**Week-3**

**SPRING CORE MAVEN HANDS ON**

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

**Program:**

<!-- 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>

        <spring.version>5.3.20</spring.version> <!-- Using a common stable version -->

    </properties>

    <dependencies>

        <!-- Spring Core Dependency -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-context</artifactId>

            <version>${spring.version}</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

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

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

                <version>3.8.1</version>

                <configuration>

                    <source>${maven.compiler.source}</source>

                    <target>${maven.compiler.target}</target>

                </configuration>

            </plugin>

        </plugins>

    </build>

</project>

```xml

<!-- src/main/resources/applicationContext.xml -->

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

<beans xmlns="[http://www.springframework.org/schema/beans](http://www.springframework.org/schema/beans)"

       xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance](http://www.w3.org/2001/XMLSchema-instance)"

       xsi:schemaLocation="[http://www.springframework.org/schema/beans](http://www.springframework.org/schema/beans)

                           [http://www.springframework.org/schema/beans/spring-beans.xsd](http://www.springframework.org/schema/beans/spring-beans.xsd)">

    <!-- Define BookRepository Bean -->

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

        <!-- No dependencies for now, but could inject data source, etc. -->

    </bean>

    <!-- Define BookService Bean and inject BookRepository -->

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

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

    </bean>

</beans>

```java

// src/main/java/com/library/repository/BookRepository.java

package com.library.repository;

import java.util.ArrayList;

import java.util.List;

/\*\*

 \* BookRepository class to simulate data access operations for books.

 \* In a real application, this would interact with a database.

 \*/

public class BookRepository {

    public BookRepository() {

        System.out.println("BookRepository initialized.");

    }

    /\*\*

     \* Simulates finding a book by its title.

     \* @param title The title of the book to find.

     \* @return A string representing the found book, or null if not found.

     \*/

    public String findBookByTitle(String title) {

        // Simulate database lookup

        if ("The Great Gatsby".equals(title)) {

            return "Book: The Great Gatsby, Author: F. Scott Fitzgerald";

        }

        return null;

    }

    /\*\*

     \* Simulates retrieving all books.

     \* @return A list of strings, each representing a book.

     \*/

    public List<String> findAllBooks() {

        List<String> books = new ArrayList<>();

        books.add("Book: 1984, Author: George Orwell");

        books.add("Book: To Kill a Mockingbird, Author: Harper Lee");

        books.add("Book: The Catcher in the Rye, Author: J.D. Salinger");

        return books;

    }

}

```java

// src/main/java/com/library/service/BookService.java

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

/\*\*

 \* BookService class to handle business logic related to books.

 \* It depends on BookRepository for data access.

 \*/

public class BookService {

    private BookRepository bookRepository;

    // Setter for dependency injection (Spring will use this)

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

        System.out.println("BookRepository injected into BookService.");

    }

    public BookService() {

        System.out.println("BookService initialized.");

    }

    /\*\*

     \* Retrieves a book by its title using the repository.

     \* @param title The title of the book.

     \* @return The book details as a string, or a message if not found.

     \*/

    public String getBookDetails(String title) {

        String book = bookRepository.findBookByTitle(title);

        if (book != null) {

            return "Found: " + book;

        } else {

            return "Book with title '" + title + "' not found.";

        }

    }

    /\*\*

     \* Retrieves all books using the repository.

     \* @return A list of all book details.

     \*/

    public List<String> getAllBooks() {

        return bookRepository.findAllBooks();

    }

}

```java

// src/main/java/com/library/MainApp.java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

/\*\*

 \* Main application class to load the Spring context and test the configured beans.

 \*/

public class MainApp {

    public static void main(String[] args) {

        System.out.println("Loading Spring application context...");

        // Load the Spring application context from the XML file

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

        System.out.println("\nSpring context loaded successfully.");

        // Retrieve the BookService bean from the context

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

        System.out.println("\n--- Testing BookService ---");

        // Test finding a specific book

        String book1 = bookService.getBookDetails("The Great Gatsby");

        System.out.println(book1);

        String book2 = bookService.getBookDetails("Non-existent Book");

        System.out.println(book2);

        // Test getting all books

        System.out.println("\nAll available books:");

        List<String> allBooks = bookService.getAllBooks();

        for (String book : allBooks) {

            System.out.println("- " + book);

