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.

CODE:

```
// ==================== pom.xml ====================

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

      http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>


  <groupId>com.library</groupId>

  <artifactId>LibraryManagement</artifactId>

  <version>1.0-SNAPSHOT</version>


  <properties>
```

```
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <spring.version>5.3.21</spring.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>${spring.version}</version>
    </dependency>
  </dependencies>
</project>

// ==================== src/main/resources/applicationContext.xml ====================
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
      http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="bookRepository" class="com.library.repository.BookRepository" />

    <bean id="bookService" class="com.library.service.BookService">
        <property name="bookRepository" ref="bookRepository" />
    </bean>

</beans>

// ==================== src/main/java/com/library/model/Book.java ====================
package com.library.model;

public class Book {
    private String id;
    private String title;
    private String author;

    public Book() {}

    public Book(String id, String title, String author) {
        this.id = id;
        this.title = title;
        this.author = author;
    }

    // Getters and Setters
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
```

```
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    @Override
    public String toString() {
        return "Book{id='" + id + "', title='" + title + "', author='" + author + "'}";
    }
}

// ==================== src/main/java/com/library/repository/BookRepository.java ====================
package com.library.repository;

import com.library.model.Book;
import java.util.ArrayList;
import java.util.List;

public class BookRepository {
    private List<Book> books;

    public BookRepository() {
        this.books = new ArrayList<>();
        initializeData();
    }
```

```
    private void initializeData() {
        books.add(new Book("1", "The Great Gatsby", "F. Scott Fitzgerald"));
        books.add(new Book("2", "To Kill a Mockingbird", "Harper Lee"));
        books.add(new Book("3", "1984", "George Orwell"));
    }

    public List<Book> findAll() {
        System.out.println("BookRepository: Fetching all books");
        return books;
    }

    public Book findById(String id) {
        System.out.println("BookRepository: Finding book by ID: " + id);
        return books.stream()
                .filter(book -> book.getId().equals(id))
                .findFirst()
                .orElse(null);
    }

    public void save(Book book) {
        System.out.println("BookRepository: Saving book: " + book.getTitle());
        books.add(book);
    }
}

// ==================== src/main/java/com/library/service/BookService.java ====================
package com.library.service;

import com.library.model.Book;
```

```
import com.library.repository.BookRepository;
import java.util.List;

public class BookService {
    private BookRepository bookRepository;

    // Setter for dependency injection
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public List<Book> getAllBooks() {
        System.out.println("BookService: Getting all books");
        return bookRepository.findAll();
    }

    public Book getBookById(String id) {
        System.out.println("BookService: Getting book by ID: " + id);
        return bookRepository.findById(id);
    }

    public void addBook(Book book) {
        System.out.println("BookService: Adding new book: " + book.getTitle());
        bookRepository.save(book);
    }

    public void displayAllBooks() {
        System.out.println("=== Library Books ===");
        List<Book> books = getAllBooks();
```

```
        books.forEach(System.out::println);
    }
}

// ==================== src/main/java/com/library/LibraryManagementApp.java ====================
package com.library;

import com.library.model.Book;
import com.library.service.BookService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {
    public static void main(String[] args) {
        System.out.println("Starting Library Management Application...");

        // Load Spring Application Context
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

        // Get BookService bean from Spring container
        BookService bookService = (BookService) context.getBean("bookService");

        // Test the application
        System.out.println("\n1. Displaying all books:");
        bookService.displayAllBooks();

        System.out.println("\n2. Finding book by ID:");
        Book book = bookService.getBookById("2");
```

```
        System.out.println("Found: " + book);

        System.out.println("\n3. Adding a new book:");
        Book newBook = new Book("4", "Brave New World", "Aldous Huxley");
        bookService.addBook(newBook);

        System.out.println("\n4. Displaying all books after addition:");
        bookService.displayAllBooks();

        System.out.println("\nApplication completed successfully!");

        // Close the context
        ((ClassPathXmlApplicationContext) context).close();
    }
}
```

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.

CODE:

```
// ==================== src/main/resources/applicationContext.xml ====================

<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- BookRepository Bean Definition -->
    <bean id="bookRepository" class="com.library.repository.BookRepository">
        <property name="databaseName" value="LibraryDB" />
        <property name="maxBooks" value="1000" />
    </bean>

    <!-- BookService Bean Definition with Dependency Injection -->
    <bean id="bookService" class="com.library.service.BookService">
        <!-- Setter-based Dependency Injection -->
        <property name="bookRepository" ref="bookRepository" />
        <property name="serviceName" value="Library Book Service" />
    </bean>

    <!-- Alternative: Constructor-based Dependency Injection -->
    <!--
    <bean id="bookServiceConstructor" class="com.library.service.BookService">
        <constructor-arg ref="bookRepository" />
        <constructor-arg value="Library Book Service" />
    </bean>
    -->

</beans>

// ==================== src/main/java/com/library/model/Book.java ====================
```

```
package com.library.model;

public class Book {
    private String id;
    private String title;
    private String author;
    private boolean isAvailable;

    public Book() {}

    public Book(String id, String title, String author) {
        this(id, title, author, true);
    }

    public Book(String id, String title, String author, boolean isAvailable) {
        this.id = id;
        this.title = title;
        this.author = author;
        this.isAvailable = isAvailable;
    }

    // Getters and Setters
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getAuthor() { return author; }
```

```
    public void setAuthor(String author) { this.author = author; }

    public boolean isAvailable() { return isAvailable; }
    public void setAvailable(boolean available) { isAvailable = available; }

    @Override
    public String toString() {
        return "Book{id='" + id + "', title='" + title + "', author='" + author +
                "', available=" + isAvailable + "}";
    }
}

// ==================== src/main/java/com/library/repository/BookRepository.java
====================
package com.library.repository;

import com.library.model.Book;
import java.util.ArrayList;
import java.util.List;

public class BookRepository {
    private List<Book> books;
    private String databaseName;
    private int maxBooks;

    public BookRepository() {
        this.books = new ArrayList<>();
        initializeData();
        System.out.println("BookRepository: Instance created");
```

```
}

private void initializeData() {
    books.add(new Book("1", "The Great Gatsby", "F. Scott Fitzgerald", true));
    books.add(new Book("2", "To Kill a Mockingbird", "Harper Lee", true));
    books.add(new Book("3", "1984", "George Orwell", false));
    books.add(new Book("4", "Pride and Prejudice", "Jane Austen", true));
    System.out.println("BookRepository: Sample data initialized");
}

// Setter methods for dependency injection
public void setDatabaseName(String databaseName) {
    this.databaseName = databaseName;
    System.out.println("BookRepository: Database name set to: " + databaseName);
}

public void setMaxBooks(int maxBooks) {
    this.maxBooks = maxBooks;
    System.out.println("BookRepository: Max books set to: " + maxBooks);
}

// Getter methods
public String getDatabaseName() { return databaseName; }
public int getMaxBooks() { return maxBooks; }

// Repository methods
public List<Book> findAll() {
    System.out.println("BookRepository: Fetching all books from " + databaseName);
    return new ArrayList<>(books);
```

```
}

public Book findById(String id) {
    System.out.println("BookRepository: Finding book by ID: " + id);
    return books.stream()
            .filter(book -> book.getId().equals(id))
            .findFirst()
            .orElse(null);
}

public void save(Book book) {
    if (books.size() >= maxBooks) {
        System.out.println("BookRepository: Cannot add book - maximum capacity reached");
        return;
    }
    System.out.println("BookRepository: Saving book: " + book.getTitle());
    books.add(book);
}

public boolean deleteById(String id) {
    System.out.println("BookRepository: Deleting book with ID: " + id);
    return books.removeIf(book -> book.getId().equals(id));
}

public List<Book> findAvailableBooks() {
    System.out.println("BookRepository: Finding available books");
    return books.stream()
            .filter(Book::isAvailable)
            .collect(ArrayList::new, ArrayList::add, ArrayList::addAll);
```

```
    }
}

// ==================== src/main/java/com/library/service/BookService.java ====================
package com.library.service;

import com.library.model.Book;
import com.library.repository.BookRepository;
import java.util.List;

public class BookService {
    private BookRepository bookRepository;
    private String serviceName;

    // Default constructor
    public BookService() {
        System.out.println("BookService: Instance created");
    }

    // Constructor for constructor-based dependency injection
    public BookService(BookRepository bookRepository, String serviceName) {
        this.bookRepository = bookRepository;
        this.serviceName = serviceName;
        System.out.println("BookService: Constructor injection completed");
    }

    // Setter method for dependency injection (REQUIRED for XML configuration)
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
```

```
    System.out.println("BookService: BookRepository dependency injected via setter");
}

public void setServiceName(String serviceName) {
    this.serviceName = serviceName;
    System.out.println("BookService: Service name set to: " + serviceName);
}

// Getter methods
public String getServiceName() { return serviceName; }

// Business methods
public List<Book> getAllBooks() {
    validateDependency();
    System.out.println("BookService: Getting all books");
    return bookRepository.findAll();
}

public Book getBookById(String id) {
    validateDependency();
    System.out.println("BookService: Getting book by ID: " + id);
    return bookRepository.findById(id);
}

public void addBook(Book book) {
    validateDependency();
    System.out.println("BookService: Adding new book: " + book.getTitle());
    bookRepository.save(book);
}
```

```
public boolean removeBook(String id) {
    validateDependency();
    System.out.println("BookService: Removing book with ID: " + id);
    return bookRepository.deleteById(id);
}

public List<Book> getAvailableBooks() {
    validateDependency();
    System.out.println("BookService: Getting available books");
    return bookRepository.findAvailableBooks();
}

public void displayAllBooks() {
    validateDependency();
    System.out.println("\n=== " + serviceName + " - All Books ===");
    List<Book> books = getAllBooks();
    if (books.isEmpty()) {
        System.out.println("No books available in the library.");
    } else {
        books.forEach(System.out::println);
    }
}

public void displayAvailableBooks() {
    validateDependency();
    System.out.println("\n=== Available Books ===");
    List<Book> availableBooks = getAvailableBooks();
    if (availableBooks.isEmpty()) {
```

```
            System.out.println("No books currently available.");
        } else {
            availableBooks.forEach(System.out::println);
        }
    }

    public void displayRepositoryInfo() {
        validateDependency();
        System.out.println("\n=== Repository Information ===");
        System.out.println("Database Name: " + bookRepository.getDatabaseName());
        System.out.println("Max Books Capacity: " + bookRepository.getMaxBooks());
        System.out.println("Current Books Count: " + bookRepository.findAll().size());
    }

    // Utility method to validate dependency injection
    private void validateDependency() {
        if (bookRepository == null) {
            throw new IllegalStateException("BookRepository dependency not injected! Check Spring
configuration.");
        }
    }
}

// ==================== src/main/java/com/library/LibraryManagementApplication.java
====================
package com.library;

import com.library.model.Book;
import com.library.service.BookService;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
    public static void main(String[] args) {
        System.out.println("=== Starting Library Management Application ===");
        System.out.println("Demonstrating Spring IoC and Dependency Injection\n");

        // Load Spring Application Context
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        System.out.println("Spring Context loaded successfully!\n");

        // Get BookService bean from Spring container
        BookService bookService = (BookService) context.getBean("bookService");

        // Test Dependency Injection
        testDependencyInjection(bookService);

        // Test Business Operations
        testBusinessOperations(bookService);

        System.out.println("\n=== Application completed successfully! ===");

        // Close the context
        ((ClassPathXmlApplicationContext) context).close();
    }

    private static void testDependencyInjection(BookService bookService) {
        System.out.println("=== Testing Dependency Injection ===");
```

```
    // Test if dependency is properly injected
    try {
        bookService.displayRepositoryInfo();
        System.out.println("✓ Dependency Injection Test PASSED");
    } catch (Exception e) {
        System.out.println("✗ Dependency Injection Test FAILED: " + e.getMessage());
        return;
    }

    // Display service information
    System.out.println("Service Name: " + bookService.getServiceName());
    System.out.println();
}

private static void testBusinessOperations(BookService bookService) {
    System.out.println("=== Testing Business Operations ===");

    // 1. Display all books
    bookService.displayAllBooks();

    // 2. Display available books
    bookService.displayAvailableBooks();

    // 3. Find specific book
    System.out.println("\n3. Finding book by ID '2':");
    Book book = bookService.getBookById("2");
    System.out.println("Found: " + book);
```

```
        // 4. Add a new book
        System.out.println("\n4. Adding a new book:");
        Book newBook = new Book("5", "Brave New World", "Aldous Huxley", true);
        bookService.addBook(newBook);

        // 5. Display all books after addition
        bookService.displayAllBooks();

        // 6. Remove a book
        System.out.println("\n6. Removing book with ID '3':");
        boolean removed = bookService.removeBook("3");
        System.out.println("Book removed: " + removed);

        // 7. Final display
        bookService.displayAllBooks();
    }
}
```

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.

// ==================== pom.xml ====================

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.library</groupId>
    <artifactId>LibraryManagement</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <spring.version>5.3.21</spring.version>
    </properties>

    <dependencies>
        <!-- Spring Core Dependencies -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>${spring.version}</version>
</dependency>

<!-- Spring AOP Dependencies -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.7</version>
</dependency>

<!-- Logging Dependencies -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.36</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
```

```
        <artifactId>logback-classic</artifactId>
        <version>1.2.11</version>
      </dependency>
    </dependencies>
</project>

// ==================== src/main/resources/applicationContext.xml ====================
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- Enable AspectJ Auto Proxy -->
    <aop:aspectj-autoproxy />

    <!-- BookRepository Bean -->
    <bean id="bookRepository" class="com.library.repository.BookRepository">
        <property name="databaseName" value="LibraryDB" />
        <property name="maxBooks" value="1000" />
    </bean>

    <!-- BookService Bean -->
    <bean id="bookService" class="com.library.service.BookService">
        <property name="bookRepository" ref="bookRepository" />
        <property name="serviceName" value="Library Book Service" />
```

```
    </bean>

    <!-- Logging Aspect Bean -->
    <bean id="loggingAspect" class="com.library.aspect.LoggingAspect" />

</beans>

// ==================== src/main/java/com/library/aspect/LoggingAspect.java ====================
package com.library.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LoggingAspect {

    private static final Logger logger = LoggerFactory.getLogger(LoggingAspect.class);

    // Pointcut for all methods in service package
    @Pointcut("execution(* com.library.service.*.*(..))")
    public void serviceLayer() {}

    // Pointcut for all methods in repository package
```

```
@Pointcut("execution(* com.library.repository.*.*(..))")
public void repositoryLayer() {}

// Combined pointcut for both service and repository layers
@Pointcut("serviceLayer() || repositoryLayer()")
public void applicationLayer() {}

// Before advice - executed before method execution
@Before("applicationLayer()")
public void logMethodStart(JoinPoint joinPoint) {
    String className = joinPoint.getTarget().getClass().getSimpleName();
    String methodName = joinPoint.getSignature().getName();
    Object[] args = joinPoint.getArgs();

    logger.info("🚀 [START] {}.{}() - Arguments: {}", className, methodName,
            args.length > 0 ? java.util.Arrays.toString(args) : "none");
}

