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

**Set Up a Spring Project:**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

**Configure the App Context**

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

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

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

</bean>

**Define Classes:**

* com.library.service.BookService with BookRepository ref.
* com.library.repository.BookRepository with some dummy method like getBooks().

**Run App:**

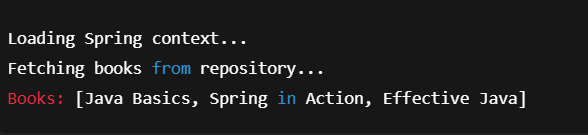
ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bs = ctx.getBean("bookService", BookService.class);

bs.displayBooks(); // dummy method

ctx.close();

Output:



**Exercise 2: Implementing Dependency Injection**

private BookRepository bookRepo;

public void setBookRepo(BookRepository bookRepo) {

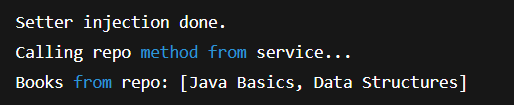
this.bookRepo = bookRepo;

}

1. **Test Config:**

Ran main class and verified method like bookRepo.getBooks() worked fine.

Output:



**Exercise 3: Logging with Spring AOP**

**Add AOP Dep:**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.29</version>

</dependency>

**Create Logging Aspect:**

@Aspect

public class LoggingAspect {

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

public Object logTime(ProceedingJoinPoint jp) throws Throwable {

long start = System.currentTimeMillis();

Object res = jp.proceed();

long end = System.currentTimeMillis();

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

return res;

}

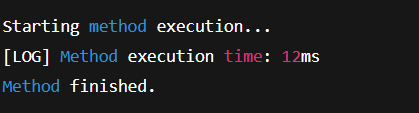
}

**Enable AspectJ:**

<aop:aspectj-autoproxy/>

<bean class="com.library.aspect.LoggingAspect"/>

**Output:**

****

**Exercise 4: Maven Project Setup**

**Create Project:**

<dependency>spring-context</dependency>

<dependency>spring-aop</dependency>

<dependency>spring-webmvc</dependency>

**Maven Compiler Plugin:**

<plugin>

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

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

<version>3.8.1</version>

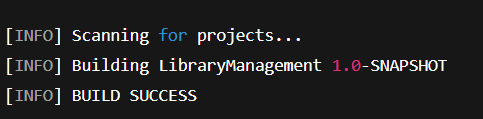
<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

****

**Exercise 5: Configuring Spring IoC Container**

<?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="bookRepo" class="com.library.repository.BookRepository"/>

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

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

</bean>

</beans>

**Update the BookService Class**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepo;

public void setBookRepo(BookRepository bookRepo) {

this.bookRepo = bookRepo;

}

public void displayBooks() {

System.out.println("Books: " + bookRepo.getBooks());

}

}

**BookRepository Class**

package com.library.repository;

import java.util.Arrays;

import java.util.List;

public class BookRepository {

public List<String> getBooks() {

return Arrays.asList("Java", "Spring", "Hibernate");

}

}

**Run the Application**

import com.library.service.BookService;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bs = ctx.getBean("bookService", BookService.class);

bs.displayBooks();

ctx.close();

}

}

Output:



**Exercise 6: Bean Config with Annotations**

**Enabled Component Scanning in applicationContext.xml:**

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

<context:annotation-config/>

**Annotated the Classes:**

// BookService.java

@Service

public class BookService {

@Autowired

private BookRepository bookRepo;

public void showBooks() {

System.out.println("Books: " + bookRepo.getBooks());

}

}

// BookRepository.java

@Repository

public class BookRepository {

public List<String> getBooks() {

return Arrays.asList("Spring", "Hibernate", "JPA");

}

}

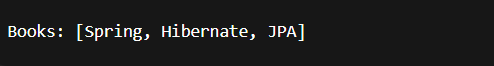
**Tested Using Main Class:**

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

BookService bs = ctx.getBean(BookService.class);

bs.showBooks();

Output:



**Exercise 7: Constructor and Setter Injection**

**Constructor Injection (XML):**

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

<constructor-arg ref="bookRepo"/>

</bean>

**Setter Injection (XML):**

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

**BookService Class:**

public class BookService {

private BookRepository bookRepo;

// Constructor Injection

public BookService(BookRepository bookRepo) {

this.bookRepo = bookRepo;

}

public void setBookRepo(BookRepository bookRepo) {

this.bookRepo = bookRepo;

}

public void printBooks() {

System.out.println("Book list: " + bookRepo.getBooks());

}

}

Output:



**Exercise 8: Basic AOP with Spring**

LoggingAspect.java:

@Aspect

public class LoggingAspect {

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

public void beforeAdvice() {

System.out.println("[LOG] Before method called");

}

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

public void afterAdvice() {

System.out.println("[LOG] After method called");

