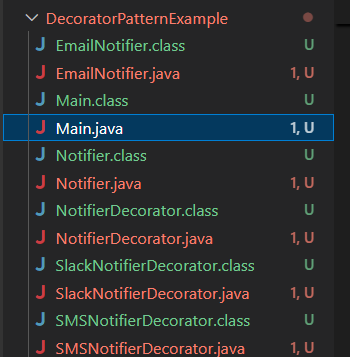
**Exercise 5: Implementing the Decorator Pattern**

**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.



**Notifier.java :-**

public interface Notifier {

void send(String message);

}

**EmailNotifier.java :-**

public class EmailNotifier implements Notifier {

@Override

public void send(String message) {

System.out.println("📧 Email sent: " + message);

}

}

**NotifierDecorator.java:-**

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); // delegate to wrapped object

}

}

**SMSNotifierDecorator.java:-**

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("📱 SMS sent: " + message);

}

}

**SlackNotifierDecorator.java:-**

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("💬 Slack message sent: " + message);

}

}

**Main.java:-**

public class Main {

public static void main(String[] args) {

// Base email notification

Notifier baseNotifier = new EmailNotifier();

// Add SMS functionality

Notifier smsNotifier = new SMSNotifierDecorator(baseNotifier);

// Add Slack on top of Email and SMS

Notifier multiChannelNotifier = new SlackNotifierDecorator(smsNotifier);

// Send message via all channels

multiChannelNotifier.send("System will be down for maintenance at 10 PM.");

}

}

**Output :-**

