|  |  |
| --- | --- |
| **WEEK 3 – Spring Core and Maven**  **Spring Data JPA with Spring Boot , Hibernate** | **Superset Id : 6429486**  **Name : Akshaya V** |

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

**Scenario:**

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

Steps:

1. **Set Up a Spring Project:**
   * Create a Maven project named LibraryManagement.
   * Add Spring Core dependencies in the pom.xml file.
2. **Configure the Application Context:**
   * Create an XML configuration file named applicationContext.xml in the src/main/resources directory.
   * Define beans for BookService and BookRepository in the XML file.
3. **Define Service and Repository Classes:**
   * Create a package com.library.service and add a class BookService.
   * Create a package com.library.repository and add a class BookRepository.
4. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Code:**

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <spring.version>6.1.2</spring.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-beans</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.11.0</version>  
 <configuration>  
 <source>17</source>  
 <target>17</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**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>  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>

**BookService.java:**

package com.library.service;  
import com.library.repository.BookRepository;  
public class BookService{  
 private BookRepository bookRepository;  
 public BookService(){  
 System.*out*.println("BookService created");  
 }  
 public void setBookRepository(BookRepository bookRepository){  
 this.bookRepository=bookRepository;  
 System.*out*.println("BookRepository injected into BookService");  
 }  
 public void showAllBooks(){  
 System.*out*.println("BookService: Showing all books");  
 bookRepository.displayBooks();  
 }  
}

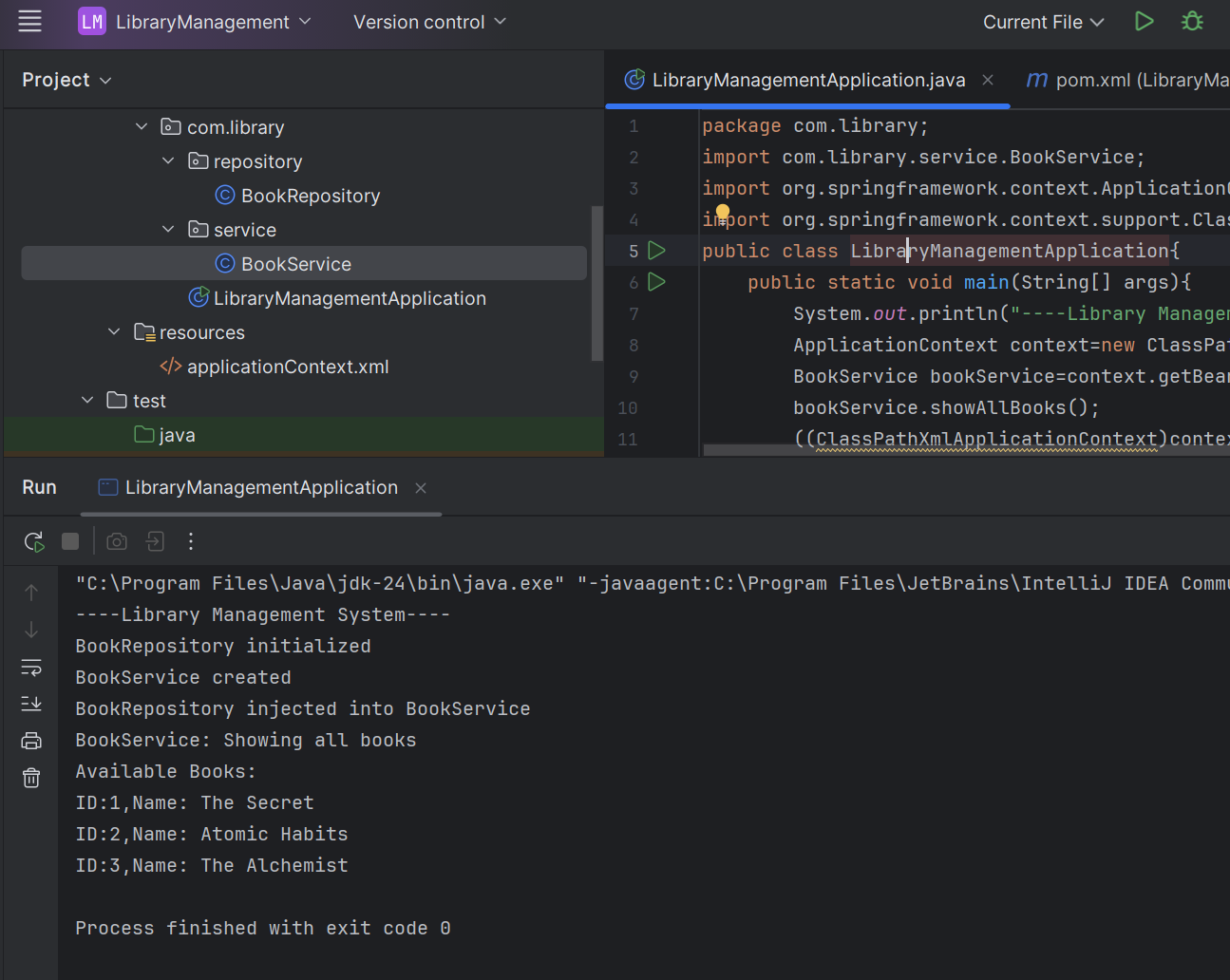
**BookRepository.java:**

package com.library.repository;  
import java.util.HashMap;  
import java.util.Map;  
public class BookRepository{  
 private Map<String,String>books;  
 public BookRepository(){  
 books=new HashMap<>();  
 books.put("1","The Secret");  
 books.put("2","Atomic Habits");  
 books.put("3","The Alchemist");  
 System.*out*.println("BookRepository initialized");  
 }  
 public void displayBooks(){  
 System.*out*.println("Available Books:");  
 books.forEach((id,name)->System.*out*.println("ID:"+id+",Name: "+name));  
 }  
}

**LibraryManagementApplication.java:(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){  
 System.*out*.println("----Library Management System----");  
 ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService=context.getBean("bookService",BookService.class);  
 bookService.showAllBooks();  
 ((ClassPathXmlApplicationContext)context).close();  
 }  
}

**Output:**

****

**Exercise 2:** **Implementing Dependency Injection**

**Scenario:**

In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

**Steps:**

1. **Modify the XML Configuration:**
   * Update applicationContext.xml to wire BookRepository into BookService.
2. **Update the BookService Class:**
   * Ensure that BookService class has a setter method for BookRepository.
3. **Test the Configuration:**
   * Run the LibraryManagementApplication main class to verify the dependency injection.

