Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Лабораторна робота №6

«Патерни проектування»

З дисципліни «**Технології розроблення програмного забезпечення**»

Виконав:

Студент групи ІА-31

Дук М. Д.

Перевірив:

Мягкий М. Ю.

Київ 2025

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

Технічне завдання:

Архіватор (strategy, adapter, factory method, facade, visitor)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, тощо) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

Шаблон «Абстрактна фабрика» (Abstract Factory)

- **Призначення:** Використовується для створення сімейств взаємопов'язаних об'єктів без вказівки їхніх конкретних класів. Дозволяє клієнту працювати з фабрикою абстрактно, забезпечуючи взаємозамінність різних сімейств продуктів.
- **Переваги:**
  - Спрощує створення об'єктів, код стає легшим для розуміння.
  - Гарантує, що об'єкти, створені однією фабрикою, узгоджені між собою (належать до одного стилю).
  - Відокремлює код створення об'єктів від їх використання, роблячи структуру чіткішою.
- **Недоліки:**
  - Додавання нового типу продукту є складним, оскільки вимагає зміни інтерфейсу фабрики та всіх її реалізацій.
  - Збільшує складність коду за рахунок створення додаткових класів.

Шаблон «Фабричний метод» (Factory Method)

- **Призначення:** Визначає інтерфейс для створення об'єктів певного базового типу, дозволяючи підкласам вирішувати, який саме клас інстанціювати. Часто називається «Віртуальний конструктор».
- **Переваги:**
  - Позбавляє основний код від жорсткої прив'язки до конкретних класів продуктів.

    - Локалізує код створення продуктів в одному місці, що спрощує підтримку.
    - Спрощує розширення програми новими видами продуктів.
- **Недоліки:**
    - Може призвести до створення великих паралельних ієрархій класів (для кожного продукту потрібен свій творець).

Шаблон «Знімок» (Memento)

- **Призначення:** Використовується для збереження і відновлення внутрішнього стану об'єкта без порушення інкапсуляції. Дозволяє створювати "контрольні точки" стану об'єкта для подальшого відновлення.
- **Переваги:**
    - Не порушує інкапсуляцію вихідного об'єкта (його стан доступний лише йому самому).
    - Спрощує структуру вихідного об'єкта, знімаючи з нього відповідальність за зберігання історії версій стану.
- **Недоліки:**
    - Вимагає значних обсягів пам'яті, якщо знімки створюються дуже часто.
    - Може спричинити витрати ресурсів, якщо не видаляти застарілі знімки.

Шаблон «Спостерігач» (Observer)

- **Призначення:** Визначає залежність «один-до-багатьох» таким чином, що при зміні стану одного об'єкта всі залежні від нього об'єкти автоматично отримують сповіщення та оновлюються. Відомий також як підписка/розсилка.
- **Переваги:**
    - Реалізує принцип слабкого зв'язку між об'єктами (суб'єкт не знає конкретних класів своїх спостерігачів).
    - Дозволяє додавати та видаляти спостерігачів у будь-який момент часу.
- **Недоліки:**
    - Послідовність отримання повідомлень підписниками не гарантується і не підтримується.

Шаблон «Декоратор» (Decorator)

- **Призначення:** Призначений для динамічного додавання нових функціональних можливостей об'єкту під час виконання програми шляхом "обгортання" його в інший об'єкт.
- **Переваги:**
    - Забезпечує більшу гнучкість, ніж статичне успадкування.
    - Дозволяє додавати обов'язки «на льоту» без зміни коду існуючих класів.
    - Дозволяє комбінувати поведінку за допомогою кількох невеликих об'єктів.
- **Недоліки:**
    - Призводить до появи великої кількості дрібних класів.
    - Важко конфігурувати об'єкти, які загорнуто в декілька обгорток одночасно.

Хід Роботи

Для реалізації механізму створення архівів різних форматів у проекті, з запропонованих патернів, було обрано породжувальний патерн проектування Factory Method (у варіації Simple/Static Factory).

