Spring Core Project: E-Commerce Java Configuration

Overview

In this project, you will be implementing an e-commerce system using Java-based configuration in Spring. The system includes defining beans for `Product`, `Customer`, and `Order` using Spring's Java Configuration mechanism with annotations. The project will also involve configuring scopes for some of the beans and verifying the configuration through predefined test cases. Your goal is to implement the configuration correctly so that the test cases pass.

Project Structure

1. **Configuration Class (`AppConfig`)**: Java-based Spring configuration class defining beans with scopes. [**This should be defined by you**]
2. **Beans (`Product`, `Customer`, `Order`)**: Java classes that represent the core entities in the e-commerce system. [**Already defined**]
3. **Main Application Class (`ConfigurationScopeBeanApp`)**: Loads Spring context and retrieves beans. [**This should be defined by you**]
4. **Test Cases**: A set of pre-written tests that will validate your Java configuration and bean scopes. [**Already defined**]

Steps to Complete the Project

Step 1: Understand the Given Code

Before you start, ensure that you understand the provided classes:
- **Product Class**: Represents a product with properties `name` and `price`.
- **Customer Class**: Represents a customer with properties `customerId` and `name`.
- **Order Class**: Represents an order with properties `orderId` and `totalAmount`.
The tests are designed to check whether the Java configuration is correct, and whether the proper annotations for bean, scope and configuration are correctly applied.

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

In the `AppConfig` class, you will define the beans using Java-based configuration:
1. **Define Product Bean (Singleton Scope)**: Create a singleton-scoped `product` bean that represents a laptop with name “Laptop” and price 1000.0.
2. **Define Product Bean (Prototype Scope)**: Create a prototype-scoped `product` bean representing a smartphone with name “Smartphone” and price 500.0.
3. **Define Customer Bean**: Create a `customer` bean with a customer ID as “CUST001”

and name as “John Doe”.
4. **Define Order Bean**: Create an `order` bean with an order ID as “ORD1234“ and total amount as 0.0.

Ensure you use the proper annotations where required.

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

In the `ConfigurationScopeBeanApp.java` class:
1. **Load the Spring Context**: Use `AnnotationConfigApplicationContext` to load the Spring context from `AppConfig.class`.
2. **Retrieve Beans**: Retrieve the `Product` (singleton and prototype), `Customer`, and `Order` beans from the Spring context.
3. **Display Bean Details**: Print the details of the `Product`, `Customer`, and `Order` beans in each different statement using `System.out.println` as:

- "Product (Singleton scope): " + product
- "Product (Prototype scope): " + productPrototype
- "Customer: " + customer
- "Order: " + order

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:**

   mvn clean package -Dmaven.test.skip
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>.war
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. **Mandatory: Before final submission run the following command:**

   mvn test

8. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**