        }

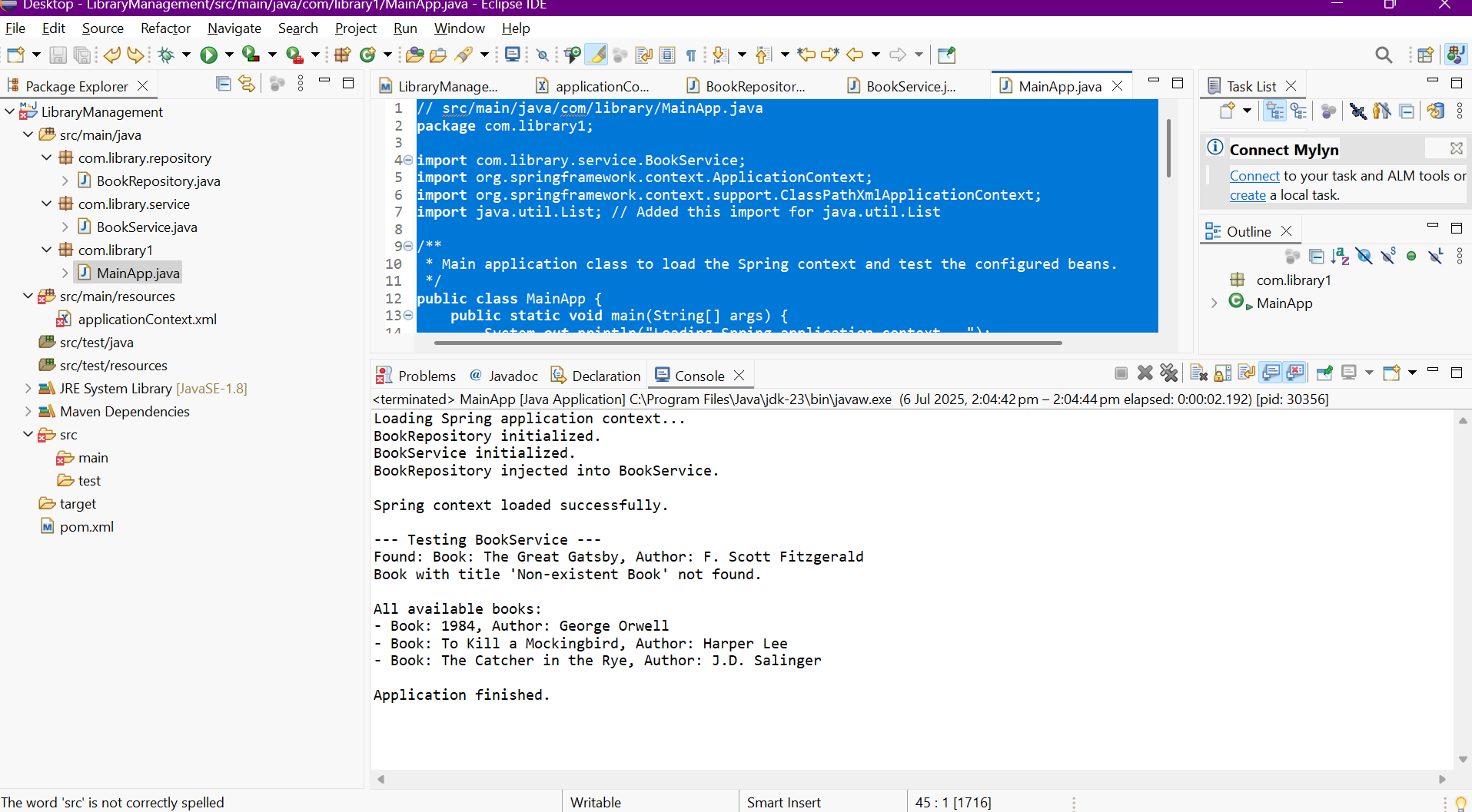
        // Close the context (important for applications that might shut down cleanly)

        ((ClassPathXmlApplicationContext) context).close();

        System.out.println("\nApplication finished.");

    }

}

**Output:**

**Exercise 2: Implementing Dependency Injection**

**Program**:

<?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">

    <!-- Define BookRepository Bean -->

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

        <!-- No dependencies for now, but could inject data source, etc. -->

    </bean>

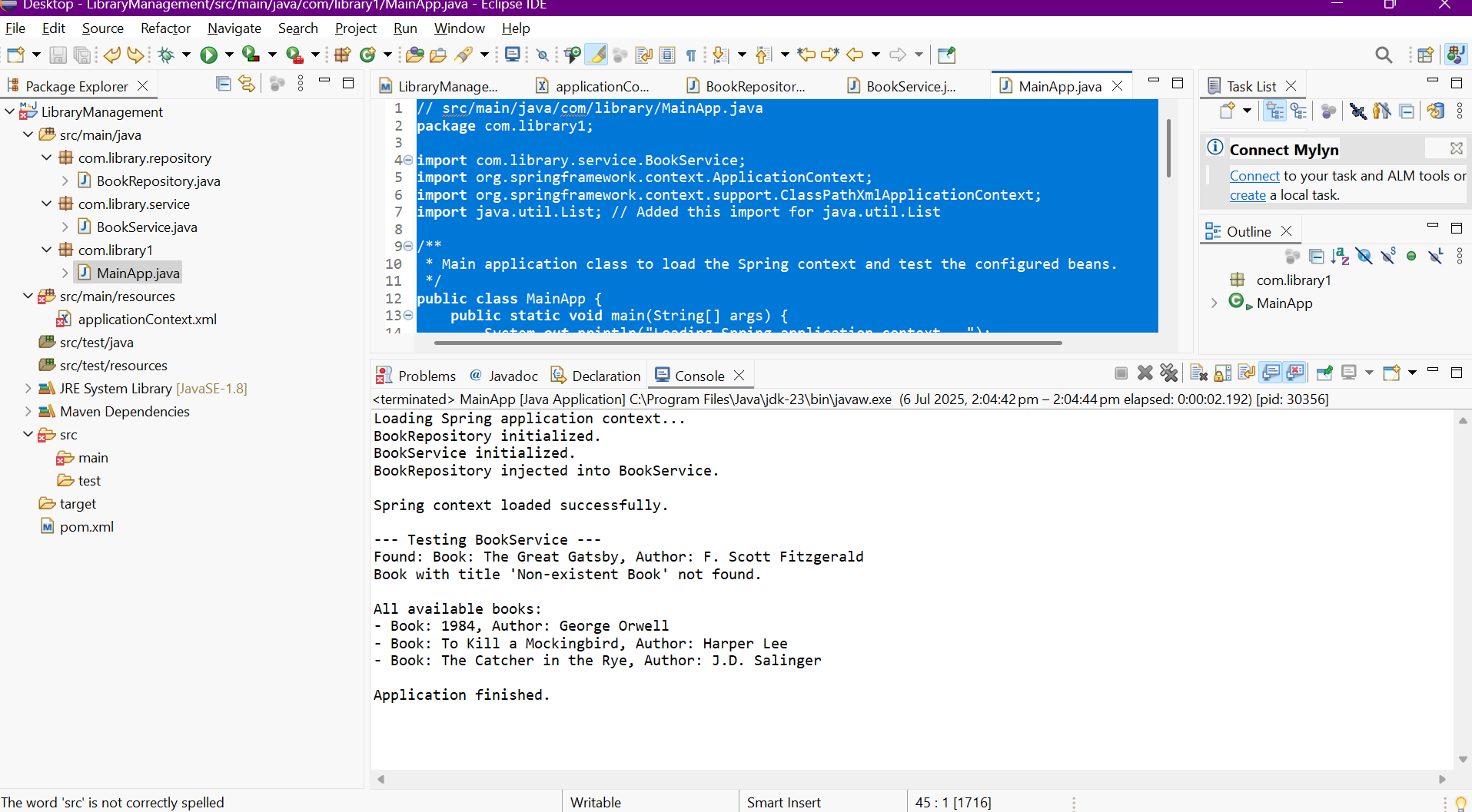
    <!-- Define BookService Bean and inject BookRepository -->

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

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

    </bean>

</beans>

****

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

**Program:**

<?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>

        <spring.version>5.3.20</spring.version> <!-- Using a common stable version -->

    </properties>

    <dependencies>

        <!-- Spring Context Dependency (for Core IoC and DI) -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-context</artifactId>

            <version>${spring.version}</version>

        </dependency>

        <!-- Spring AOP Dependency (for Aspect-Oriented Programming) -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-aop</artifactId>

            <version>${spring.version}</version>

        </dependency>

        <!-- Spring WebMVC Dependency (for Web Application Development) -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-webmvc</artifactId>

            <version>${spring.version}</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <!-- Configure the Maven Compiler Plugin for Java 1.8 -->