**Code:**

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <spring.version>6.1.2</spring.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-beans</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.11.0</version>  
 <configuration>  
 <source>17</source>  
 <target>17</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**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>  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>

**BookService.java:**

package com.library.service;  
import com.library.repository.BookRepository;  
import java.util.List;  
public class BookService{  
 private BookRepository bookRepository;  
 public BookService(){  
 System.*out*.println("BookService created");  
 }  
 public void setBookRepository(BookRepository bookRepository){  
 this.bookRepository=bookRepository;  
 System.*out*.println("BookRepository injected into BookService");  
 }  
 public List<String>getAllBooks(){  
 System.*out*.println("BookService: Processing request to get all books");  
 if(bookRepository==null){  
 throw new IllegalStateException("BookRepository not initialized");  
 }  
 return bookRepository.getAllBooks();  
 }  
 public String getBookById(String id){  
 System.*out*.println("BookService: Processing request to get book with ID: "+id);  
 if(bookRepository==null){  
 throw new IllegalStateException("BookRepository not initialized");  
 }  
 if(id==null||id.trim().isEmpty()){  
 throw new IllegalArgumentException("Book ID cannot be null or empty");  
 }  
 return bookRepository.getBookById(id);  
 }  
 public void addBook(String id,String title){  
 System.*out*.println("BookService: Processing request to add book - ID: "+id+", Title: "+title);  
 if(bookRepository==null){  
 throw new IllegalStateException("BookRepository not initialized");  
 }  
 if(id==null||id.trim().isEmpty()){  
 throw new IllegalArgumentException("Book ID cannot be null or empty");  
 }  
 if(title==null||title.trim().isEmpty()){  
 throw new IllegalArgumentException("Book title cannot be null or empty");  
 }  
 bookRepository.addBook(id,title);  
 }  
 public boolean deleteBook(String id){  
 System.*out*.println("BookService: Processing request to delete book with ID: "+id);  
 if(bookRepository==null){  
 throw new IllegalStateException("BookRepository not initialized");  
 }  
 if(id==null||id.trim().isEmpty()){  
 throw new IllegalArgumentException("Book ID cannot be null or empty");  
 }  
 return bookRepository.deleteBook(id);  
 }  
 public String getLibraryStats(){  
 System.*out*.println("BookService: Processing request for library statistics");  
 if(bookRepository==null){  
 throw new IllegalStateException("BookRepository not initialized");  
 }  
 int totalBooks=bookRepository.getTotalBooks();  
 return "Library Statistics: Total Books = "+totalBooks;  
 }  
 public List<String>searchBooks(String searchTerm){  
 System.*out*.println("BookService: Processing search request for: "+searchTerm);  
 if(bookRepository==null){  
 throw new IllegalStateException("BookRepository not initialized");  
 }  
 if(searchTerm==null||searchTerm.trim().isEmpty()){  
 return getAllBooks();  
 }  
 return bookRepository.getAllBooks().stream().filter(book->book.toLowerCase().contains(searchTerm.toLowerCase())).collect(java.util.stream.Collectors.*toList*());  
 }  
}

