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.

Solution

`pom.xml`

```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.shelian</groupId>
 <artifactId>LibraryManagement</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>LibraryManagement</name>
 <!-- FIXME change it to the project's website -->
 <url>http://www.example.com</url>
 <properties>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <maven.compiler.release>17</maven.compiler.release>
 </properties>
 <dependencyManagement>
   <dependencies>
```

```
     <dependency>
       <groupId>org.junit</groupId>
       <artifactId>junit-bom</artifactId>
       <version>5.11.0</version>
       <type>pom</type>
       <scope>import</scope>
     </dependency>
   </dependencies>


 </dependencyManagement>
 <dependencies>
   <dependency>
     <groupId>org.junit.jupiter</groupId>
     <artifactId>junit-jupiter-api</artifactId>
     <scope>test</scope>
   </dependency>
   <!-- Optionally: parameterized tests support -->
   <dependency>
     <groupId>org.junit.jupiter</groupId>
     <artifactId>junit-jupiter-params</artifactId>
     <scope>test</scope>
   </dependency>

   <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
             <dependency>
                 <groupId>org.springframework</groupId>
                 <artifactId>spring-core</artifactId>
                 <version>6.2.7</version>
             </dependency>


       <!--
https://mvnrepository.com/artifact/org.springframework/spring-context -->
       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-context</artifactId>
           <version>6.2.7</version>
       </dependency>
 </dependencies>
 <build>
   <pluginManagement><!-- lock down plugins versions to avoid using Maven
defaults (may be moved to parent pom) -->
     <plugins>
       <!-- clean lifecycle, see
https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle
-->
       <plugin>
         <artifactId>maven-clean-plugin</artifactId>
         <version>3.4.0</version>
```

```
      </plugin>
      <!-- default lifecycle, jar packaging: see
https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin_bin
dings_for_jar_packaging -->
      <plugin>
        <artifactId>maven-resources-plugin</artifactId>
        <version>3.3.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.13.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.3.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.4.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-install-plugin</artifactId>
        <version>3.1.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>3.1.2</version>
      </plugin>
      <!-- site lifecycle, see
https://maven.apache.org/ref/current/maven-core/lifecycles.html#site_Lifecycle
-->
      <plugin>
        <artifactId>maven-site-plugin</artifactId>
        <version>3.12.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>3.6.1</version>
      </plugin>
    </plugins>
  </pluginManagement>
 </build>
</project>
```

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="service"
class="com.library.service.BookService"></bean>
            <bean id="repository"
class="com.library.repository.BookRepository"></bean>
</beans>
```

BookService.java

