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 № 4

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Вступ до паттернів проектування»

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

Перевірив: Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

2. **Теоретичні відомості:**

Singleton: Забезпечує єдиний екземпляр класу з глобальним доступом. Використовується для унікальних ресурсів (наприклад, конфігураційні файли), але вважається антипатерном через глобальний стан.
Iterator: Дозволяє послідовний доступ до елементів колекції без розкриття її внутрішньої структури. Підтримує різні методи обходу (наприклад, у глибину, випадково).
Proxy: Діє як замінник або заглушка для іншого об’єкта, додаючи функціонал (наприклад, ледаче завантаження, контроль доступу). Зменшує кількість запитів до зовнішніх сервісів (наприклад, оптимізація DocuSign).
State: Дозволяє змінювати поведінку об’єкта залежно від його стану (наприклад, типи карток або режими системи). Використовує окремі класи для кожного стану.
Strategy: Уможливлює заміну алгоритмів поведінки об’єкта (наприклад, методи сортування чи маршрути). Відокремлює логіку алгоритмів від контексту для гнучкості.

3. **Хід роботи:**

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

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.

Обґрунтування застосування патерну

Для системи реалізовано патерн State для керування життєвим циклом збірки (Build). Оскільки збірка може перебувати в різних станах (Pending, Running, Success, Failed), і поведінка системи (наприклад, можливість скасування або запуску) залежить від поточного стану, використання великих конструкцій switch-case або if-else у класі Build призвело б до заплутаного коду. Патерн State

дозволяє інкапсулювати поведінку кожного статусу в окремий клас, роблячи переходи між станами (Transitions) прозорими та керованими.