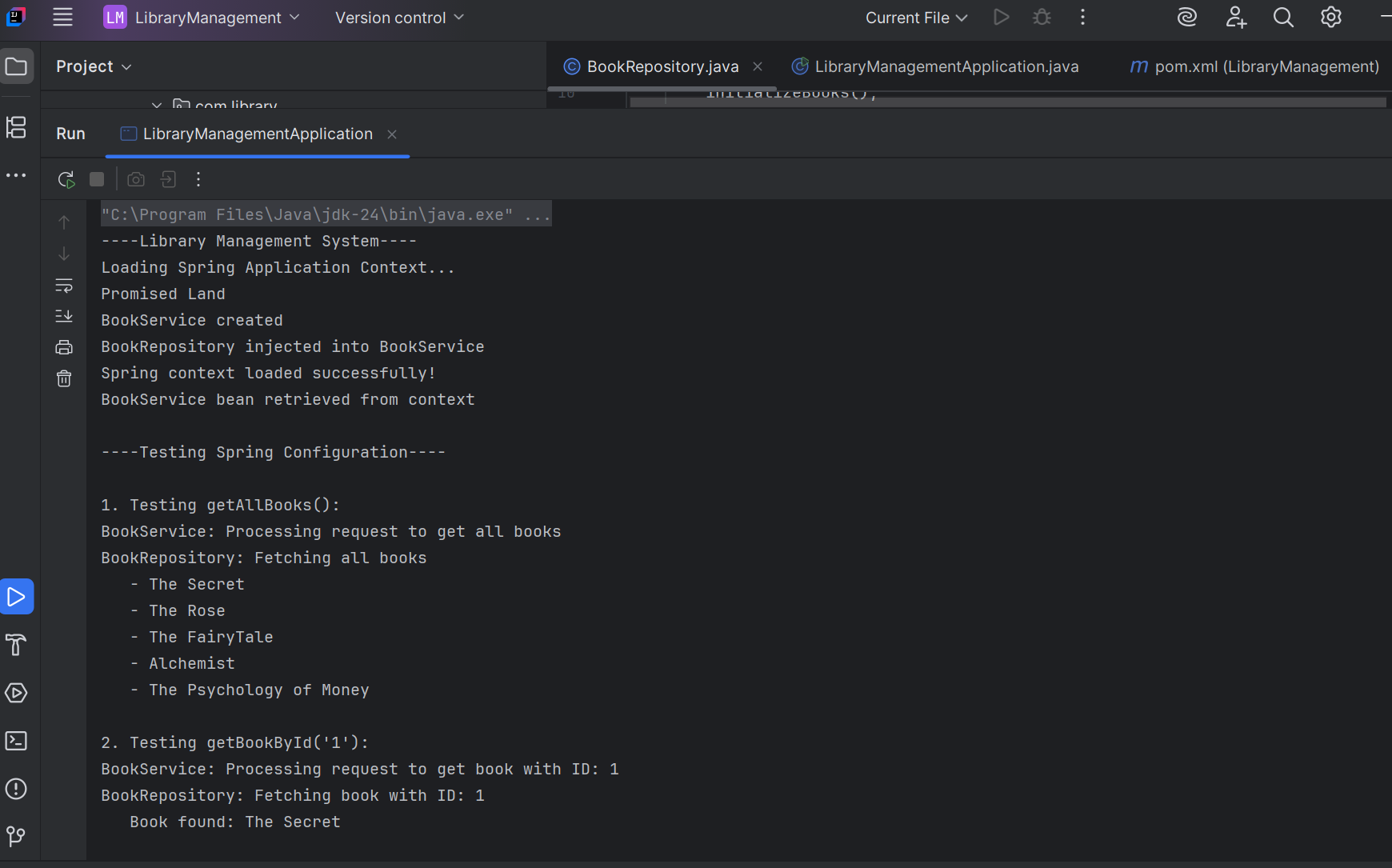
**BookRepository.java:**

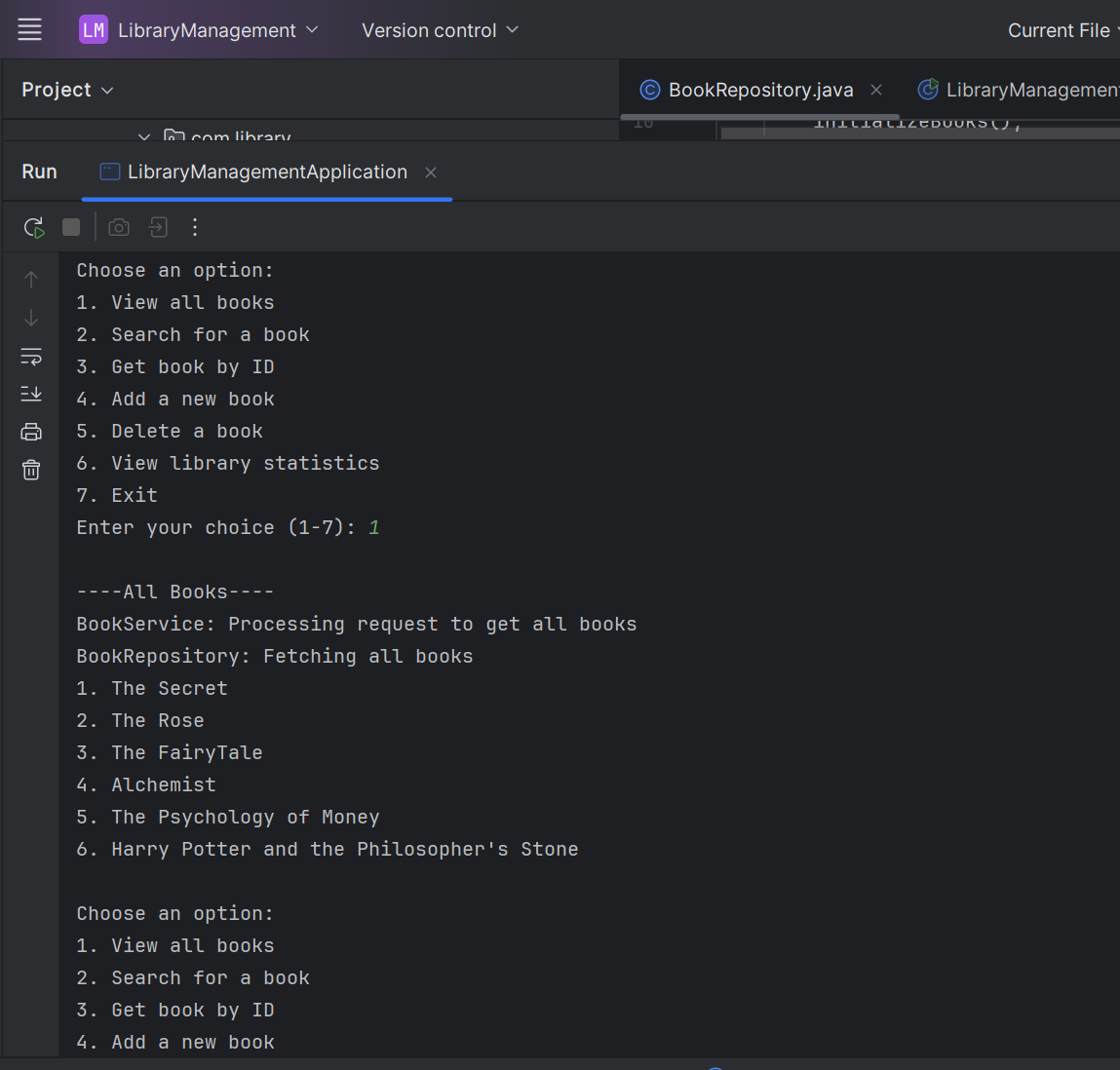
package com.library.repository;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
public class BookRepository{  
 private Map<String,String>books;  
 public BookRepository(){  
 this.books=new HashMap<>();  
 initializeBooks();  
 }  
 private void initializeBooks(){  
 books.put("1","The Secret");  
 books.put("2","The Rose");  
 books.put("3","The FairyTale");  
 books.put("4","Alchemist");  
 books.put("5","The Psychology of Money");  
 System.*out*.println("Promised Land");  
 }  
 public List<String>getAllBooks(){  
 System.*out*.println("BookRepository: Fetching all books");  
 return new ArrayList<>(books.values());  
 }  
 public String getBookById(String id){  
 System.*out*.println("BookRepository: Fetching book with ID: "+id);  
 return books.get(id);  
 }  
 public void addBook(String id,String title){  
 System.*out*.println("BookRepository: Adding book - ID: "+id+", Title: "+title);  
 books.put(id,title);  
 }  
 public boolean deleteBook(String id){  
 System.*out*.println("BookRepository: Deleting book with ID: "+id);  
 return books.remove(id)!=null;  
 }  
 public int getTotalBooks(){  
 return books.size();  
 }  
}

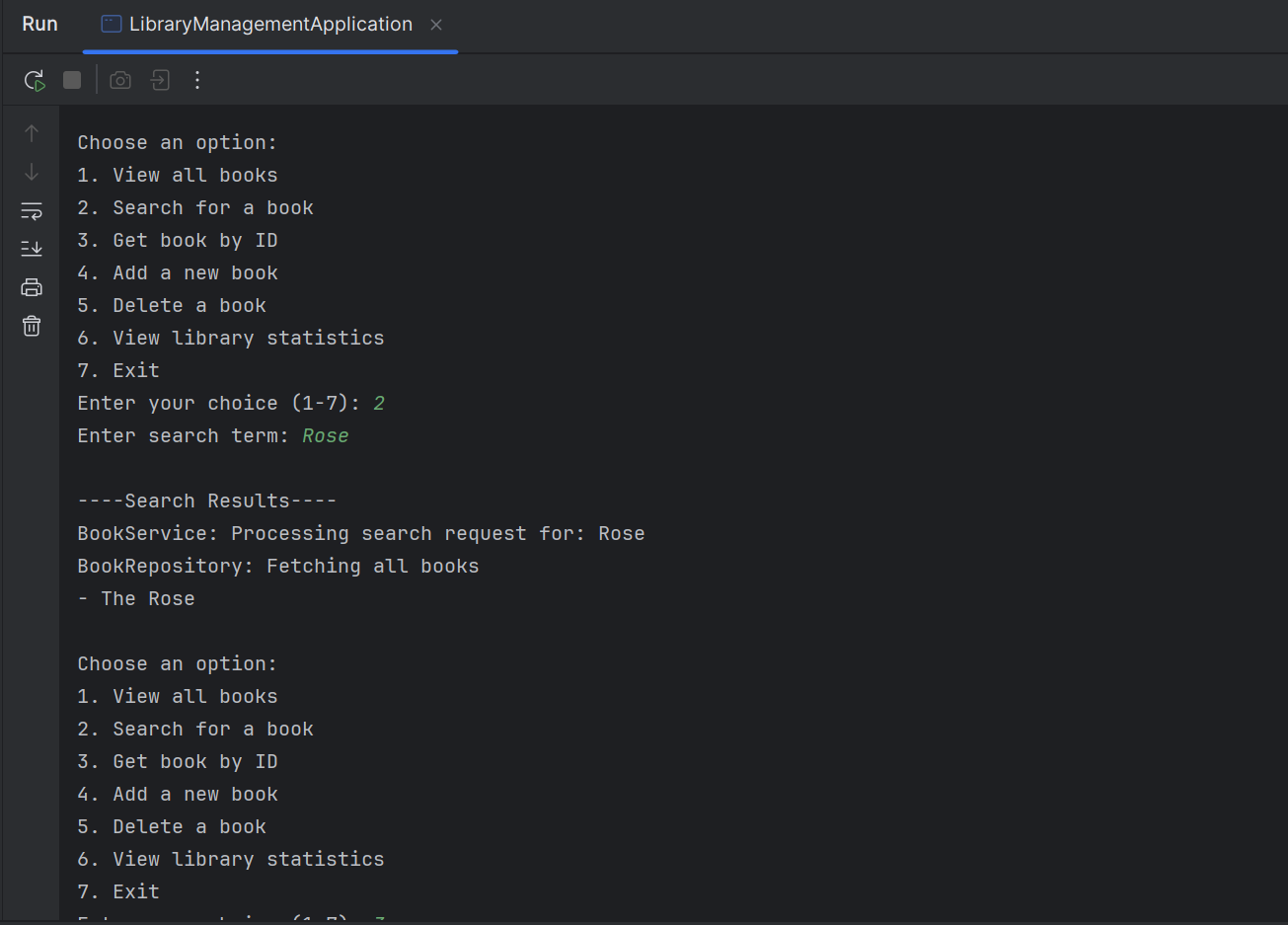
**LibraryManagementApplication.java:(main class)**

