Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 6

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав
Студент групи ІА-31:
Губар Б .О.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

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

2. **Теоретичні відомості**

Abstract Factory: Шаблон для створення сімейств пов’язаних об’єктів без вказівки їх конкретних класів. Використовується, коли потрібно забезпечити узгодженість об’єктів одного стилю або типу. Приклад: створення об’єктів різних стилів у грі (стіни, двері, меблі).

Factory Method: Визначає інтерфейс для створення об’єктів, дозволяючи підкласам вирішувати, який саме об’єкт створювати. Підходить для розширення системи новими типами без зміни існуючого коду.

Memento: (Знімок) Дозволяє зберігати і відновлювати стан об’єкта без порушення інкапсуляції. Стан зберігається в об’єкті-знімку, доступному лише вихідному об’єкту.

Observer: (Спостерігач) Визначає залежність «один-до-багатьох»: зміна стану одного об’єкта сповіщає всіх підписаних. Приклад: підписка на канал або повідомлення про зміну даних.

Decorator: (Декоратор) Дозволяє динамічно додавати об’єктам нову функціональність без зміни їхнього коду. Декоратор «обгортає» базовий об’єкт і розширює його поведінку.

3. **Хід роботи**

Тема : CI server (state, command, decorator, mediator, visitor, soa)

Обґрунтування застосування патерну

Для системи реалізовано патерн Decorator (Декоратор) для додання функціоналу логування та звітності до команд збірки. У початковій реалізації команди (CloneRepositoryCommand, CompileCodeCommand) відповідали лише за виконання своєї безпосередньої роботи. Однак, для CI сервера критично важливо відправляти статус виконання ("Start", "Success", "Error") на сервер у реальному часі. Внесення коду HTTP-запитів безпосередньо в класи команд порушило б Single Responsibility Principle (клас відповідав би і за компіляцію, і за мережеву взаємодію) та Open/Closed Principle (довелося б змінювати робочий код). Патерн Decorator дозволив створити клас-обгортку HttpReportDecorator, який перехоплює виклик Execute(), відправляє лог на сервер, а потім делегує виконання вкладеній команді. Це дозволило додати нову поведінку динамічно, не змінюючи код самих команд.

