.secondary.username=sa

app.datasource.secondary.password=password

### Configuration Class for Secondary Data Source

### config/SecondaryDataSourceConfig.java

package com.example.employeemanagementsystem.config;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.boot.autoconfigure.jdbc.DataSourceProperties;

import org.springframework.boot.context.properties.ConfigurationProperties;

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

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;import org.springframework.jdbc.datasource.DataSourceTransactionManager;

import javax.sql.DataSource;

@Configuration@EnableJpaRepositories(

basePackages = "com.example.employeemanagementsystem.secondary.repository",

entityManagerFactoryRef = "secondaryEntityManagerFactory",

transactionManagerRef = "secondaryTransactionManager"

)public class SecondaryDataSourceConfig {

@Bean

@ConfigurationProperties("app.datasource.secondary")

public DataSourceProperties secondaryDataSourceProperties() {

return new DataSourceProperties();

}

@Bean

public DataSource secondaryDataSource() {

return secondaryDataSourceProperties().initializeDataSourceBuilder().build();

}

@Bean

public LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory(

EntityManagerFactoryBuilder builder) {

return builder

.dataSource(secondaryDataSource())

.packages("com.example.employeemanagementsystem.secondary.entity")

.persistenceUnit("secondary")

.build();

}

@Bean

public PlatformTransactionManager secondaryTransactionManager(

@Qualifier("secondaryEntityManagerFactory") LocalContainerEntityManagerFactoryBean emf) {

return new DataSourceTransactionManager(secondaryDataSource());

}

}

## **Exercise 10: Hibernate-Specific Features**

### **1. Hibernate-Specific Annotations**

Employee.java example:

@CreationTimestamp@Column(updatable = false)private LocalDateTime createdAt;

@UpdateTimestampprivate LocalDateTime updatedAt;

@DynamicInsert@DynamicUpdate@BatchSize(size = 20)@Entitypublic class Employee {

//...

}

2. Configure Hibernate Properties for Performance

application.properties:

spring.jpa.properties.hibernate.jdbc.batch\_size=30

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

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

spring.jpa.properties.hibernate.generate\_statistics=false

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

spring.jpa.properties.hibernate.cache.use\_query\_cache=false

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

### Batch Processing with Hibernate

### EmployeeService.java (bulk insert example):

@Transactionalpublic void saveAllEmployees(List<Employee> employees) {

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

employeeRepository.save(employees.get(i));

if (i % 30 == 0) { // Flush and clear every 30

employeeRepository.flush(); }

}

}