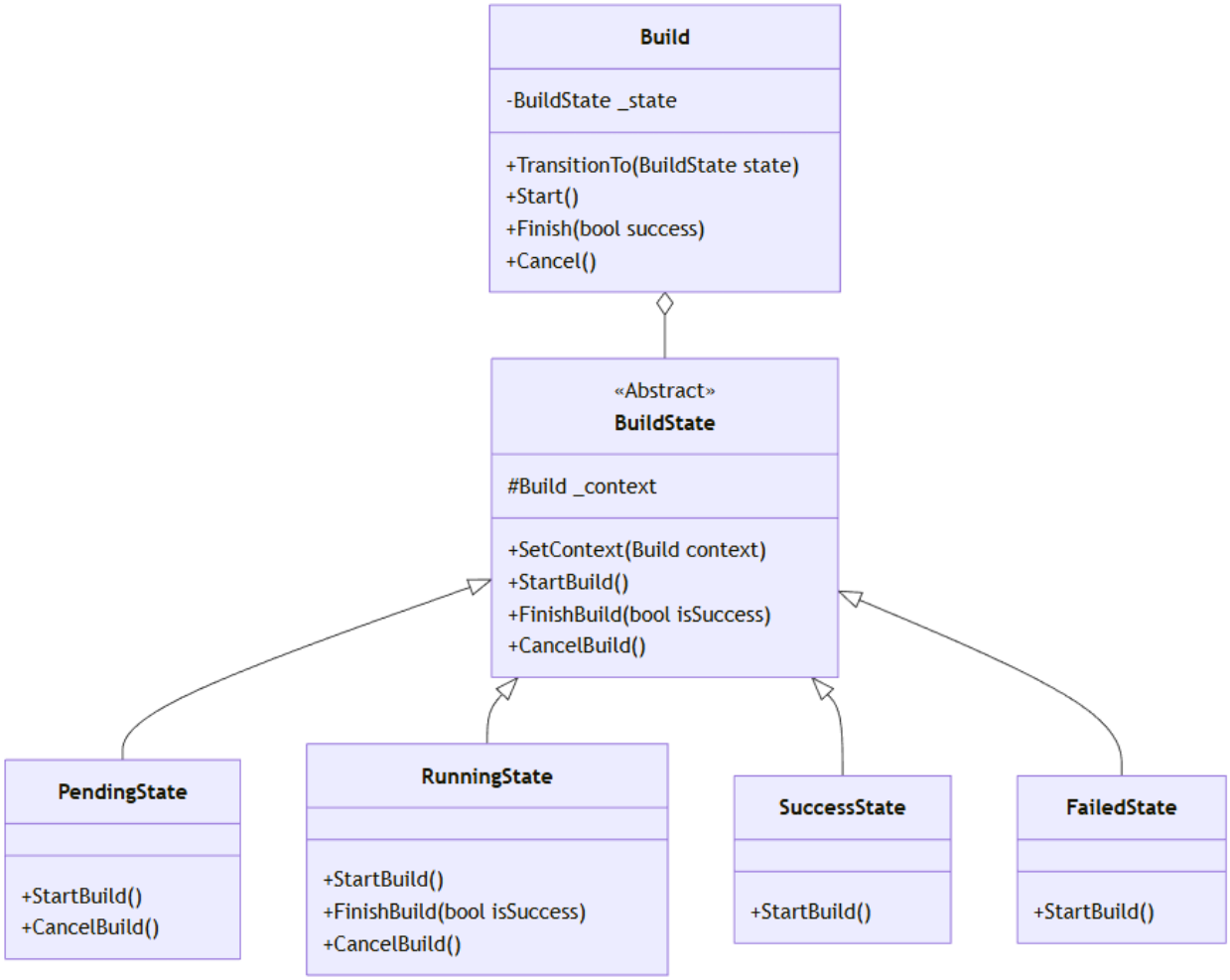


Рисунок 1 - Структура патерну State

Build (Context) – це центральна сутність, яка зберігає посилання на поточний стан (_state) і делегує йому виконання операцій (Start, Finish, Cancel). Вона також містить метод TransitionTo, що дозволяє змінювати стан під час виконання.
BuildState (State) – це абстрактний клас, який визначає інтерфейс для всіх можливих станів збірки. Він задає методи StartBuild, FinishBuild та CancelBuild, які повинні бути реалізовані або перевизначені в конкретних станах.
Concrete States (Pending, Running, Success, Failed) – це конкретні реалізації станів.

PendingState: Початковий стан. Дозволяє перехід у RunningState (початок збірки) або FailedState (скасування).

RunningState: Активний стан. Дозволяє завершити збірку успішно (SuccessState) або з помилкою (FailedState). Блокує повторний запуск.

SuccessState / FailedState: Фінальні стани. Вони забороняють будь-які подальші дії (збірка вже завершена), гарантуючи цілісність даних.

```
public void RestoreState()
{
    switch (Status)
    {
        case BuildStatus.Pending:
            _state = new PendingState();
            break;
        case BuildStatus.Running:
            _state = new RunningState();
            break;
        case BuildStatus.Success:
            _state = new SuccessState();
            break;
        case BuildStatus.Failed:
            _state = new FailedState();
            break;
        default:
            _state = new PendingState();
            break;
    }
    _state.SetContext(this);
}
```

```
public void TransitionTo(BuildState state)
{
    _state = state;
    _state.SetContext(this);

    if (_state is PendingState)
        Status = BuildStatus.Pending;
    else if (_state is RunningState)
        Status = BuildStatus.Running;
    else if (_state is SuccessState)
        Status = BuildStatus.Success;
    else if (_state is FailedState)
        Status = BuildStatus.Failed;
}
```

Рисунок 2 - Build.cs (Контекст)

```
namespace CiServer.Core.States;

0 references
public abstract class BuildState
{
    1 reference
    protected Build _context;

    0 references
    public void SetContext(Build context)
    {
        _context = context;
    }

    0 references
    public abstract void StartBuild();
    0 references
    public abstract void FinishBuild(bool isSuccess);
    0 references
    public abstract void CancelBuild();
```