            <plugin>

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

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

                <version>3.8.1</version>

                <configuration>

                    <source>${maven.compiler.source}</source>

                    <target>${maven.compiler.target}</target>

                </configuration>

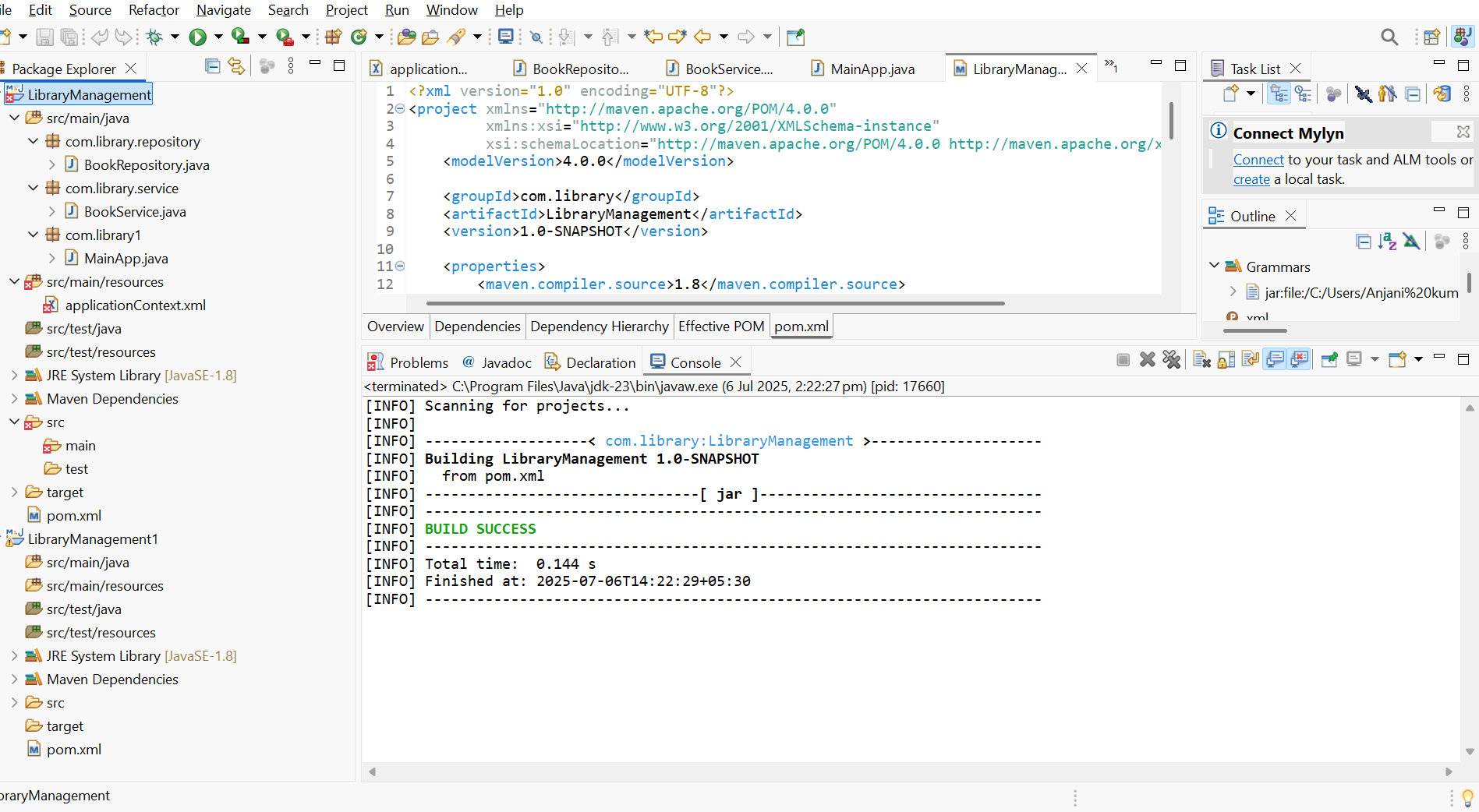
            </plugin>

        </plugins>

    </build>

</project>

Output:



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

Program:

**package** com.library1;

**import** com.library.service.BookService;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** java.util.List;

/\*\*

\* Main application class to load the Spring context and test the configured beans.

