**Exercise 5: Implementing the Decorator Pattern**

**Notifier.java**

**package** mypackage;

**public** **interface** Notifier {

**void** send(String message);

}

### Implement Concrete Component

**EmailNotifier.java**

**package** mypackage;

**public** **class** EmailNotifier **implements** Notifier {

@Override

**public** **void** send(String message) {

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

}

}

### Implement Decorator Classes

#### Abstract Decorator Class

**NotifierDecorator.java**

**package** mypackage;

**public** **abstract** **class** NotifierDecorator **implements** Notifier {

**protected** Notifier notifier;

**public** NotifierDecorator(Notifier notifier) {

**this**.notifier = notifier;

}

**public** **void** send(String message) {

notifier.send(message);

}

}

#### Concrete Decorator: SMS

**SMSNotifierDecorator.java**

**package** mypackage;

**public** **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);

}

}

#### Concrete Decorator: Slack

**SlackNotifierDecorator.java**

**package** mypackage;

**public** **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);

}

}

### Test the Decorator Implementation

**TestDecoratorPattern.java**

**package** mypackage;

**public** **class** TestDecoratorPattern {

**public** **static** **void** main(String[] args) {

// Base notifier - Email only

Notifier emailNotifier = **new** EmailNotifier();

// Decorate with SMS

Notifier smsNotifier = **new** SMSNotifierDecorator(emailNotifier);

// Decorate with Slack on top of SMS

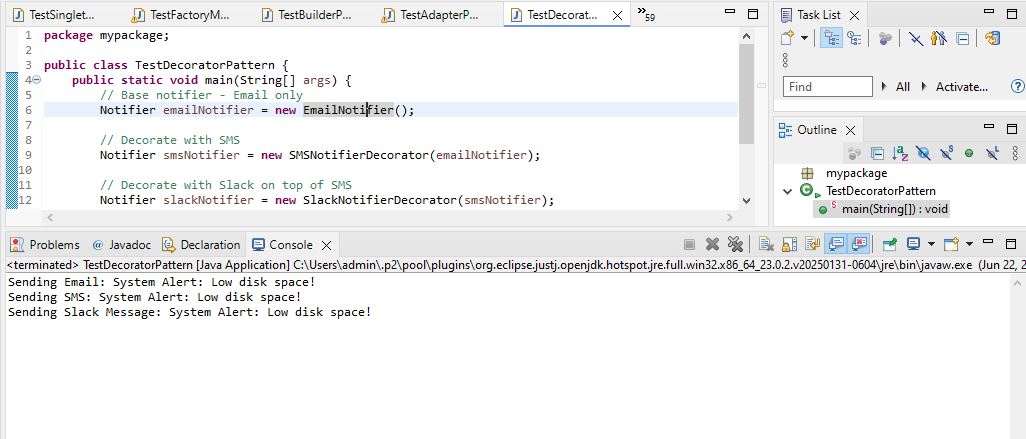
Notifier slackNotifier = **new** SlackNotifierDecorator(smsNotifier);

// Final notification will go via Email, SMS, and Slack

slackNotifier.send("System Alert: Low disk space!");

}

}



We are developing a notification system where notifications can be sent via multiple channels like Email, SMS, and Slack. The Decorator Pattern allows us to dynamically add new notification channels without modifying existing code.

**1. Component Interface**  
We define the interface Notifier with the method send(). This is the base interface that all notifiers implement.

**2. Concrete Component**  
EmailNotifier implements Notifier and provides basic email notification functionality.

**3. Decorator Classes**

* NotifierDecorator is an abstract class that implements Notifier and wraps another Notifier object.
* Concrete decorators like SMSNotifierDecorator and SlackNotifierDecorator extend NotifierDecorator and add extra behavior.

**4. Advantages of Decorator Pattern**

* Allows dynamic addition of new behaviors to existing objects at runtime.
* Promotes code reusability and flexibility.
* Follows the open-closed principle (open for extension, closed for modification).
* Avoids subclass explosion compared to inheritance-based extension.

**5. Time Complexity**

* Each decorator adds O(1) time as it simply calls the next layer of notifier.
* Total time complexity depends on the number of decorators chained.

**6. Real-life Applications**

* Notification systems (Email, SMS, Push, Slack, etc.).
* I/O Streams in Java (BufferedReader, BufferedInputStream are decorators).
* UI components where multiple decorations (border, color, effects) are added.