Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проєктування»

Варіант: E-mail клієнт

Виконав:
студент групи ІА-32
Карповець Роман

Київ - 2025

Тема: Патерни проєктування.

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

Тема проєкту: E-mail клієнт

Опис: Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

Зміст

Теоретичні відомості

Призначення: Шаблон «Decorator» призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми. Декоратор деяким чином «обертає» (за рахунок агрегації) початковий об'єкт зі збереженням його функцій, проте дозволяє додати додаткові дії. Такий шаблон надає гнучкіший спосіб зміни поведінки об'єкту чим просте спадкоємство, оскільки початкова функціональність зберігається в повному об'ємі. Більше того, таку поведінку можна застосовувати до окремих об'єктів, а не до усієї системи в цілому.

Простим прикладом є накладення смуги прокрутки до усіх візуальних елементів. Кожен об'єкт, який може прокручуватися, обертається в «прокручуваному» елементі, і при необхідності з'являється полоса прокрутки. Початкові функції елементу (наприклад, рядки статусу) залишаються незмінними.

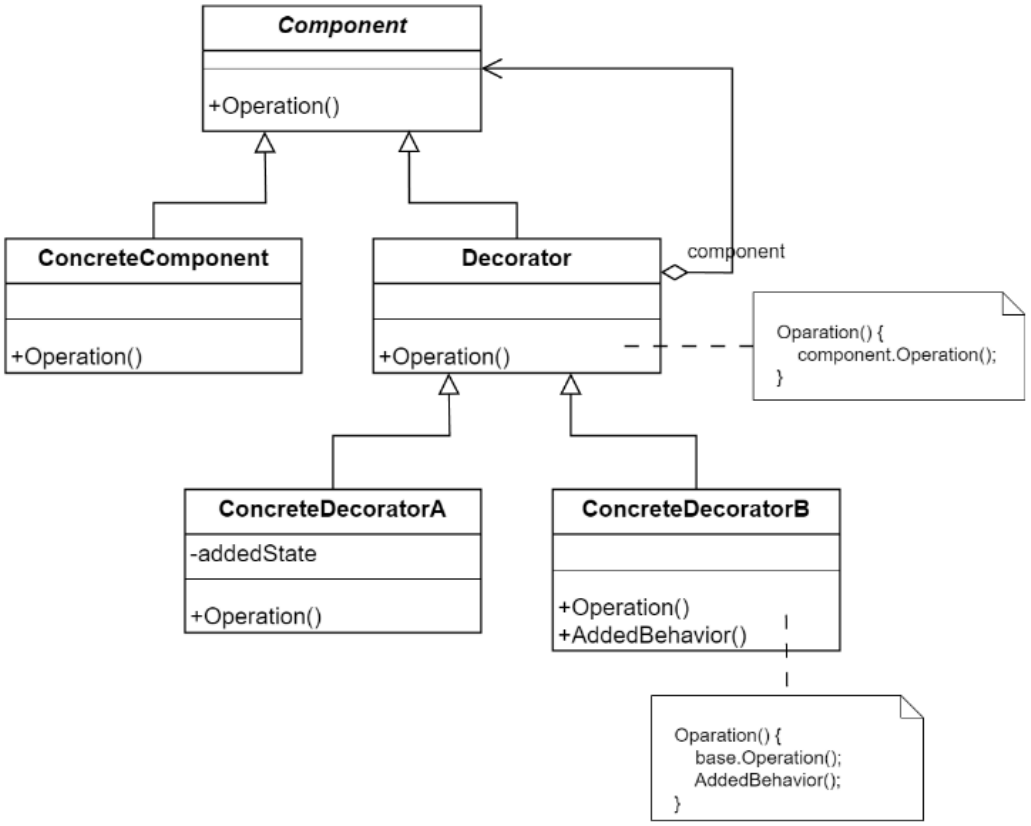

Рис. 2. Структура патерну «Декоратор»

Переваги та недоліки:

+ Дозволяє мати кілька дрібних об’єктів, замість одного об’єкта «на всі випадки життя».

+ Дозволяє додавати обов’язки «на льоту».

+ Більша гнучкість, ніж у спадкування.

- Велика кількість крихітних класів.

- Важко конфігурувати об’єкти, які загорнуто в декілька обгорток одночасно.