\*/

**public** **class** MainApp {

**public** **static** **void** main(String[] args) {

System.***out***.println("Loading Spring application context...");

// Load the Spring application context from the XML file

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

System.***out***.println("\nSpring context loaded successfully.");

// Retrieve the BookService bean from the context

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

System.***out***.println("\n--- Testing BookService ---");

// Test finding a specific book

String book1 = bookService.getBookDetails("The Great Gatsby");

System.***out***.println(book1);

String book2 = bookService.getBookDetails("Non-existent Book");

System.***out***.println(book2);

// Test getting all books

System.***out***.println("\nAll available books:");

List<String> allBooks = bookService.getAllBooks();

**for** (String book : allBooks) {

System.***out***.println("- " + book);

}

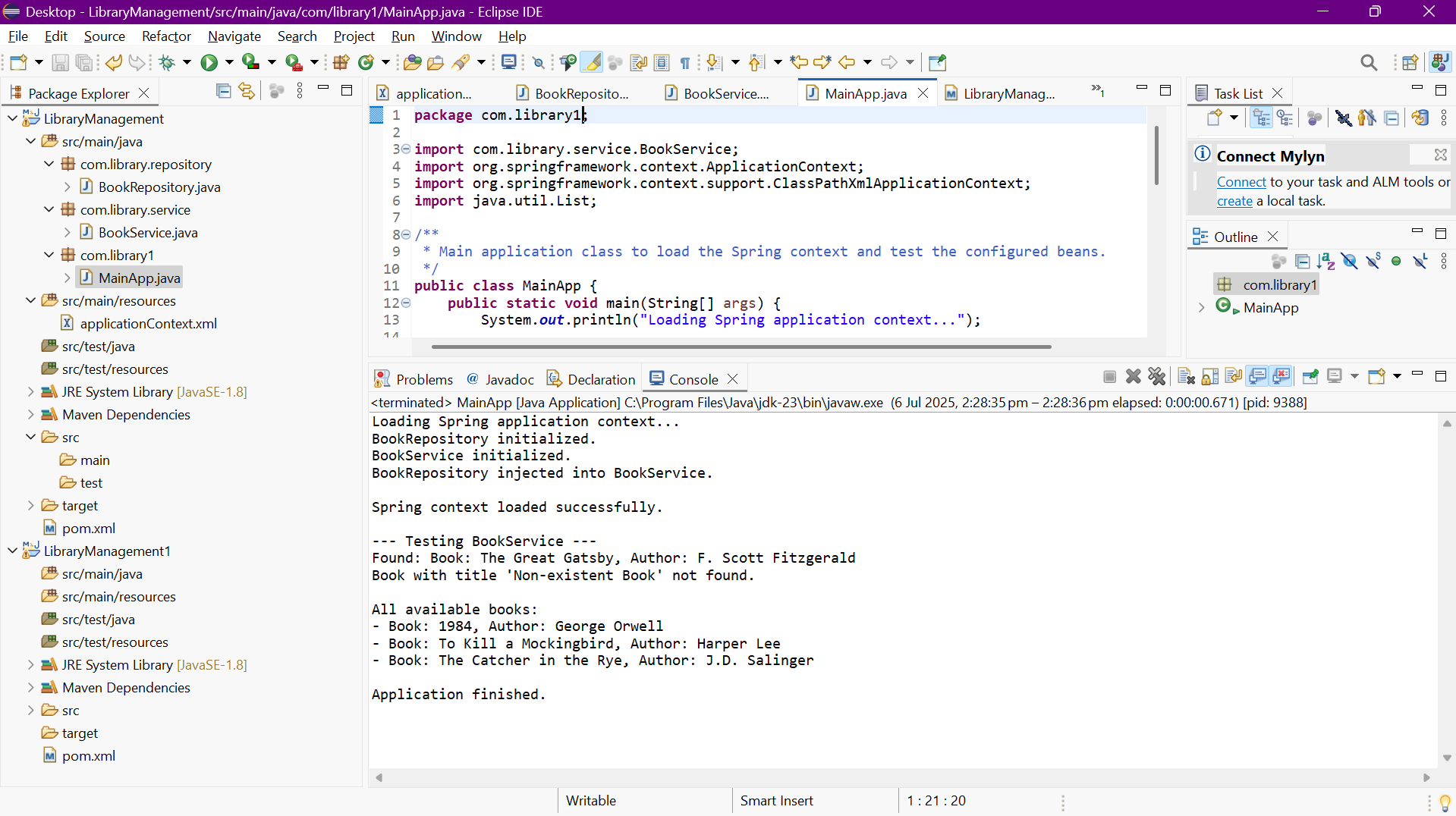
// Close the context (important for applications that might shut down cleanly)

((ClassPathXmlApplicationContext) context).close();

System.***out***.println("\nApplication finished.");

}

}



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

Program:

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

/\*\*

 \* BookService class to handle business logic related to books.