Рисунок 3 – Абстрактний клас BuildState

```
namespace CiServer.Core.States;

0 references
public class PendingState : BuildState
{
    0 references
    public override void StartBuild()
    {
        Console.WriteLine("Перехід зі стану PENDING у RUNNING...");
        _context.TransitionTo(new RunningState());
    }

    0 references
    public override void FinishBuild(bool isSuccess)
    {
        Console.WriteLine("Помилка: Не можна завершити збірку, яка ще не почалася.");
    }

    0 references
    public override void CancelBuild()
    {
        Console.WriteLine("Збірку скасовано до початку.");
        _context.TransitionTo(new FailedState());
    }
}
```

Рисунок 4 – приклад класу (PendingState)

```
!!! Pipeline failed on command HttpReportDecorator: Compilation failed.
[JOB FINISHED] Status: Failed
```

Build ID: 59a74616-1f33-4e1d-bb85-8fbc23c1deb
Status: Failed

Рисунок 5 – Результат виконання

4. Висновок

У ході виконання лабораторної роботи було розглянуто базові шаблони проєктування, зокрема Singleton, Iterator, Proxy, State та Strategy. Отримані знання допомогли зрозуміти їхню роль у створенні гнучкої та масштабованої архітектури.

На прикладі системи «CI Server» було реалізовано шаблон State для керування життєвим циклом збірки (Build). Логіка переходів між статусами (Pending, Running, Success, Failed) винесена в окремі класи станів, що дозволило позбутися громіздких умовних конструкцій у класі контексту. Такий підхід робить поведінку системи передбачуваною, спрощує додавання нових статусів та гарантує, що неможливі переходи (наприклад, скасування вже завершеної збірки) будуть коректно

оброблені. Це підтверджує ефективність використання патернів станів для моделювання складних бізнес-процесів.

5. Контрольні питання:

1. Що таке шаблон проєктування?

Шаблон проєктування — це універсальне рішення для типових проблем у розробці ПЗ, яке описує структуру та взаємодію об’єктів.

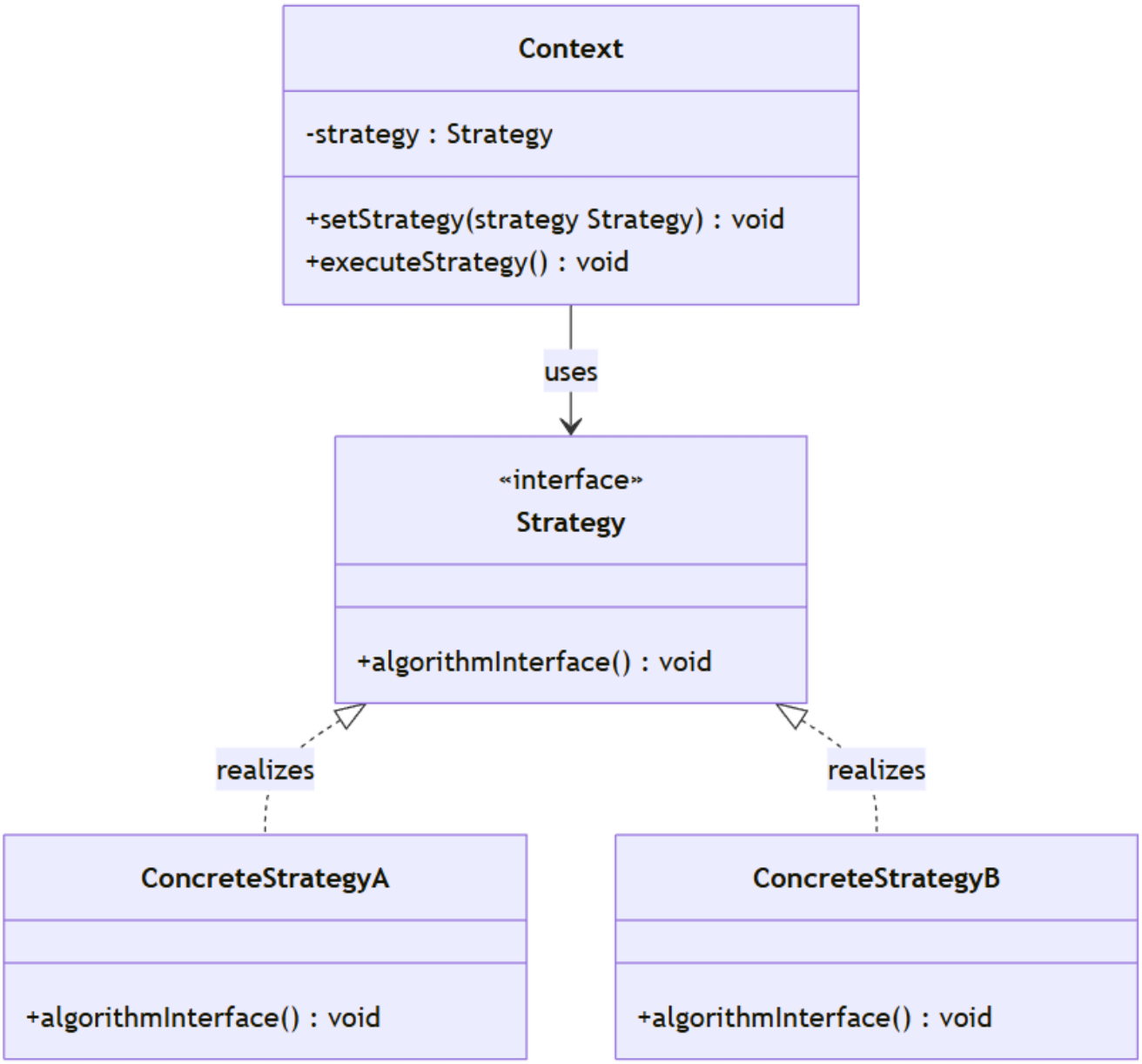
2. Навіщо використовувати шаблони проєктування?

