**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.

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  
 https://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("Saving book: " + bookName);  
 }  
}

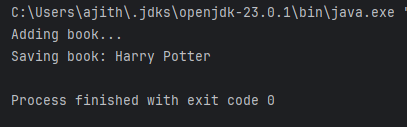
BookService.java

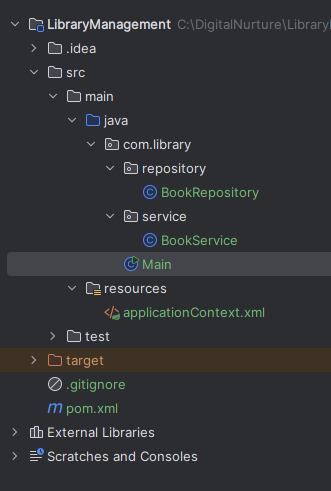
package com.library.service;  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook(String name) {  
 System.*out*.println("Adding book...");  
 bookRepository.saveBook(name);  
 }  
}

Main.java

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Main{  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService = (BookService) context.getBean("bookService");  
 bookService.addBook("Harry Potter");  
 }  
}

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.

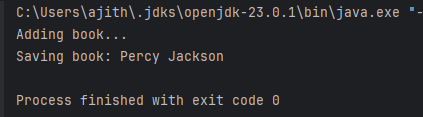
applicationContext.xml

<?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  
 https://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>

Main.java

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Main{  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 BookService bookService = (BookService) context.getBean("bookService");  
 bookService.addBook("Percy Jackson");  
 }  
}

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.

<?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>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.32</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.32</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.32</version>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <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>  
 </plugins>  
 </build>  
  
 <properties>  
 <maven.compiler.source>23</maven.compiler.source>  
 <maven.compiler.target>23</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
</project>

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

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**Steps:**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

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  
 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <!-- Define Repository Bean -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository" />  
  
 <!-- Define Service Bean with Dependency Injection -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
</beans>

**Exercise 7: Implementing Constructor and Setter Injection**

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Steps:**

1. **Configure Constructor Injection:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.
2. **Configure Setter Injection:**
   * Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.
3. **Test the Injection:**
   * Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public BookService() {  
 System.*out*.println("BookService created with default constructor");  
 }  
  
 public BookService(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 System.*out*.println("BookService created with constructor injection");  
 }  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 System.*out*.println("BookRepository injected via setter");  
 }  
  
 public void addBook(String name) {  
 System.*out*.println("Adding book...");  
 bookRepository.saveBook(name);  
 }  
}

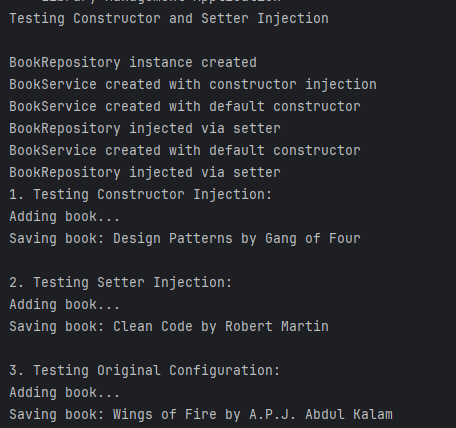
applicationContext.xml

<?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  
 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="bookRepository" class="com.library.repository.BookRepository" />  
  
 <bean id="bookServiceConstructor" class="com.library.service.BookService">  
 <constructor-arg ref="bookRepository" />  
 </bean>  
  
 <bean id="bookServiceSetter" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
</beans>

Main.java  
package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("=== Library Management Application ===");  
 System.*out*.println("Testing Constructor and Setter Injection\n");  
  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 System.*out*.println("1. Testing Constructor Injection:");  
 BookService bookServiceConstructor = (BookService) context.getBean("bookServiceConstructor");  
 bookServiceConstructor.addBook("Design Patterns by Gang of Four");  
  
 System.*out*.println("\n2. Testing Setter Injection:");  
 BookService bookServiceSetter = (BookService) context.getBean("bookServiceSetter");  
 bookServiceSetter.addBook("Clean Code by Robert Martin");  
  
 System.*out*.println("\n3. Testing Original Configuration:");  
 BookService bookService = (BookService) context.getBean("bookService");  
 bookService.addBook("Wings of Fire by A.P.J. Abdul Kalam");  
  
 }  
}

BookRepository.java

package com.library.repository;  
  
public class BookRepository {  
   
 public BookRepository() {  
 System.*out*.println("BookRepository instance created");  
 }  
   
 public void saveBook(String bookName) {  
 System.*out*.println("Saving book: " + bookName);  
 }  
}



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

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

1. **Create a Spring Boot Project:**
   * Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.
2. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.
4. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

<?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>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.7.18</version>  
 <relativePath/>  
 </parent>  
  
 <groupId>com.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
 <name>LibraryManagement</name>  
 <description>Spring Boot Library Management System</description>  
  