 \* This version demonstrates both constructor and setter injection.

 \*/

public class BookService {

    private BookRepository bookRepository;

    private String libraryName; // New property for setter injection

    /\*\*

     \* Constructor for BookService, demonstrating constructor injection.

     \* Spring will use this constructor to inject the BookRepository bean.

     \* @param bookRepository The BookRepository instance to be injected.

     \*/

    public BookService(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

        System.out.println("BookRepository injected via Constructor into BookService.");

    }

    /\*\*

     \* Setter method for BookRepository, demonstrating setter injection.

     \* While BookRepository is primarily constructor-injected here,

     \* this setter is kept to show how both types of injection can coexist

     \* for different properties or for re-injecting.

     \* @param bookRepository The BookRepository instance to set.

     \*/

    public void setBookRepository(BookRepository bookRepository) {

        // This setter might not be called by Spring if constructor injection is used for the same dependency,

        // but it's here to show the mechanism.

        // For this exercise, we're using it to show \*how\* a setter would be present.

        // The primary injection for bookRepository is via the constructor.

        // If you were to inject a \*different\* dependency via setter, this method would be for that.

        this.bookRepository = bookRepository;

        System.out.println("BookRepository (re)injected via Setter into BookService.");

    }

    /\*\*

     \* Setter method for libraryName, demonstrating setter injection.

     \* Spring will use this setter to inject the value defined in applicationContext.xml.

     \* @param libraryName The name of the library.

     \*/

    public void setLibraryName(String libraryName) {

        this.libraryName = libraryName;

        System.out.println("Library Name '" + libraryName + "' injected via Setter into BookService.");

    }

    /\*\*

     \* Retrieves a book by its title using the repository.

     \* Includes the library name for demonstration.

     \* @param title The title of the book.

     \* @return The book details as a string, or a message if not found.

     \*/

    public String getBookDetails(String title) {

        String book = bookRepository.findBookByTitle(title);

        if (book != null) {

            return "Found at " + libraryName + ": " + book;

        } else {

            return "Book with title '" + title + "' not found at " + libraryName + ".";

        }

    }

    /\*\*

     \* Retrieves all books using the repository.

     \* @return A list of all book details.

     \*/

    public List<String> getAllBooks() {

        return bookRepository.findAllBooks();

    }

}

<?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">

<!-- Define BookRepository Bean -->

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

<!-- This bean will be managed by Spring -->

</bean>

<!-- Define BookService Bean

- BookRepository is injected via constructor injection.

- libraryName is injected via setter injection.

