1. **Define a NotificationService Interface:**

Create a Java interface named **NotificationService** with a method **sendNotification(String message)**.

1. **Implement Two NotificationService Implementations:**

Create two classes that implement the **NotificationService** interface:

* + **EmailNotificationService**: Sends notifications via email.
  + **SMSNotificationService**: Sends notifications via SMS.

1. **Create a NotificationManager Class:**

Implement a class named **NotificationManager** that depends on the **NotificationService** interface. It should have a constructor that takes a **NotificationService** as a dependency and a method **sendNotification(String message)** that delegates to the injected service.

1. **Configure Beans in Spring:**

Write a Spring configuration class (**AppConfig**) where you define beans for the two **NotificationService** implementations (**emailNotificationService** and **smsNotificationService**) and a bean for the **NotificationManager** class (**notificationManager**). Use dependency injection to wire the services into the manager.

1. **Run the Program:**

In a **MainApp** class, use the Spring context to obtain instances of the **NotificationManager** for both email and SMS services and invoke the **sendNotification** method.

When you run the program, it should produce output similar to:

Sending email notification: Hello, this is an **email notification!**

Sending SMS notification: Hello, this is an **SMS notification!**

**Answer**

**1. Define an NotificationService interface:**

// NotificationService.java

public interface NotificationService {

void sendNotification(String message);

}

**2. Create two implementations of the NotificationService interface:**

// EmailNotificationService.java

public class EmailNotificationService implements NotificationService {

@Override

public void sendNotification(String message) {

System.out.println("Sending email notification: " + message);

}

}

// SMSNotificationService.java

public class SMSNotificationService implements NotificationService {

@Override

public void sendNotification(String message) {

System.out.println("Sending SMS notification: " + message);

}

}

**3. Create a NotificationManager class that depends on the NotificationService interface:**

// NotificationManager.java

public class NotificationManager {

private NotificationService notificationService;

// Constructor with NotificationService interface dependency

public NotificationManager(NotificationService notificationService) {

this.notificationService = notificationService;

}

public void sendNotification(String message) {

notificationService.sendNotification(message);

}

}

**Create a Spring configuration class:**

// AppConfig.java

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class AppConfig {

@Bean

@Primary

public NotificationService emailNotificationService() {

return new EmailNotificationService();

}

@Bean

public NotificationService smsNotificationService() {

return new SMSNotificationService();

}

@Bean

public NotificationManager notificationManager(NotificationService notificationService) {

return new NotificationManager(notificationService);

}

}

In the **MainApp** class, we demonstrate the use of different implementations of the **NotificationService** interface (**EmailNotificationService** and **SMSNotificationService**) with the **NotificationManager** class.

// MainApp.java

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {

public static void main(String[] args) {

try (AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class)) {

// Use EmailNotificationService

NotificationManager emailNotificationManager = context.getBean(NotificationManager.class);

emailNotificationManager.sendNotification("Hello, this is an email notification!");

// Use SMSNotificationService

NotificationManager smsNotificationManager = context.getBean(NotificationManager.class);

smsNotificationManager.sendNotification("Hello, this is an SMS notification!");

}

}

}

**Exercise Explanation:**

This exercise illustrates loose coupling by creating a notification system where the **NotificationManager** class is loosely coupled with the **NotificationService** interface. The choice of notification service implementation is determined at runtime through Spring dependency injection.

* The **NotificationManager** class is not concerned with the details of how notifications are sent. It depends on the abstraction provided by the **NotificationService** interface.
* The Spring container injects different implementations of the **NotificationService** interface into the **NotificationManager** class based on the configuration.
* Switching between email and SMS notification services is achieved by changing the configuration, not by modifying the **NotificationManager** class.

When you run the **MainApp** class, you should see output similar to:

**Sending email notification: Hello, this is an email notification!**

**Sending SMS notification: Hello, this is an SMS notification!**