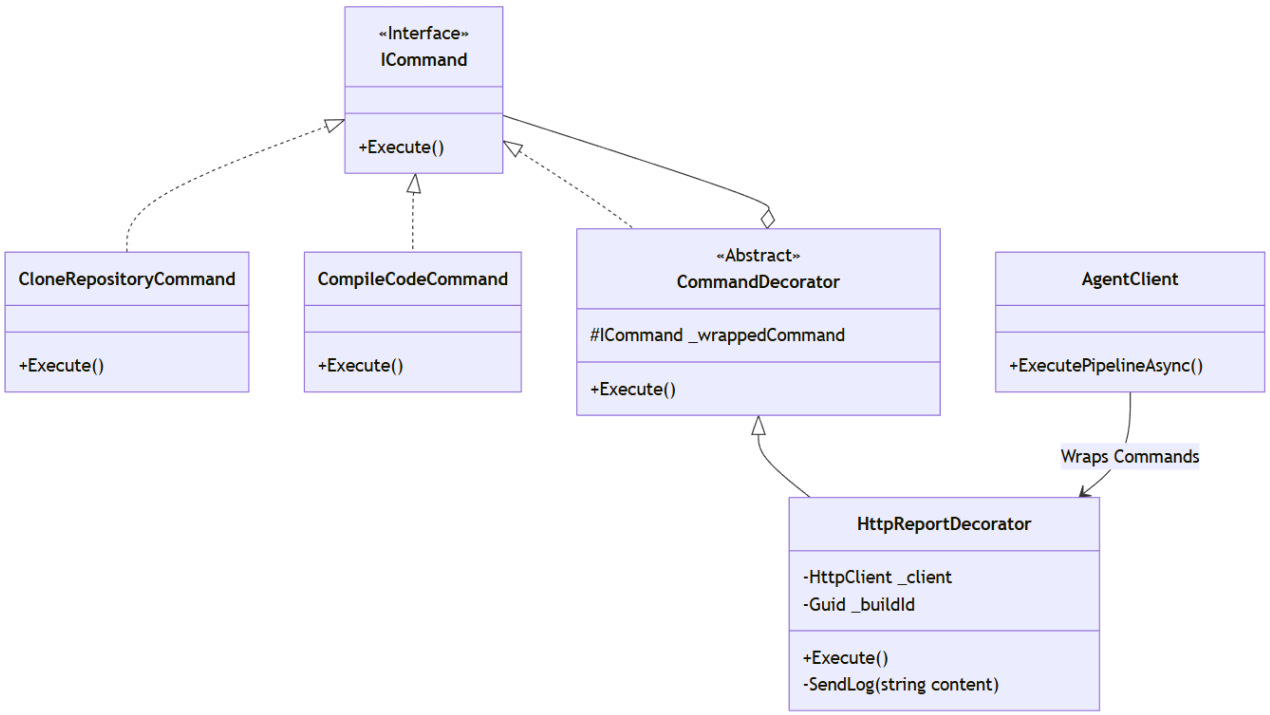


Рисунок 1 - Структура патерну Decorator

ICommand (Component) – спільний інтерфейс для компонентів і декораторів. Гарантує, що декоратор можна використовувати там же, де і звичайну команду.

Concrete Components (CloneRepositoryCommand, CompileCodeCommand) – "чисті" класи, які виконують основну роботу (робота з Git, компіляція). Вони нічого не знають про звіти чи HTTP.

CommandDecorator (Decorator) – абстрактний клас, що зберігає посилання на об'єкт ICommand (_wrappedCommand) і делегує йому виконання методу Execute.

HttpReportDecorator (Concrete Decorator) – конкретна обгортка. У методі Execute вона:

- Відправляє лог про початок роботи.
- Викликає базовий Execute (реальну роботу).
- У разі успіху або помилки відправляє відповідний лог на сервер.

AgentClient (Client) – клієнтський код, який "одягає" команди в декоратори перед додаванням їх у пайплайн.

```
namespace CiServer.Core.Commands;

1 reference
public abstract class CommandDecorator : ICommand
{
    2 references
    protected readonly ICommand _wrappedCommand;

    0 references
    public CommandDecorator(ICommand command)
    {
        _wrappedCommand = command;
    }

    0 references
    public virtual void Execute()
    {
        _wrappedCommand.Execute();
    }
}
```

Рисунок 2 - Абстрактний клас CommandDecorator.cs

```
public HttpReportDecorator(ICommand command, HttpClient client, Guid buildId) : base(command)
{
    _client = client;
    _buildId = buildId;
}

0 references
public override void Execute()
{
    string name = _wrappedCommand.GetType().Name;
    SendLog($"[Start] {name}");

    try
    {
        base.Execute();
        SendLog($"[Success] {name}");
    }
    catch (Exception ex)
    {
        SendLog($"[Error] {name}: {ex.Message}");
        throw;
    }
}

3 references
private void SendLog(string content)
{
    var log = new BuildLog { BuildId = _buildId, Content = content };
    _client.PostAsJsonAsync("/api/agent/log", log);
    Console.WriteLine($"   -> Sent log: {content}");
}
```

Рисунок 3 – Конкретний декоратор

```
var decoratedGit = new HttpReportDecorator(gitCmd, client, build.BuildId);
var decoratedBuild = new HttpReportDecorator(buildCmd, client, build.BuildId);
var decoratedTest = new HttpReportDecorator(testCmd, client, build.BuildId);

pipeline.AddCommand(decoratedGit);
pipeline.AddCommand(decoratedBuild);
pipeline.AddCommand(decoratedTest);
```

Рисунок 4 – Створюємо обгортки

```
-> Sent log: [Success] CloneRepositoryCommand
-> Sent log: [Start] CompileCodeCommand
```

```
[Start] CloneRepositoryCommand
[Success] CloneRepositoryCommand
```