-->

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

<!-- Constructor Injection for BookRepository -->

<constructor-arg ref="bookRepository"/>

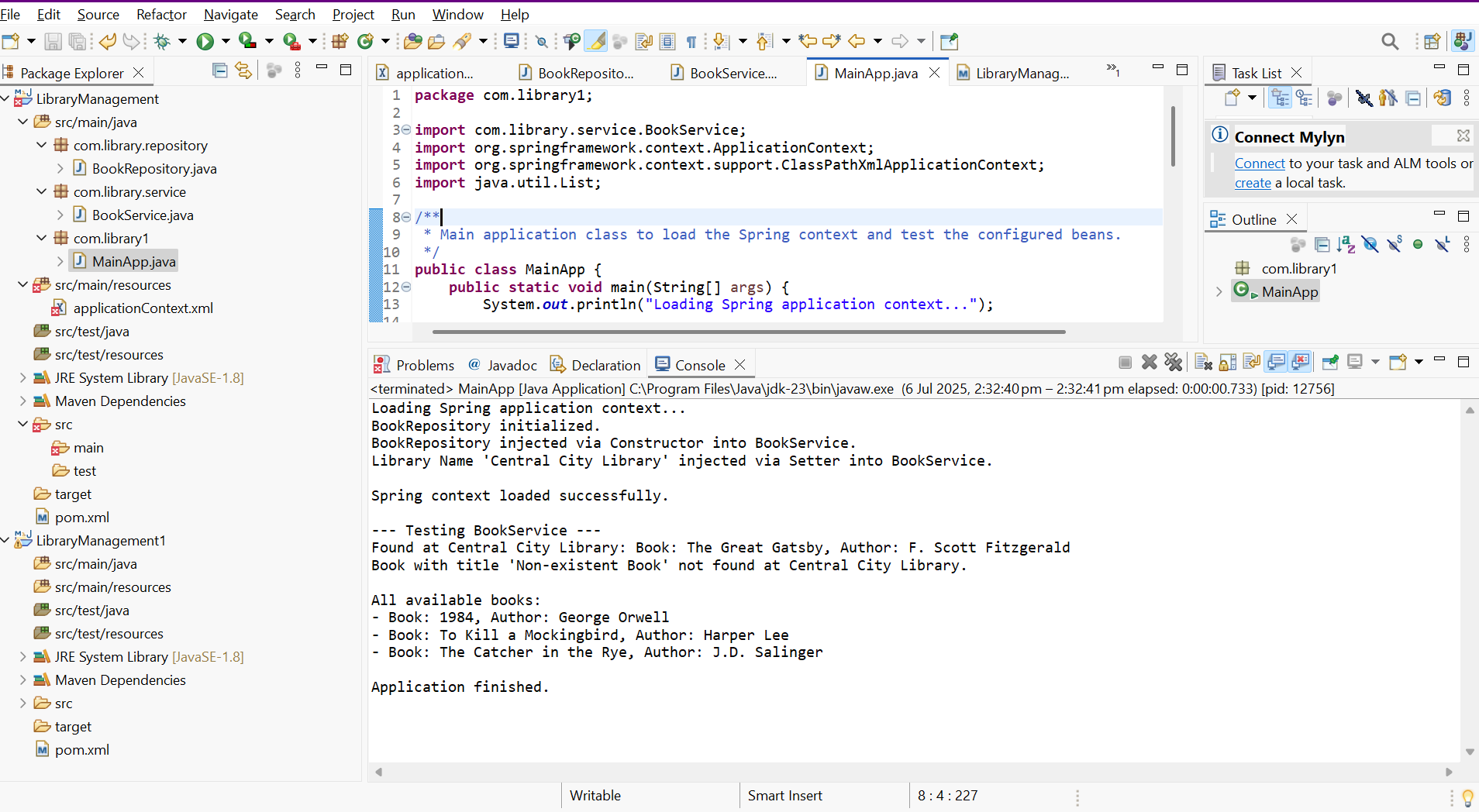
<!-- Setter Injection for a new property: libraryName -->

<property name="libraryName" value="Central City Library"/>

</bean>

</beans>

**OUTPUT:**

****

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

**Program:**

<!-- pom.xml -->

<!-- Path: YourProjectRoot/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 https://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>3.2.7</version> <!-- Use the Spring Boot version you selected (e.g., 3.2.7) -->

        <relativePath/> <!-- lookup parent from repository -->

    </parent>

    <groupId>com.library</groupId>

    <artifactId>LibraryManagement</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <name>LibraryManagement</name>

    <description>Library Management System with Spring Boot</description>

    <properties>

        <java.version>17</java.version> <!-- Match the Java version you selected (e.g., 17) -->

    </properties>

    <dependencies>

        <dependency>

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

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

        </dependency>

        <dependency>

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

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

        </dependency>

        <dependency>

            <groupId>com.h2database</groupId>

            <artifactId>h2</artifactId>

            <scope>runtime</scope>

        </dependency>

        <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>

</project>

```properties

# application.properties

# Path: YourProjectRoot/src/main/resources/application.properties

# H2 Database Configuration

spring.h2.console.enabled=true

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

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

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

spring.jpa.hibernate.ddl-auto=update

# Creates/updates schema automatically - Moved comment to its own line

spring.jpa.show-sql=true

# Show SQL queries in console - Moved comment to its own line

```java

// Book.java (Entity)

// Path: YourProjectRoot/src/main/java/com/library/entity/Book.java

package com.library.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

/\*\*

 \* Represents a Book entity in the library management system.

 \* Mapped to a database table by Spring Data JPA.