// After advice - executed after method execution (success or failure)
@After("applicationLayer()")
public void logMethodEnd(JoinPoint joinPoint) {
    String className = joinPoint.getTarget().getClass().getSimpleName();
    String methodName = joinPoint.getSignature().getName();

    logger.info("🏁 [END] {}.{}()", className, methodName);
}

// After returning advice - executed after successful method execution
@AfterReturning(pointcut = "applicationLayer()", returning = "result")
```

```
public void logMethodSuccess(JoinPoint joinPoint, Object result) {
    String className = joinPoint.getTarget().getClass().getSimpleName();
    String methodName = joinPoint.getSignature().getName();

    String resultInfo = result != null ? result.toString() : "void";
    if (result instanceof java.util.Collection) {
        resultInfo = "Collection[" + ((java.util.Collection<?>) result).size() + " items]";
    }

    logger.info("✅ [SUCCESS] {}.{}() - Result: {}", className, methodName, resultInfo);
}

// After throwing advice - executed when method throws exception
@AfterThrowing(pointcut = "applicationLayer()", throwing = "exception")
public void logMethodException(JoinPoint joinPoint, Throwable exception) {
    String className = joinPoint.getTarget().getClass().getSimpleName();
    String methodName = joinPoint.getSignature().getName();

    logger.error("❌ [ERROR] {}.{}() - Exception: {}", className, methodName,
            exception.getMessage());
}

// Around advice - wraps method execution for performance monitoring
@Around("applicationLayer()")
public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {
    String className = joinPoint.getTarget().getClass().getSimpleName();
    String methodName = joinPoint.getSignature().getName();

    long startTime = System.currentTimeMillis();
```

```
logger.info("⏱️ [TIMING] {}.{}() - Started", className, methodName);

try {
    // Proceed with method execution
    Object result = joinPoint.proceed();

    long endTime = System.currentTimeMillis();
    long executionTime = endTime - startTime;

    logger.info("⏱️ [TIMING] {}.{}() - Completed in {} ms",
            className, methodName, executionTime);

    // Log performance warning for slow methods
    if (executionTime > 100) {
        logger.warn("⚠️ [PERFORMANCE] {}.{}() - Slow execution: {} ms",
                className, methodName, executionTime);
    }

    return result;

} catch (Throwable throwable) {
    long endTime = System.currentTimeMillis();
    long executionTime = endTime - startTime;

    logger.error("⏱️ [TIMING] {}.{}() - Failed after {} ms",
            className, methodName, executionTime);
    throw throwable;
}
```

```
    }

    // Specific advice for critical operations
    @Before("execution(* com.library.service.BookService.addBook(..)) || " +
            "execution(* com.library.service.BookService.removeBook(..))")
    public void logCriticalOperation(JoinPoint joinPoint) {
        String methodName = joinPoint.getSignature().getName();
        Object[] args = joinPoint.getArgs();

        logger.warn("🔒 [CRITICAL] {} operation initiated - Args: {}",
                methodName, java.util.Arrays.toString(args));
    }

    // Pointcut for getter methods (for demonstration)
    @Pointcut("execution(* get*(..))")
    public void getterMethods() {}

    // Light logging for getter methods
    @Around("getterMethods() && applicationLayer()")
    public Object logGetterMethods(ProceedingJoinPoint joinPoint) throws Throwable {
        String methodName = joinPoint.getSignature().getName();
        Object result = joinPoint.proceed();

        logger.debug("📖 [GETTER] {} - Returned: {}", methodName,
                result != null ? result.toString() : "null");

        return result;
    }
}
```

```
// ==================== src/main/java/com/library/model/Book.java ====================
package com.library.model;

public class Book {
    private String id;
    private String title;
    private String author;
    private boolean isAvailable;

    public Book() {}

    public Book(String id, String title, String author) {
        this(id, title, author, true);
    }

    public Book(String id, String title, String author, boolean isAvailable) {
        this.id = id;
        this.title = title;
        this.author = author;
        this.isAvailable = isAvailable;
    }

    // Getters and Setters
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
```

```
    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    public boolean isAvailable() { return isAvailable; }
    public void setAvailable(boolean available) { isAvailable = available; }

    @Override
    public String toString() {
        return "Book{id='" + id + "', title='" + title + "', author='" + author +
                "', available=" + isAvailable + "}";
    }
}

// ==================== src/main/java/com/library/repository/BookRepository.java ====================
package com.library.repository;

import com.library.model.Book;
import java.util.ArrayList;
import java.util.List;

public class BookRepository {
    private List<Book> books;
    private String databaseName;
    private int maxBooks;

    public BookRepository() {
        this.books = new ArrayList<>();
```

```
    initializeData();
}

private void initializeData() {
    // Simulate some processing time
    try {
        Thread.sleep(50);
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }

    books.add(new Book("1", "The Great Gatsby", "F. Scott Fitzgerald", true));
    books.add(new Book("2", "To Kill a Mockingbird", "Harper Lee", true));
    books.add(new Book("3", "1984", "George Orwell", false));
    books.add(new Book("4", "Pride and Prejudice", "Jane Austen", true));
}

// Setter methods for dependency injection
public void setDatabaseName(String databaseName) {
    this.databaseName = databaseName;
}

public void setMaxBooks(int maxBooks) {
    this.maxBooks = maxBooks;
}

// Getter methods
public String getDatabaseName() { return databaseName; }
public int getMaxBooks() { return maxBooks; }
```

```
// Repository methods
public List<Book> findAll() {
   // Simulate database query time
   simulateProcessingTime(30);
   return new ArrayList<>(books);
}

public Book findById(String id) {
   // Simulate database lookup time
   simulateProcessingTime(20);
   return books.stream()
         .filter(book -> book.getId().equals(id))
         .findFirst()
         .orElse(null);
}

public void save(Book book) {
   // Simulate database save time
   simulateProcessingTime(40);
   if (books.size() >= maxBooks) {
      throw new RuntimeException("Maximum book capacity reached");
   }
   books.add(book);
}

public boolean deleteById(String id) {
   // Simulate database delete time
   simulateProcessingTime(35);
```

```
        return books.removeIf(book -> book.getId().equals(id));
    }

    public List<Book> findAvailableBooks() {
        // Simulate complex query time
        simulateProcessingTime(60);
        return books.stream()
                .filter(Book::isAvailable)
                .collect(ArrayList::new, ArrayList::add, ArrayList::addAll);
    }

    private void simulateProcessingTime(long millis) {
        try {
            Thread.sleep(millis);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

// ==================== src/main/java/com/library/service/BookService.java ====================
package com.library.service;

import com.library.model.Book;
import com.library.repository.BookRepository;
import java.util.List;

public class BookService {
    private BookRepository bookRepository;
```

```
private String serviceName;

public BookService() {}

// Setter methods for dependency injection
public void setBookRepository(BookRepository bookRepository) {
    this.bookRepository = bookRepository;
}

public void setServiceName(String serviceName) {
    this.serviceName = serviceName;
}

// Getter methods
public String getServiceName() { return serviceName; }
public BookRepository getBookRepository() { return bookRepository; }

// Business methods
public List<Book> getAllBooks() {
    validateDependency();
    return bookRepository.findAll();
}

public Book getBookById(String id) {
    validateDependency();
    if (id == null || id.trim().isEmpty()) {
        throw new IllegalArgumentException("Book ID cannot be null or empty");
    }
    return bookRepository.findById(id);
```

```
}

public void addBook(Book book) {
    validateDependency();
    if (book == null) {
        throw new IllegalArgumentException("Book cannot be null");
    }
    bookRepository.save(book);
}

public boolean removeBook(String id) {
    validateDependency();
    if (id == null || id.trim().isEmpty()) {
        throw new IllegalArgumentException("Book ID cannot be null or empty");
    }
    return bookRepository.deleteById(id);
}

public List<Book> getAvailableBooks() {
    validateDependency();
    return bookRepository.findAvailableBooks();
}

public void displayAllBooks() {
    List<Book> books = getAllBooks();
    System.out.println("\n=== " + serviceName + " - All Books ===");
    books.forEach(System.out::println);
}
```

```
    public void displayAvailableBooks() {
        List<Book> availableBooks = getAvailableBooks();
        System.out.println("\n=== Available Books ===");
        availableBooks.forEach(System.out::println);
    }

    private void validateDependency() {
        if (bookRepository == null) {
            throw new IllegalStateException("BookRepository dependency not injected!");
        }
    }
}

// ==================== src/main/java/com/library/LibraryManagementApplication.java ====================
package com.library;

import com.library.model.Book;
import com.library.service.BookService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
    private static final Logger logger = LoggerFactory.getLogger(LibraryManagementApplication.class);

    public static void main(String[] args) {
        logger.info(" 🚀 Starting Library Management Application with AOP Logging");
```

```
    // Load Spring Application Context
    ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
    logger.info(" 📦 Spring Context loaded successfully");

    // Get BookService bean (proxy will be created by AOP)
    BookService bookService = (BookService) context.getBean("bookService");
    logger.info(" 📚 BookService bean retrieved from Spring context");

    // Test AOP logging with various operations
    testAopLogging(bookService);

    logger.info(" 🎉 Application completed successfully");

    // Close the context
    ((ClassPathXmlApplicationContext) context).close();
}

private static void testAopLogging(BookService bookService) {
    logger.info("=== Testing AOP Logging Functionality ===");

    try {
        // Test 1: Display all books (will trigger multiple aspects)
        logger.info("--- Test 1: Display All Books ---");
        bookService.displayAllBooks();

        // Test 2: Get specific book by ID
        logger.info("--- Test 2: Get Book by ID ---");
        Book book = bookService.getBookById("2");
```

```
logger.info("Retrieved book: {}", book);

// Test 3: Get available books (complex query)
logger.info("--- Test 3: Get Available Books ---");
bookService.displayAvailableBooks();

// Test 4: Add new book (critical operation)
logger.info("--- Test 4: Add New Book ---");
Book newBook = new Book("5", "Brave New World", "Aldous Huxley", true);
bookService.addBook(newBook);
logger.info("Book added successfully");

// Test 5: Remove book (critical operation)
logger.info("--- Test 5: Remove Book ---");
boolean removed = bookService.removeBook("3");
logger.info("Book removal result: {}", removed);

