**Implementing Basic AOP with Spring**

The library management application requires basic Aspect-Oriented Programming (AOP) functionality to separate cross-cutting concerns like logging and transaction management.

**Understanding the Problem:**

AOP is used in Spring to manage cross-cutting concerns such as logging, security, and transaction management. It helps to keep the code modular by separating these concerns from the main business logic.

**Implementation Steps:**

1. **Define an Aspect**:
   * Created a package com.library.aspect.
   * Added a class LoggingAspect in the com.library.aspect package.

An aspect is a class annotated with @Aspect. In this case, LoggingAspect is created to handle logging concerns. The @Component annotation registers it as a Spring bean.

1. **Create Advice Methods**:
   * Defined two advice methods in LoggingAspect:
     + logBefore method, which logs a message before the execution of any method in the BookService class.
     + logAfter method, which logs a message after the execution of any method in the BookService class.

* @Before("execution(\* com.library.service.BookService.\*(..))") specifies that the logBefore method should run before any method in the BookService class.
* @After("execution(\* com.library.service.BookService.\*(..))") specifies that the logAfter method should run after any method in the BookService class.

1. **Configure the Aspect**:
   * Updated applicationContext.xml to register the aspect and enable AspectJ auto-proxying.
   * Enabled component scanning to automatically detect Spring components.

* In applicationContext.xml, context:component-scan is used to enable component scanning.
* aop:aspectj-autoproxy is enabled to allow the use of AspectJ annotations for defining aspects.

1. **Test the Aspect**:
   * Ran the MainApp class to verify the AOP functionality.

* The MainApp class loads the Spring context from applicationContext.xml and retrieves the BookService bean.
* When methods in BookService are called, the logging aspect is triggered, demonstrating the AOP functionality.

**Extra Notes:**

* The implementation successfully logs messages before and after method execution, proving the AOP setup works as intended.
* To improve performance, using more specific pointcuts to target only the necessary methods, means for e.g., instead of logging every method in BookService, logging only critical methods like addBook or removeBook reduces unnecessary overhead of logging every method and focuses on important parts of the application.

**Conclusion:**

This exercise shows how to implement basic AOP functionality in a Spring application to manage concerns like logging. Using aspects helps to keep the code clean and modular.