- Спрощують розробку.
- Покращують читабельність і масштабування коду.
- Допомагають уникнути помилок.
- Забезпечують гнучкість і повторне використання.

3. Яке призначення шаблону «Стратегія»?

Шаблон Стратегія дозволяє динамічно вибирати алгоритми, інкапсулюючи їх у взаємозамінні класи, щоб уникнути умовних конструкцій.

6. Нарисуйте структуру шаблону «Стратегія».



5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

Класи:

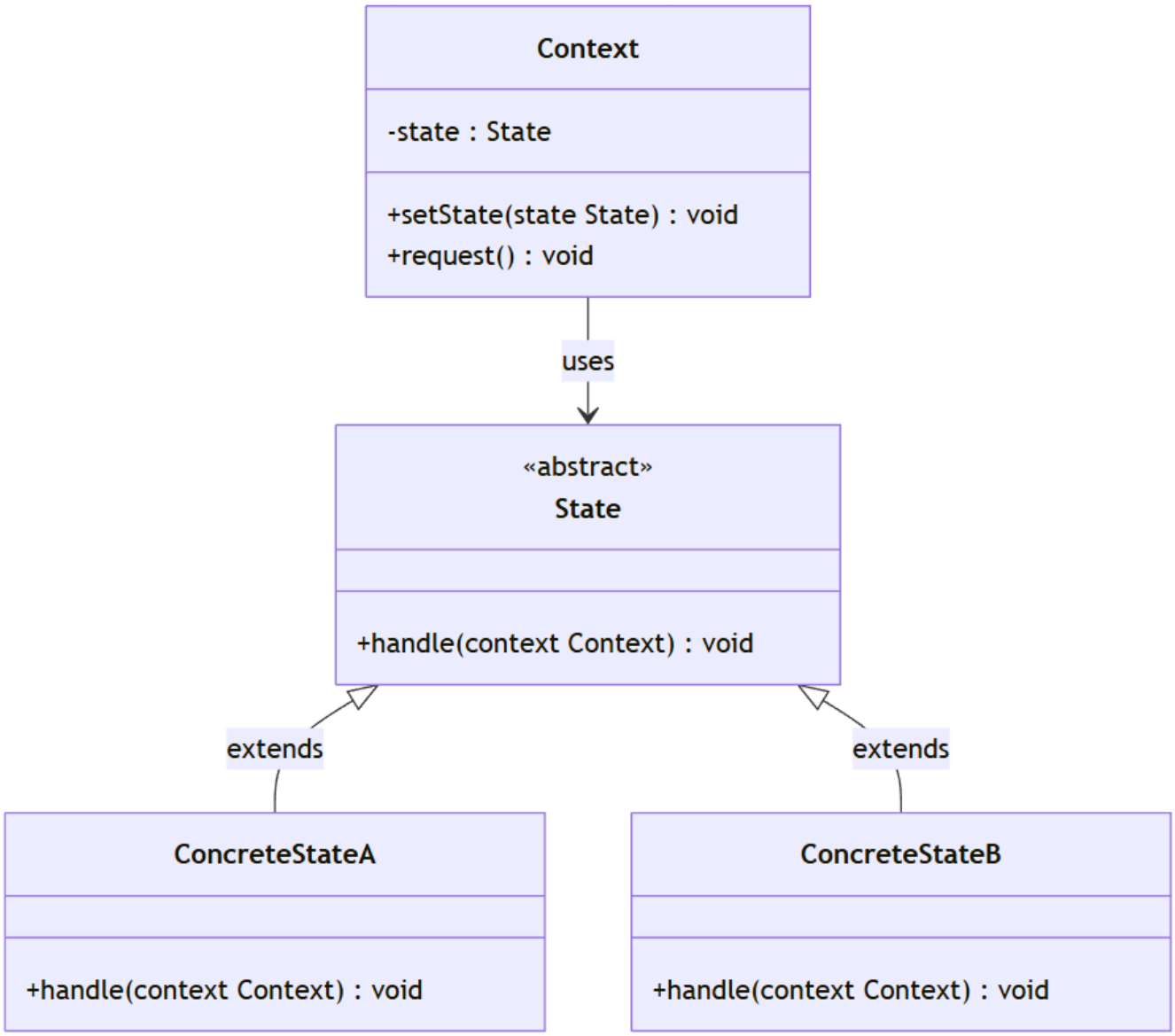
- Strategy: Інтерфейс із методом algorithm().
- ConcreteStrategy: Реалізації алгоритмів.
- Context: Використовує Strategy через setStrategy() і викликає algorithm().

Взаємодія: Контекст делегує виконання алгоритму об’єкту Strategy, який клієнт встановлює динамічно.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» дозволяє об’єкту змінювати поведінку залежно від стану, інкапсулюючи стани в окремі класи.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

Класи:

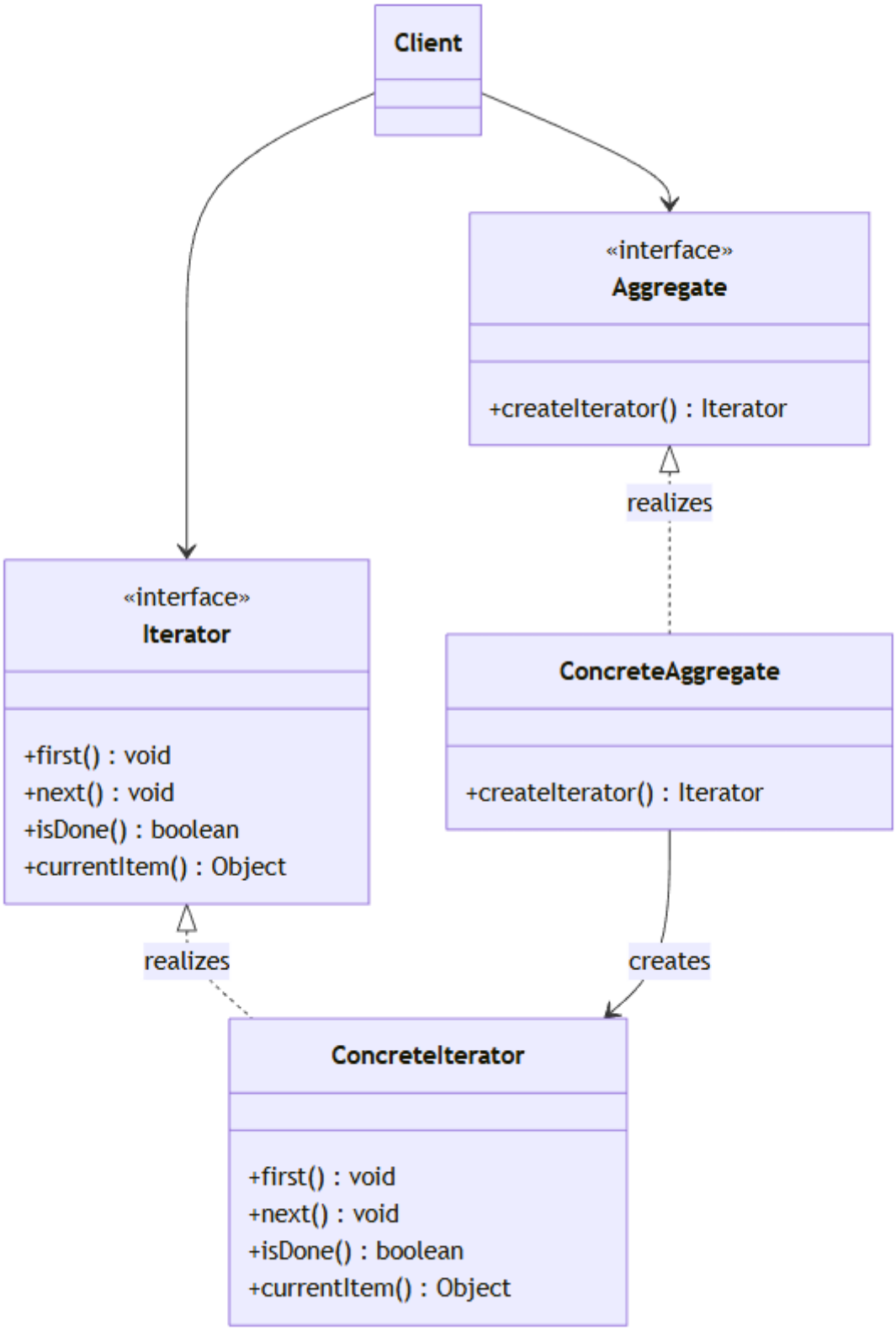
- State: Інтерфейс із методом handle().
- ConcreteState: Реалізації поведінки для стану.
- Context: Зберігає стан, делегує виконання handle().

Взаємодія: Контекст викликає handle() поточного стану, який може змінити стан через setState().

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» забезпечує послідовний доступ до елементів колекції, приховуючи її внутрішню структуру.

10. Нарисуйте структуру шаблону «Ітератор».



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

Класи:

- Iterator: Інтерфейс із методами next(), hasNext().

- ConcreteIterator: Реалізація обходу.
- Aggregate: Інтерфейс із createIterator().
- ConcreteAggregate: Створює ConcreteIterator.

Взаємодія: Клієнт отримує ітератор через createIterator() і використовує його для обходу колекції.

12. В чому полягає ідея шаблона «Одинак»?

Шаблон «Одинак» забезпечує єдиний екземпляр класу з глобальним доступом до нього.