 <dependencies>  
 <!-- Spring Boot Web Starter -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <!-- Spring Boot Data JPA Starter -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 <!-- H2 Database -->  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
 <!-- Spring Boot Test Starter -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
 <properties>  
 <java.version>11</java.version>  
 <maven.compiler.source>11</maven.compiler.source>  
 <maven.compiler.target>11</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
</project>

DataLoader.java

package com.library.config;  
  
import com.library.entity.Book;  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.stereotype.Component;  
  
@Component  
public class DataLoader implements CommandLineRunner {  
  
 @Autowired  
 private BookRepository bookRepository;  
  
 @Override  
 public void run(String... args) throws Exception {  
 // Check if data already exists  
 if (bookRepository.count() == 0) {  
 // Create sample books  
 Book book1 = new Book(  
 "Clean Code: A Handbook of Agile Software Craftsmanship",  
 "Robert C. Martin",  
 "978-0132350884",  
 2008,  
 5  
 );  
  
 Book book2 = new Book(  
 "Design Patterns: Elements of Reusable Object-Oriented Software",  
 "Gang of Four",  
 "978-0201633610",  
 1994,  
 3  
 );  
  
 Book book3 = new Book(  
 "Spring Boot in Action",  
 "Craig Walls",  
 "978-1617292545",  
 2015,  
 4  
 );  
  
 Book book4 = new Book(  
 "Effective Java",  
 "Joshua Bloch",  
 "978-0134685991",  
 2017,  
 6  
 );  
  
 Book book5 = new Book(  
 "Wings of Fire: An Autobiography",  
 "A.P.J. Abdul Kalam",  
 "978-8173711466",  
 1999,  
 2  
 );  
  
 // Save books to database  
 bookRepository.save(book1);  
 bookRepository.save(book2);  
 bookRepository.save(book3);  
 bookRepository.save(book4);  
 bookRepository.save(book5);  
  
 System.*out*.println("Sample data loaded successfully!");  
 System.*out*.println("Total books in database: " + bookRepository.count());  
 }  
 }  
}

BookController.java

package com.library.controller;  
  
import com.library.entity.Book;  
import com.library.service.BookService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/books")  
@CrossOrigin(origins = "\*")  
public class BookController {  
  
 private final BookService bookService;  
  
 @Autowired  
 public BookController(BookService bookService) {  
 this.bookService = bookService;  
 }  
  
   
 @PostMapping  
 public ResponseEntity<?> createBook(@RequestBody Book book) {  
 try {  
 Book createdBook = bookService.createBook(book);  
 return new ResponseEntity<>(createdBook, HttpStatus.*CREATED*);  
 } catch (RuntimeException e) {  
 return new ResponseEntity<>(e.getMessage(), HttpStatus.*BAD\_REQUEST*);  
 }  
 }  
  
  
 @GetMapping  
 public ResponseEntity<List<Book>> getAllBooks() {  
 List<Book> books = bookService.getAllBooks();  
 return new ResponseEntity<>(books, HttpStatus.*OK*);  
 }  
  
 @GetMapping("/{id}")  
 public ResponseEntity<?> getBookById(@PathVariable Long id) {  
 Optional<Book> book = bookService.getBookById(id);  
 if (book.isPresent()) {  
 return new ResponseEntity<>(book.get(), HttpStatus.*OK*);  
 } else {  
 return new ResponseEntity<>("Book not found with id: " + id, HttpStatus.*NOT\_FOUND*);  
 }  
 }  
  
  
 @PutMapping("/{id}")  
 public ResponseEntity<?> updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {  
 try {  
 Book updatedBook = bookService.updateBook(id, bookDetails);  
 return new ResponseEntity<>(updatedBook, HttpStatus.*OK*);  
 } catch (RuntimeException e) {  
 return new ResponseEntity<>(e.getMessage(), HttpStatus.*NOT\_FOUND*);  
 }  
 }  
  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity<?> deleteBook(@PathVariable Long id) {  
 try {  
 bookService.deleteBook(id);  
 return new ResponseEntity<>("Book deleted successfully", HttpStatus.*OK*);  
 } catch (RuntimeException e) {  
 return new ResponseEntity<>(e.getMessage(), HttpStatus.*NOT\_FOUND*);  
 }  
 }  
  
  
 @GetMapping("/search/title")  
 public ResponseEntity<List<Book>> searchBooksByTitle(@RequestParam String title) {  
 List<Book> books = bookService.searchBooksByTitle(title);  
 return new ResponseEntity<>(books, HttpStatus.*OK*);  
 }  
  
 // Search books by author  
 @GetMapping("/search/author")  
 public ResponseEntity<List<Book>> searchBooksByAuthor(@RequestParam String author) {  
 List<Book> books = bookService.searchBooksByAuthor(author);  
 return new ResponseEntity<>(books, HttpStatus.*OK*);  
 }  
  
  
 @GetMapping("/available")  
 public ResponseEntity<List<Book>> getAvailableBooks() {  
 List<Book> books = bookService.getAvailableBooks();  
 return new ResponseEntity<>(books, HttpStatus.*OK*);  
 }  
  
