Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Лабораторна робота №4

«Вступ до паттернів 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 «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Теоретичні відомості

Шаблон «Одинак» (Singleton)

- **Призначення:** Гарантує, що клас матиме лише один екземпляр, і надає глобальну точку доступу до цього екземпляра.
- **Переваги:**
  - Гарантована наявність єдиного екземпляра класу.
  - Наявність глобальної точки доступу до нього.
- **Недоліки:**
  - Порушує принцип єдиної відповідальності класу.
  - Може приховувати проблеми в дизайні системи.

Шаблон «Ітератор» (Iterator)

- **Призначення:** Надає уніфікований спосіб послідовного доступу до елементів колекції (набору даних), не розкриваючи її внутрішньої структури. Логіка обходу колекції виноситься з самої колекції в окремий об'єкт-ітератор.
- **Переваги:**
  - Дозволяє реалізувати різноманітні способи обходу однієї і тієї ж структури даних.
  - Спрощує класи, що відповідають за зберігання даних.
- **Недоліки:**
  - Не виправданий, якщо для обходу колекції можна обійтися простим циклом.

Шаблон «Замісник» (Proxy)

- **Призначення:** Створює об'єкт-заступник, який контролює доступ до іншого, реального об'єкта. Це дозволяє додати додаткову логіку

(наприклад, кешування, контроль доступу, відкладену ініціалізацію) без зміни клієнтського коду.

- **Переваги:**
  - Дозволяє керувати життєвим циклом та доступом до реального об'єкта непомітно для клієнта.
  - Легко впроваджується без переробки існуючого клієнтського коду.
- **Недоліки:**
  - Може знизити швидкість роботи системи через впровадження додаткових операцій.
  - Існує ризик неадекватної заміни відповіді від реального об'єкта.

Шаблон «Стан» (State)

- **Призначення:** Дозволяє об'єкту змінювати свою поведінку при зміні його внутрішнього стану. Створюється враження, ніби об'єкт змінив свій клас. Логіка, що залежить від стану, виноситься в окремі класи.
- **Переваги:**
  - Спрощує код основного об'єкта, усуваючи складні умовні конструкції (if/switch).
  - Локалізує поведінку, специфічну для кожного стану, в окремих класах.
  - Полегшує додавання нових станів.
- **Недоліки:**
  - Може ускладнити архітектуру, якщо логіка переходів між станами є заплутаною.

Шаблон «Стратегія» (Strategy)

- **Призначення:** Визначає сімейство алгоритмів, інкапсулює кожен з них і робить їх взаємозамінними. Це дозволяє вибирати потрібний алгоритм під час виконання програми, не змінюючи код об'єкта, який його використовує.
- **Переваги:**

    - Дозволяє динамічно змінювати алгоритми під час виконання.
    - Відокремлює реалізацію алгоритмів від коду, що їх використовує.
    - Зменшує кількість умовних операторів для вибору поведінки.
- **Недоліки:**
    - Може невиправдано ускладнити код, якщо алгоритмів небагато і вони прості.
    - Клієнтський код повинен знати про відмінності між стратегіями для вибору найбільш відповідної.

Хід Роботи

В ході аналізу системи, було обрано шаблон Strategy для реалізації адже він як найкраще підходить до архітектури додатку.

Цей шаблон дозволяє визначати сімейство схожих алгоритмів, інкапсулювати кожен з них та робити їх взаємозамінними. Це дає можливість змінювати алгоритм незалежно від клієнтського коду, який його використовує. Таким чином, вибір конкретної реалізації алгоритму може відбуватися під час виконання програми.

Основна ідея полягає у винесенні різних алгоритмів в окремі класи (стратегії), які мають спільний інтерфейс. Клас-контекст, який використовує ці алгоритми, працює з ними через цей загальний інтерфейс і не залежить від конкретної реалізації.

Переваги:

- **Гнучкість:** Дозволяє змінювати алгоритми "на льоту" під час виконання програми.
- **Розширюваність:** Легко додавати нові алгоритми, не змінюючи код контексту.
- **Ізоляція коду:** Реалізація алгоритмів відокремлена від клієнтського коду, що спрощує їх тестування та супровід.
- **Спрощення контексту:** Прибирає з класу-контексту складні умовні оператори (switch, if-else) для вибору алгоритму.