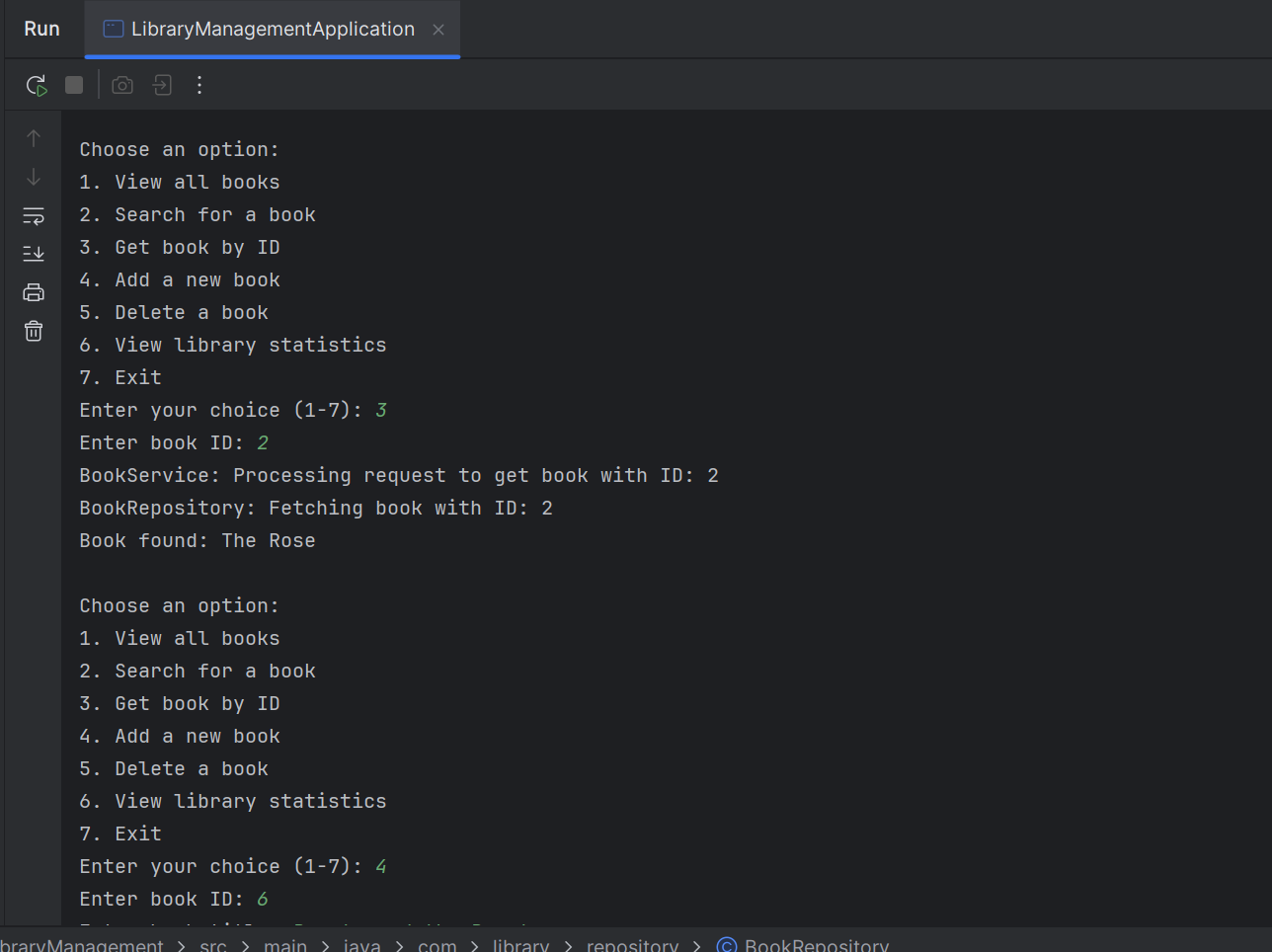
package com.library;  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import java.util.List;  
import java.util.Scanner;  
public class LibraryManagementApplication{  
 private static BookService *bookService*;  
 private static Scanner *scanner*;  
 public static void main(String[] args){  
 System.*out*.println("----Library Management System----");  
 System.*out*.println("Loading Spring Application Context...");  
 ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");  
 *bookService*=context.getBean("bookService",BookService.class);  
 System.*out*.println("Spring context loaded successfully!");  
 System.*out*.println("BookService bean retrieved from context");  
 *scanner*=new Scanner(System.*in*);  
 *testConfiguration*();  
 *showInteractiveMenu*();  
 *scanner*.close();  
 ((ClassPathXmlApplicationContext)context).close();  
 System.*out*.println("Application terminated successfully!");  
 }  
 private static void testConfiguration(){  
 System.*out*.println("\n----Testing Spring Configuration----");  
 try{  
 System.*out*.println("\n1. Testing getAllBooks():");  
 List<String>books=*bookService*.getAllBooks();  
 books.forEach(book->System.*out*.println(" - "+book));  
 System.*out*.println("\n2. Testing getBookById('1'):");  
 String book=*bookService*.getBookById("1");  
 System.*out*.println(" Book found: "+book);  
 System.*out*.println("\n3. Testing addBook():");  
 *bookService*.addBook("6","Harry Potter and the Philosopher's Stone");  
 System.*out*.println(" Book added successfully!");  
 System.*out*.println("\n4. Testing getLibraryStats():");  
 String stats=*bookService*.getLibraryStats();  
 System.*out*.println(" "+stats);  
 System.*out*.println("\n5. Testing searchBooks('Great'):");  
 List<String>searchResults=*bookService*.searchBooks("Great");  
 searchResults.forEach(result->System.*out*.println(" - "+result));  
 System.*out*.println("\n----Configuration Test Completed Successfully!----");  
 }catch(Exception e){  
 System.*err*.println("Error during configuration test: "+e.getMessage());  
 e.printStackTrace();  
 }  
 }  
 private static void showInteractiveMenu(){  
 System.*out*.println("\n----Interactive Library Management----");  
 while(true){  
 System.*out*.println("\nChoose an option:");  
 System.*out*.println("1. View all books");  
 System.*out*.println("2. Search for a book");  
 System.*out*.println("3. Get book by ID");  
 System.*out*.println("4. Add a new book");  
 System.*out*.println("5. Delete a book");  
 System.*out*.println("6. View library statistics");  
 System.*out*.println("7. Exit");  
 System.*out*.print("Enter your choice (1-7): ");  
 try{  
 int choice=Integer.*parseInt*(*scanner*.nextLine());  
 switch(choice){  
 case 1:{  
 *viewAllBooks*();  
 break;  
 }  
 case 2:{  
 *searchBooks*();  
 break;  
 }  
 case 3:{  
 *getBookById*();  
 break;  
 }  
 case 4:{  
 *addNewBook*();  
 break;  
 }  
 case 5:{  
 *deleteBook*();  
 break;  
 }  
 case 6:{  
 *viewLibraryStats*();  
 break;  
 }  
 case 7:{  
 System.*out*.println("Exiting interactive mode...");  
 return;  
 }  
 default:{  
 System.*out*.println("Invalid choice! Please enter a number between 1-7.");  
 break;  
 }  
 }  
 }catch(NumberFormatException e){  
 System.*out*.println("Invalid input! Please enter a valid number.");  
 }catch(Exception e){  
 System.*out*.println("Error: "+e.getMessage());  
 }  
 }  
 }  
 private static void viewAllBooks(){  
 System.*out*.println("\n----All Books----");  
 List<String>books=*bookService*.getAllBooks();  
 if(books.isEmpty()){  
 System.*out*.println("No books found in the library.");  
 }else{  
 for(int i=0;i<books.size();i++){  
 System.*out*.println((i+1)+". "+books.get(i));  
 }  
 }  
 }  
 private static void searchBooks(){  
 System.*out*.print("Enter search term: ");  
 String searchTerm=*scanner*.nextLine();  
 System.*out*.println("\n----Search Results----");  
 List<String>results=*bookService*.searchBooks(searchTerm);  
 if(results.isEmpty()){  
 System.*out*.println("No books found matching: "+searchTerm);  
 }else{  
 results.forEach(book->System.*out*.println("- "+book));  
 }  
 }  
 private static void getBookById(){  
 System.*out*.print("Enter book ID: ");  
 String id=*scanner*.nextLine();  
 String book=*bookService*.getBookById(id);  
 if(book!=null){  
 System.*out*.println("Book found: "+book);  
 }else{  
 System.*out*.println("No book found with ID: "+id);  
 }  
 }  
 private static void addNewBook(){  
 System.*out*.print("Enter book ID: ");  
 String id=*scanner*.nextLine();  
 System.*out*.print("Enter book title: ");  
 String title=*scanner*.nextLine();  
 *bookService*.addBook(id,title);  
 System.*out*.println("Book added successfully!");  
 }  
 private static void deleteBook(){  
 System.*out*.print("Enter book ID to delete: ");  
 String id=*scanner*.nextLine();  
 boolean deleted=*bookService*.deleteBook(id);  
 if(deleted){  
 System.*out*.println("Book deleted successfully!");  
 }else{  
 System.*out*.println("No book found with ID: "+id);  
 }  
 }  
 private static void viewLibraryStats(){  
 String stats=*bookService*.getLibraryStats();  
 System.*out*.println("\n----Library Statistics----");  
 System.*out*.println(stats);  
 }  
}