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Хід роботи

Реалізація патерну Decorator

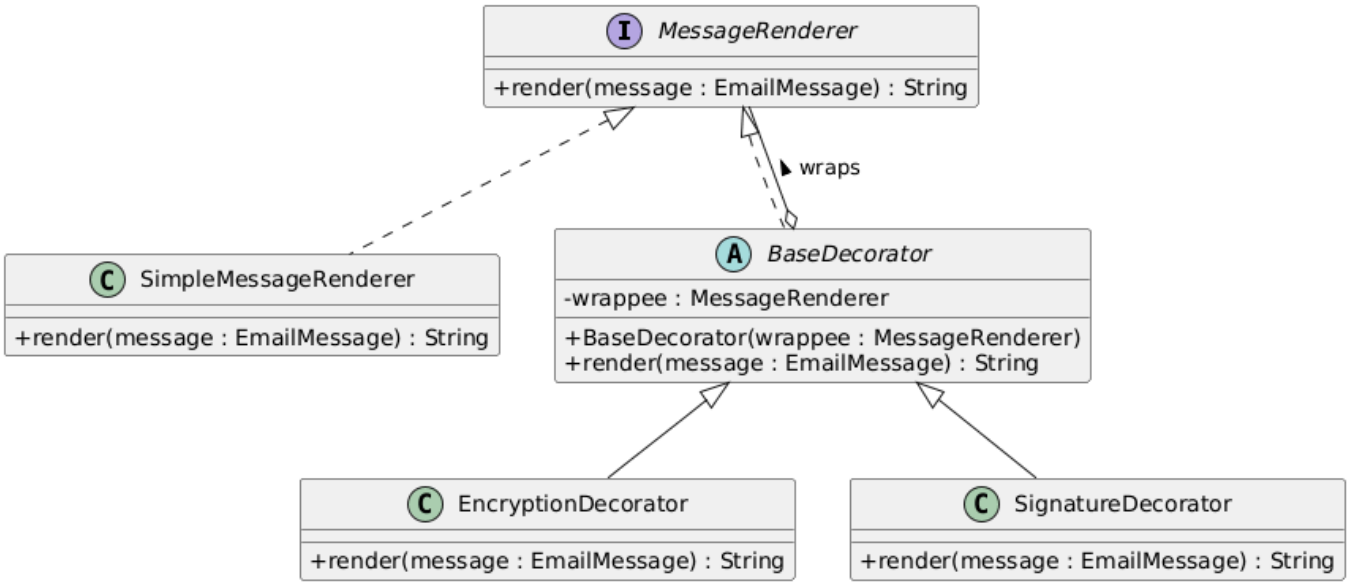

Рис. 2. Діаграма класів з використанням патерну Builder

Фрагмент коду з реалізацією патерну Decorator

MessageRenderer.java

```java
package com.emailclient.service.decorator;

import com.emailclient.model.EmailMessage;

public interface MessageRenderer {
   String render(EmailMessage message);
}
```

SimpleMessageRenderer.java

```java
package com.emailclient.service.decorator;

import com.emailclient.model.EmailMessage;

public class SimpleMessageRenderer implements MessageRenderer {
   @Override
   public String render(EmailMessage message) {
       return message.getBody();
   }
}
```

BaseDecorator.java

```java
package com.emailclient.service.decorator;

import com.emailclient.model.EmailMessage;

public abstract class BaseDecorator implements MessageRenderer {
   protected MessageRenderer wrappee; // Об'єкт, який ми декоруємо

   public BaseDecorator(MessageRenderer wrappee) {
       this.wrappee = wrappee;
   }

   @Override
   public String render(EmailMessage message) {
       return wrappee.render(message);
   }
}
```

EncryptionDecorator.java

```java
package com.emailclient.service.decorator;

import com.emailclient.model.EmailMessage;

public abstract class BaseDecorator implements MessageRenderer {
    protected MessageRenderer wrappee; // Об'єкт, який ми декоруємо

    public BaseDecorator(MessageRenderer wrappee) {
        this.wrappee = wrappee;
    }

    @Override
    public String render(EmailMessage message) {
        return wrappee.render(message);
    }
}
```

SignatureDecorator.java

```java
package com.emailclient.service.decorator;

import com.emailclient.model.EmailMessage;

public class SignatureDecorator extends BaseDecorator {

    public SignatureDecorator(MessageRenderer wrappee) {
        super(wrappee);
    }

    @Override
    public String render(EmailMessage message) {
        String content = super.render(message);

        // Додаємо підпис
        return content + "\n\n-- \nЗ повагою, \nВаш E-mail клієнт (Lab 6)";
    }
}
```

Висновок: я вивчив структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчився застосовувати їх в реалізації програмної системи.