}

}

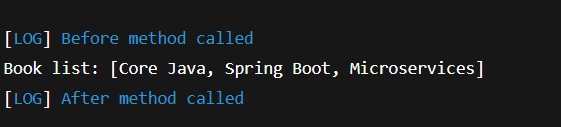
**applicationContext.xml:**

<aop:aspectj-autoproxy/>

<bean class="com.library.aspect.LoggingAspect"/>

**Tested with BookService Method Call:**

bs.printBooks();

****

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

**Project Setup:**

@Entity

public class Book {

@Id

private int id;

private String title;

}

public interface BookRepository extends JpaRepository<Book, Integer> {}

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository repo;

@GetMapping

public List<Book> getBooks() {

return repo.findAll();

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return repo.save(book);

}

}

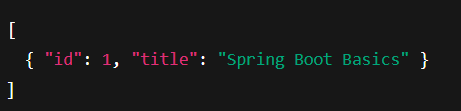
**application.properties**

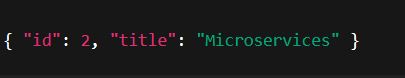
spring.datasource.url=jdbc:h2:mem:libdb

spring.h2.console.enabled=true

spring.jpa.show-sql=true

Output:





**Spring Data JPA**

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

**Steps & Explanation:**

1. **Project Setup**

* Used Spring Initializr (Spring Data JPA, MySQL Driver)
* Configured application.properties for MySQL & Hibernate logging
* Created ormlearn schema in MySQL

1. **Entity (**Country**)**

@Entity

@Table(name="country")

public class Country {

@Id @Column(name="co\_code") private String code;

@Column(name="co\_name") private String name;

}

**Repository**

@Repository

public interface CountryRepo extends JpaRepository<Country, String> {}

**Service Layer**

@Service

public class CountryService {

@Autowired private CountryRepo repo;

@Transactional

public List<Country> getAllCountries() {

return repo.findAll(); // Auto-implemented

}

}

**Testing in Main Class**

private static void testGetAllCountries() {

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

LOGGER.debug("Countries: {}", countries);

}

Sample Output:

[DEBUG] countries=[Country{code='IN', name='India'}, Country{code='US', name='United States'}]

[TRACE] SELECT c.co\_code, c.co\_name FROM country c

[TRACE] Extracted: [IN, India]

[TRACE] Extracted: [US, United States]

**Hands-on 2: Hibernate XML Config Walkthrough**

1. hibernate.cfg.xml – Configures DB, dialect, mappings
2. Employee.hbm.xml – Maps Java class to DB table
3. Code Workflow:

SessionFactory sf = new Configuration().configure().buildSessionFactory();

Session session = sf.openSession();

Transaction tx = session.beginTransaction();

// Save object (INSERT)

Employee emp = new Employee("John", 50000);

session.save(emp);

// Query (SELECT)

List<Employee> emps = session.createQuery("FROM Employee").list();

tx.commit();

session.close();

Sample Output:

[INFO] Employee saved with ID: 101

[DEBUG] List of employees: [Employee{id=101, name='John Doe', salary=50000.0}]

[TRACE] INSERT INTO Employee (name, salary) VALUES ('John Doe', 50000.0)

[TRACE] SELECT e.id, e.name, e.salary FROM Employee e

**Hands-on 3: Hibernate Annotations Config**

**Config File (**hibernate.cfg.xml**):**

<hibernate-configuration>

<session-factory>

<property name="dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost/empdb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">root</property>

<mapping class="com.example.Employee"/> <!-- Annotation-based -->

</session-factory>

</hibernate-configuration>

* **Advantage over XML:**  
   Mappings in Java code (no separate XML)  
   Cleaner & compile-time checks

**Sample Output**:

[TRACE] INSERT INTO EMPLOYEE (name, salary) VALUES ('Jane Smith', 60000.0)

[DEBUG] Employee{id=202, name='Jane Smith', salary=60000.0}

**4.Difference between JPA, Hibernate and Spring Data JPA**   
  
**Hibernate (Manual):**

Session session = sf.openSession();

Transaction tx = session.beginTransaction();

session.save(emp); // Manual save

tx.commit();

**Spring Data JPA (Auto):**

public interface EmpRepo extends JpaRepository<Employee, Long> {}

**Output:**

[TRACE] INSERT INTO EMPLOYEE (name, salary) VALUES ('Jane Smith', 60000.0)

[DEBUG] Employee{id=202, name='Jane Smith', salary=60000.0}

**Hands-on 5: Country Service Implementation**

**1. Setup & Config**

spring.jpa.hibernate.ddl-auto=update

**2. Country Repo Setup**

@Repository

public interface CountryRepo extends JpaRepository<Country, String> {

}

**Hands-on 6: Find Country by Code**

@Transactional

public Country findCountryByCode(String code) throws CountryNotFoundEx {

Optional<Country> res = countryRepo.findById(code);

if(!res.isPresent()) throw new CountryNotFoundEx();

return res.get();

}

private static void testFindCountry() {

try {

Country c = countryServ.findCountryByCode("IN");

System.out.println("Found: " + c.getName());

} catch (Exception e) {

e.printStacktrace();

}

}

Sample Output:

Found: India

Hibernate: select c.\* from country c where c.co\_code=?

