WEEK-3

Spring Core and Maven

Exercise 1: Configuring a Basic Spring Application

applicationContext.xml

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

<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 saveBook(String bookName) {

System.out.println("BookRepository: Saving book - " + bookName);

}

}

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 addBook(String bookName) {

System.out.println("BookService: Adding book - " + bookName);

bookRepository.saveBook(bookName);

}

}

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");

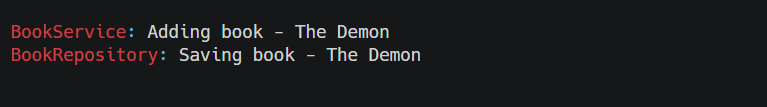
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Demon");

((ClassPathXmlApplicationContext) context).close();

}

}



Exercise 5: Configuring the Spring IoC Container

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 addBook(String bookName) {

System.out.println("BookService: Adding book - " + bookName);

bookRepository.saveBook(bookName);

}

}

BookRepository.java

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("BookRepository: Saving book - " + bookName);

}

}

MainApp.java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

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

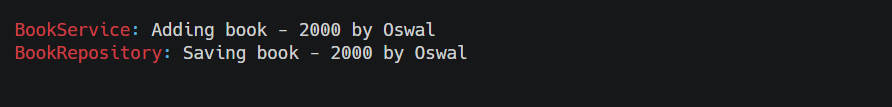
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("2000 by Oswal");

((ClassPathXmlApplicationContext) context).close();

}

}



Exercise 2: Implementing Dependency Injection

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 addBook(String bookName) {

System.out.println("BookService: Adding book - " + bookName);

bookRepository.saveBook(bookName);

}

}

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");

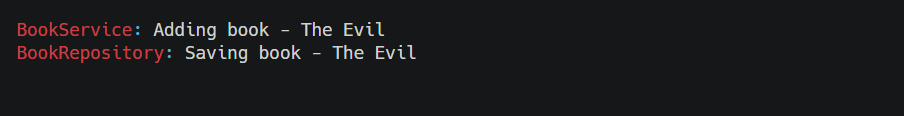
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Evil");

((ClassPathXmlApplicationContext) context).close();

}

}



Exercise 7: Implementing Constructor and Setter Injection

BookService.java

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String serviceName;

public BookService(String serviceName) {

this.serviceName = serviceName;

System.out.println("Constructor Injection: serviceName = " + serviceName);

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Setter Injection: bookRepository injected");

}

public void addBook(String bookName) {

System.out.println(serviceName + ": Adding book - " + bookName);

bookRepository.saveBook(bookName);

}

}

BookRepository.java

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("BookRepository: Saving book - " + bookName);

}

}

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");

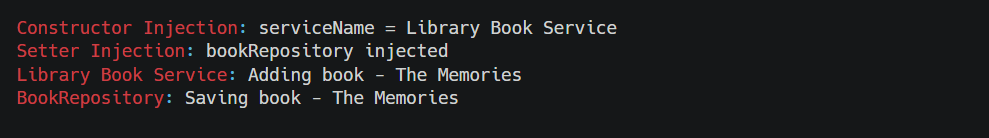
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Memories ");

((ClassPathXmlApplicationContext) context).close();

}

}



Exercise 4: Creating and Configuring a Maven Project

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);

}

}

BookRepository.java

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("BookRepository: Saving book - " + bookName);

}

}

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 final BookRepository bookRepository;

@Autowired

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.out.println("BookService: Adding book - " + bookName);

bookRepository.saveBook(bookName);

}

}



Exercise 9: Creating a Spring Boot Application

Book.java

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 Book() {}

public Book(String title, String author) {

this.title = title;

this.author = 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> {

}

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;

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

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

}

@PutMapping("/{id}")

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

return bookRepository.findById(id).map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

return bookRepository.save(book);

}).orElse(null);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**Hands on 1**

**Spring Data JPA - Quick Example**

@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");

