Notification System with Decorator Pattern

public class DecoratorPatternExample {

public static void main(String[] args) {

Notifier emailNotifier = new EmailNotifier();

System.out.println("SIMPLE EMAIL NOTIFICATION:");

emailNotifier.send("Server is down!");

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

System.out.println("\nEMAIL + SMS NOTIFICATION:");

emailAndSms.send("Database backup completed");

Notifier allNotifications = new SlackNotifierDecorator(

new SMSNotifierDecorator(

new EmailNotifier()));

System.out.println("\nEMAIL + SMS + SLACK NOTIFICATION:");

allNotifications.send("New user registered");

Notifier smsAndFacebook = new FacebookNotifierDecorator(

new SMSNotifierDecorator(

new BasicNotifier()));

System.out.println("\nSMS + FACEBOOK NOTIFICATION:");

smsAndFacebook.send("Your order has shipped");

Notifier complexNotification = new SlackNotifierDecorator(

new FacebookNotifierDecorator(

new SMSNotifierDecorator(

new EmailNotifier())));

System.out.println("\nCOMPLEX NOTIFICATION CHAIN:");

complexNotification.send("Urgent: Security patch required");

}

interface Notifier {

void send(String message);

}

// Concrete Component - Basic Notifier

static class BasicNotifier implements Notifier {

@Override

public void send(String message) {

// Base notifier does nothing (can be used as starting point)

}

}

// Concrete Component - Email Notifier

static class EmailNotifier implements Notifier {

@Override

public void send(String message) {

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

}

}

// Abstract Decorator

static abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

@Override

public void send(String message) {

wrappedNotifier.send(message);

}

}

static 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);

}

}

static 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);

}

}

static class FacebookNotifierDecorator extends NotifierDecorator {

public FacebookNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("Posting to Facebook: " + message);

}

}

}

Key Features:

1. **Flexible Composition**: Notifiers can be combined in any order
2. **Dynamic Behavior**: Functionality can be added at runtime
3. **Single Responsibility**: Each decorator handles one notification type
4. **Open/Closed Principle**: New notifiers can be added without modifying existing code
5. **Base Component**: BasicNotifier allows creating pure decorator chains

Decorator Pattern Benefits:

1. **Dynamic Functionality**: Add responsibilities to objects dynamically
2. **Avoids Class Explosion**: No need for classes like EmailAndSmsNotifier
3. **Flexible**: Decorators can be combined in any order
4. **Transparent**: Decorators implement the same interface as components
5. **Incremental Modification**: Each decorator adds small pieces of functionality

**Output**