В класичному розумінні GoF (Gang of Four) Фабричний метод реалізується через спадкування творців (Creator -> ConcreteCreator), у моєму ж проекті, використана варіація Parameterized Factory (Параметризована фабрика) або Simple Factory, яка централізує логіку створення об'єктів на основі вхідних даних (розширення файлу).

Необхідність використання цього патерну зумовлена вимогою системи підтримувати декілька форматів архівації (.zip, .tar, .tar.gz) та потенційною необхідністю розширення цього списку в майбутньому.

Сутність реалізації
У додатку реалізовано клас-фабрику ArchiveFactory, який інкапсулює логіку вибору конкретного алгоритму архівації залежно від розширення файлу.

- Абстракція (Product): Інтерфейс IArchiveStrategy визначає єдиний контракт для роботи з будь-яким типом архіву.
- Конкретні продукти (Concrete Products): Класи ZipAdapter та TarAdapter реалізують специфіку роботи з бібліотеками System.IO.Compression та System.Formats.Tar відповідно.
- Фабрика (Creator): Метод CreateFor класу ArchiveFactory приймає шлях до файлу, аналізує розширення та повертає готовий екземпляр потрібного адаптера.

Переваги та недоліки використання в проекті

Переваги:

1. Розширюваність: Додавання нового формату (наприклад, .7z або .rar) вимагає лише створення нового класу-адаптера та додавання одного рядка в switch-вираз фабрики. Код клієнта (ArchiveManager) при цьому не змінюється.

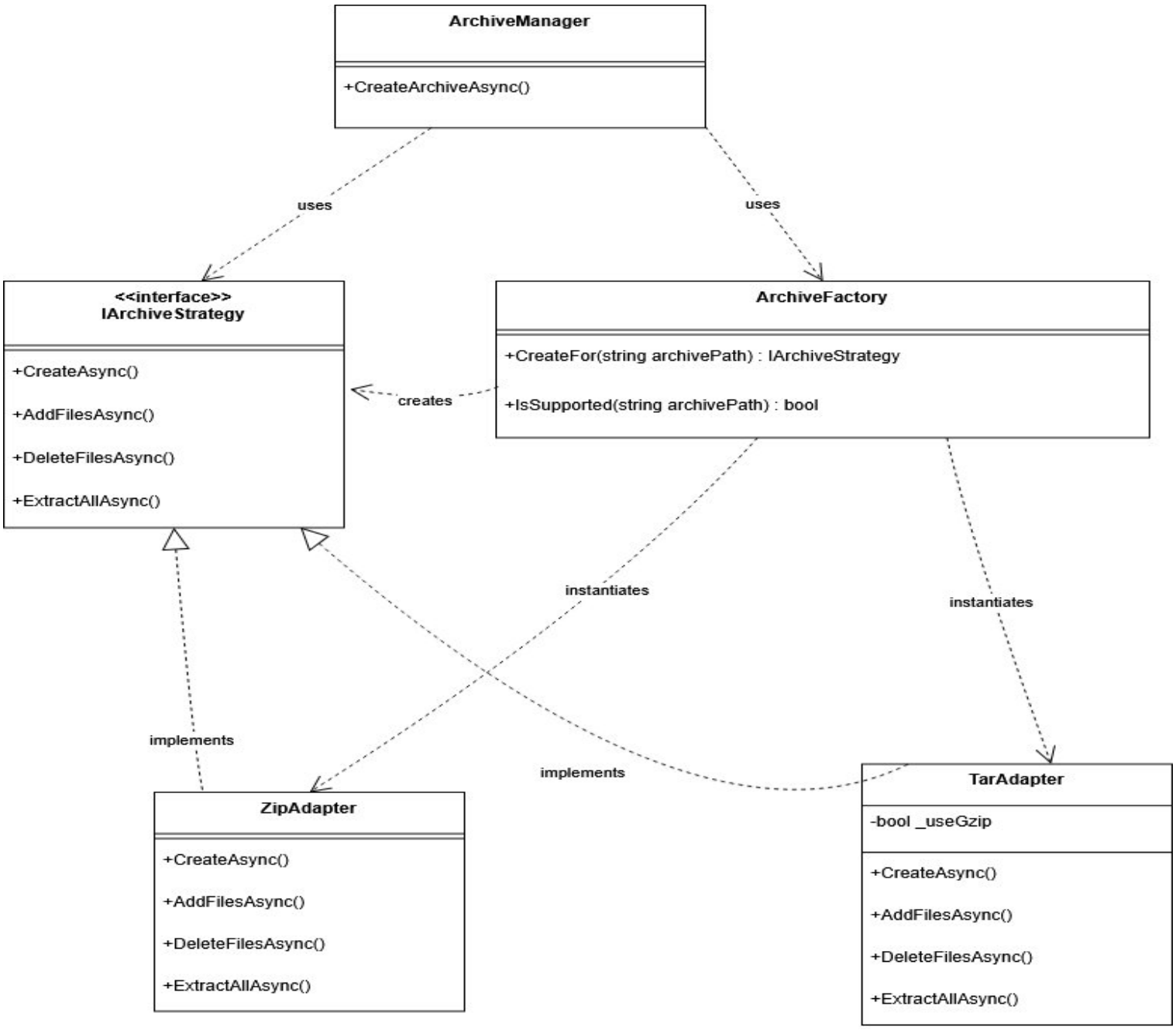
2. Інкапсуляція створення: Клієнтський код не знає про деталі ініціалізації конкретних адаптерів (наприклад, про налаштування GZipStream для .tar.gz).

