**Implementing Dependency Injection in Library Management Application**

This document explains how to implement Dependency Injection (DI) using Spring's Inversion of Control (IoC) container in the library management application. DI is a design pattern that promotes loose coupling between classes by separating object creation and dependency management.

**Understanding Dependency Injection**

* The BookService class relies on the BookRepository to interact with book data.
* Traditionally, BookService might create a BookRepository instance itself, leading to tight coupling.
* Spring's IoC container manages object creation and injects dependencies during object initialization.

**Modifying the XML Configuration (Step 1)**

* We update applicationContext.xml to define the dependency between BookService and BookRepository.
* The <bean> element for BookService now includes a <property> element.
* This <property> specifies the setter method (setBookRepository) to be used for injecting the BookRepository bean (referenced by its id "bookRepository").

**Updating the BookService Class (Step 2)**

* We ensure BookService has a setter method named setBookRepository that takes a BookRepository argument.
* Spring will call this method during object creation to inject the dependency.

**Testing the Configuration (Step 3)**

* We run the MainApp class to verify DI functionality.
* The MainApp class retrieves the BookService bean from the Spring context.
* When bookService.manageBooks() is called, Spring injects the BookRepository instance before executing the method.
* The output confirms successful DI, with BookService managing books and BookRepository saving a book.

**Benefits of Dependency Injection**

* Promotes loose coupling: Classes don't need to know how to create their dependencies.
* Improves testability: Easier to mock and test isolated components by injecting test doubles.
* Enhances maintainability: Changes in dependencies are centralized in the Spring configuration.

This exercise demonstrates a fundamental concept in Spring development. By leveraging DI, developers can build more flexible and maintainable applications.