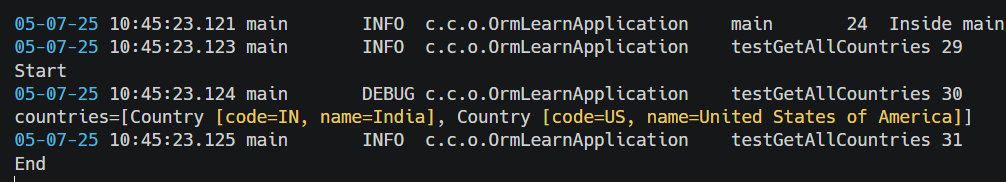
List<Country> countries = countryService.getAllCountries();

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

LOGGER.info("End");

}

}



Implement services for managing Country

Country.java

package com.example.countryapp.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

public Country() {}

public Country(String code, String name) {

this.code = code;

this.name = 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;

}

@Override

public String toString() {

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

}

}

CountryRepository.java

package com.example.countryapp.repository;

import com.example.countryapp.model.Country;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String name);

}

CountryService.java

package com.example.countryapp.service;

import com.example.countryapp.model.Country;

import com.example.countryapp.repository.CountryRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

public Country findCountryByCode(String code) {

return countryRepository.findById(code)

.orElseThrow(() -> new RuntimeException("Country not found: " + code));

}

public Country addCountry(Country country) {

return countryRepository.save(country);

}

public Country updateCountry(String code, String newName) {

Country existing = findCountryByCode(code);

existing.setName(newName);

return countryRepository.save(existing);

}

public void deleteCountry(String code) {

if (!countryRepository.existsById(code)) {

throw new RuntimeException("Country not found: " + code);

}

countryRepository.deleteById(code);

}

public List<Country> findCountriesByPartialName(String partialName) {

return countryRepository.findByNameContainingIgnoreCase(partialName);

}

}

CountryApplication.java

package com.example.countryapp;

import com.example.countryapp.model.Country;

import com.example.countryapp.service.CountryService;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class CountryAppApplication implements CommandLineRunner {

@Autowired

private CountryService countryService;

public static void main(String[] args) {

SpringApplication.run(CountryAppApplication.class, args);

}

@Override

public void run(String... args) {

countryService.addCountry(new Country("IN", "India"));

System.out.println("Find by code: " + countryService.findCountryByCode("IN"));

countryService.updateCountry("IN", "Bharat");

System.out.println("Search: " + countryService.findCountriesByPartialName("rat"));

countryService.deleteCountry("IN");

}

}

**Difference between JPA, Hibernate and Spring Data JPA**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature / Aspect** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| **Type** | Specification (JSR 338) | Implementation of JPA (also a standalone ORM) | Framework/Abstraction built on top of JPA and Hibernate |
| **Provides implementation?** | No | Yes | Uses Hibernate (or other JPA providers) |
| **ORM Tool** | No | Yes | Yes (through Hibernate) |
| **Boilerplate Code Reduction** | No | Minimal | Significant (interfaces like JpaRepository) |
| **Standard API?** | Yes | Implements JPA API + custom APIs | Uses JPA + additional Spring abstractions |
| **Part of Java EE / Jakarta EE?** | Yes | No | No |
| **Query Language Support** | JPQL | JPQL + HQL (Hibernate Query Language) | JPQL + Derived Query Methods + @Query with native SQL |
| **Ease of Use** | Moderate – requires EntityManager handling | Slightly better – uses SessionFactory and advanced features | High – simple CRUD with repository interfaces |
| **Auto Schema Generation** | Depends on provider | Yes (e.g., hibernate.hbm2ddl.auto) | Yes (through Hibernate configs in application.properties) |
| **Caching Support** | Depends on provider | First- and second-level caching | Yes (inherited from Hibernate) |
| **Pagination and Sorting** | Manual implementation | Manual or with HQL | Built-in support in PagingAndSortingRepository |
| **Vendor Neutral** | Yes (works with Hibernate, EclipseLink, etc.) | Tied to Hibernate | Yes (but typically uses Hibernate underneath) |
| **Integration with Spring Boot** | Manual setup | Manual setup | Seamless integration with Spring Boot |
| **Custom Features** | No | Yes (e.g., native queries, batch processing, interceptors) | Yes (query derivation, projections, specifications, etc.) |
| **Documentation** | Standardized via JSR | Extensive official + community support | Extensive Spring + community support |