3. Слабкий зв'язок (Loose Coupling): Система залежить від абстракції IArchiveStrategy, а не від конкретних класів реалізації.

Недоліки:

1. Необхідність перекомпіляції: Для додавання нового типу продукту необхідно вносити зміни у код самої фабрики (порушення Open/Closed Principle у найпростішій реалізації, що є допустимим компромісом для даного масштабу задачі).

Діаграма Класів реалізованого шаблону



Код реалізованого шаблону

```
11 usages  PriMerro23
public class ArchiveFactory
{
    /// <summary>
    /// Створює відповідну реалізацію IArchiveStrategy на основі розширення файлу.
    /// </summary>
    9 usages  PriMerro23
    public static IArchiveStrategy CreateFor(string archivePath)
    {
        var extension :string = Path.GetExtension(archivePath).ToLowerInvariant();

        return extension switch
        {
            ".zip" => new Adapters.ZipAdapter(),
            ".tar" => new Adapters.TarAdapter(useGzip: false),
            ".gz" when archivePath.EndsWith(".tar.gz", StringComparison.OrdinalIgnoreCase)
                => new Adapters.TarAdapter(useGzip: true),
            _ => throw new NotSupportedException($"Archive format '{extension}' is not supported. Supported formats: .zip, .tar, .t
        };
    }

    /// <summary>
    /// Перевіряє, чи підтримується розширення файлу.
    /// </summary>
    1 usage  PriMerro23
    public static bool IsSupported(string archivePath)
    {
        var extension :string = Path.GetExtension(archivePath).ToLowerInvariant();
        return extension == ".zip"
            || extension == ".tar"
            || archivePath.EndsWith(".tar.gz", StringComparison.OrdinalIgnoreCase);
    }

    /// <summary>
    /// Отримує всі підтримувані розширення.
    /// </summary>
    1 usage  PriMerro23
    public static string[] GetSupportedExtensions()
    {
        return new[] { ".zip", ".tar", ".tar.gz" };
    }
}
```

