## Exercise 5: Implementing the Decorator Pattern

Scenario:  
Developing a notification system where notifications can be sent via multiple channels (e.g., Email, SMS).  
Use the Decorator Pattern to add functionalities dynamically.

## Step-by-step Implementation:

### Step 1: Create a New Java Project

Create a new Java project named DecoratorPatternExample.

### Step 2: Define Component Interface

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

### Step 3: Implement Concrete Component

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

### Step 4: Implement Decorator Classes

public abstract class NotifierDecorator implements Notifier {  
 protected Notifier notifier;  
  
 public NotifierDecorator(Notifier notifier) {  
 this.notifier = notifier;  
 }  
  
 @Override  
 public void send(String message) {  
 notifier.send(message);  
 }  
}  
  
public class SMSNotifierDecorator extends NotifierDecorator {  
 public SMSNotifierDecorator(Notifier notifier) {  
 super(notifier);  
 }  
  
 @Override  
 public void send(String message) {  
 super.send(message);  
 System.out.println("Sending SMS: " + message);  
 }  
}  
  
public class SlackNotifierDecorator extends NotifierDecorator {  
 public SlackNotifierDecorator(Notifier notifier) {  
 super(notifier);  
 }  
  
 @Override  
 public void send(String message) {  
 super.send(message);  
 System.out.println("Sending Slack message: " + message);  
 }  
}

### Step 5: Test the Decorator Implementation

public class TestDecoratorPattern {  
 public static void main(String[] args) {  
 Notifier emailNotifier = new EmailNotifier();  
  
 Notifier smsNotifier = new SMSNotifierDecorator(emailNotifier);  
 Notifier slackNotifier = new SlackNotifierDecorator(smsNotifier);  
  
 slackNotifier.send("System maintenance at 10 PM tonight.");  
 }  
}  
  
OUTPUT:

