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 проектування»

«2. HTTP-сервер»

Виконав:
студент групи - ІА-32
Непотачев Іван Дмитрович

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Патерни проектування.

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

Тема проєкту: HTTP-сервер (state, builder, factory method, mediator, composite, p2p) Сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), надавати сторінки chtml (html сторінки з додаванням найпростіших C# конструкцій на розсуд студента), вести статистику вхідних запитів, обробку запитів у багатопотоковому/подієвому режимах.

Зміст

Теоретичні відомості

Шаблон «Factory Method»
Призначення: Шаблон «Фабричний метод» визначає інтерфейс для створення об'єктів певного базового типу [6]. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів. Основна ідея полягає саме в заміні об'єктів їх підтипами, що при цьому зберігає ту ж функціональність; інша частина поведінки об'єктів не є інтерфейсною (AnOperation) і дозволяє взаємодіяти із створеними об'єктами як з об'єктами базового типу. Тому шаблон «Фабричний метод» носить ще назву «Віртуальний конструктор».

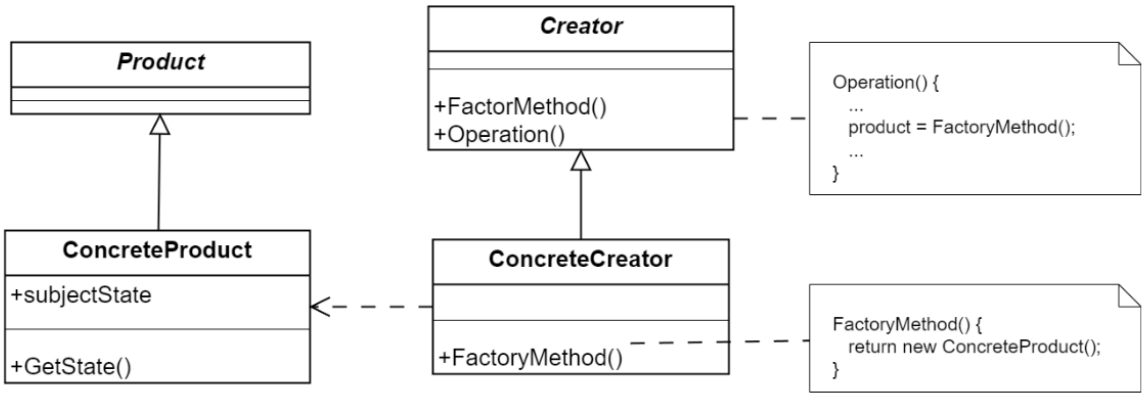


Рисунок 6.2. Структура патерну «Фабричний Метод»

Розглянемо простий приклад. Нехай наш застосунок працює з мережевими драйвер-мі і використовує клас Packet для зберігання даних, що передаються в мережу. Залежно від використовуваного протоколу, існує два перевантаження – TcpPacket, UdpPacket. І відповідно два створюючі об'єкти (TcpCreator, UdpCreator) з фабричним методом (який створює відповідні реалізації). Проте базова функціональність (передача пакету, прийом пакету, заповнення пакету даними) нічим не відрізняється один від одного, відповідно поміщається у базовий клас PacketCreator. Таким чином поведінка системи залишається тим же, проте з'являється можливість підстановки власних об'єктів в процес створення і роботи з пакетами.
Переваги та недоліки:
+ Позбавляє клас від прив’язки до конкретних класів продуктів.
+ Виділяє код виробництва продуктів в одне місце, спрощуючи підтримку коду.
+ Спрощує додавання нових продуктів до програми.
- Може призвести до створення великих паралельних ієрархій класів.

Хід роботи

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.

Реалізація шаблону проєктування для майбутньої системи

Для реалізації HTTP-сервера використано шаблон проєктування Factory Method, оскільки він забезпечує гнучке створення різних типів HTTP-відповідей в залежності від контексту їх використання. Це дозволяє розділити логіку створення успішних та помилкових відповідей та спрощує додавання нових типів відповідей у майбутньому.