**Find a country based on country code**

CountryService.java

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.repository.CountryRepository;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import jakarta.transaction.Transactional;

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

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found with code: " + countryCode);

}

return result.get();

}

}

OrmLearnApplication.java

package com.cognizant.springlearn;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OrmLearnApplication {

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

@Autowired

private CountryService countryService;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Autowired

public void testFindCountryByCode(CountryService countryService) {

try {

getAllCountriesTest(countryService);

} catch (CountryNotFoundException e) {

LOGGER.error("Exception: {}", e.getMessage());

}

}

private void getAllCountriesTest(CountryService countryService) throws CountryNotFoundException {

LOGGER.info("Start");

Country country = countryService.findCountryByCode("IN");

LOGGER.debug("Country: {}", country);

LOGGER.info("End");

}

}

CountryService.java

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.repository.CountryRepository;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import jakarta.transaction.Transactional;

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

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found with code: " + countryCode);

}

return result.get();

}

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

}

OrmLearnApplication.java

private void testAddCountry(CountryService countryService) throws CountryNotFoundException {

LOGGER.info("Start");

Country newCountry = new Country();

newCountry.setCode("XY");

newCountry.setName("Xyland");

countryService.addCountry(newCountry);

Country addedCountry = countryService.findCountryByCode("XY");

LOGGER.debug("Added Country: {}", addedCountry);

LOGGER.info("End");

}

Demonstrate implementation of Query Methods feature of Spring Data JPA

package com.cognizant.springlearn.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@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; }

@Override

public String toString() {

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

}

}

Repository.java

package com.cognizant.springlearn.repository;

import com.cognizant.springlearn.model.Country;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

Country findByName(String name);

List<Country> findByNameContaining(String keyword);

List<Country> findByNameStartingWith(String prefix);

List<Country> findByNameEndingWith(String suffix);

}

Demonstrate implementation of O/R Mapping

@Autowired

BookRepository bookRepo;

@Autowired

AuthorRepository authorRepo;

@Autowired

PublisherRepository publisherRepo;

@PostConstruct

public void init() {

Publisher pub = new Publisher();

pub.setName("Pearson");

publisherRepo.save(pub);

Author author1 = new Author();

author1.setName("J.K. Rowling");

Author author2 = new Author();

author2.setName("Tolkien");

authorRepo.saveAll(List.of(author1, author2));

Book book1 = new Book();

book1.setTitle("Harry Potter");

book1.setPublisher(pub);

book1.setAuthors(List.of(author1));

Book book2 = new Book();

book2.setTitle("Lord of the Rings");

book2.setPublisher(pub);

book2.setAuthors(List.of(author2));

bookRepo.saveAll(List.of(book1, book2));

}

CountryRepository.java

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

@Query("SELECT c FROM Country c WHERE c.population > :population ORDER BY c.population DESC")

List<Country> getHighPopulationCountries(@Param("population") Long population);

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

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

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

long countTotalCountries();

@Query("SELECT c FROM Country c JOIN FETCH c.states WHERE c.code = :code")

Country fetchCountryWithStates(@Param("code") String code); // assuming OneToMany with State

@Query(value = "SELECT \* FROM country WHERE co\_name = :name", nativeQuery = true)

Country findByCountryNameNative(@Param("name") String name);

@Query(value = "SELECT \* FROM country ORDER BY population DESC LIMIT 5", nativeQuery = true)

List<Country> findTop5PopulousCountriesNative();

}

CountryService.java

@Autowired

private CountryRepository countryRepo;

public void runQueryExamples() {

System.out.println("High population countries:");

countryRepo.getHighPopulationCountries(50\_000\_000L).forEach(System.out::println);

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

countryRepo.searchCountriesByName("land").forEach(System.out::println);

System.out.println("Total countries: " + countryRepo.countTotalCountries());

System.out.println("Fetch country by name (Native):");

System.out.println(countryRepo.findByCountryNameNative("India"));

System.out.println("Top 5 most populous countries:");

countryRepo.findTop5PopulousCountriesNative().forEach(System.out::println);

}