Рисунок 5 – Скрін виконання як в командному рядку, так і на сторінці

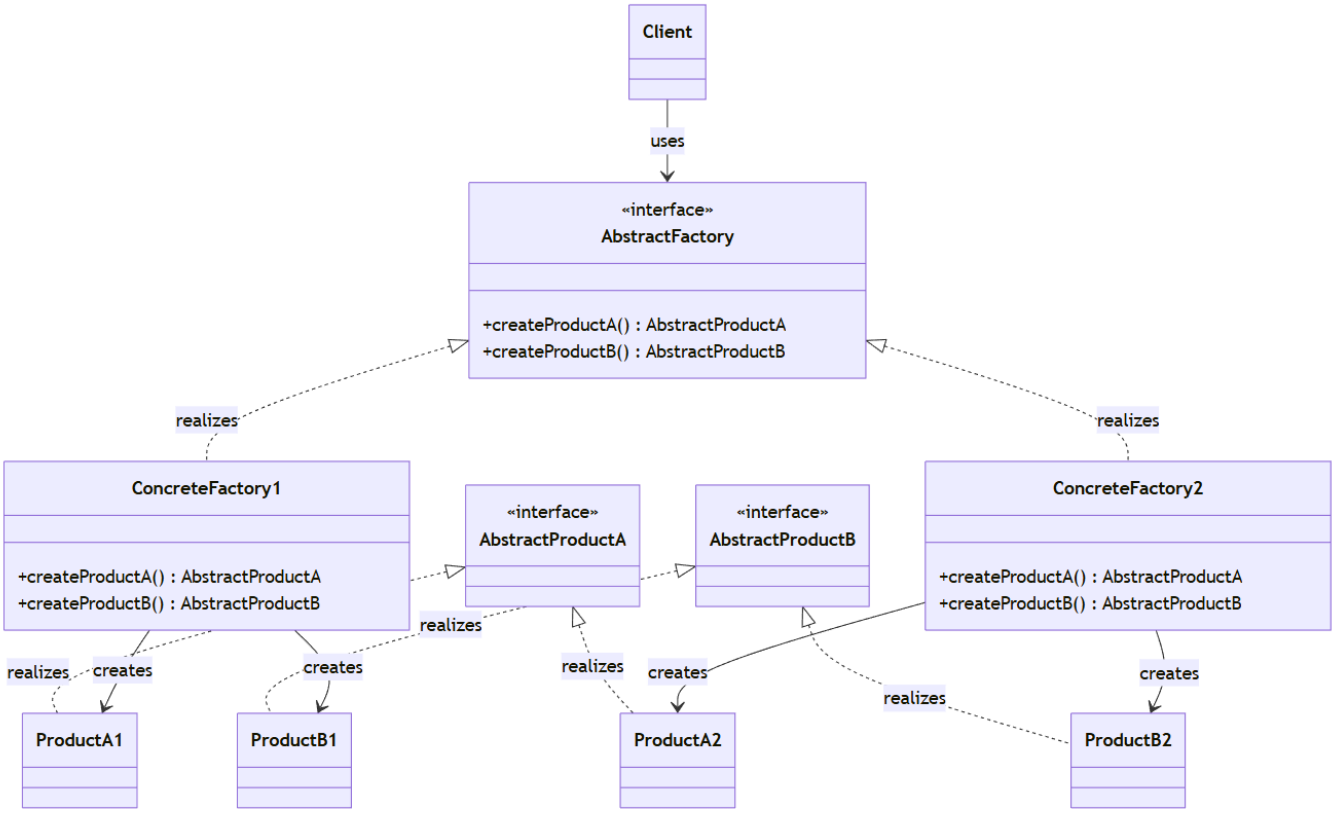
4. Висновок

У ході виконання лабораторної роботи було розглянуто структурні шаблони проєктування. На прикладі системи «CI Server» було реалізовано шаблон Decorator для розширення функціональності команд збірки. Замість модифікації існуючих класів команд, було створено клас HttpReportDecorator, який динамічно додає можливість мережевого логування до будь-якої команди, що реалізує інтерфейс ICommand. Це дозволило чітко розділити бізнес-логіку (виконання процесів ОС) та інфраструктурну логіку (звітність перед сервером). Використання цього патерну значно спростило код системи та підвищило її гнучкість, дозволяючи вмикати або вимикати логування при формуванні пайплайну без зміни структури самих команд.