```
package com.library.service;
public class BookService {
      public void useBookService() {
            System.out.println("You have availed this service!");
      }
}
```

BookRepository.java

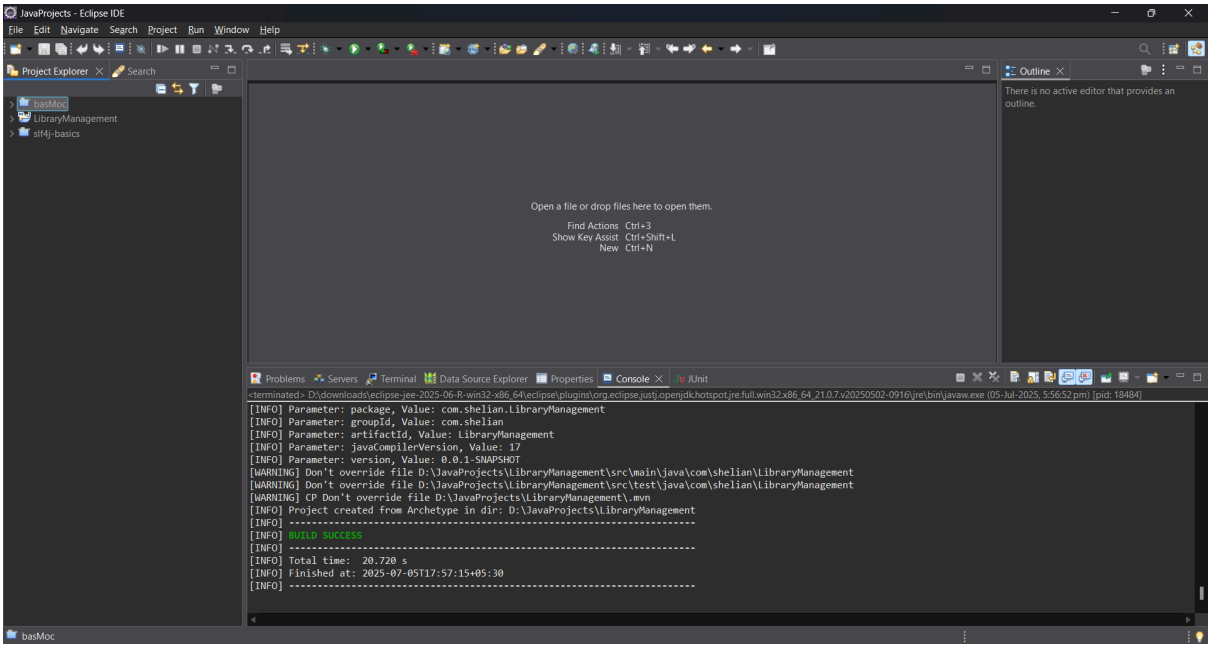
```
package com.library.repository;
public class BookRepository {
      public void browseRepo() {
            System.out.println("Browsing through the repo...");
      }
}
```

App.java

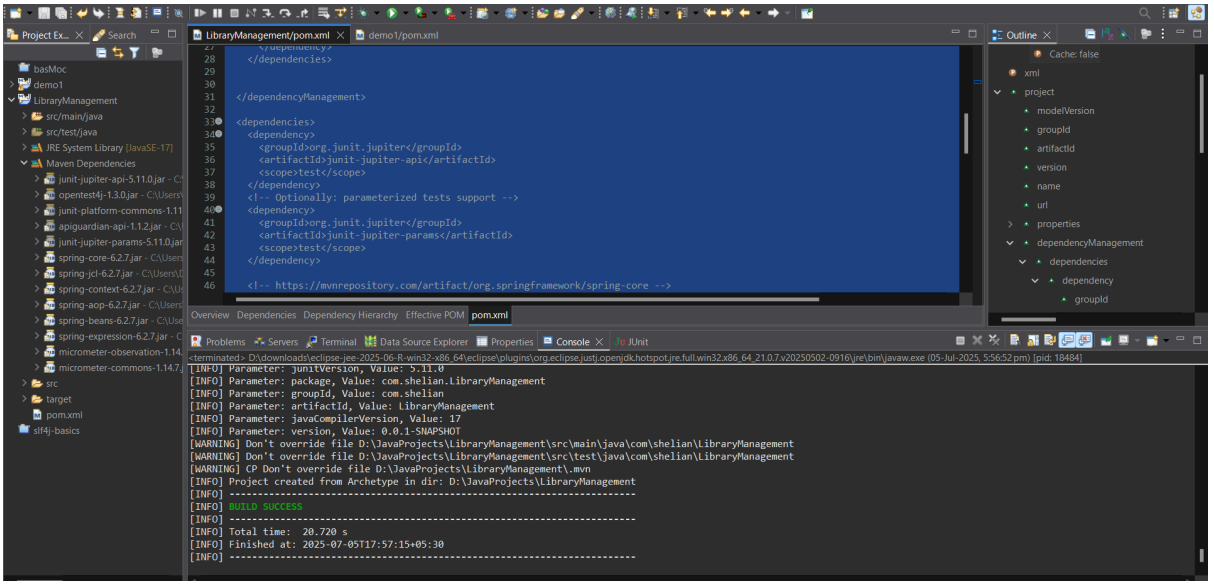
```
package com.shelian.LibraryManagement;
import com.library.repository.BookRepository;
import com.library.service.BookService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class App {
   public static void main(String[] args) {
       ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
       BookService bs = (BookService) context.getBean("service");
       BookRepository br = (BookRepository) context.getBean("repository");

       bs.useBookService();
       br.browseRepo();
   }
}
```

Output Screenshots

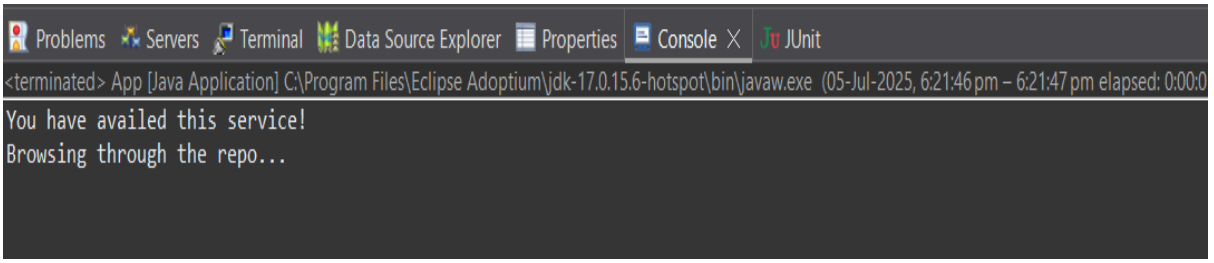
1. Create the project



2. Add dependencies



3. Output of App.java



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.

Solution

`applicationContext.xml`

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

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

</beans>
```