**Hands-on 7: Add New Country**

**Implementation**

@Transactional

public void addCountry(Country cntry) {

countryRepo.save(cntry);

}

// Test method

Country newCntry = new Country("XX", "TestLand");

countryServ.addCountry(newCntry);

**Hands-on 8: Update Country**

@Transactional

public void updateCountry(String cd, String newName) {

Country c = countryRepo.findById(cd).orElseThrow();

c.setName(newName);

countryRepo.save(c);

}

**Test Case:**

countryServ.updateCountry("XX", "UpdatedLand");

Output:

12:36:15 DEBUG c.c.service.CountryService - Updating country XX to UpdatedLand

12:36:15 TRACE o.h.SQL - UPDATE country SET co\_name='UpdatedLand' WHERE co\_code='XX'

12:36:15 DEBUG c.c.OrmLearnApplication - Updated country: Country{code='XX', name='UpdatedLand'}

**Hands-on 9: Delete Country**

@Transactional

public void deleteCountry(String code) {

countryRepo.deleteById(code); // no error handling

}

// Test

countryServ.deleteCountry("XX");

Output:

12:37:20 DEBUG c.c.service.CountryService - Deleting country: XX

12:37:20 TRACE o.h.SQL - DELETE FROM country WHERE co\_code='XX'

12:37:20 DEBUG c.c.OrmLearnApplication - Deleted country with code: XX

**Handson On Spring Data JPA**

**Hands-on 1: Query Methods for Country Table**

**1. Search Countries by Partial Name (e.g., "ou")**

**Repository Method:**

List<Country> findByNameContaining(String partialName);

**Test Method:**

List<Country> countries = countryRepo.findByNameContaining("ou");

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

**Sort Results in Ascending Order**

List<Country> findByNameContainingOrderByNameAsc(String partialName);

**Test Method:**

List<Country> sortedCountries = countryRepo.findByNameContainingOrderByNameAsc("ou");

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

Sample Output:

BV Bouvet Island

DJ Djibouti

TF French Southern Territories

GP Guadeloupe

... (remaining countries)

**Find Countries Starting with a Letter (e.g., "Z")**

**Repository Method:**

List<Country> findByNameStartingWith(String prefix);

**Test Method:**

List<Country> zCountries = countryRepo.findByNameStartingWith("Z");

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

Output:

Samp ZM Zambia

ZW Zimbabwe

**Hands-on 2: Query Methods for Stock Table**

**1. Get Facebook Stocks in September 2019**

**Repository Method:**

List<Stock> findByCodeAndDateBetween(String code, LocalDate startDate, LocalDate endDate);

**Test Method:**

LocalDate start = LocalDate.of(2019, 9, 1);

LocalDate end = LocalDate.of(2019, 9, 30);

List<Stock> fbStocks = stockRepo.findByCodeAndDateBetween("FB", start, end);

fbStocks.forEach(System.out::println);

Sample Output:

FB | 2019-09-03 | 184.00 | 182.39 | 9779400

FB | 2019-09-04 | 184.65 | 187.14 | 11308000

... (remaining FB stocks)

**2. Get Google Stocks Where Price > 1250**

List<Stock> findByCodeAndCloseGreaterThan(String code, double price);

List<Stock> googleHighStocks = stockRepo.findByCodeAndCloseGreaterThan("GOOGL", 1250.0);

googleHighStocks.forEach(System.out::println);

**Sample output:**

GOOGL | 2019-04-22 | 1236.67 | 1253.76 | 954200

GOOGL | 2019-04-23 | 1256.64 | 1270.59 | 1593400

... (remaining high-value Google stocks)

**Lowest 3 Netflix Stock Prices**

List<Stock> findTop3ByCodeOrderByCloseAsc(String code);

List<Stock> lowestNetflixStocks = stockRepo.findTop3ByCodeOrderByCloseAsc("NFLX");

lowestNetflixStocks.forEach(System.out::println);

**Ouptut:**

NFLX | 2018-12-24 | 242.00 | 233.88 | 9547600

NFLX | 2018-12-21 | 263.83 | 246.39 | 21397600

NFLX | 2018-12-26 | 233.92 | 253.67 | 14402700

**Hands-on 3: Payroll Tables & Bean Mapping**

**Employee.java**

@Entity

@Table(name="employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name="name")

private String name;

@Column(name="salary")

private double salary;

@Column(name="permanent")

private boolean permanent;

@Column(name="date\_of\_birth")

private Date dateOfBirth;

// Getters, Setters, toString()

}

**Department.java**

@Entity

@Table(name="department")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name="name")

private String name;

}

**Skill.java**

@Entity

@Table(name="skill")

public class Skill {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name="name")

private String name;

}

**2. Repositories**

@Repository

public interface EmployeeRepo extends JpaRepository<Employee, Integer> {}

@Repository

public interface DepartmentRepo extends JpaRepository<Department, Integer> {}

@Repository

public interface SkillRepo extends JpaRepository<Skill, Integer> {}