5. Контрольні питання

1. Яке призначення шаблону «Абстрактна фабрика» Шаблон «Абстрактна фабрика» використовується для створення сімейств об'єктів без вказівки їх конкретних класів.

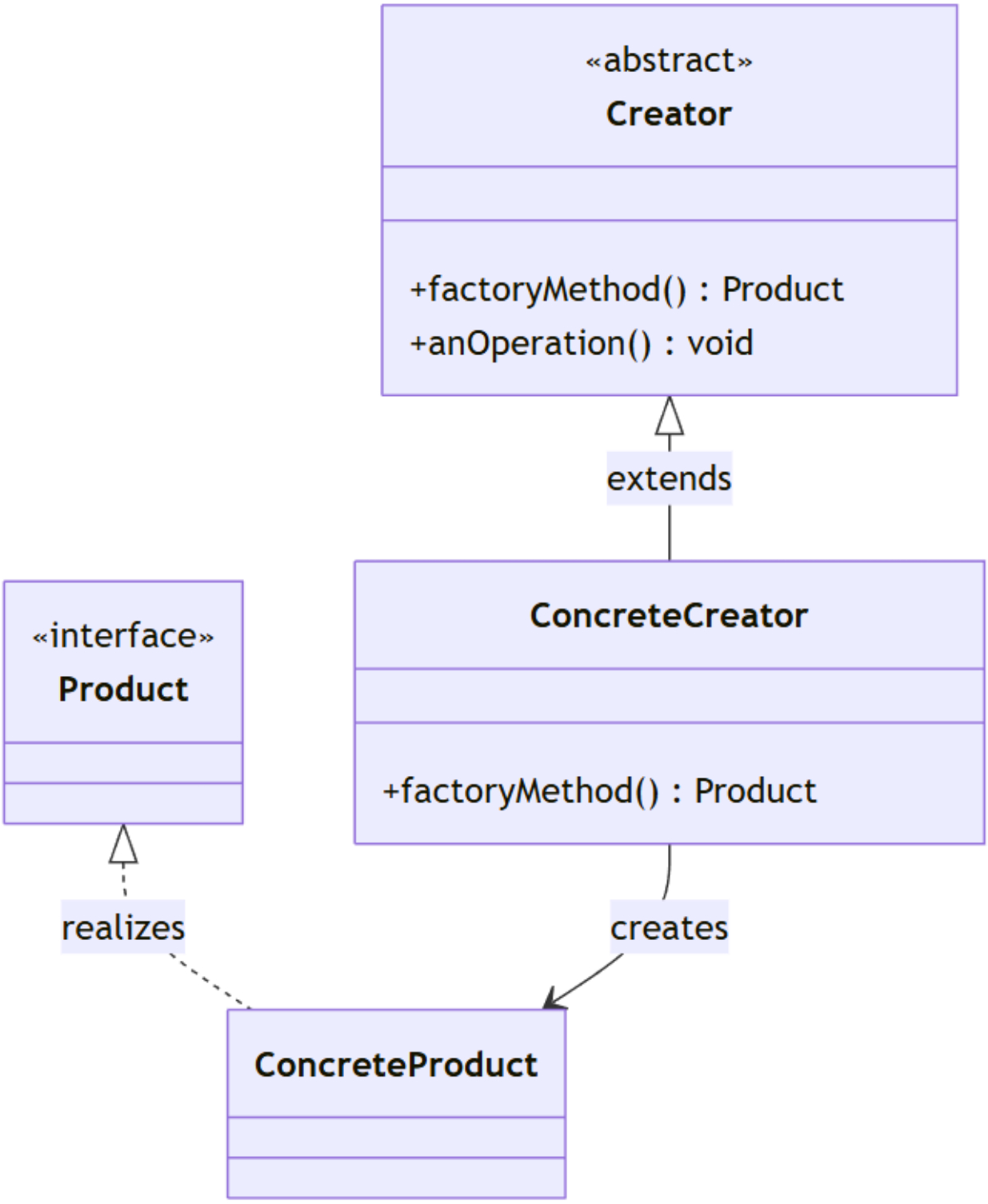
2. Нарисуйте структуру шаблону «Абстрактна фабрика»



3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія? Класи: AbstractFactory, ConcreteFactory, AbstractProductA/B, ConcreteProductA/B, Client. Взаємодія: Client використовує AbstractFactory для створення об'єктів через інтерфейси продуктів; ConcreteFactory реалізує методи створення конкретних продуктів одного сімейства.

4. Яке призначення шаблону «Фабричний метод»? Шаблон «Фабричний метод» визначає інтерфейс для створення об'єктів певного базового типу, залишаючи реалізацію підтипам.

5. Нарисуйте структуру шаблону «Фабричний метод»



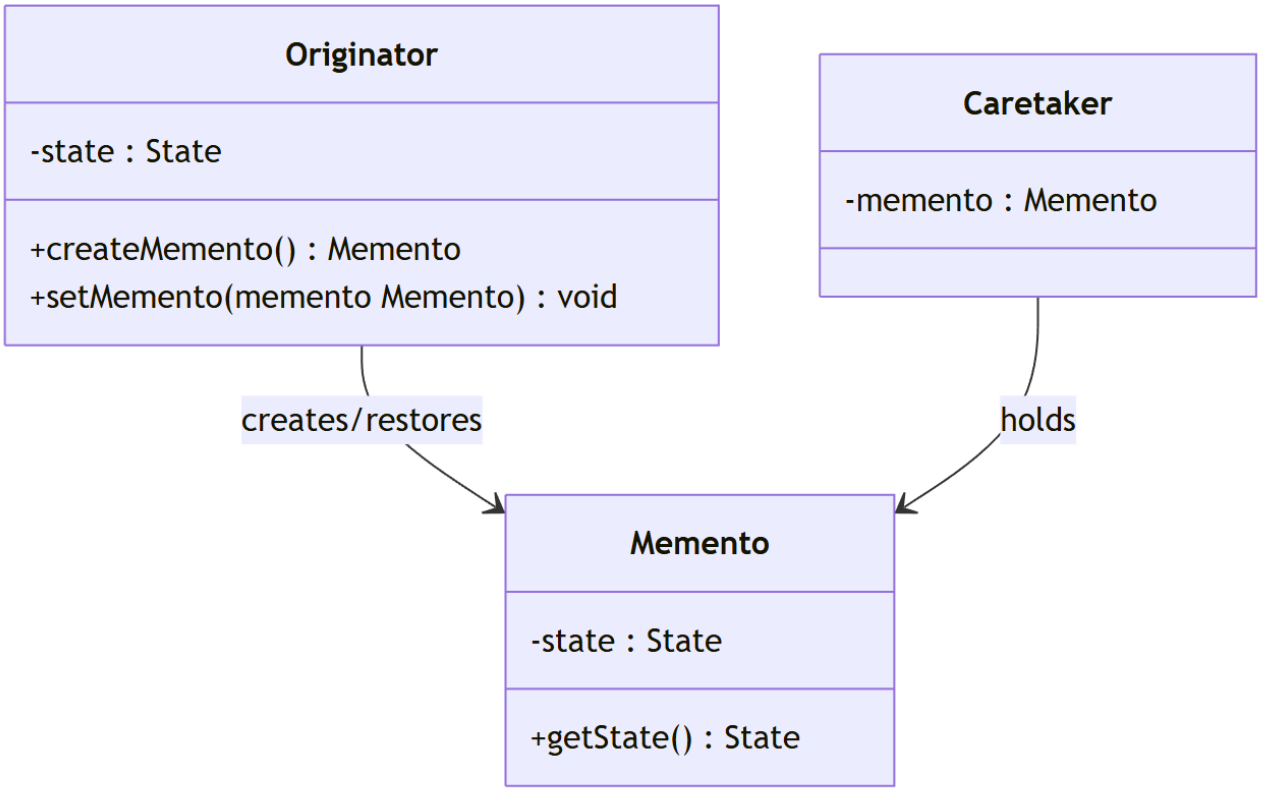
6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія? Класи: Creator, ConcreteCreator, Product, ConcreteProduct. Взаємодія: Creator викликає FactoryMethod() для створення об'єкта; ConcreteCreator повертає конкретний ConcreteProduct.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»? «Абстрактна фабрика» створює сімейства пов’язаних об'єктів через набір методів;

«Фабричний метод» створює один об'єкт через один віртуальний метод, реалізований у підкласах.

8. Яке призначення шаблону «Знімок»? Шаблон «Знімок» використовується для збереження і відновлення стану об'єктів без порушення інкапсуляції.

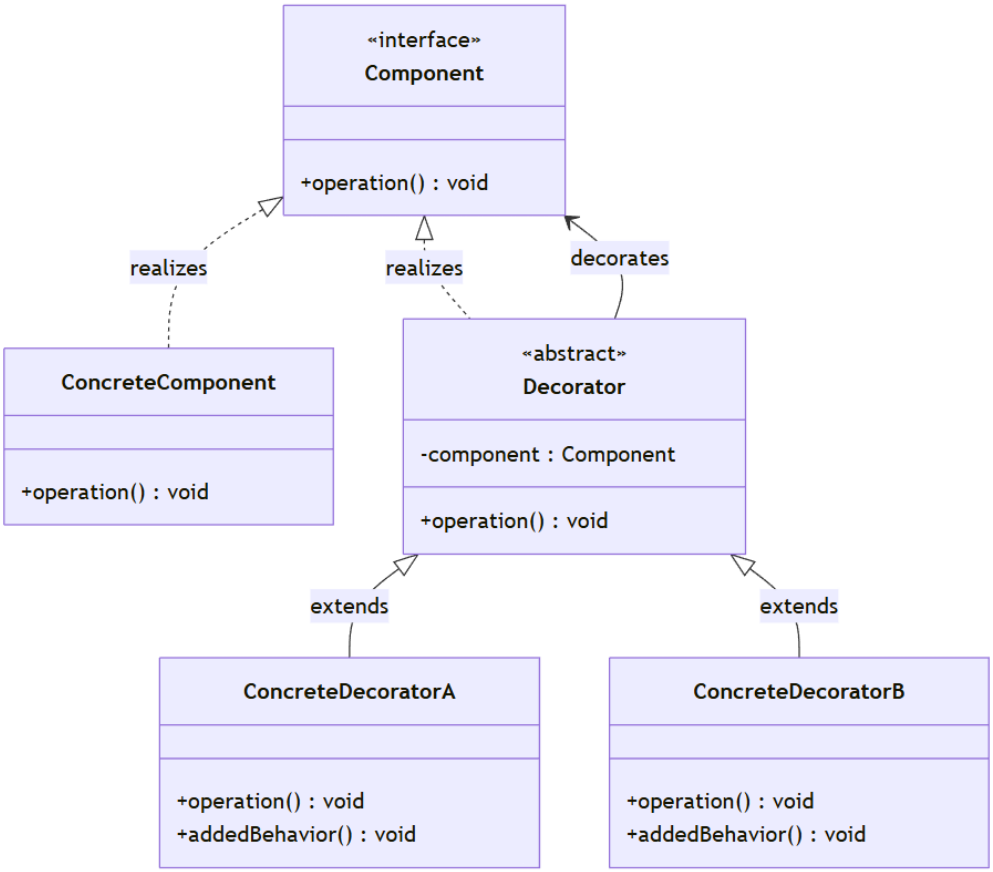
9. Нарисуйте структуру шаблону «Знімок»



10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія? Класи: Originator, Memento, Caretaker. Взаємодія: Originator створює/відновлює Memento; Caretaker зберігає Memento, не маючи доступу до його вмісту.

11. Яке призначення шаблону «Декоратор»? Шаблон «Декоратор» призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми.

12. Нарисуйте структуру шаблону «Декоратор»



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія? Класи: Component, ConcreteComponent, Decorator, ConcreteDecorator. Взаємодія: Decorator обгортає Component, викликає його Operation() і додає власну поведінку.

14. Які є обмеження використання шаблону «Декоратор»? Велика кількість крихітних класів; важко конфігурувати об’єкти, загорнуті в декілька обгорток одночасно.