**Output:**

****

****

****

****

****

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

**Scenario:**

You need to set up a new Maven project for the library management application and add Spring dependencies.

**Steps:**

1. **Create a New Maven Project:**
   * Create a new Maven project named LibraryManagement.
2. **Add Spring Dependencies in pom.xml:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Code:**

**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  
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.10.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**BookRepository.java:**

package com.library.repository;  
import java.util.ArrayList;  
import java.util.List;  
public class BookRepository{  
 private final List<String> books=new ArrayList<>();  
 public BookRepository(){  
 books.add("Pride and Prejudice by Jane Austen");  
 books.add("1984 by George Orwell");  
 books.add("The Great Gatsby by F. Scott Fitzgerald");  
 books.add("Harry Potter and the Sorcerer's Stone by J.K. Rowling");  
 }  
 public List<String> findAllBooks(){  
 return books;  
 }  
 public int getTotalBooks(){  
 return books.size();  
 }  
}

**BookService.java:**

package com.library.service;  
import com.library.repository.BookRepository;  
import java.util.List;  
public class BookService{  
 private BookRepository bookRepository;  
 public void setBookRepository(BookRepository bookRepository){  
 this.bookRepository=bookRepository;  
 }  
 public void displayAllBooks(){  
 System.*out*.println("\nFetching all books in the library...\n");  
  
 List<String> books=bookRepository.findAllBooks();  
 if(books.isEmpty()){  
 System.*out*.println("No books found in the library.");  
 }else{  
 System.*out*.println("Found the following books:\n");  
 int count=1;  
 for(String book:books){  
 System.*out*.println(count+". "+book);  
 count++;  
 }  
 }  
 System.*out*.println("\nTotal Books in Library: "+bookRepository.getTotalBooks());  
 System.*out*.println("Status: Library is Open for Borrowing\n");  
 }  
}