Шаблон Factory Method реалізовано у класах HttpResponseCreator, SuccessResponseCreator та ErrorResponseCreator, які відповідають за створення специфічних типів HTTP-відповідей. Такий підхід є особливо ефективним у роботі HTTP-сервера, де необхідно динамічно створювати різні типи відповідей залежно від результатів обробки запитів.

```java
package factory;

import model.HttpResponse;

public abstract class HttpResponseCreator {  4 usages  2 inheritors
    public abstract HttpResponse createResponse(int statusCode, String body);  1 usage  2 implementations
}
```

Рис. 1 – Код класу HttpResponseCreator

```java
package factory;

import builder.HttpResponseBuilder;
import builder.IHttpResponseBuilder;
import model.HttpResponse;

import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

public class SuccessResponseCreator extends HttpResponseCreator {  2 usages
    @Override  1 usage
    public HttpResponse createResponse(int statusCode, String body) {
        IHttpResponseBuilder builder = new HttpResponseBuilder();

        return builder
                .setStatusCode(statusCode)
                .setHeader("Server", "JavaHTTP/1.0")
                .setHeader("Content-Type", "text/html; charset=UTF-8")
                .setHeader("Date", ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME))
                .setBody(body)
                .build();
    }
}
```

Рис. 2 – Код класу SuccessResponseCreator

```java
package factory;

import builder.HttpResponseBuilder;
import builder.IHttpResponseBuilder;
import model.HttpResponse;

import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

public class ErrorResponseCreator extends HttpResponseCreator {  2 usages
    @Override  1 usage
    public HttpResponse createResponse(int statusCode, String body) {
        IHttpResponseBuilder builder = new HttpResponseBuilder();

        return builder
                .setStatusCode(statusCode)
                .setHeader("Server", "JavaHTTP/1.0")
                .setHeader("Content-Type", "text/html; charset=UTF-8")
                .setHeader("Date", ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME))
                .setBody(body)
                .build();
    }
}
```

Рис. 3 – Код класу ErrorResponseCreator

```java
package server;

import factory.ErrorResponseCreator;
import factory.HttpResponseCreator;
import factory.SuccessResponseCreator;
import model.HttpRequest;
import model.HttpResponse;

public class RequestHandler {  3 usages
    public RequestHandler(HttpServer server) {}  1 usage

    public HttpResponse Handle(HttpRequest req) {  1 usage
        String url = req.getUrl();
        String body = switch (url) {
            case "/home" -> "<h1>Welcome to Home!</h1>";
            case "/about" -> "<h1>About us page.</h1>";
            case "/contact" -> "<h1>Contact information here.</h1>";
            default -> "<h1>404 Page Not Found</h1>";
        };

        int statusCode = (url.equals("/home") || url.equals("/about") || url.equals("/contact")) ? 200 : 404;

        HttpResponseCreator creator = (statusCode == 200)
                ? new SuccessResponseCreator()
                : new ErrorResponseCreator();

        return creator.createResponse(statusCode, body);
    }
}
```

Рис. 4 – Код класу RequestHandler

Застосування цього шаблону забезпечує:

1. Розділення логіки створення об'єктів: Класи SuccessResponseCreator та ErrorResponseCreator інкапсулюють специфічну логіку створення успішних та помилкових відповідей. Це дозволяє змінювати процес створення окремих типів відповідей без впливу на основну логіку сервера.
2. Гнучкість та розширюваність: При необхідності додати новий тип відповіді достатньо створити новий клас-нащадок HttpResponseCreator без змін в існуючому коді обробки запитів.
3. Спрощення коду клієнта: Клас RequestHandler не потребує знання про конкретні деталі створення різних типів відповідей. Він просто використовує відповідну фабрику залежно від статусу відповіді.
4. Уніфікація процесу створення: Всі типи відповідей створюються через єдиний інтерфейс createResponse(), що робить код більш читабельним та легшим для підтримки.

У нашому випадку шаблон Factory Method забезпечує чітке розділення відповідальності між логікою обробки запитів та логікою створення відповідей. Це дозволяє легко розширювати функціонал сервера, додаючи нові типи відповідей, та спрощує тестування окремих компонентів системи.

