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.

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.

Exercise 3: Implementing Logging with Spring AOP

Scenario:

The library management application requires logging capabilities to track method execution times.

Steps:

1. **Add Spring AOP Dependency:**
    - Update **pom.xml** to include Spring AOP dependency.
2. **Create an Aspect for Logging:**
    - Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.
3. **Enable AspectJ Support:**
    - Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.
4. **Test the Aspect:**
    - Run the **LibraryManagementApplication** main class and observe the console for log messages indicating method execution times.

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.

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.

Exercise 6: Configuring Beans with Annotations

Scenario:

You need to simplify the configuration of beans in the library management application using annotations.

Steps:

1. **Enable Component Scanning:**
    - Update **applicationContext.xml** to include component scanning for the **com.library** package.
2. **Annotate Classes:**
    - Use **@Service** annotation for the **BookService** class.
    - Use **@Repository** annotation for the **BookRepository** class.
3. **Test the Configuration:**
    - Run the **LibraryManagementApplication** main class to verify the annotation-based configuration.

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.

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.

Steps:

1. **Define an Aspect:**
    - Create a package **com.library.aspect** and add a class **LoggingAspect**.
2. **Create Advice Methods:**
    - Define advice methods in **LoggingAspect** for logging before and after method execution.
3. **Configure the Aspect:**
    - Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.
4. **Test the Aspect:**
    - Run the **LibraryManagementApplication** main class to verify the AOP functionality.

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.