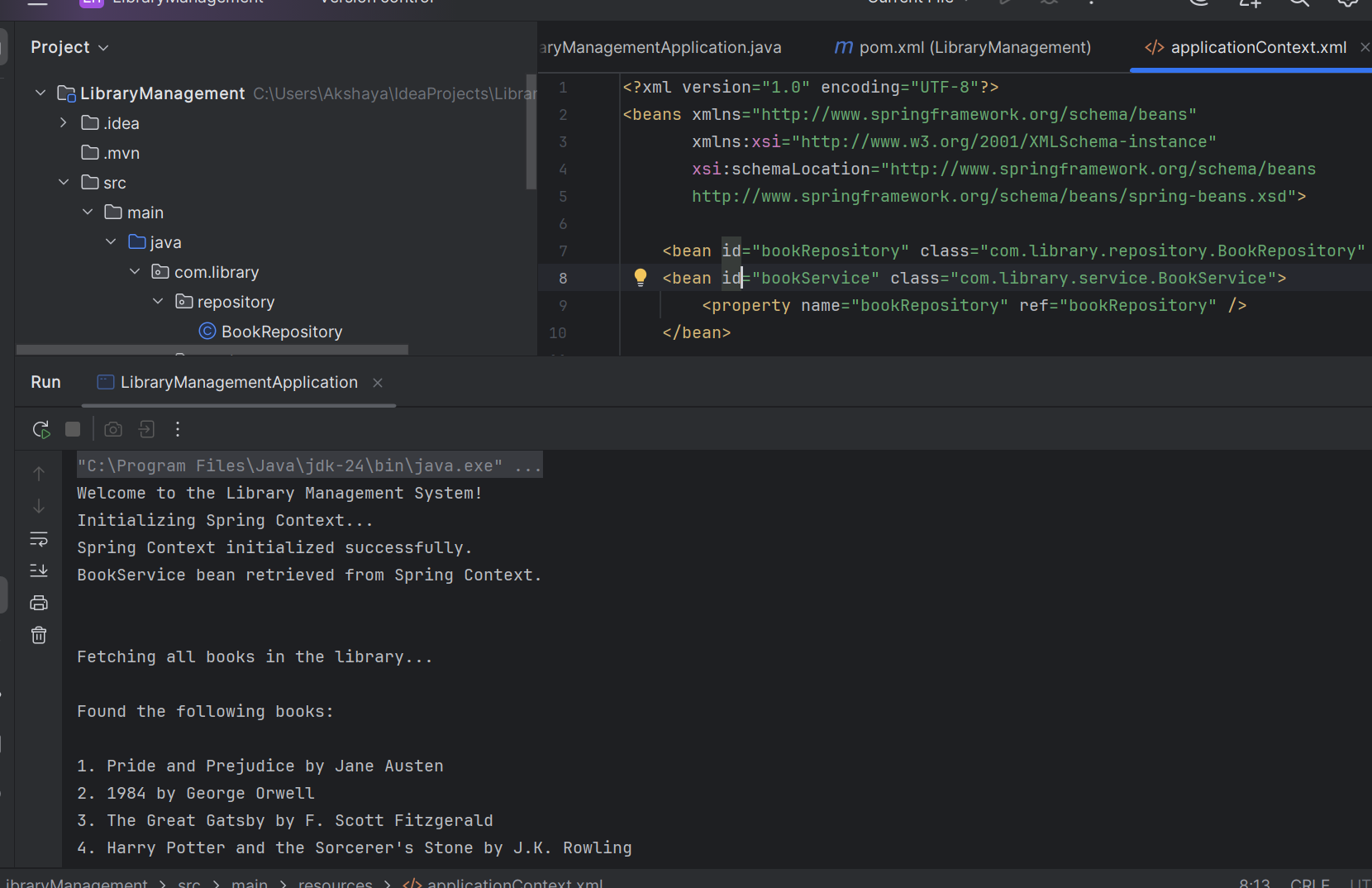
**LibraryManagementApplication.java:**

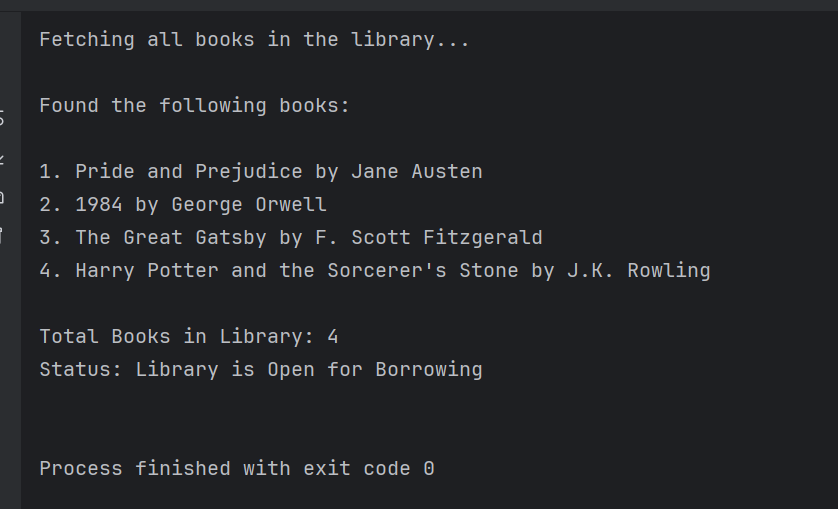
package com.library;  
import com.library.service.BookService;  
import com.library.repository.BookRepository;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
public class LibraryManagementApplication{  
 public static void main(String[] args){  
 System.*out*.println("Welcome to the Library Management System!");  
 System.*out*.println("Initializing Spring Context...");  
 ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService=context.getBean("bookService", BookService.class);  
 System.*out*.println("Spring Context initialized successfully.");  
 System.*out*.println("BookService bean retrieved from Spring Context.\n");  
 bookService.displayAllBooks();  
 }  
}

**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>

**Output:**

****

****

**Spring Data JPA with Spring Boot,Hibernate:**

**Spring Data JPA - Quick Example**

**Code:**

**Country.java:**

package com.cognizant.ormlearn.model;  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
import jakarta.persistence.Column;  
import jakarta.persistence.Table;  
@Entity  
@Table(name="country")  
public class Country{  
 @Id  
 @Column(name="code")  
 private String code;  
 @Column(name="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.cognizant.ormlearn.repository;  
import org.springframework.stereotype.Repository;  
import com.cognizant.ormlearn.model.Country;  
import org.springframework.data.jpa.repository.JpaRepository;  
public interface CountryRepository extends JpaRepository<Country,String>{  
}

**CountryService.java:**

package com.cognizant.ormlearn.service;  
import com.cognizant.ormlearn.model.Country;  
import com.cognizant.ormlearn.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 List<Country> getAllCountries(){  
 return countryRepository.findAll();  
 }  
 public void saveCountry(Country country){  
 countryRepository.save(country);  
 }  
}