Рис. 1 –ArchiveFactory.cs

```
/// Інтерфейс патерну Стратегія, який визначає уніфіковані операції для всіх форматів архівів.
/// Також служить як цільовий інтерфейс для патерну Адаптер.
/// </summary>
4 usages  2 inheritors  PriMerro23  1 exposing API
public interface IArchiveStrategy
{
    /// <summary>
    /// Створює новий архів із вказаними файлами.
    /// </summary>
    1 usage  2 implementations  PriMerro23
    Task CreateAsync(string archivePath, IEnumerable<string> filePaths, IProgress<int>? progress = null, CancellationToken cancellationToken = default);

    /// <summary>
    /// Додає файли до існуючого архіву.
    /// </summary>
    1 usage  2 implementations  PriMerro23
    Task AddFilesAsync(string archivePath, IEnumerable<string> filePaths, IProgress<int>? progress = null, CancellationToken cancellationToken = default);

    /// <summary>
    /// Видаляє файли з існуючого архіву.
    /// </summary>
    1 usage  2 implementations  PriMerro23
    Task DeleteFilesAsync(string archivePath, IEnumerable<string> fileNamesToDelete, IProgress<int>? progress = null, CancellationToken cancellationToken =

    /// <summary>
    /// Розпаковує всі файли з архіву до вказаної директорії.
    /// </summary>
    1 usage  2 implementations  PriMerro23
    Task ExtractAllAsync(string archivePath, string extractPath, IProgress<int>? progress = null, CancellationToken cancellationToken = default);

    /// <summary>
    /// Отримує всі записи (файли) в архіві.
    /// </summary>
    2 usages  2 implementations  PriMerro23
    Task<IEnumerable<Models.ArchiveEntry>> GetEntriesAsync(string archivePath, CancellationToken cancellationToken = default);

    /// <summary>
    /// Застосовує відвідувача до всіх записів в архіві.
    /// </summary>
    5 usages  2 implementations  PriMerro23
    Task AcceptVisitorAsync(string archivePath, IArchiveVisitor visitor, IProgress<int>? progress = null, CancellationToken cancellationToken = default);
```

Рис. 2 –IArchiveStrategy.cs

```
10 usages  PriMerro23
public class TarAdapter : IArchiveStrategy
{
    private readonly bool _useGzip;

    8 usages  PriMerro23
    public TarAdapter(bool useGzip = false)
    {
        _useGzip = useGzip;
    }

    6+1 usages  PriMerro23
    public async Task CreateAsync(string archivePath, IEnumerable<string> filePaths, IProgress<int>? progress = null, CancellationToken cancellationToken

    1+1 usages  PriMerro23
    public async Task AddFilesAsync(string archivePath, IEnumerable<string> filePaths, IProgress<int>? progress = null, CancellationToken cancellationToke

    1+1 usages  PriMerro23
    public async Task DeleteFilesAsync(string archivePath, IEnumerable<string> fileNamesToDelete, IProgress<int>? progress = null, CancellationToken cance

    1+1 usages  PriMerro23
    public async Task ExtractAllAsync(string archivePath, string extractPath, IProgress<int>? progress = null, CancellationToken cancellationToken = defau

    5+2 usages  PriMerro23
    public async Task<IEnumerable<ArchiveEntry>> GetEntriesAsync(string archivePath, CancellationToken cancellationToken = default){...}

    0+5 usages  PriMerro23
    public async Task AcceptVisitorAsync(string archivePath, IArchiveVisitor visitor, IProgress<int>? progress = null, CancellationToken cancellationToken

    4 usages  PriMerro23
    private void AddFileToTar(TarWriter writer, string filePath, string entryName){...}

    4 usages  PriMerro23
    private void AddDirectoryToTar(TarWriter writer, string directoryPath, string entryBaseName){...}

    /// <summary>
    /// Перевіряє, чи є потік стиснутим за допомогою GZip, шляхом перевірки магічних байтів.
    /// </summary>
    5 usages  PriMerro23
    private bool IsGzipCompressed(Stream stream){...}
}
```

Рис. 3 –TarAdapter

Рис. 5 –ZipAdapter.

Висновок: Під час виконання лабораторної роботи я поглибив свої знання у сфері об'єктно-орієнтованого проєктування та детально ознайомився зі структурою та призначенням патернів проєктування: Abstract Factory, Factory Method, Memento, Observer та Decorator.

На практиці було проаналізовано предметну область розробки архіватора та визначено необхідність роботи з різними форматами файлів (.zip, .tar, .tar.gz). Для вирішення задачі гнучкого створення об'єктів, що відповідають за обробку конкретних форматів, я обрав та реалізував породжувальний патерн **Factory Method** (Фабричний метод).