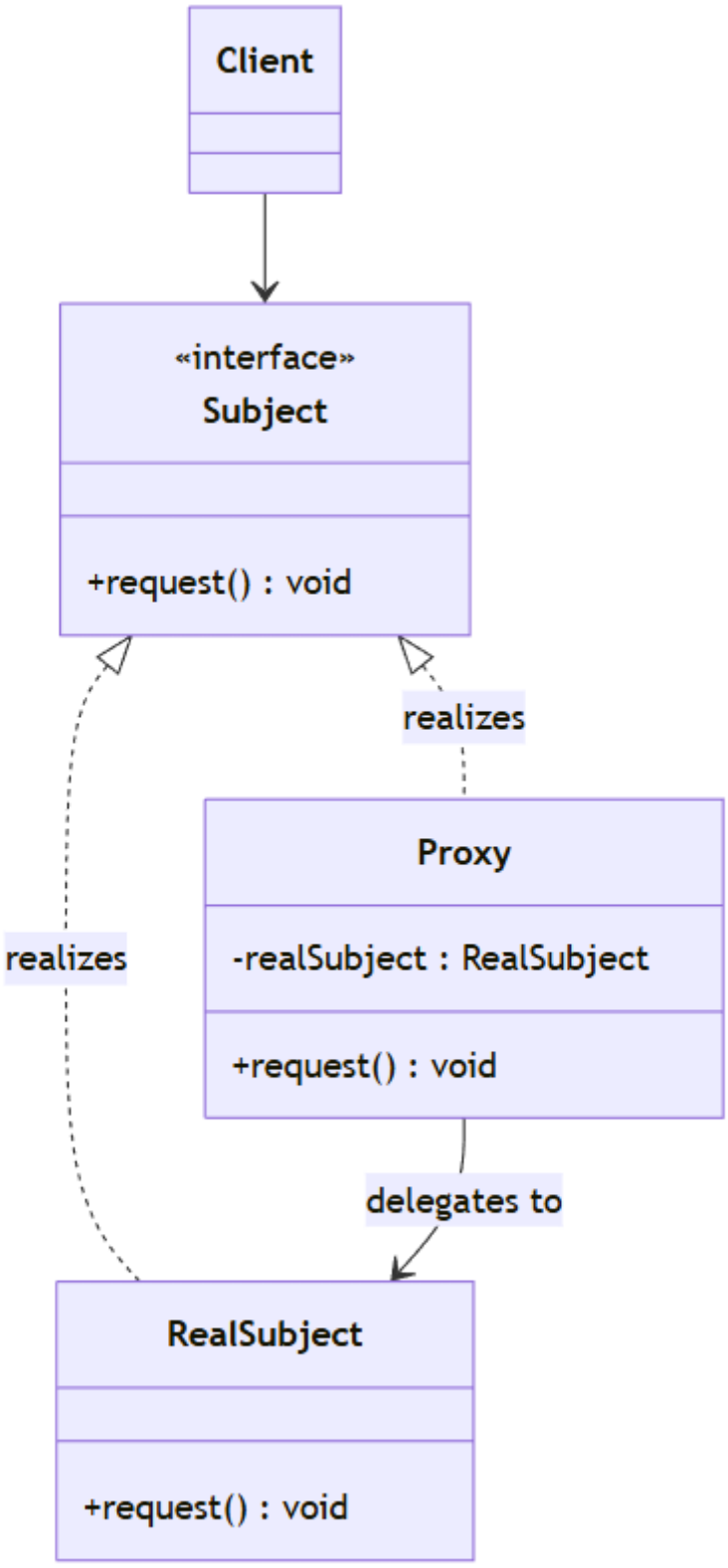
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Бо він порушує принципи ООП: ускладнює тестування, створює приховані залежності та глобальний стан

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» контролює доступ до об’єкта, додаючи функціонал, як-от ледарська ініціалізація чи перевірка прав.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

Класи:

- Subject: Інтерфейс із методом request().

- RealSubject: Виконує основну роботу.
- Proxy: Контролює доступ до RealSubject.

Взаємодія: Клієнт викликає request() через Proxy, який делегує виклик до RealSubject або додає логіку.