**Module 1 - Design Patterns and Principles**

**Exercise 5: Implementing the Decorator Pattern**

**Aim:**

To develop a notification system where notifications can be sent via multiple channels (e.g., Email, SMS) using the Decorator Pattern to add functionalities dynamically.

**Code:**

interface Notifier {

    void send();

}

class EmailNotifier implements Notifier {

    public void send() {

        System.out.println("Sending Email Notification");

    }

}

abstract class NotifierDecorator implements Notifier {

    protected Notifier notifier;

    public NotifierDecorator(Notifier notifier) {

        this.notifier = notifier;

    }

    public void send() {

        notifier.send();

    }

}

class SMSNotifierDecorator extends NotifierDecorator {

    public SMSNotifierDecorator(Notifier notifier) {

        super(notifier);

    }

    public void send() {

        super.send();

        System.out.println("Sending SMS Notification");

    }

}

class SlackNotifierDecorator extends NotifierDecorator {

    public SlackNotifierDecorator(Notifier notifier) {

        super(notifier);

    }

    public void send() {

        super.send();

        System.out.println("Sending Slack Notification");

    }

}

public class DecoratorPatternExample {

    public static void main(String[] args) {

        Notifier email = new EmailNotifier();

        email.send();

        Notifier sms = new SMSNotifierDecorator(new EmailNotifier());

        sms.send();

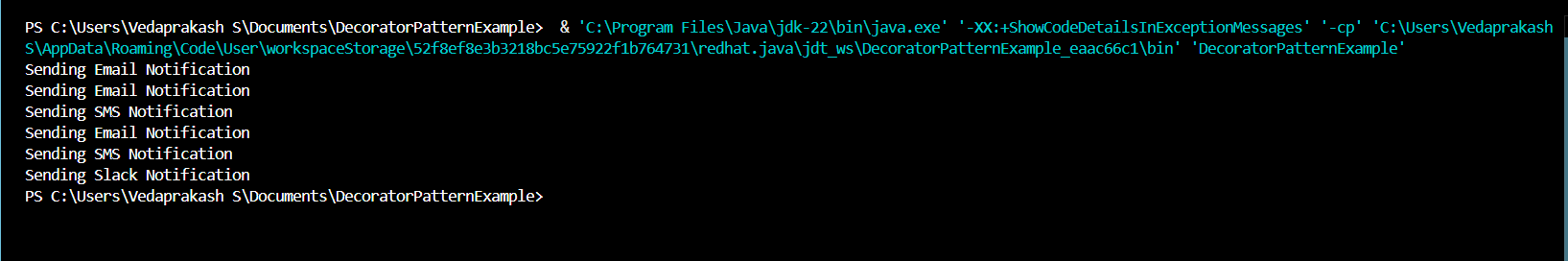
        Notifier all = new SlackNotifierDecorator(new SMSNotifierDecorator(new EmailNotifier()));

        all.send();

    }

}

**Output:**

****