В ході програмної реалізації мною було розроблено:

1. Інтерфейс продукту (IArchiveStrategy) — який уніфікує методи роботи з архівами (створення, додавання, вилучення файлів) незалежно від їх формату.
2. Конкретні продукти (ZipAdapter, TarAdapter) — класи, що імплементують цей інтерфейс та містять специфічну логіку обробки архівів через бібліотеки System.IO.Compression та System.Formats.Tar.
3. Фабрику (ArchiveFactory) — клас, що містить логіку вибору та інстанціювання необхідного адаптера на основі розширення файлу.

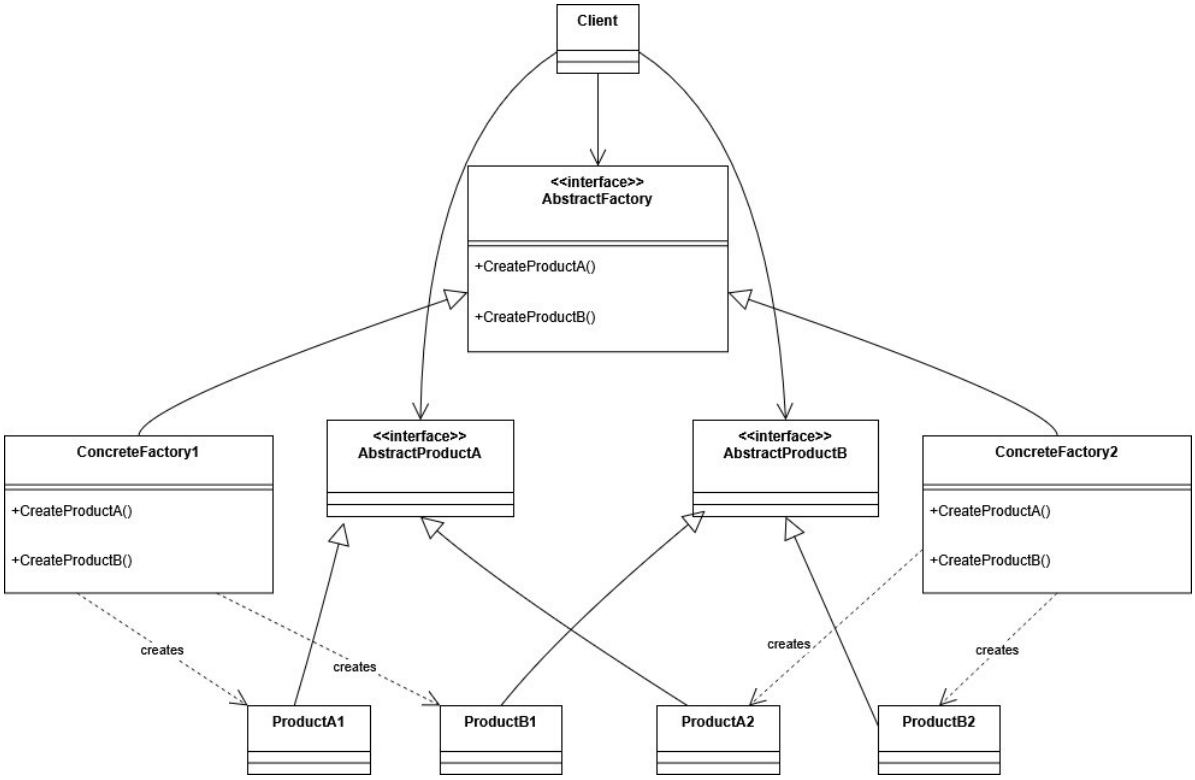
Застосування патерну Factory Method дозволило досягти слабкої зв'язаності (loose coupling) між клієнтським кодом (ArchiveManager) та конкретними алгоритмами архівації. Це забезпечило дотримання принципу відкритості/закритості (Open/Closed Principle): додавання підтримки нових форматів (наприклад, .7z) тепер можливе шляхом створення нового класу адаптера та мінімальних змін у фабриці, без необхідності переписування основної бізнес-логіки програми.

Питання до лабораторної роботи

1. Яке призначення шаблону «Абстрактна фабрика»?

Шаблон «Абстрактна фабрика» призначений для створення сімейств взаємопов'язаних або взаємозалежних об'єктів без вказівки їхніх конкретних класів. Він дозволяє клієнту працювати з об'єктами різних сімейств через єдиний інтерфейс, забезпечуючи підміну цілих сімейств продуктів (наприклад, компонентів інтерфейсу для різних ОС).

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



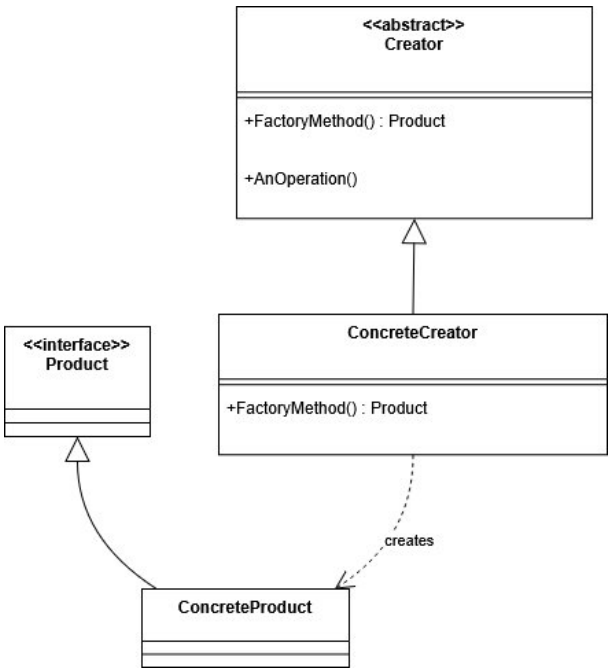
3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

- AbstractFactory: Оголошує інтерфейс для створення продуктів.
- ConcreteFactory: Реалізує створення конкретних продуктів певної серії.
- AbstractProduct: Оголошує інтерфейс для типу продукту.
- ConcreteProduct: Визначає об'єкт, що створюється відповідною фабрикою.
- Client: Використовує виключно інтерфейси AbstractFactory та AbstractProduct, що дозволяє підміняти конкретні фабрики під час виконання.

4. Яке призначення шаблону «Фабричний метод»?

Призначенням є визначення інтерфейсу для створення об'єкта, при цьому класам-спадкоємцям надається право вирішувати, який саме клас інстанціювати. Це дозволяє делегувати створення об'єктів підкласам.

5. Нарисуйте структуру шаблону «Фабричний метод».



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

- Product: Інтерфейс об'єктів, які створюються.
- ConcreteProduct: Реалізація інтерфейсу Product.
- Creator: Оголошує фабричний метод, що повертає об'єкт типу Product.
- ConcreteCreator: Перевизначає фабричний метод для повернення екземпляра ConcreteProduct.

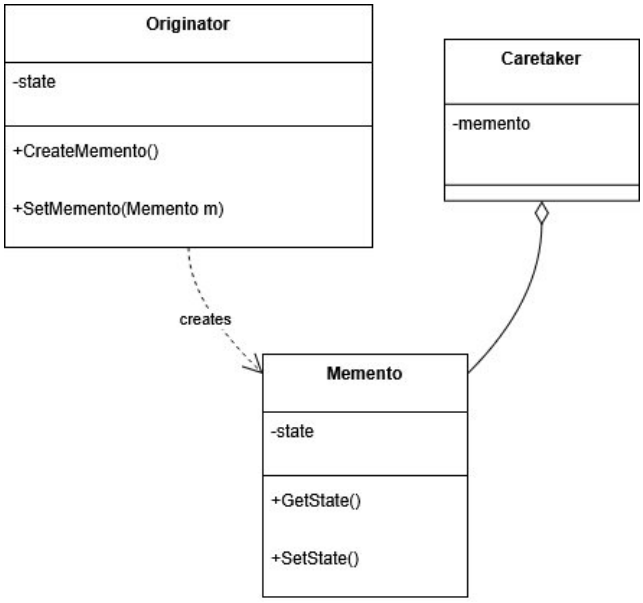
7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

- Масштаб: Абстрактна фабрика створює сімейства продуктів, а Фабричний метод — один продукт.
- Реалізація: Абстрактна фабрика зазвичай використовує композицію (об'єкт фабрики передається клієнту), а Фабричний метод — спадкування (підкласи вирішують, що створити).
- Методи Абстрактної фабрики часто реалізуються за допомогою Фабричних методів.

8. Яке призначення шаблону «Знімок»?

Збереження і відновлення стану об'єкта без порушення його інкапсуляції.

9. Нарисуйте структуру шаблону «Знімок».



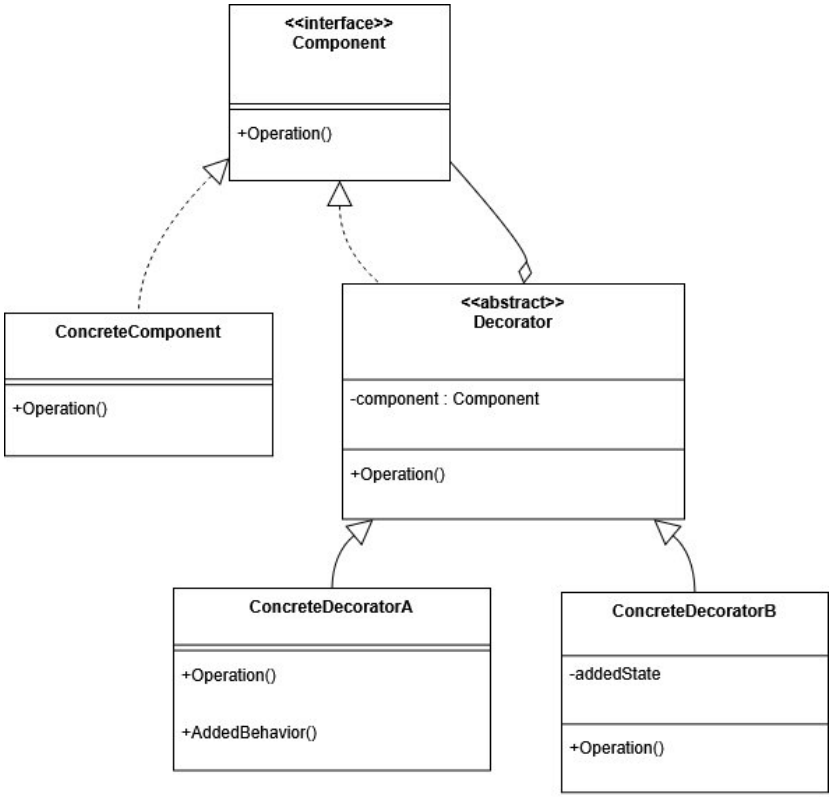
10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

- Originator: Об'єкт, стан якого зберігається. Створює Memento.
- Memento: Зберігає стан Originator-а. Захищає дані від доступу інших об'єктів.
- Caretaker: Відповідає за зберігання знімка, але не має доступу до його внутрішніх даних.

11. Яке призначення шаблону «Декоратор»?

Динамічне додавання нових обов'язків об'єкту. Це гнучка альтернатива породженню підкласів для розширення функціональності.

12. Нарисуйте структуру шаблону «Декоратор».



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

- Component: Інтерфейс для об'єктів, до яких можна динамічно додавати обов'язки.
- ConcreteComponent: Базовий об'єкт, який декорується.
- Decorator: Зберігає посилання на Component і реалізує той самий інтерфейс.
- ConcreteDecorator: Додає функціональність до компонента.

14. Які є обмеження використання шаблону «декоратор»?

- Утворюється велика кількість дрібних об'єктів, що ускладнює розуміння та налагодження системи.
- Складно конфігурувати об'єкт, загорнутий у багато шарів декораторів (важливий порядок обгортання).
- Ідентичність декорованого об'єкта не дорівнює ідентичності оригінального об'єкта.