Spring Core Project: Library Book Inventory with Proxy

Overview

In this project, you will implement a Library Book Inventory System for showing **Spring-Managed Bean Lifecycle with Proxy and Profile Configuration** using Java-based Spring configuration and a proxy pattern. The system manages a library with books and allows operations like adding a book and issuing it. You will also use a proxy class to perform library actions like issuing books. Additionally, the project makes use of Spring's Dependency Injection and AOP-like behavior using proxies.

Project Structure

1. **Configuration Class (\`AppConfig\`)**: Java-based Spring configuration class defining beans for \`Book\`, \`Library\`, and \`LibraryServiceProxy\`. [**This should be defined by you**]
2. **Bean (\`Book\`)**: Java class representing books**. [Already defined]**

3. **Bean (\`Library\`)**: Java class representing books and the library with operations such as adding and issuing books. [**Partially defined**]
4. **Proxy Class (\`LibraryServiceProxy\`)**: A proxy that wraps around the \`Library\` service to add additional logic (like logging) before and after method invocations. [**This should be defined by you**]
5. **Main Application Class (\`BookInventoryApplication\`)**: Boots the Spring context and performs actions like adding and issuing books. [**This should be defined by you**]
6. **Test Cases**: A set of pre-written tests that validate the correctness of your Spring configuration and ensure the proxy works correctly. [**Already defined**]

Steps to Complete the Project

Step 1: Understand the Given Code

Before you start, ensure that you understand the provided classes:
- **Book Class**: Represents a book with properties such as \`name\` and \`quantity\`.
- **Library Class**: Represents a library, which holds a book. It has methods to add and issue a book. Along with this it should have lifecycle annotations like \`@PostConstruct\` and \`@PreDestroy\`.
- **LibraryServiceProxy**: A proxy class that wraps the \`Library\` service methods, allowing you to add additional logic (like logging) before and after method execution.

Step 2: Implement the Configuration Class (`AppConfig.java`)

In the `AppConfig` class, you will define the following beans:
1. **Book Bean**: Define a `Book` bean with a name (`Java Basics`) and quantity (`50`).
2. **Library Bean**: Define a `Library` bean with a name (`Central Library`).
3. **LibraryServiceProxy Bean**: Define a `LibraryServiceProxy` bean to act as a proxy to the `Library` service.

Step 3: Implement the Main Application Class (`BookInventoryApplication.java`)

In the `BookInventoryApplication.java` class:
1. **Load the Spring Context**: Use `AnnotationConfigApplicationContext` to load the Spring context from `AppConfig.class`.
2. **Retrieve Beans**: Retrieve the `Book` and `Library` beans from the Spring context and perform actions like adding books to the library.
3. **Perform Library Action via Proxy**: Use the `LibraryServiceProxy` to issue a book (via the `performLibraryAction()` method) and verify the book quantity has decreased.

Step 4: Implement the Proxy Class (`LibraryServiceProxy.java`)

In the `LibraryServiceProxy` class:
1. **Proxy Logic**: Implement a method `performLibraryAction(Library library)` that wraps the `issueBook()` method of the `Library` class, adding additional logic before and after the method call by adding System.out.println statements with message as “Proxy: Before calling the actual method” and “Proxy: After calling the actual method.” respectively.

Step 5: Implement the `Library` Class (`Library`)

In the Library class, you will implement two lifecycle methods: init() and destroy(). These methods allow you to perform custom actions when the bean is initialized and when it is about to be destroyed by the Spring container.

- **init() Method**:
  After the Library bean is created and all its dependencies (such as the Book object) are injected, this method will be automatically called. In this method, you should print a message indicating that the Library bean has been initialized as "Library bean initialized: " + name. Additionally, if a Book is associated with the library, print the name of the book as "Book initialized in library: " + book.getName().

- **destroy() Method**:
  Just before the Library bean is destroyed, this method will be called automatically. Here, you should print a message as “Library bean destroyed: " + name” that indicates the Library bean is being destroyed. If a Book is associated with the library, print the name of the book that is being destroyed as "Book destroyed: " + book.getName().

Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through**

Command Terminal.

2. To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. cd into your backend project folder.
4. To build your project use command:

   sudo JAVA_HOME=$JAVA_HOME /usr/share/maven/bin/mvn clean package -Dmaven.test.skip

   ***If it asks for the password, provide password : pass@word1**
5. To launch your application, move into the target folder (cd target). Run the following command to run the application:

   java -jar <your application jar file name>
6. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN. Please use 127.0.0.1 instead of localhost to test rest endpoints.
7. Default credentials for MySQL:
   a. Username: root
   b. Password: pass@word1
8. To login to mysql instance: Open new terminal and use following command:
   a. sudo systemctl enable mysql
   b. sudo systemctl start mysql

   NOTE: After typing any of the above commands you might encounter any warnings.

   >> Please note that this warning is expected and can be disregarded. Proceed to the next step.

   c. mysql -u root -p

   The last command will ask for password which is 'pass@word1'
9. Mandatory: Before final submission run the following command:

   sudo JAVA_HOME=$JAVA_HOME /usr/share/maven/bin/mvn test

   ***If it asks for the password, provide password : pass@word1**