**Exercise 5: Implementing the Decorator Pattern**

**Code:**

**Notifier Interface: -**

public interface Notifier {  
 void send(String msg);  
}

**EmailNotifier Class: -**

public class EmailNotifier implements Notifier{  
 @Override  
 public void send(String msg) {  
 System.*out*.println("Sending Email: "+msg);  
 }  
}

**NotifierDecorator Class: -**

public abstract class NotifierDecorator implements Notifier {  
 protected Notifier decoratedNotifier;  
 public NotifierDecorator(Notifier notifier){  
 this.decoratedNotifier=notifier;  
 }  
 @Override  
 public void send(String msg){  
 decoratedNotifier.send(msg);  
 }  
}

**SlackNotifierDecorator Class: -**

public class SlackNotifierDecorator extends NotifierDecorator {  
 public SlackNotifierDecorator(Notifier notifier) {  
 super(notifier);  
 }  
 @Override  
 public void send(String msg) {  
 super.send(msg);  
 sendSlack(msg);  
 }  
 private void sendSlack(String msg) {  
 System.*out*.println("Sending Slack Message: " + msg);  
 }  
}

**SMSNotifierDecorator Class: -**

public class SMSNotifierDecorator extends NotifierDecorator {  
 public SMSNotifierDecorator(Notifier notifier){  
 super(notifier);  
 }  
 @Override  
 public void send(String msg){  
 super.send(msg);  
 sendSMS(msg);  
 }  
 private void sendSMS(String msg){  
 System.*out*.println("Sending SMS: "+msg);  
 }  
}

**TestClass (Main Class): -**

public class TestClass {  
 public static void main(String[] args){  
 Notifier baseNotifier=new EmailNotifier();  
 Notifier smsNotifier=new SMSNotifierDecorator(baseNotifier);  
 Notifier fullNotifier=new SlackNotifierDecorator(smsNotifier);  
 fullNotifier.send("Hello there!");  
 }  
}

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**