**Exercise 5: Implementing the Decorator Pattern**

**Scenario:**

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

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **DecoratorPatternExample**.
2. **Define Component Interface:**
   * Create an interface **Notifier** with a method **send()**.
3. **Implement Concrete Component:**
   * Create a class **EmailNotifier** that implements Notifier.
4. **Implement Decorator Classes:**
   * Create abstract decorator class **NotifierDecorator** that implements **Notifier** and holds a reference to a **Notifier** object.
   * Create concrete decorator classes like **SMSNotifierDecorator**, **SlackNotifierDecorator** that extend **NotifierDecorator**.
5. **Test the Decorator Implementation:**
   * Create a test class to demonstrate sending notifications via multiple channels using decorators.

## Answer:

EmailNotifier.java

public class EmailNotifier implements Notifier{

    String email;

    public EmailNotifier(String emailAddress) {

        this.email = emailAddress;

    }

    @Override

    public void send(String msg) {

        System.out.println("Sending email to " + email + ": " + msg);

    }

}

Main.java

public class Main {

    public static void main(String[] args) {

        Notifier email = new EmailNotifier("soumik@cognizant.com");

        System.out.println("For email only: ");

        email.send("Hello This is mail only.");

        Notifier emailandSms = new SMSNotifierDecorator(new EmailNotifier("soumik@cognizant.com"), 745136452);

        System.out.println("\nFor email and phone number: ");

        emailandSms.send("Hello This is mail and phone number.");

        Notifier emailandSmsandSlack = new SlackNotifierDecorator(emailandSms, "#Added Slack");

        System.out.println("\nFor email, phone number and slack:");

        emailandSmsandSlack.send("Hello This is mail, phone number and slack.");

    }

}

Notifier.java

public interface Notifier{

    void send(String message);

}

public abstract class NotifierDecorator implements Notifier {

    Notifier wrap;

    public NotifierDecorator(Notifier wrap) {

        this.wrap = wrap;

    }

    @Override

    public void send(String msg) {

        wrap.send(msg);

    }

}

SlackNotifierDecorator.java

public class SlackNotifierDecorator extends NotifierDecorator {

    String channel;

    public SlackNotifierDecorator(Notifier wrap, String channel) {

        super(wrap);

        this.channel = channel;

    }

    @Override

    public void send(String msg) {

        super.send(msg);

        System.out.println("Sending slack to " + channel + ": " + msg);

    }

}

SMSNotifierDecorator.java

public class SMSNotifierDecorator  extends NotifierDecorator {

    long pno;

    public SMSNotifierDecorator(Notifier wrap, long pno) {

        super(wrap);

        this.pno = pno;

    }

    @Override

    public void send(String msg) {

        super.send(msg);

        System.out.println("Sending SMS to " + pno + ": " + msg);

    }

}

Output:

