**Exercise 8: Implementing Basic AOP with Spring**

Scenario:

The library management application requires basic AOP functionality to separate cross-cutting concerns like logging and transaction management.

**1. Introduction**

In this exercise, we enhance the Library Management Application by introducing basic Aspect-Oriented Programming (AOP) with Spring. AOP helps in separating cross-cutting concerns like logging and transaction management from the business logic. We will implement logging using Spring AOP to demonstrate this functionality.

**2. Step-by-Step Implementation**

**Step 1: Defining the Aspect**

**Update:**  
We define an aspect in a new package, com.library.aspect. The aspect class is named LoggingAspect, and it is annotated with @Aspect and @Component to designate it as a Spring-managed aspect.

**Step 2: Creating Advice Methods**

**Update:**  
In the LoggingAspect class, we created two advice methods:

* logBeforeMethod(): Executes before any method in the com.library.service package.
* logAfterMethod(): Executes after any method in the com.library.service package.

These methods will log messages before and after method execution, allowing us to monitor the flow of the application

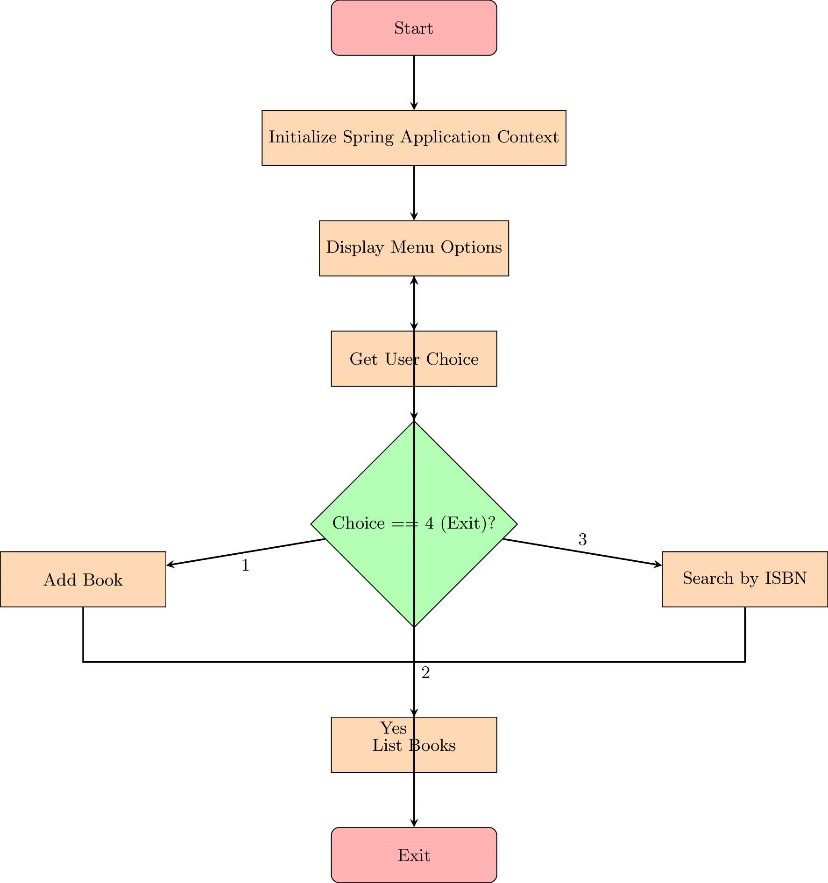
**Step 3: Configuring the Aspect**

**Update:**  
We need to update the applicationContext.xml to register the aspect and enable AspectJ auto-proxying. This ensures that Spring will automatically create proxies for beans where aspects are applied.

**Step 4: Testing the Aspect**

Run the LibraryManagementApplication main class to test the AOP functionality. It runs without an error.

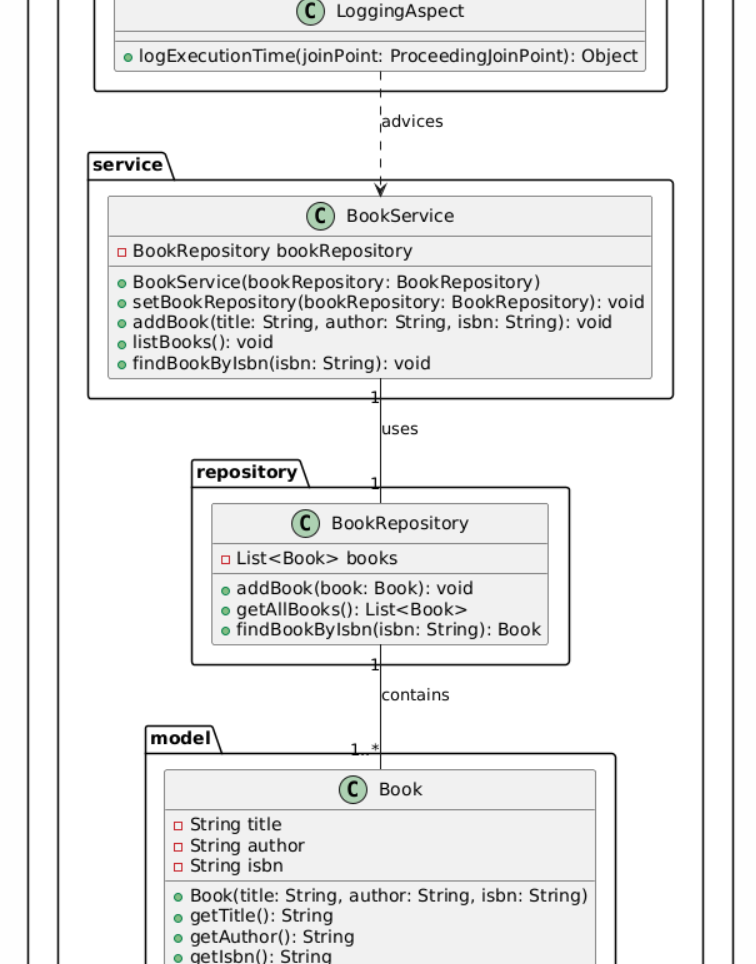
**FLOWCHART of the program :**



The flowchart depicts the flow of control in the LibraryManagementApplication class:

* **Start:** The application initializes the Spring Application Context.
* **Menu Display:** The user is presented with options to add a book, list all books, search by ISBN, or exit the application.
* **User Choice:** The user's choice is captured and processed:
  + **Add Book:** Prompts the user for book details and adds the book to the repository.
  + **List Books:** Retrieves and displays all books in the library.
  + **Search Book:** Searches for a book by its ISBN.
  + **Exit:** Terminates the application.
* **Decision Points:** The application checks if the user wants to exit. If not, the process loops back to display the menu again

**CLASS DIAGRAM** :



The class diagram illustrates the structure of the classes and their relationships:

* **Book:** Represents a book entity with attributes like title, author, and ISBN.
* **BookRepository:** Manages a collection of Book objects and provides methods to add, retrieve, and search books.
* **BookService:** Provides services related to books, such as adding and listing books. It uses BookRepository for these operations.
* **LoggingAspect:** An aspect that logs method executions in the BookService class. It is applied before and after method calls to capture execution details.