Зображення структури шаблону

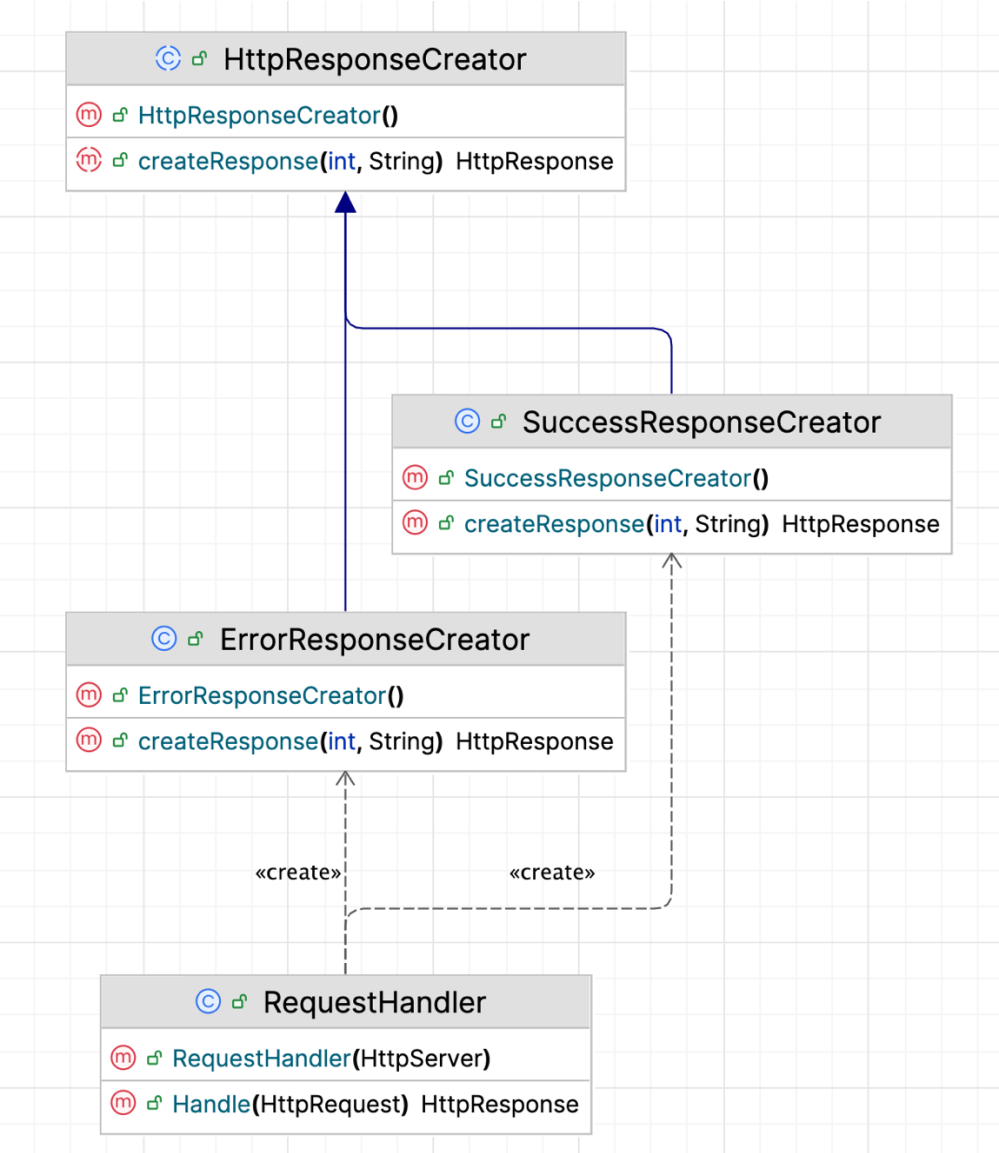


Рис. 2 – Структура шаблону Factory Method

Структура реалізації:

- HttpResponseCreator - абстрактний клас, що визначає фабричний метод createResponse().