 \*/

@Entity // Marks this class as a JPA entity

public class Book {

    @Id // Specifies the primary key

    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-generates ID

    private Long id;

    private String title;

    private String author;

    private String isbn;

    // Default constructor is required by JPA

    public Book() {

    }

    public Book(String title, String author, String isbn) {

        this.title = title;

        this.author = author;

        this.isbn = isbn;

    }

    // Getters and Setters

    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;

    }

    public String getIsbn() {

        return isbn;

    }

    public void setIsbn(String isbn) {

        this.isbn = isbn;

    }

    @Override

    public String toString() {

        return "Book{" +

               "id=" + id +

               ", title='" + title + '\'' +

               ", author='" + author + '\'' +

               ", isbn='" + isbn + '\'' +

               '}';

    }

}

```java

// BookRepository.java (Interface)

// Path: YourProjectRoot/src/main/java/com/library/repository/BookRepository.java

package com.library.repository;

import com.library.entity.Book;

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

import org.springframework.stereotype.Repository;

/\*\*

 \* Repository interface for Book entities.

 \* Spring Data JPA automatically provides CRUD operations.

 \*/

@Repository // Optional, but good practice for clarity

public interface BookRepository extends JpaRepository<Book, Long> {

    // You can add custom query methods here if needed, e.g.,

    // List<Book> findByTitleContaining(String title);

}

```java

// BookController.java (REST Controller)

// Path: YourProjectRoot/src/main/java/com/library/controller/BookController.java

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

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;

/\*\*

 \* REST Controller for managing Book resources.

 \* Handles HTTP requests for CRUD operations on books.

 \*/

@RestController // Marks this class as a REST controller

@RequestMapping("/api/books") // Base path for all endpoints in this controller

public class BookController {

    @Autowired // Injects the BookRepository dependency

    private BookRepository bookRepository;

    /\*\*

     \* Get all books.

     \* GET /api/books

     \* @return A list of all books.

     \*/

    @GetMapping

    public List<Book> getAllBooks() {

        return bookRepository.findAll();

    }

    /\*\*

     \* Get a book by ID.

     \* GET /api/books/{id}

     \* @param id The ID of the book.

     \* @return The book if found, or 404 Not Found.

     \*/

    @GetMapping("/{id}")

    public ResponseEntity<Book> getBookById(@PathVariable Long id) {

        Optional<Book> book = bookRepository.findById(id);

        return book.map(ResponseEntity::ok)

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

    }

    /\*\*

     \* Create a new book.

     \* POST /api/books

     \* @param book The book object to create.

     \* @return The created book with its generated ID.

     \*/

    @PostMapping

    @ResponseStatus(HttpStatus.CREATED) // Returns 201 Created status

    public Book createBook(@RequestBody Book book) {

        return bookRepository.save(book);

    }

    /\*\*

     \* Update an existing book.

     \* PUT /api/books/{id}

     \* @param id The ID of the book to update.

     \* @param bookDetails The updated book details.

     \* @return The updated book if found, or 404 Not Found.

     \*/

    @PutMapping("/{id}")

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

        Optional<Book> optionalBook = bookRepository.findById(id);

        if (optionalBook.isPresent()) {

            Book existingBook = optionalBook.get();

            existingBook.setTitle(bookDetails.getTitle());

            existingBook.setAuthor(bookDetails.getAuthor());

            existingBook.setIsbn(bookDetails.getIsbn());

            Book updatedBook = bookRepository.save(existingBook);

            return ResponseEntity.ok(updatedBook);

        } else {

            return ResponseEntity.notFound().build();

        }

    }

    /\*\*

     \* Delete a book by ID.

     \* DELETE /api/books/{id}

     \* @param id The ID of the book to delete.

     \* @return 204 No Content if successful, or 404 Not Found.

     \*/

    @DeleteMapping("/{id}")

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

        if (bookRepository.existsById(id)) {

            bookRepository.deleteById(id);

            return ResponseEntity.noContent().build(); // 204 No Content

        } else {

            return ResponseEntity.notFound().build();

        }

    }

}

```java

// LibraryManagementApplication.java (Main Spring Boot Class)

// Path: YourProjectRoot/src/main/java/com/library/LibraryManagementApplication.java

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

/\*\*

 \* Main Spring Boot application class for the Library Management System.

 \* This class serves as the entry point for the application.

 \*/

@SpringBootApplication // Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan

public class LibraryManagementApplication {

    public static void main(String[] args) {

        SpringApplication.run(LibraryManagementApplication.class, args);

        System.out.println("Spring Boot Library Management Application started successfully!");

        System.out.println("Access H2 Console at: http://localhost:8080/h2-console (JDBC URL: jdbc:h2:mem:librarydb)");

        System.out.println("Test REST Endpoints (e.g., with Postman or curl):");

        System.out.println("  - GET all books: http://localhost:8080/api/books");

        System.out.println("  - POST create book: http://localhost:8080/api/books (with JSON body)");

    }

}

Output: