Implementing Dependency Injection (DI) in Java Using Interfaces and Classes

Project Abstract

The purpose of this project is to demonstrate the concept of Dependency Injection (DI) in Java, using both interfaces and classes. Dependency Injection is a design pattern that allows us to implement loose coupling in our application by passing dependencies from outside rather than creating them within a class. The project aims to show how DI can be used effectively with both interfaces and classes, and how methods in DI-injected classes can be accessed and executed.

Tasks Overview

Task 1: Implementing Dependency Injection Interface

Objective: Define an interface for a service and implement it in concrete classes using Dependency Injection.

Detailed Description: In this task, you will create an interface called MessageService, which will have a method to send messages. You will then implement this interface in two classes: EmailService and SMSService. Each class will define its own way of sending messages.

- Steps:

1. Create the MessageService Interface:
    - Declare a method sendMessage(String message) in the interface.
2. Create the EmailService Class:
    - Implement the MessageService interface.
    - Provide the implementation of sendMessage to simulate sending an email and print message as "Sending email with message: " + message.
3. Create the SMSService Class:
    - Implement the MessageService interface.
    - Provide the implementation of sendMessage to simulate sending an SMS and print message as "Sending SMS with message: " + message.

Task 2: Implementing Dependency Injection in Classes

Objective: Implement Dependency Injection by passing the dependency to a class via the constructor.

Detailed Description: In this task, you will create the MyApplication class, which will receive a MessageService dependency. The MessageService will be injected through the constructor of MyApplication. The class will use the injected service to send messages.

- Steps:

4. Create the MyApplication Class:
    - Create a constructor in MyApplication that accepts a MessageService object and assigns it to an instance variable.
    - Define a method processMessage(String message) that calls the sendMessage method of the injected MessageService.
5. In main():
    - Create an object of EmailService as emailService and assign it to MessageService.
    - Create an object of MyApplication as app and pass emailService to it
    - Invoke processMessage("Hello, Dependency Injection!") from app.
    - Create an object of SMSService as smsService and assign it to MessageService.
    - Create an object of MyApplication as appSMS and pass smsService to it
    - Invoke processMessage("Hello, Dependency Injection via SMS!") from appSMS.

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 Application menu (Three horizontal lines at left top) → Terminal →New Terminal.**
3. **This editor Auto Saves the code.**
4. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
5. **To run your project use command:**

```
sudo JAVA_HOME=$JAVA_HOME /usr/share/maven/bin/mvn compile exec:java
-Dexec.mainClass="com.yaksha.assignment.DependencyInjectionAssignment"
```

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

6. To test your project test cases, use the command

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

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