- SuccessResponseCreator - конкретна фабрика для створення успішних HTTP-відповідей.

- ErrorResponseCreator - конкретна фабрика для створення помилкових HTTP-відповідей.

- RequestHandler - використовує фабрики для створення відповідних типів відповідей залежно від результату обробки запиту.

Такий підхід особливо ефективний у поєднанні з патерном State, оскільки кожен стан сервера може використовувати відповідні фабрики для створення оптимальних відповідей для поточного стану системи.

Посилання на репозиторій: https://github.com/IvanGodPro24/trpz
Посилання на звіти: https://github.com/IvanGodPro24/trpz_reports

Висновки

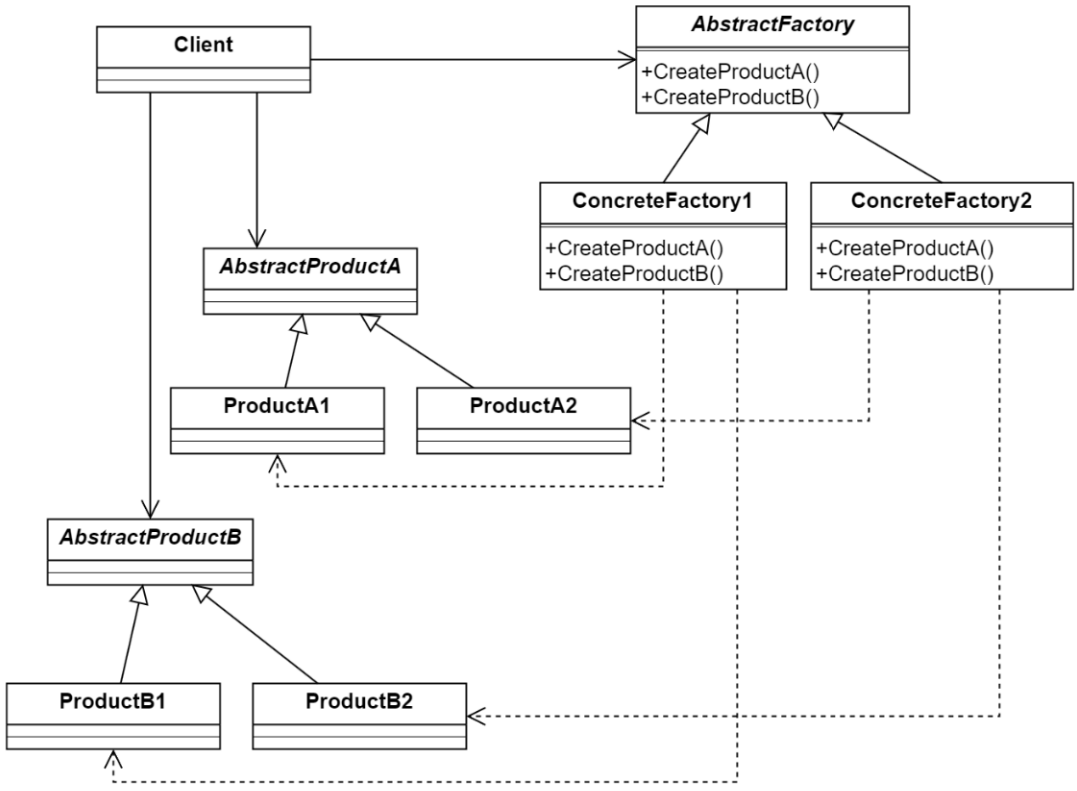
Висновки: під час виконання лабораторної роботи, ми вивчили структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчилися застосовувати їх в реалізації програмної системи.

Питання до лабораторної роботи

1. Яке призначення шаблону «Абстрактна фабрика»?

Шаблон «Абстрактна фабрика» (Abstract Factory) забезпечує створення сімейств взаємопов’язаних об’єктів без зазначення їх конкретних класів.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

AbstractFactory – інтерфейс із методами створення об’єктів.

ConcreteFactory – створює конкретні варіанти продуктів.

AbstractProduct – інтерфейс продукту.

ConcreteProduct – конкретна реалізація продукту.

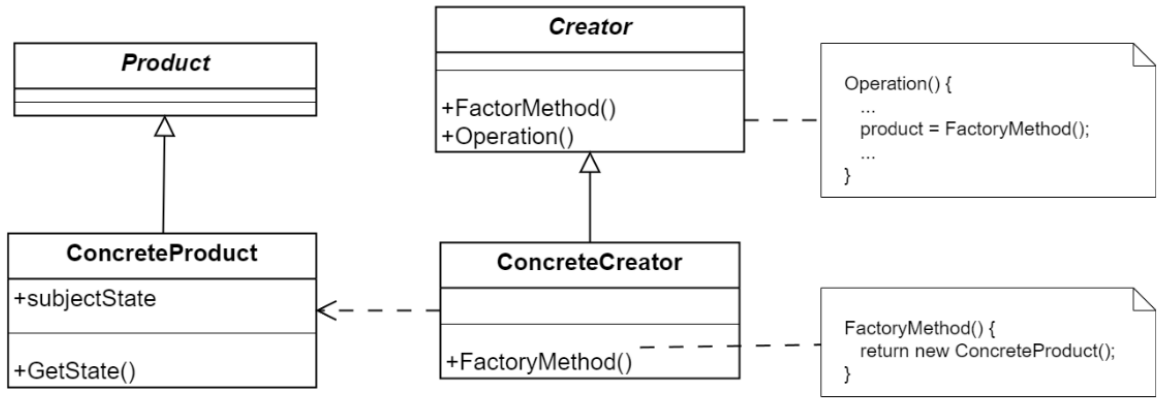
Client – використовує об’єкти, створені фабрикою.

Client звертається до AbstractFactory для створення об’єктів, не знаючи, які конкретні класи створюються.

4. Яке призначення шаблону «Фабричний метод»?

Шаблон «Фабричний метод» (Factory Method) визначає інтерфейс для створення об’єктів, але дозволяє підкласам вирішувати, який клас створювати.

5. Нарисуйте структуру шаблону «Фабричний метод».



6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

Creator – оголошує метод factoryMethod(), який повертає об’єкт типу Product.

ConcreteCreator – перевизначає метод, створюючи конкретний продукт.

Product – інтерфейс або базовий клас для продуктів.

ConcreteProduct – конкретна реалізація продукту.

Creator викликає factoryMethod() для створення продукту, а конкретний підклас визначає, який саме продукт створити.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний

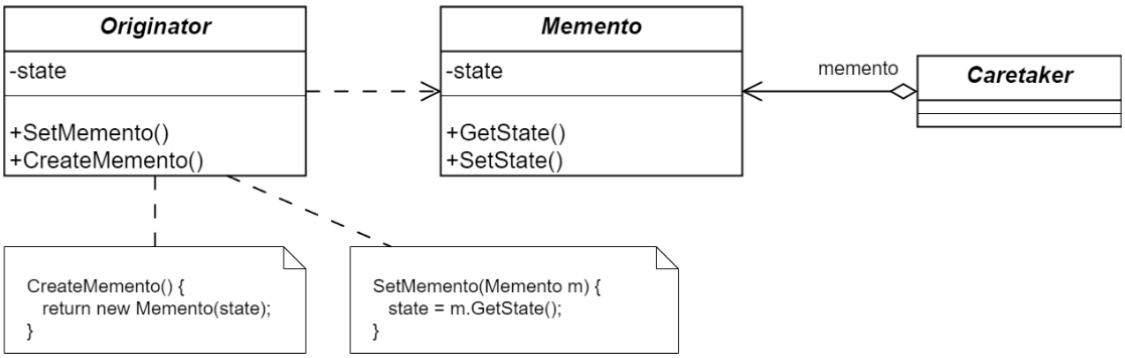
метод»?

Абстрактна фабрика створює сімейства пов’язаних об’єктів, Фабричний метод у свою чергу, створює один тип об’єкта. Абстрактна фабрика містить кілька фабричних методів, а Фабричний метод є лише складовою абстрактної фабрики, тому Абстрактна фабрика має вищий рівень абстракції. Абстрактну фабрику краще використовувати, коли потрібна узгоджена група продуктів, а Фабричний метод, коли створюється один об’єкт через підкласи.

8. Яке призначення шаблону «Знімок»?

Шаблон «Знімок» (Memento) зберігає внутрішній стан об’єкта, щоб потім можна було відновити його без порушення інкапсуляції.

9. Нарисуйте структуру шаблону «Знімок».



10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

Originator – створює знімок власного стану й може його відновити.

Memento – об’єкт, що зберігає стан Originator.

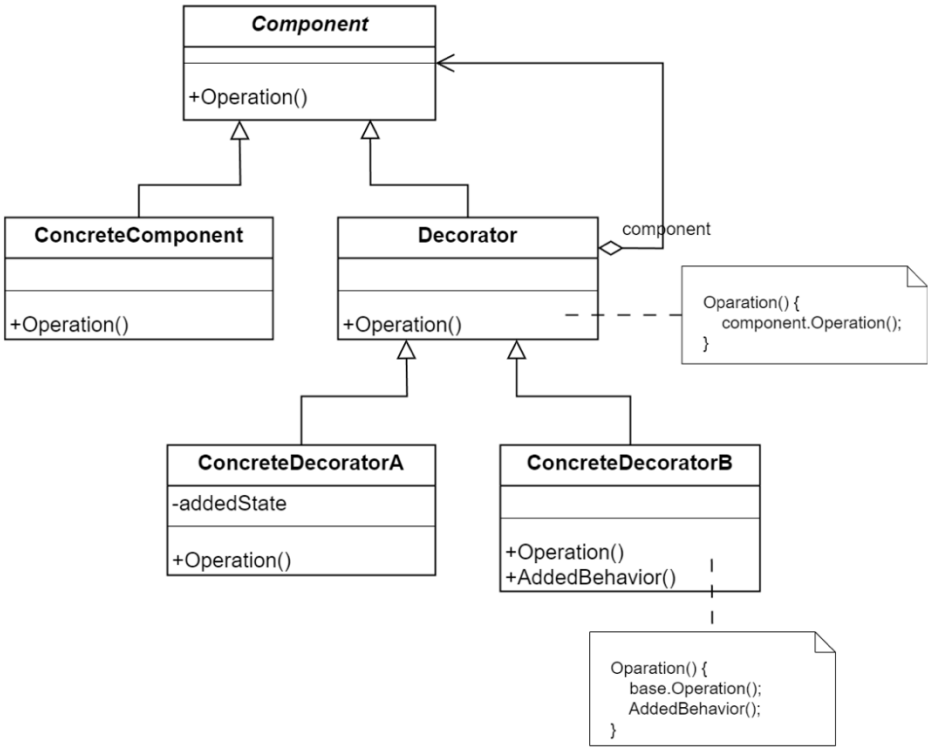
Caretaker – зберігає Memento, але не змінює його.

Originator створює Memento, Caretaker його зберігає, а потім може передати назад для відновлення стану.

11. Яке призначення шаблону «Декоратор»?

Шаблон «Декоратор» (Decorator) дозволяє динамічно додавати об’єктам нову функціональність без зміни їхнього коду або створення підкласів.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

Component – базовий інтерфейс об’єкта.

ConcreteComponent – об’єкт, до якого додаються нові функції.

Decorator – зберігає посилання на компонент і реалізує той самий інтерфейс.

ConcreteDecorator – додає нову поведінку до компонента.

Decorator викликає метод базового компонента й додає додаткову поведінку до або після нього.

14. Які є обмеження використання шаблону «декоратор»?

Основні обмеження використання шаблону «декоратор» включають:

- Велика кількість дрібних класів може ускладнити структуру програми.

- Складно відлагоджувати через вкладеність декораторів.
- Порядок застосування декораторів може впливати на результат.
- Декоратори не завжди легко поєднуються між собою.