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»

Варіант - 17

Виконав
Студент групи ІА-31:
Лисенко В. Є

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. Завдання:

1) Ознайомитись з короткими теоретичними відомостями.
2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3) Реалізувати один з розглянутих шаблонів за обраною темою.
4) Реалізувати не менше 3-х класів відповідно до обраної теми.
5) 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.

4. Хід роботи:

Мій варіант:

17. **System activity monitor** (iterator, command, abstract factory, bridge, visitor, SOA) Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

Паттерн Abstract Factory дозволяє:

1. Об'єднати створення пов'язаних об'єктів (сенсорів CPU та RAM) у єдині сімейства.
2. Гарантувати, що система використовує сенсори лише з однієї родини одночасно (неможливо випадково змішати "критичний" CPU зі "звичайним" RAM).
3. Ізолювати клієнтський код (SystemController) від конкретних класів сенсорів. Клієнт працює лише з абстрактними інтерфейсами ICpuSensor та IRamSensor.

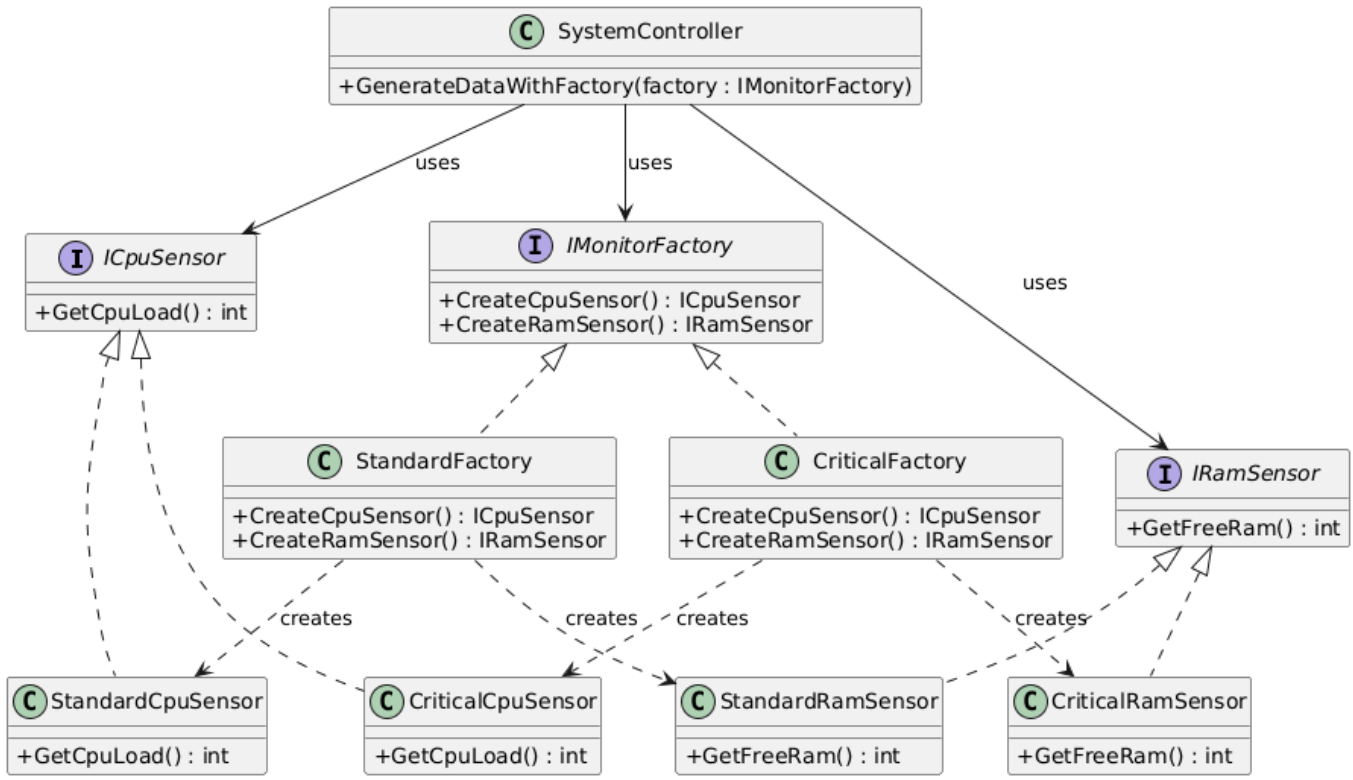


Рисунок 1 - Структура патерну Abstract factory

IMonitorFactory	Абстрактна фабрика	Інтерфейс, що оголошує методи створення абстрактних продуктів (CreateCpuSensor, CreateRamSensor).
StandardFactory / CriticalFactory	Конкретні фабрики	Реалізують методи створення, повертаючи відповідні екземпляри сенсорів для конкретного режиму (звичайного або критичного).
ICpuSensor / IRamSensor	Абстрактні продукти	Інтерфейси для окремих типів об'єктів (сенсорів).
StandardCpuSensor / CriticalCpuSensor	Конкретні продукти	Реалізація сенсорів, що містять специфічну логіку генерації даних (наприклад, Random(10, 40) для звичайного та Random(90, 100) для критичного режиму).
SystemController	Клієнт	Клас, що використовує фабрику для отримання даних. Він не знає, яка саме фабрика йому передана, і працює з продуктами виключно через їхні інтерфейси.

Програмна реалізація:

IMonitorFactory.cs:

```
namespace SystemActivityMonitor.Data.Patterns.AbstractFactory
{
    public interface IMonitorFactory
    {
        ICpuSensor CreateCpuSensor();
        IRamSensor CreateRamSensor();
    }
}
```

ISensors.cs:

```
namespace SystemActivityMonitor.Data.Patterns.AbstractFactory
{
    public interface ICpuSensor
    {
        int GetCpuLoad();
    }

    public interface IRamSensor
    {
        int GetFreeRam();
    }
}
```

CriticalFamily.cs:

```
    public class CriticalCpuSensor : ICpuSensor
    {
        public int GetCpuLoad() => new Random().Next(90, 100);
```

```
    }

    public class CriticalRamSensor : IRamSensor
    {
        public int GetFreeRam() => new Random().Next(50, 500);
    }

    public class CriticalFactory : IMonitorFactory
    {
        public ICpuSensor CreateCpuSensor() => new CriticalCpuSensor();
        public IRamSensor CreateRamSensor() => new CriticalRamSensor();
    }
}
```

SystemController.cs:

```
public void GenerateDataWithFactory(IMonitorFactory factory)
        {
            ICpuSensor cpuSensor = factory.CreateCpuSensor();
            IRamSensor ramSensor = factory.CreateRamSensor();

            using (var db = new MonitorDbContext())
            {
                var admin = db.Users.FirstOrDefault(u => u.Username == "admin");
                if (admin == null) return;

                var session = new Session
                {
                    UserId = admin.Id,
                    MachineName = "FULL-PC",
                    OSVersion = "Windows 11 Pro"
                };
                db.Sessions.Add(session);
                db.SaveChanges();

                var rnd = new Random();
                string[] windows = { "Google Chrome", "Visual Studio", "Google Chrome", "Telegram", "Word" };

                for (int i = 0; i < 10; i++)
                {
                    db.ResourceLogs.Add(new ResourceLog
                    {
                        SessionId = session.Id,
                        CpuLoad = cpuSensor.GetCpuLoad(),
```

```
                RamUsage = ramSensor.GetFreeRam(),
                ActiveWindow = windows[rnd.Next(windows.Length)],
                CreatedAt = DateTime.UtcNow.AddSeconds(i * 2)
            });
        }

        string[] keyActions = { "Ctrl+C", "Ctrl+V", "Enter", "Alt+Tab", "Space" };
        string[] mouseActions = { "Left Click", "Right Click", "Scroll Down", "Double Click" };

        for (int i = 0; i < 5; i++)
        {
            bool isKeyboard = rnd.Next(0, 2) == 0;

            db.InputEvents.Add(new InputEvent
            {
                SessionId = session.Id,
                EventType = isKeyboard ? "Keyboard" : "Mouse",
                Details = isKeyboard
                    ? keyActions[rnd.Next(keyActions.Length)]
                    : mouseActions[rnd.Next(mouseActions.Length)],
                CreatedAt = DateTime.UtcNow.AddSeconds(i * 3)
            });
        }

        db.SaveChanges();
    }
}

}
```

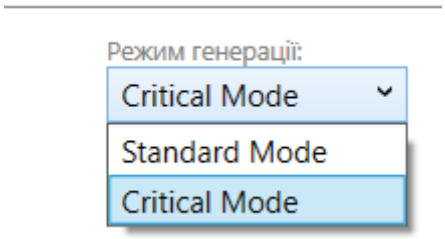


Рисунок 2 – Генерація даних у режимі «Critical Mode»

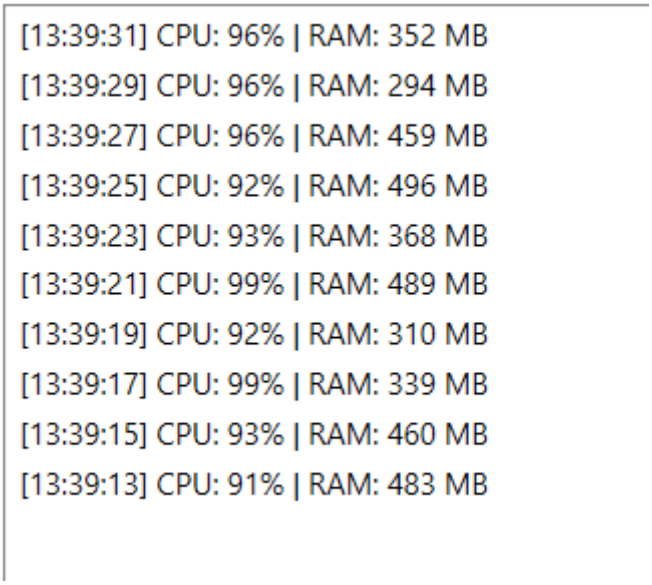


Рисунок 3 – Результат виконання

5. Висновок

У ході виконання лабораторної роботи було вивчено та реалізовано породжуючий патерн проєктування Abstract Factory (Абстрактна фабрика) . Цей шаблон було застосовано для створення механізму генерації тестових даних системного моніторингу в різних режимах роботи.

В рамках системи було розроблено:

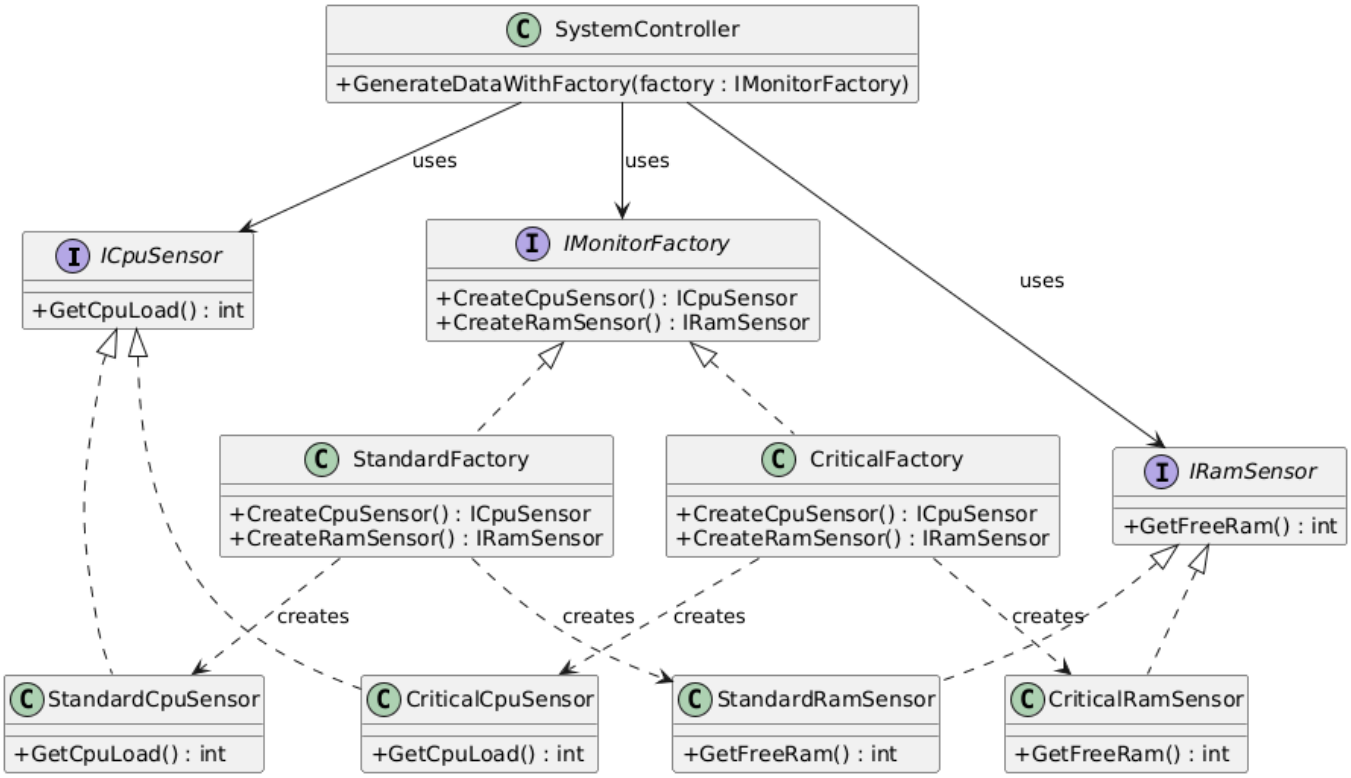
1. Абстрактний інтерфейс фабрики IMonitorFactory та інтерфейси продуктів ICpuSensor, IRamSensor.
2. Дві конкретні реалізації фабрик: StandardFactory (для імітації штатної роботи системи) та CriticalFactory (для імітації стрес-тестів та аварійних ситуацій). 3. Сімейства взаємопов'язаних об'єктів-сенсорів, що забезпечують генерацію узгоджених даних (наприклад, високе навантаження CPU корелює з низьким обсягом вільної пам'яті в критичному режимі).

Використання патерну дозволило ізолювати процес створення об'єктів від клієнтського коду (SystemController) та забезпечити легку заміну сімейств алгоритмів генерації даних під час виконання програми через графічний інтерфейс користувача.

6. Контрольні питання:

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

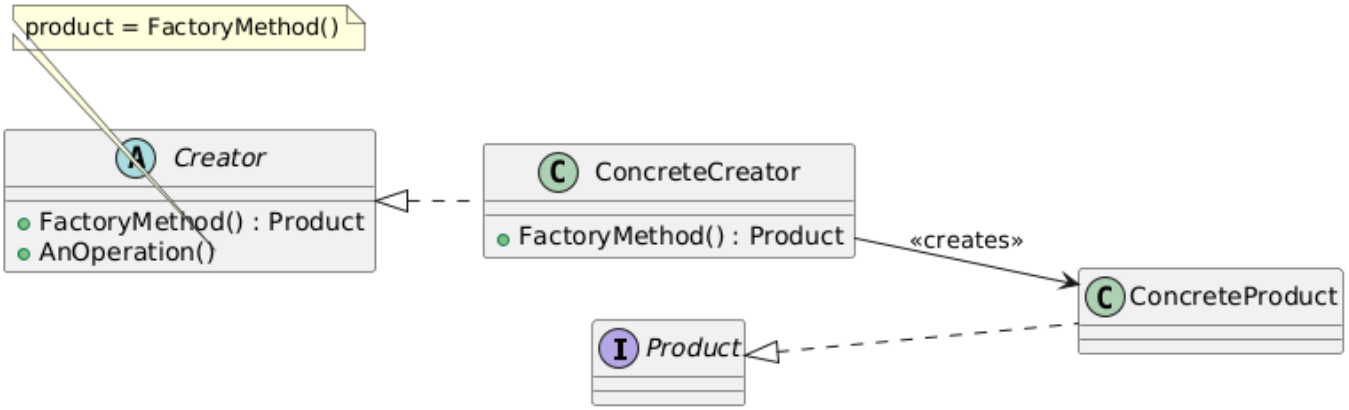
2. Нарисуйте структуру шаблону «Абстрактна фабрика»



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»



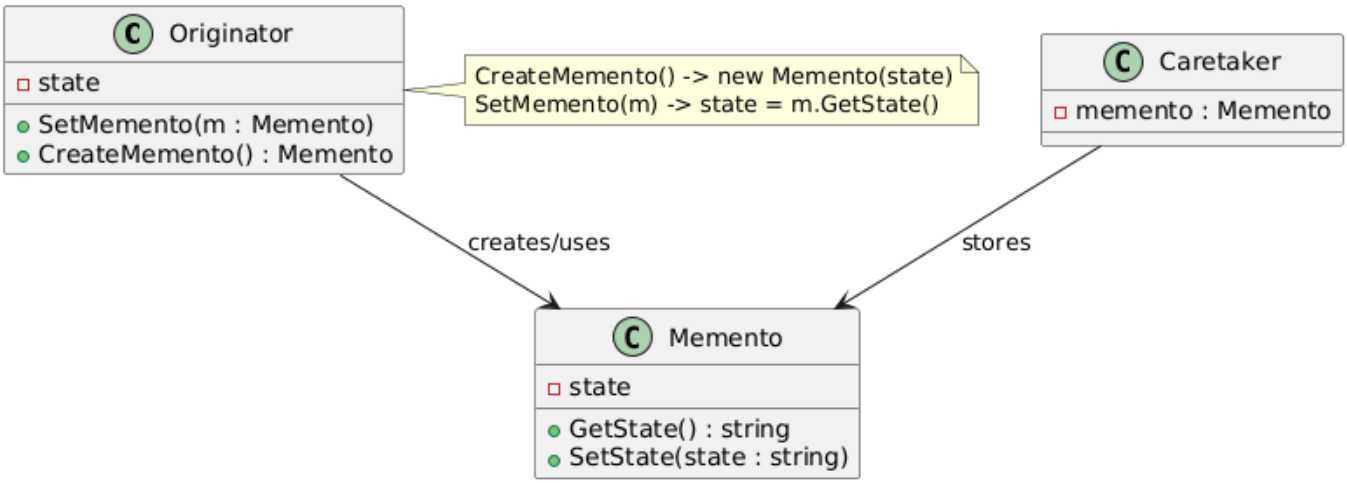
6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія? Класи: Creator, ConcreteCreator, Product, ConcreteProduct. Взаємодія: Creator

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.

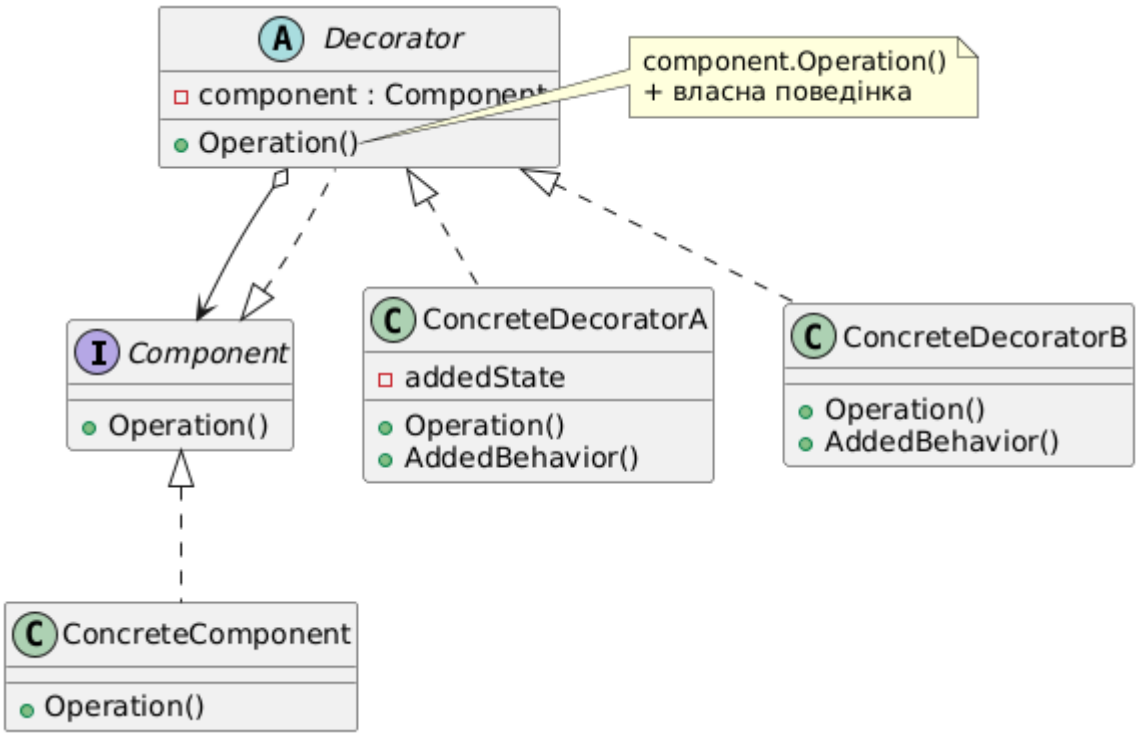
9. Нарисуйте структуру шаблону «Знімок»



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.