`BookService.java`

```java
package com.library.service;
import com.library.repository.BookRepository;
public class BookService {
      private BookRepository bookrepository;

      // setter
      public void setBookRepository(BookRepository bookrepository) {
            this.bookrepository = bookrepository;
      }
```

```
        public void useBookService() {
                System.out.println("You have availed this service!");
                System.out.println("We will now redirect you to our repository!");

                System.out.println();

                bookrepository.browseRepo();
        }
}
```

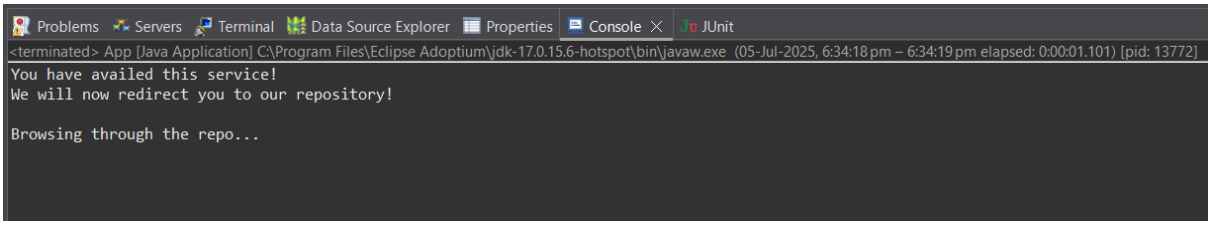
BookRepository.java

```
package com.library.repository;
public class BookRepository {
        public void browseRepo() {
                System.out.println("Browsing through the repo...");
        }
}
```

App.java

```
package com.shelian.LibraryManagement;
import com.library.service.BookService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class App {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
        BookService bs = (BookService) context.getBean("service",
BookService.class);
        bs.useBookService();
    }
}
```

Output Screenshots



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.

Solution

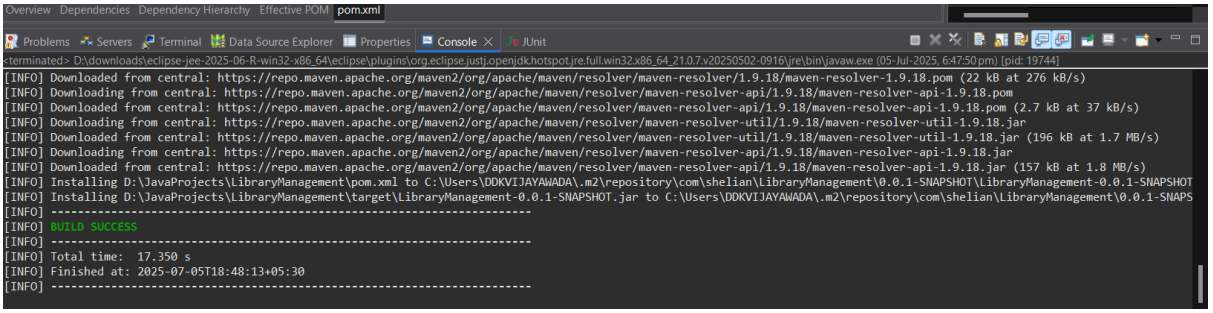
`LibraryManagement/pom.xml`

```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.shelian</groupId>
 <artifactId>LibraryManagement</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>LibraryManagement</name>
 <!-- FIXME change it to the project's website -->
 <url>http://www.example.com</url>
 <properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <maven.compiler.source>1.8</maven.compiler.source>
 <maven.compiler.target>1.8</maven.compiler.target>
</properties>
 <dependencyManagement>
   <dependencies>
     <dependency>
       <groupId>org.junit</groupId>
       <artifactId>junit-bom</artifactId>
       <version>5.11.0</version>
       <type>pom</type>
       <scope>import</scope>
     </dependency>
   </dependencies>
```

```
 </dependencyManagement>
 <dependencies>
   <dependency>
     <groupId>org.junit.jupiter</groupId>
     <artifactId>junit-jupiter-api</artifactId>
     <scope>test</scope>
   </dependency>
   <!-- Optionally: parameterized tests support -->
   <dependency>
     <groupId>org.junit.jupiter</groupId>
     <artifactId>junit-jupiter-params</artifactId>
     <scope>test</scope>
   </dependency>

   <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
             <dependency>
                 <groupId>org.springframework</groupId>
                 <artifactId>spring-core</artifactId>
                 <version>6.2.7</version>
             </dependency>


       <!--
https://mvnrepository.com/artifact/org.springframework/spring-context -->
       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-context</artifactId>
           <version>6.2.7</version>
       </dependency>


       <!-- https://mvnrepository.com/artifact/org.springframework/spring-aop
-->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-aop</artifactId>
   <version>6.2.6</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>6.2.6</version>
</dependency>
 </dependencies>

 <build>
 <plugins>
```

```
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.10.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
  </project>
```

Output Screenshots



ꕥ —— [Additional Important Hands-On] —— ꕥ

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.

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.

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.