**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.

* **Introduction**

This document outlines the steps taken to configure a basic Spring application for managing a library. The application consists of a backend that handles book-related operations, using the Spring Framework to manage dependencies and configurations.

**1. Setting Up the Spring Project**

**Project Creation**

A Maven project named **LibraryManagement** was created to manage the library system. Maven was chosen for dependency management and project structure.

**Dependencies**

* The Spring Core dependencies were added to the pom.xml file. These dependencies are essential for setting up the Spring Context and handling the beans within the application.

**2. Configuring the Application Context**

* An XML configuration file named applicationContext.xml was created in the ***src/main/resources*** directory. This file defines the beans for BookService and ***BookRepository***. This configuration file establishes the relationships between the ***BookService*** and ***BookRepository*** classes, allowing Spring to manage their lifecycle.

## 3. Service and Repository Classes

### BookService

The *BookService* class handles the business logic for managing books. It depends on *BookRepository* to perform operations like adding and listing books.

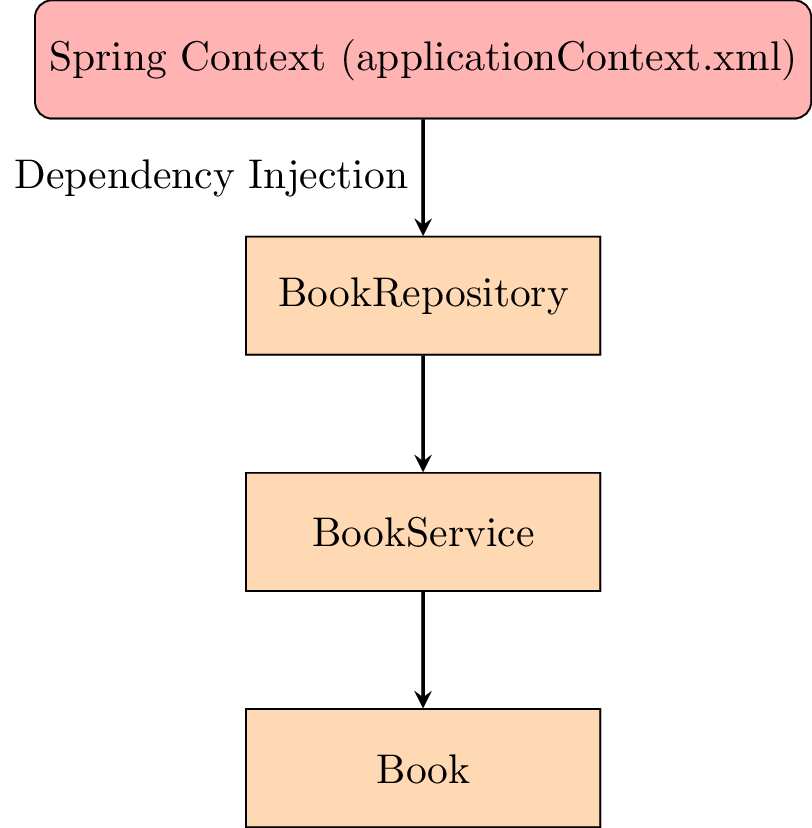
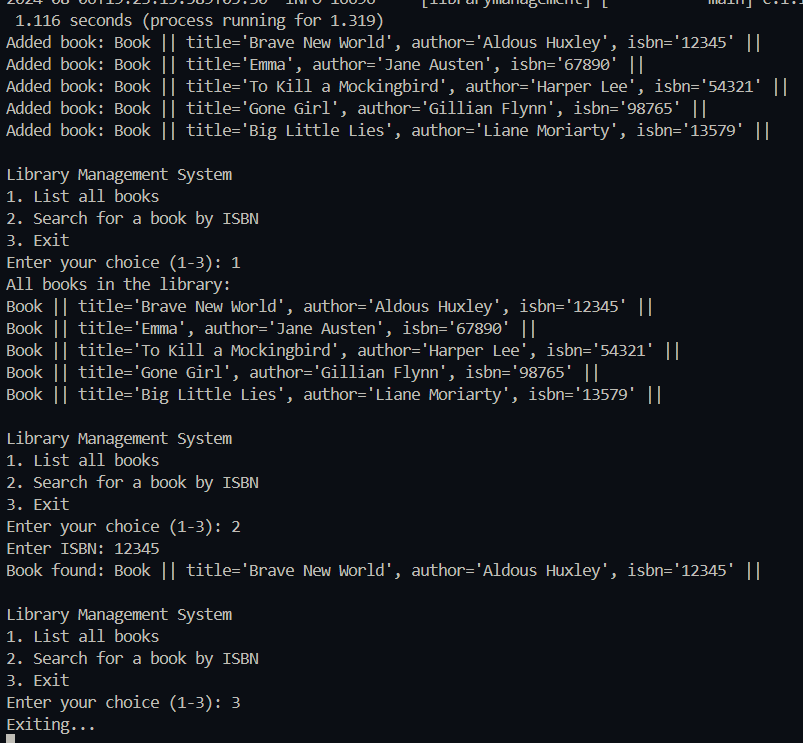
### BookRepository

The *BookRepository* class manages a collection of Book objects, providing methods to add, retrieve, and find books by their ISBN.

## 4. Running the Application

### Main Class

A main class was created to load the Spring context and test the configuration. This class initializes the Spring framework and verifies that the *BookService* and *BookRepository* beans are correctly wired.