// Test 6: Display updated books
logger.info("--- Test 6: Display Updated Books ---");
bookService.displayAllBooks();

// Test 7: Error handling (invalid ID)
logger.info("--- Test 7: Error Handling ---");
try {
   bookService.getBookById("");
} catch (IllegalArgumentException e) {
   logger.error("Expected error caught: {}", e.getMessage());
}
```

```
        // Test 8: Getter methods
        logger.info("--- Test 8: Getter Methods ---");
        String serviceName = bookService.getServiceName();
        logger.info("Service name: {}", serviceName);

    } catch (Exception e) {
        logger.error("Error during AOP testing: {}", e.getMessage(), e);
    }
  }
}
```

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.

CODE:

```
// ==================== Project Structure ====================
/*
LibraryManagement/
 ├── pom.xml
 ├── src/
|   ├── main/
```

```
|   |   ├── java/
|   |   |   └── com/
|   |   |       └── library/
|   |   |           ├── LibraryManagementApplication.java
|   |   |           ├── config/
|   |   |           |   └── WebConfig.java
|   |   |           ├── controller/
|   |   |           |   └── BookController.java
|   |   |           ├── service/
|   |   |           |   └── BookService.java
|   |   |           ├── repository/
|   |   |           |   └── BookRepository.java
|   |   |           ├── model/
|   |   |           |   └── Book.java
|   |   |           └── aspect/
|   |   |               └── LoggingAspect.java
|   |   ├── resources/
|   |   |   ├── applicationContext.xml
|   |   |   ├── logback.xml
|   |   |   └── static/
|   |   |       └── css/
|   |   |           └── style.css
|   |   └── webapp/
|   |       ├── WEB-INF/
|   |       |   ├── web.xml
|   |       |   └── views/
|   |       |       └── books.jsp
|   |       └── index.html
```

```
|   └── test/
|       └── java/
|           └── com/
|               └── library/
|                   └── LibraryManagementApplicationTest.java
 ├── target/
└── README.md
*/

// ==================== pom.xml ====================
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- Project Information -->
    <groupId>com.library</groupId>
    <artifactId>LibraryManagement</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>LibraryManagement</name>
    <description>Library Management System using Spring Framework</description>
    <url>http://www.library.com</url>

    <!-- Properties -->
    <properties>
```

```
    <java.version>1.8</java.version>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <!-- Spring Framework Version -->
    <spring.version>5.3.21</spring.version>

    <!-- Other Dependencies Versions -->
    <aspectj.version>1.9.7</aspectj.version>
    <servlet.version>4.0.1</servlet.version>
    <jsp.version>2.3.3</jsp.version>
    <jstl.version>1.2</jstl.version>
    <junit.version>4.13.2</junit.version>
    <slf4j.version>1.7.36</slf4j.version>
    <logback.version>1.2.11</logback.version>

    <!-- Maven Plugin Versions -->
    <maven.compiler.plugin.version>3.8.1</maven.compiler.plugin.version>
    <maven.war.plugin.version>3.2.3</maven.war.plugin.version>
    <maven.surefire.plugin.version>2.22.2</maven.surefire.plugin.version>
    <maven.failsafe.plugin.version>2.22.2</maven.failsafe.plugin.version>
    <tomcat.maven.plugin.version>2.2</tomcat.maven.plugin.version>
</properties>

<!-- Dependencies -->
<dependencies>
```

```
<!-- Spring Context (Core IoC Container) -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>

<!-- Spring AOP (Aspect-Oriented Programming) -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${spring.version}</version>
</dependency>

<!-- Spring WebMVC (Model-View-Controller) -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
</dependency>

<!-- Additional Spring Dependencies -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
```

```
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-expression</artifactId>
    <version>${spring.version}</version>
</dependency>

<!-- AspectJ Weaver for AOP -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${aspectj.version}</version>
</dependency>

<!-- Servlet API -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>${servlet.version}</version>
```

```
    <scope>provided</scope>
</dependency>

<!-- JSP API -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>${jsp.version}</version>
    <scope>provided</scope>
</dependency>

<!-- JSTL -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>${jstl.version}</version>
</dependency>

<!-- Jackson for JSON processing -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.3</version>
</dependency>

<!-- Logging Dependencies -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
```

```
    <version>${slf4j.version}</version>
</dependency>

<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
</dependency>

<!-- Test Dependencies -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring.version}</version>
    <scope>test</scope>
</dependency>

<!-- Mock dependencies for testing -->
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>3.12.4</version>
```

```
        <scope>test</scope>
    </dependency>

</dependencies>

<!-- Build Configuration -->
<build>
    <finalName>LibraryManagement</finalName>

    <plugins>

        <!-- Maven Compiler Plugin (Java 1.8) -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>${maven.compiler.plugin.version}</version>
            <configuration>
                <source>${java.version}</source>
                <target>${java.version}</target>
                <encoding>${project.build.sourceEncoding}</encoding>
                <showWarnings>true</showWarnings>
                <showDeprecation>true</showDeprecation>
            </configuration>
        </plugin>

        <!-- Maven WAR Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
```

```
    <version>${maven.war.plugin.version}</version>
    <configuration>
      <warSourceDirectory>src/main/webapp</warSourceDirectory>
      <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
  </plugin>

