**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.

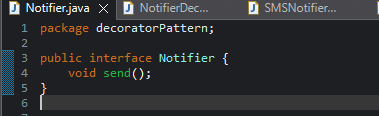
**Steps:**

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.

**Solution:**

| **Element** | **Role** | **Purpose** |
| --- | --- | --- |
| Notifier | Component Interface | Declares the core send() method |
| EmailNotifier | Concrete Component | Base notifier that sends email |
| NotifierDecorator | Abstract Decorator | Wraps a Notifier and delegates the send() call |
| SMSNotifierDecorator, SlackNotifierDecorator | Concrete Decorators | Add behavior to send() dynamically |

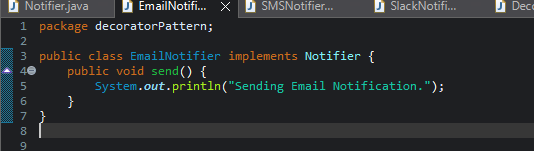
**Notifier.java**

****

**Explanation:**

* This is the common interface that all notifiers will implement.
* It contains the core method: send().

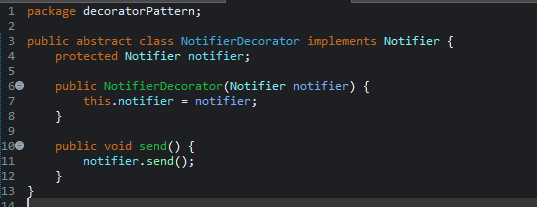
**EmailNotifier.java**

****

**Explanation:**

* This is the base class that sends notifications through email.
* It is the starting point for decorating.

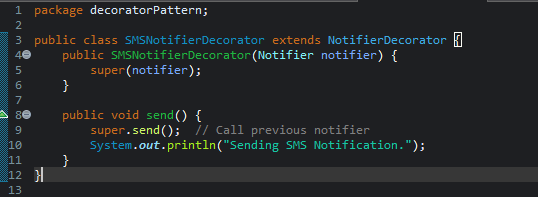
**NotifierDecorator.java**

****

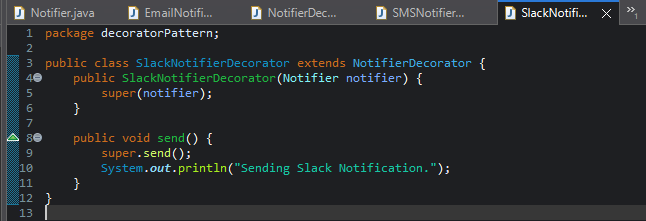
**Explanation:**

* This class wraps a Notifier and implements the same interface.
* It delegates the send() call to the wrapped object.
* Serves as a base for concrete decorators.

**SMSNotifierDecorator.java**

****

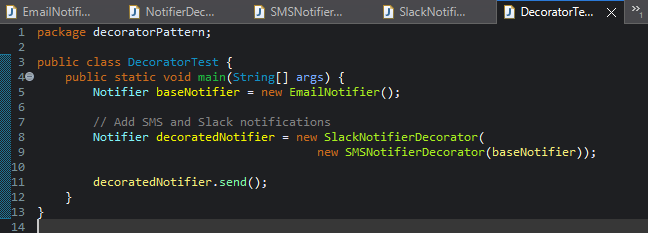
**SlackNotifierDecorator.java**

****

**Explanation (both):**

* These classes extend the base NotifierDecorator.
* They add their own logic after calling super.send() to maintain the chain.
* You can stack them in any order.

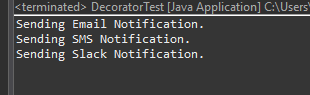
**DecoratorTest.java**

****

**Explanation:**

* The base notifier sends an email.
* Decorators are added on top: SMS, then Slack.
* All three notifications are sent in order.

**Output:**

****

**Benefits of the Decorator Pattern**

* ✔ Flexible layering of functionalities at runtime.
* ✔ Avoids large inheritance trees.
* ✔ Open for extension, closed for modification (SOLID principle).
* ✔ Easy to maintain and test each decorator separately.