****

**FLOWCHART of the program :**

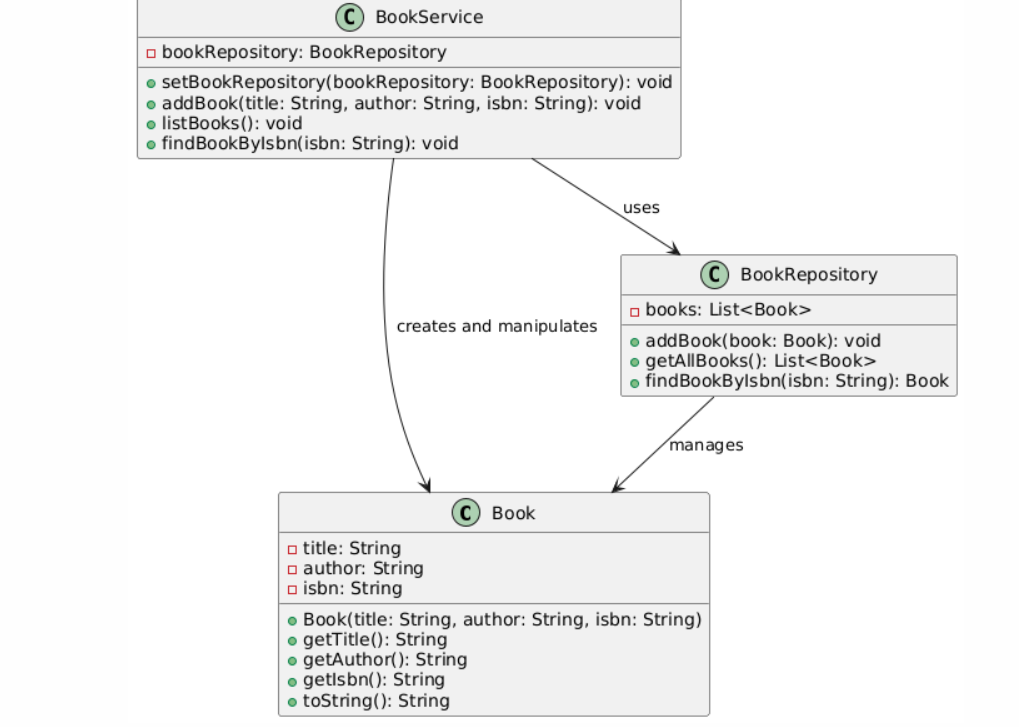
* **Output:**

**Flowchart Explanation**

The flowchart represents the interactions between the key components of the Library Management application:

1. **Spring Context (applicationContext.xml)**:
   * This is where the Spring beans are defined and managed. It acts as the configuration center, handling the creation and injection of dependencies between different components.
2. **BookRepository**:
   * The BookRepository bean is created by Spring as per the configuration in applicationContext.xml. This class is responsible for managing the collection of Book objects. It provides methods to add new books, retrieve all books, and find a book by its ISBN.
3. **BookService**:
   * The BookService bean is also managed by Spring and depends on BookRepository. The Spring context injects BookRepository into BookService, allowing BookService to delegate book management tasks to BookRepository.
4. **Book**:
   * BookService uses the Book class to create instances of books and interact with them. This class represents the data model for books, holding attributes like title, author, and ISBN.

**Flow of Execution:**

* The Spring context initializes and wires the BookRepository and BookService beans.
* BookService interacts with BookRepository to perform operations on Book objects.
* All interactions are managed by Spring, ensuring that dependencies are correctly injected and managed throughout the application lifecycle.
* **CLASS DIAGRAM :**

**Class Diagram Explanation**

The class diagram provides a static view of the system, showing the structure of the classes and their relationships:

1. **Book Class**:
   * **Attributes**: title, author, isbn (all private).
   * **Methods**:
     + Book(String title, String author, String isbn): Constructor to initialize a book.
     + getTitle(), getAuthor(), getIsbn(): Getter methods to retrieve the respective attributes.
     + toString(): A method to return a string representation of the book.
   * This class represents the basic data model for books in the application.
2. **BookRepository Class**:
   * **Attributes**: books (a list of Book objects).
   * **Methods**:
     + addBook(Book book): Adds a new book to the repository.
     + getAllBooks(): Returns all the books in the repository.
     + findBookByIsbn(String isbn): Finds a book in the repository by its ISBN.
   * This class is responsible for managing the collection of books. It provides methods to perform basic CRUD operations on the book data.
3. **BookService Class**:
   * **Attributes**: bookRepository (a reference to BookRepository).
   * **Methods**:
     + setBookRepository(BookRepository bookRepository): Injects the BookRepository dependency.
     + addBook(String title, String author, String isbn): Creates a new book and adds it to the repository.
     + listBooks(): Lists all books in the repository.
     + findBookByIsbn(String isbn): Searches for a book by its ISBN.
   * This class contains the business logic of the application. It uses BookRepository to interact with the data model (Book), and it is responsible for higher-level operations that may involve multiple books or more complex logic.

**Relationships:**

* **Dependency (BookService to BookRepository)**: BookService relies on BookRepository to manage the actual data of the books.
* **Association (BookRepository to Book)**: BookRepository manages a collection of Book objects.
* **Usage (BookService to Book)**: BookService creates and manipulates Book objects using the methods provided by BookRepository.