**Design Pattern And Principles**

**EXERCISE 5: Implementing the Decorator Pattern**

**Source Code**

// Main.java

// Step 2: Component Interface

interface Notifier {

void send(String message);

}

// Step 3: Concrete Component

class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

// Step 4: Abstract Decorator

abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

public void send(String message) {

wrappedNotifier.send(message);

}

}

// Concrete Decorator: SMS

class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

// Concrete Decorator: Slack

class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

// Step 5: Test Class

public class Main {

public static void main(String[] args) {

System.out.println("=== Single Channel Notification ===");

Notifier email = new EmailNotifier();

email.send("Your order has been shipped….");

System.out.println("\n=== Email + SMS Notification ===");

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

emailAndSMS.send("Your OTP is 749375");

System.out.println("\n=== Email + SMS + Slack Notification ===");

Notifier multiChannel = new SlackNotifierDecorator(

new SMSNotifierDecorator(

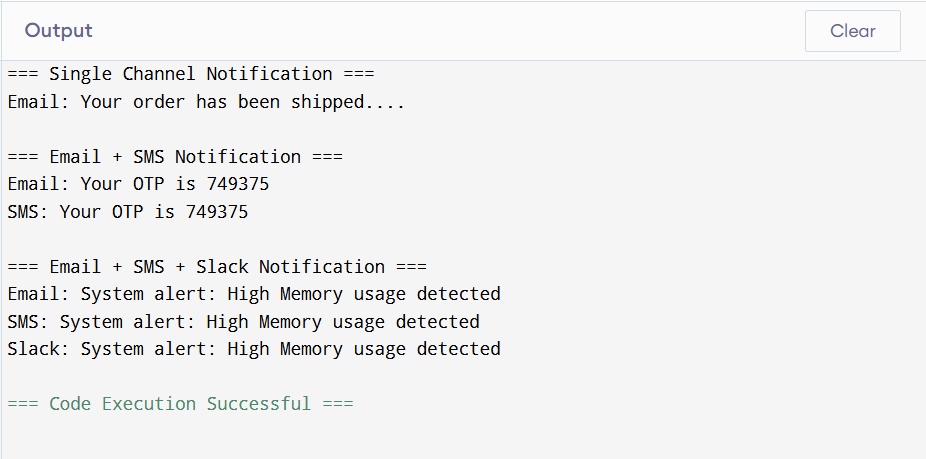
new EmailNotifier()));

multiChannel.send("System alert: High Memory usage detected");

}

}

**OUTPUT**

****