  <!-- Maven Surefire Plugin for Unit Tests -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>${maven.surefire.plugin.version}</version>
    <configuration>
      <skipTests>false</skipTests>
      <testFailureIgnore>false</testFailureIgnore>
    </configuration>
  </plugin>

  <!-- Maven Failsafe Plugin for Integration Tests -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-failsafe-plugin</artifactId>
    <version>${maven.failsafe.plugin.version}</version>
    <executions>
      <execution>
        <goals>
          <goal>integration-test</goal>
          <goal>verify</goal>
        </goals>
```

```
      </execution>
    </executions>
  </plugin>

  <!-- Tomcat Maven Plugin for Development -->
  <plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>${tomcat.maven.plugin.version}</version>
    <configuration>
      <path>/library</path>
      <port>8080</port>
    </configuration>
  </plugin>

  <!-- Maven Resources Plugin -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.2.0</version>
    <configuration>
      <encoding>${project.build.sourceEncoding}</encoding>
    </configuration>
  </plugin>

  <!-- Maven Clean Plugin -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-clean-plugin</artifactId>
```

```
            <version>3.1.0</version>
        </plugin>

        <!-- Maven Install Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-install-plugin</artifactId>
            <version>2.5.2</version>
        </plugin>

        <!-- Maven Deploy Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-deploy-plugin</artifactId>
            <version>2.8.2</version>
        </plugin>

    </plugins>
</build>

<!-- Maven Repositories -->
<repositories>
    <repository>
        <id>central</id>
        <name>Maven Central Repository</name>
        <url>https://repo1.maven.org/maven2/</url>
    </repository>
    <repository>
        <id>spring-releases</id>
```

```
        <name>Spring Releases</name>
        <url>https://repo.spring.io/release</url>
    </repository>
</repositories>

<!-- Plugin Repositories -->
<pluginRepositories>
    <pluginRepository>
        <id>central</id>
        <name>Maven Plugin Repository</name>
        <url>https://repo1.maven.org/maven2/</url>
    </pluginRepository>
</pluginRepositories>

<!-- Profiles -->
<profiles>
    <!-- Development Profile -->
    <profile>
        <id>dev</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <properties>
            <log.level>DEBUG</log.level>
            <database.url>jdbc:h2:mem:testdb</database.url>
        </properties>
    </profile>

    <!-- Production Profile -->
```

```
    <profile>
      <id>prod</id>
      <properties>
        <log.level>INFO</log.level>
        <database.url>jdbc:mysql://localhost:3306/library</database.url>
      </properties>
    </profile>

    <!-- Test Profile -->
    <profile>
      <id>test</id>
      <properties>
        <log.level>DEBUG</log.level>
        <database.url>jdbc:h2:mem:testdb</database.url>
      </properties>
    </profile>
  </profiles>

  <!-- Reporting -->
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>3.1.2</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-report-plugin</artifactId>
```

```
            <version>${maven.surefire.plugin.version}</version>
          </plugin>
        </plugins>
    </reporting>

</project>

// ==================== README.md ====================
# Library Management System

A comprehensive library management application built with Spring Framework and Maven.

## Features
- Spring Context for IoC and DI
- Spring AOP for cross-cutting concerns
- Spring WebMVC for web layer
- Maven build system
- JUnit testing support

## Technologies Used
- Java 1.8
- Spring Framework 5.3.21
- Maven 3.x
- JUnit 4.13.2
- Logback for logging

## Project Structure
```
LibraryManagement/
```
```

```
├── pom.xml
├── src/main/java/       # Java source files
├── src/main/resources/  # Configuration files
├── src/main/webapp/     # Web resources
└── src/test/java/       # Test files
```

Build Commands

```bash
# Compile the project
mvn clean compile

# Run tests
mvn test

# Package as WAR
mvn clean package

# Run with Tomcat
mvn tomcat7:run
```

Maven Goals

- `mvn clean` - Clean target directory
- `mvn compile` - Compile source code
- `mvn test` - Run unit tests
- `mvn package` - Create WAR file
- `mvn install` - Install to local repository
- `mvn deploy` - Deploy to remote repository

Profiles

- `dev` - Development profile (default)

- `prod` - Production profile

- `test` - Test profile

Use: `mvn clean package -Pprod`

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.

CODE:

```
// --- File: BookRepository.java ---

package com.example.library;


public class BookRepository {

   @Override

   public String toString() {

      return "BookRepository instance";

   }

}
```

```
// --- File: BookService.java ---
package com.example.library;

public class BookService {
    private BookRepository bookRepository;

    // Setter injection
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public void display() {
        System.out.println("BookService working with: " + bookRepository);
    }
}

// --- File: MainApp.java ---
package com.example.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        BookService bookService = (BookService) context.getBean("bookService");
        bookService.display();
    }
```

```
}

/*
--- File: src/main/resources/applicationContext.xml ---

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="bookRepository" class="com.example.library.BookRepository"/>
   <bean id="bookService" class="com.example.library.BookService">
      <property name="bookRepository" ref="bookRepository"/>
   </bean>

</beans>
*/
```

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.

Code:

```
// --- File: BookRepository.java ---
package com.library;

import org.springframework.stereotype.Repository;