  
 @PutMapping("/{id}/borrow")  
 public ResponseEntity<?> borrowBook(@PathVariable Long id) {  
 try {  
 Book book = bookService.borrowBook(id);  
 return new ResponseEntity<>(book, HttpStatus.*OK*);  
 } catch (RuntimeException e) {  
 return new ResponseEntity<>(e.getMessage(), HttpStatus.*BAD\_REQUEST*);  
 }  
 }  
  
  
 @PutMapping("/{id}/return")  
 public ResponseEntity<?> returnBook(@PathVariable Long id) {  
 try {  
 Book book = bookService.returnBook(id);  
 return new ResponseEntity<>(book, HttpStatus.*OK*);  
 } catch (RuntimeException e) {  
 return new ResponseEntity<>(e.getMessage(), HttpStatus.*BAD\_REQUEST*);  
 }  
 }  
  
 @GetMapping("/health")  
 public ResponseEntity<String> healthCheck() {  
 return new ResponseEntity<>("Library Management API is running!", HttpStatus.*OK*);  
 }  
}

interface BookRepository.java

package com.library.repository;  
  
import com.library.entity.Book;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
import java.util.Optional;  
  
@Repository  
public interface BookRepository extends JpaRepository<Book, Long> {  
   
 List<Book> findByTitleContainingIgnoreCase(String title);  
   
 List<Book> findByAuthorContainingIgnoreCase(String author);  
   
 Optional<Book> findByIsbn(String isbn);  
   
 List<Book> findByPublicationYear(Integer year);  
   
 @Query("SELECT b FROM Book b WHERE b.availableCopies > 0")  
 List<Book> findAvailableBooks();  
   
 @Query("SELECT b FROM Book b WHERE LOWER(b.title) LIKE LOWER(CONCAT('%', :title, '%')) AND LOWER(b.author) LIKE LOWER(CONCAT('%', :author, '%'))")  
 List<Book> findByTitleAndAuthor(@Param("title") String title, @Param("author") String author);  
}

BookService.java

package com.library.service;  
  
import com.library.entity.Book;  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class BookService {  
   
 private final BookRepository bookRepository;  
  
 @Autowired  
 public BookService(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 // Create a new book  
 public Book createBook(Book book) {  
 // Check if book with same ISBN already exists  
 if (bookRepository.findByIsbn(book.getIsbn()).isPresent()) {  
 throw new RuntimeException("Book with ISBN " + book.getIsbn() + " already exists");  
 }  
 return bookRepository.save(book);  
 }  
  
 // Get all books  
 public List<Book> getAllBooks() {  
 return bookRepository.findAll();  
 }  
  
 // Get book by ID  
 public Optional<Book> getBookById(Long id) {  
 return bookRepository.findById(id);  
 }  
  
 // Update book  
 public Book updateBook(Long id, Book bookDetails) {  
 Book book = bookRepository.findById(id)  
 .orElseThrow(() -> new RuntimeException("Book not found with id: " + id));  
  
 book.setTitle(bookDetails.getTitle());  
 book.setAuthor(bookDetails.getAuthor());  
 book.setIsbn(bookDetails.getIsbn());  
 book.setPublicationYear(bookDetails.getPublicationYear());  
 book.setTotalCopies(bookDetails.getTotalCopies());  
 book.setAvailableCopies(bookDetails.getAvailableCopies());  
  
 return bookRepository.save(book);  
 }  
  
 // Delete book  
 public void deleteBook(Long id) {  
 Book book = bookRepository.findById(id)  
 .orElseThrow(() -> new RuntimeException("Book not found with id: " + id));  
 bookRepository.delete(book);  
 }  
  
 // Search books by title  
 public List<Book> searchBooksByTitle(String title) {  
 return bookRepository.findByTitleContainingIgnoreCase(title);  
 }  
  
 // Search books by author  
 public List<Book> searchBooksByAuthor(String author) {  
 return bookRepository.findByAuthorContainingIgnoreCase(author);  
 }  
  
 // Get available books  
 public List<Book> getAvailableBooks() {  
 return bookRepository.findAvailableBooks();  
 }  
  
 // Borrow a book  
 public Book borrowBook(Long id) {  
 Book book = bookRepository.findById(id)  
 .orElseThrow(() -> new RuntimeException("Book not found with id: " + id));  
  
 if (book.getAvailableCopies() <= 0) {  
 throw new RuntimeException("No copies available for book: " + book.getTitle());  
 }  
  
 book.setAvailableCopies(book.getAvailableCopies() - 1);  
 return bookRepository.save(book);  
 }  
  
 // Return a book  
 public Book returnBook(Long id) {  
 Book book = bookRepository.findById(id)  
 .orElseThrow(() -> new RuntimeException("Book not found with id: " + id));  
  
 if (book.getAvailableCopies() >= book.getTotalCopies()) {  
 throw new RuntimeException("All copies are already available for book: " + book.getTitle());  
 }  
  
 book.setAvailableCopies(book.getAvailableCopies() + 1);  
 return bookRepository.save(book);  
 }  
}

output