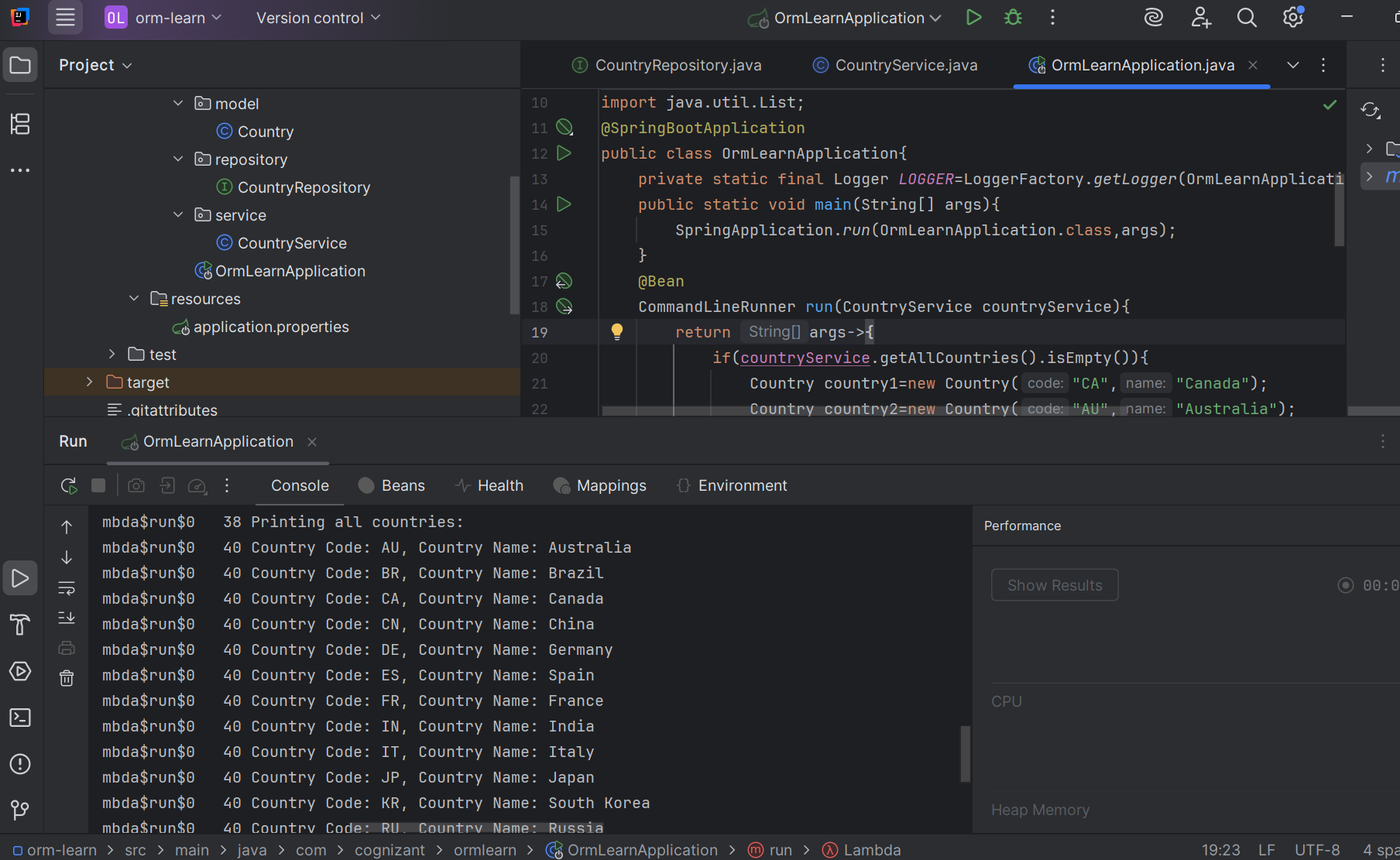
**OrmLearnApplication.java:**

package com.cognizant.ormlearn;  
import com.cognizant.ormlearn.model.Country;  
import com.cognizant.ormlearn.service.CountryService;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.Bean;  
import java.util.List;  
@SpringBootApplication  
public class OrmLearnApplication{  
 private static final Logger *LOGGER*=LoggerFactory.*getLogger*(OrmLearnApplication.class);  
 public static void main(String[] args){  
 SpringApplication.*run*(OrmLearnApplication.class,args);  
 }  
 @Bean  
 CommandLineRunner run(CountryService countryService){  
 return args->{  
 if(countryService.getAllCountries().isEmpty()){  
 Country country1=new Country("CA","Canada");  
 Country country2=new Country("AU","Australia");  
 Country country3=new Country("SG","Singapore");  
 Country country4=new Country("JP","Japan");  
 Country country5=new Country("DE","Germany");  
 Country country6=new Country("FR","France");  
 Country country7=new Country("IT","Italy");  
 countryService.saveCountry(country1);  
 countryService.saveCountry(country2);  
 countryService.saveCountry(country3);  
 countryService.saveCountry(country4);  
 countryService.saveCountry(country5);  
 countryService.saveCountry(country6);  
 countryService.saveCountry(country7);  
 *LOGGER*.info("Inserted sample data");  
 }  
 List<Country> countries=countryService.getAllCountries();  
 *LOGGER*.info("Printing all countries:");  
 for(Country country:countries){  
 *LOGGER*.debug("Country Code: {}, Country Name: {}",country.getCode(),country.getName());  
 }  
 };  
 }  
}

**application.properties:**

spring.application.name=orm-learn  
logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
logging.level.org.hibernate.SQL=trace  
logging.level.org.hibernate.type.descriptor.sql=trace  
logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn  
spring.datasource.username=root  
spring.datasource.password=root@123  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**Output:**



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

**Java Persistence API:**

* JPA is a Java specification for managing relational data in Java applications. It defines how Java objects (entities) can be mapped to database tables.
* JPA is only a specification (JSR 338). It does NOT provide actual implementation, instead it provides a set of interfaces and annotations for object-relational mapping (ORM).
* To use JPA, you need an implementation like Hibernate, EclipseLink, or OpenJPA.

**Key Features:**

* Simplifies database operations with annotations (@Entity, @Table, @Id, etc.).
* Standardizes ORM in Java EE and Spring applications.
* Provides a query language called JPQL (Java Persistence Query Language).

**Hibernate**

* Hibernate is the most popular implementation of JPA.
* It is an ORM framework that maps Java objects to database tables and handles CRUD operations.
* Hibernate provides many advanced features beyond the JPA specification.

**Key Features:**

* Implements JPA interfaces and annotations.
* Provides lazy loading, caching, and batch processing.
* Supports HQL (Hibernate Query Language) in addition to JPQL.
* Handles database transactions, connections, and schema generation automatically.

**Spring Data JPA**

* Spring Data JPA is a module of Spring that simplifies JPA-based data access.
* It does not provide its own implementation of JPA; instead, it works on top of JPA implementations like Hibernate.
* It eliminates boilerplate code by providing ready-made repository interfaces like JpaRepository, CrudRepository, etc.

**Key Features:**

* Reduces the need to write DAO (Data Access Object) classes.
* Automatically implements CRUD methods based on method names (e.g., findById(), findByName()).
* Integrates seamlessly with Spring Boot to manage transactions and dependencies.
* Provides pagination, sorting, and dynamic query creation.

|  |  |  |  |
| --- | --- | --- | --- |
| Aspect | JPA | Hibernate | Spring Data JPA |
| Type | Specification (JSR 338) | Implementation of JPA | Abstraction layer over JPA |
| Implementation | No implementation | Provides implementation | Relies on JPA implementations like Hibernate |
| Boilerplate Code | Requires developer to write DAOs | Slightly less but still needs DAO methods | Eliminates boilerplate DAO code |
| ORM Support | Defines mapping rules (via annotations) | Provides full ORM support | Uses ORM features from JPA implementations |
| Query Support | JPQL | JPQL + HQL + Criteria API | Derived Queries + JPQL + Criteria API |
| Transaction Management | Must manage manually or with EJB | Provides transaction management | Spring manages transactions automatically |
| Ease of Use | Medium complexity | Moderate complexity | Very easy (focus only on business logic) |