Spring Core Maven Exercises - Solutions

# Exercise 1: Configuring a Basic Spring Application

1. Set Up a Spring Project:  
- Create a Maven project named LibraryManagement.  
- In pom.xml, add dependencies:

xml  
<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.9</version>  
 </dependency>  
</dependencies>  
  
  
2. Configure the Application Context:  
- src/main/resources/applicationContext.xml:

xml  
<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>  
  
  
3. Define Service and Repository Classes:

java  
package com.library.service;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void issueBook() {  
 System.out.println("Issuing book...");  
 bookRepository.save();  
 }  
}

java:  
package com.library.repository;  
  
public class BookRepository {  
 public void save() {  
 System.out.println("Saving book to database...");  
 }  
}  
  
  
4. Run the Application:

java:  
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 service = context.getBean("bookService", BookService.class);  
 service.issueBook();  
 }  
}

# Exercise 2: Implementing Dependency Injection

1. Modify the XML Configuration:

xml  
<bean id="bookRepository" class="com.library.repository.BookRepository"/>  
<bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
</bean>  
  
  
2. Update the BookService Class:

java  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void issueBook() {  
 System.out.println("Issuing book...");  
 bookRepository.save();  
 }  
}  
  
  
3. Test the Configuration: Run the same `LibraryManagementApplication` main class.

# Exercise 3: Implementing Logging with Spring AOP

1. Add Spring AOP Dependency in pom.xml:

xml  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.9</version>  
</dependency>  
<dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.9.6</version>  
</dependency>  
  
  
2. Create LoggingAspect Class:

java  
@Aspect  
public class LoggingAspect {  
 @Around("execution(\* com.library..\*(..))")  
 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;  
 }  
}  
  
  
3. Enable AspectJ Support in `applicationContext.xml`:

xml  
<aop:aspectj-autoproxy/>  
<bean class="com.library.aspect.LoggingAspect"/>  
  
  
4. Test: Run LibraryManagementApplication to see logging output.

# Exercise 4: Creating and Configuring a Maven Project

1. Create a new Maven Project: LibraryManagement  
  
2. Add Spring Dependencies:

xml  
<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.9</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.9</version>  
 </dependency>  
</dependencies>  
  
  
3. Configure Maven Compiler Plugin:

xml  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.0</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
</build>

# Exercise 5: Configuring the Spring IoC Container

- Use the same applicationContext.xml as earlier.  
- Ensure BookService has a setter method for BookRepository.  
- Load the Spring context in LibraryManagementApplication and invoke methods on bookService bean.

# Exercise 6: Configuring Beans with Annotations

1. Enable Component Scanning:

xml  
<context:component-scan base-package="com.library"/>  
<context:annotation-config/>  
  
  
2. Annotate Classes:

java  
@Service  
public class BookService {  
 @Autowired  
 private BookRepository bookRepository;  
}  
  
  
java  
@Repository  
public class BookRepository {  
}  
  
  
3. Run the `LibraryManagementApplication` to test annotation-based config.

# Exercise 7: Implementing Constructor and Setter Injection

1. Constructor Injection in XML:

xml  
<bean id="bookService" class="com.library.service.BookService">  
 <constructor-arg ref="bookRepository"/>  
</bean>

2. BookService.java:

java  
public class BookService {  
 private BookRepository bookRepository;  
  
 public BookService(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
}  
  
  
3. Run LibraryManagementApplication to test.

# Exercise 8: Implementing Basic AOP with Spring

- Define LoggingAspect class with @Before and @After advice.

java  
@Before("execution(\* com.library.service.\*.\*(..))")  
public void logBefore() {  
 System.out.println("Before method execution");  
}  
  
@After("execution(\* com.library.service.\*.\*(..))")  
public void logAfter() {  
 System.out.println("After method execution");  
}  
  
  
- Register this aspect in applicationContext.xml and enable <aop:aspectj-autoproxy/>

# Exercise 9: Creating a Spring Boot Application

1. Use [Spring Initializr](https://start.spring.io/) with dependencies:  
- Spring Web  
- Spring Data JPA  
- H2 Database  
  
2. Configure application.properties:

properties  
spring.datasource.url=jdbc:h2:mem:testdb  
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  
  
  
3. Define Book entity:

java  
@Entity  
public class Book {  
 @Id @GeneratedValue  
 private Long id;  
 private String title;  
}  
  
  
4. BookRepository:

java  
public interface BookRepository extends JpaRepository<Book, Long> {}  
  
  
5. REST Controller:

java  
@RestController  
@RequestMapping("/books")  
public class BookController {  
 @Autowired  
 private BookRepository repository;  
  
 @GetMapping  
 public List<Book> findAll() {  
 return repository.findAll();  
 }  
}  
  
  
6. Run the application using main() from LibraryManagementApplication.java