@Repository
public class BookRepository {
   @Override
   public String toString() {
      return "BookRepository instance";
   }
}

// --- File: BookService.java ---
package com.library;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class BookService {

   private BookRepository bookRepository;

   @Autowired // Constructor or setter injection works too
```

```
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public void display() {
        System.out.println("BookService working with: " + bookRepository);
    }
}

// --- File: LibraryManagementApplication.java ---
package com.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        BookService bookService = context.getBean(BookService.class);
        bookService.display();
    }
}

/*
--- File: src/main/resources/applicationContext.xml ---

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd">

   <!-- Enable component scanning for the com.library package -->
   <context:component-scan base-package="com.library"/>

</beans>
*/
```

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.

Code:

```
// --- File: BookRepository.java ---
package com.library;
```

```
public class BookRepository {
    @Override
    public String toString() {
        return "BookRepository instance";
    }
}

// --- File: BookService.java ---
package com.library;

public class BookService {

    private String serviceName;        // Injected by constructor
    private BookRepository bookRepository; // Injected by setter

    // Constructor injection
    public BookService(String serviceName) {
        this.serviceName = serviceName;
    }

    // Setter injection
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public void display() {
        System.out.println("Service Name: " + serviceName);
        System.out.println("BookRepository: " + bookRepository);
    }
```

```
}

// --- File: LibraryManagementApplication.java ---
package com.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        BookService bookService = (BookService) context.getBean("bookService");
        bookService.display();
    }
}

/*
--- File: src/main/resources/applicationContext.xml ---

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- BookRepository Bean -->
    <bean id="bookRepository" class="com.library.BookRepository"/>
```

```
    <!-- BookService Bean with constructor and setter injection -->
    <bean id="bookService" class="com.library.BookService">
        <!-- Constructor injection -->
        <constructor-arg value="Central Library Service"/>
        <!-- Setter injection -->
        <property name="bookRepository" ref="bookRepository"/>
    </bean>

</beans>
*/
```

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.

Code:

```
// --- File: BookRepository.java ---
package com.library;

public class BookRepository {
    public void saveBook() {
```

```
        System.out.println("BookRepository: Saving book to database.");
    }
}

// --- File: BookService.java ---
package com.library;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class BookService {

    @Autowired
    private BookRepository bookRepository;

    public void addBook() {
        System.out.println("BookService: Adding book.");
        bookRepository.saveBook();
    }
}

// --- File: LoggingAspect.java ---
package com.library.aspect;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
```

```
@Aspect
public class LoggingAspect {

   @Before("execution(* com.library.BookService.*(..))")
   public void beforeAdvice() {
      System.out.println("LoggingAspect: Before method execution.");
   }

   @After("execution(* com.library.BookService.*(..))")
   public void afterAdvice() {
      System.out.println("LoggingAspect: After method execution.");
   }
}

// --- File: LibraryManagementApplication.java ---
package com.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
      BookService bookService = context.getBean(BookService.class);
      bookService.addBook();
   }
}

/*
```

--- File: src/main/resources/applicationContext.xml ---

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd
         http://www.springframework.org/schema/aop
         http://www.springframework.org/schema/aop/spring-aop.xsd">

   <!-- Enable component scanning for BookService -->
   <context:component-scan base-package="com.library"/>

   <!-- Register the Aspect bean -->
   <bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>

   <!-- Enable AspectJ auto-proxying -->
   <aop:aspectj-autoproxy/>

</beans>
*/
```

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.

Code:

```
// --- File: src/main/java/com/example/librarymanagement/LibraryManagementApplication.java ---

package com.example.librarymanagement;


import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication

public class LibraryManagementApplication {

   public static void main(String[] args) {

      SpringApplication.run(LibraryManagementApplication.class, args);

   }

}
```

```
// --- File: src/main/java/com/example/librarymanagement/entity/Book.java ---
package com.example.librarymanagement.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String author;

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }
}

// --- File: src/main/java/com/example/librarymanagement/repository/BookRepository.java ---
```

```
package com.example.librarymanagement.repository;

import com.example.librarymanagement.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
}

// --- File: src/main/java/com/example/librarymanagement/controller/BookController.java ---
package com.example.librarymanagement.controller;

import com.example.librarymanagement.entity.Book;
import com.example.librarymanagement.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/books")
public class BookController {

    @Autowired
    private BookRepository bookRepository;

    @GetMapping
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }
```

```
    @PostMapping
    public Book addBook(@RequestBody Book book) {
        return bookRepository.save(book);
    }

    @GetMapping("/{id}")
    public Book getBookById(@PathVariable Long id) {
        return bookRepository.findById(id).orElse(null);
    }

    @PutMapping("/{id}")
    public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {
        return bookRepository.findById(id)
                .map(book -> {
                    book.setTitle(updatedBook.getTitle());
                    book.setAuthor(updatedBook.getAuthor());
                    return bookRepository.save(book);
                })
                .orElse(null);
    }

    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable Long id) {
        bookRepository.deleteById(id);
    }
}

/*
```

--- File: src/main/resources/application.properties ---

spring.datasource.url=jdbc:h2:mem:librarydb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

spring.jpa.hibernate.ddl-auto=update

*/