Exercise 1: Configuring a Basic Spring Application

Step 1: Set Up a Spring Project

1. **Create a Maven project named LibraryManagement**:
    - Use your IDE to create a new Maven project named LibraryManagement.
2. **Add Spring Core dependencies in the pom.xml file**:

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.21</version>
  </dependency>
</dependencies>
```

Step 2: Configure the Application Context

1. **Create an XML configuration file named applicationContext.xml in the src/main/resources directory**:

```
<?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">

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

  <!-- Define BookService bean and inject BookRepository -->
  <bean id="bookService" class="com.library.service.BookService">
    <property name="bookRepository" ref="bookRepository"/>
  </bean>
</beans>
```

Step 3: Define Service and Repository Classes

1. **Create a package com.library.service and add a class BookService**:

```
package com.library.service;
import com.library.repository.BookRepository;
public class BookService {
   private BookRepository bookRepository;
   public void setBookRepository(BookRepository bookRepository) {
      this.bookRepository = bookRepository;
   }
   // Add service methods here
}
```

2. **Create a package com.library.repository and add a class BookRepository**:

```
package com.library.repository;
public class BookRepository {
   // Add repository methods here
}
```

Step 4: Run the Application

1. **Create a main class to load the Spring context and test the configuration**:

```
package com.library;
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) {
      ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

      BookService bookService = context.getBean("bookService", BookService.class);
      // Test the bookService methods here
   }
}
```

Exercise 3: Implementing Logging with Spring AOP

Step 1: Add Spring AOP Dependency

1. **Update pom.xml to include Spring AOP dependency**:

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.21</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>5.3.21</version>
  </dependency>
</dependencies>
```

Step 2: Create an Aspect for Logging

1. **Create a package com.library.aspect and add a class LoggingAspect**:

```
package com.library.aspect;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;
@Aspect
@Component
public class LoggingAspect {
  @Around("execution(* com.library.service.*.*(..))")
  public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {
    long start = System.currentTimeMillis();
    Object proceed = joinPoint.proceed();
    long executionTime = System.currentTimeMillis() - start;
    System.out.println(joinPoint.getSignature() + " executed in " + executionTime + "ms");
    return proceed;
  }
```

}

Step 3: Enable AspectJ Support

1. **Update applicationContext.xml to enable AspectJ support and register the aspect**:

```
<?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 support -->
  <aop:aspectj-autoproxy/>
  <!-- Define BookRepository bean -->
  <bean id="bookRepository" class="com.library.repository.BookRepository"/>
  <!-- Define BookService bean and inject BookRepository -->
  <bean id="bookService" class="com.library.service.BookService">
    <property name="bookRepository" ref="bookRepository"/>
  </bean>
  <!-- Register LoggingAspect -->
  <bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>
</beans>
```

Exercise 4: Creating and Configuring a Maven Project

Step 1: Create a New Maven Project

1. **Create a new Maven project named LibraryManagement**: (Already done in Exercise 1)

Step 2: Add Spring Dependencies in pom.xml

1. **Include dependencies for Spring Context, Spring AOP, and Spring WebMVC**:

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
```

```
    <version>5.3.21</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>5.3.21</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.21</version>
  </dependency>
</dependencies>
```

Step 3: Configure Maven Plugins

1. **Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file**:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Exercise 5: Configuring the Spring IoC Container

Step 1: Create Spring Configuration File

1. **Create an XML configuration file named applicationContext.xml in the src/main/resources directory**: (Already done in Exercise 1)

Step 2: Update the BookService Class

1. **Ensure that the BookService class has a setter method for BookRepository**: (Already done in Exercise 1)

Step 3: Run the Application

1. **Create a main class to load the Spring context and test the configuration**: (Already done in Exercise 1)

Exercise 6: Configuring Beans with Annotations

Step 1: Enable Component Scanning

1. **Update applicationContext.xml to include component scanning for the com.library package**:

```
<?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:context="http://www.springframework.org/schema/context"
    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 -->
  <context:component-scan base-package="com.library"/>
  <!-- Enable AspectJ support -->
  <aop:aspectj-autoproxy/>
</beans>
```

Step 2: Annotate Classes

1. **Use @Service annotation for the BookService class**:

```
package com.library.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.library.repository.BookRepository;
@Service
```

```
public class BookService {
    private BookRepository bookRepository;
    @Autowired
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
    // Add service methods here
}
```

2. **Use @Repository annotation for the BookRepository class**:

```
package com.library.repository;
import org.springframework.stereotype.Repository;
@Repository
public class BookRepository {
    // Add repository methods here
}
```

Step 3: Test the Configuration

1. **Run the LibraryManagementApplication main class to verify the annotation-based configuration**: (No changes needed from Exercise 1

Exercise 7: Implementing Constructor and Setter Injection

Step 1: Configure Constructor Injection

1. **Update applicationContext.xml to configure constructor injection for BookService**:

```
<?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">
    <!-- Define BookRepository bean -->
    <bean id="bookRepository" class="com.library.repository.BookRepository"/>
    <!-- Define BookService bean with constructor injection -->
    <bean id="bookService" class="com.library.service.BookService">
        <constructor-arg ref="bookRepository"/>
```

```
    </bean>
</beans>
```

2. **Update the BookService class to use constructor injection**:

```
package com.library.service;
import com.library.repository.BookRepository;
public class BookService {
    private final BookRepository bookRepository;
    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
}
```

Step 2: Configure Setter Injection

1. **Ensure that the BookService class has a setter method for BookRepository**: (Already done in Exercise 1)

Step 3: Test the Injection

1. **Run the LibraryManagementApplication main class to verify both constructor and setter injection**: (No changes needed from Exercise 1)

Exercise 8: Implementing Basic AOP with Spring

Step 1: Define an Aspect

1. **Create a package com.library.aspect and add a class LoggingAspect**: (Already done in Exercise 3)

Step 2: Create Advice Methods

1. **Define advice methods in LoggingAspect for logging before and after method execution**:

```
package com.library.aspect;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;
@Aspect
@Component
public class LoggingAspect {
```

```
    @Before("execution(* com.library.service.*.*(..))")
    public void logBefore() {
        System.out.println("Method execution started");
    }
    @After("execution(* com.library.service.*.*(..))")
    public void logAfter() {
        System.out.println("Method execution finished");
    }
}
```

Step 3: Configure the Aspect

1. **Update applicationContext.xml to register the aspect and enable AspectJ auto-proxying**: (Already done in Exercise 3)

Step 4: Test the Aspect

1. **Run the LibraryManagementApplication main class to verify the AOP functionality**: (No changes needed from Exercise 1)

Exercise 9: Creating a Spring Boot Application

Step 1: Create a Spring Boot Project

1. **Use Spring Initializr to create a new Spring Boot project named LibraryManagement**:
    - Go to Spring Initializr.
    - Set the project name to LibraryManagement.
    - Select dependencies: Spring Web, Spring Data JPA, H2 Database.
    - Generate the project and unzip it.

Step 2: Add Dependencies

1. **Include dependencies for Spring Web, Spring Data JPA, and H2 Database**: (Already done by Spring Initializr)

Step 3: Create Application Properties

1. **Configure database connection properties in application.properties**:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

```
spring.h2.console.enabled=true
```

Step 4: Define Entities and Repositories

1. **Create Book entity**:

```
package com.library.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Book {

   @Id

   @GeneratedValue(strategy = GenerationType.IDENTITY)

   private Long id;

   private String title;

   private String author;


   // Getters and setters

}
```

2. **Create BookRepository interface**:

```
package com.library.repository;

import com.library.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;


@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

}
```

Step 5: Create a REST Controller

1. **Create BookController class to handle CRUD operations**:

```
package com.library.controller;

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

```
import com.library.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 createBook(@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 bookDetails) {
        Book book = bookRepository.findById(id).orElse(null);
        if (book != null) {
            book.setTitle(bookDetails.getTitle());
            book.setAuthor(bookDetails.getAuthor());
            return bookRepository.save(book);
        }
        return null;
    }
```

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

Step 6: Run the Application

1. **Run the Spring Boot application and test the REST endpoints**:
    - Run the main application class generated by Spring Initializr.
    - Use a tool like Postman or curl to test the CRUD operations at http://localhost:8080/books.

This setup covers all the exercises for configuring and implementing various features in a Spring or Spring Boot application for managing a library.