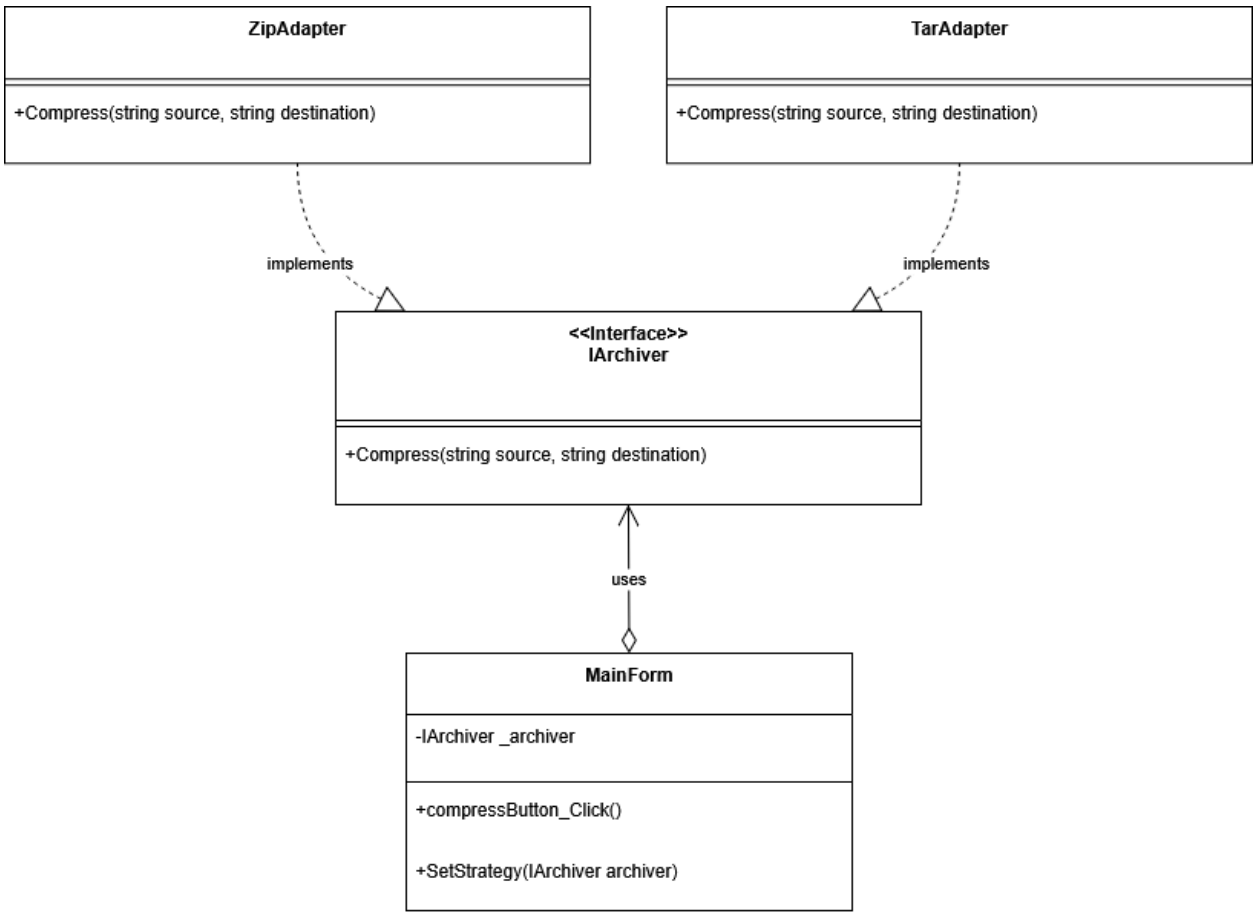
Оскільки архіватор має надавати користувачеві можливість стискати обрані файли в архіви форматів ZIP, TAR та ін.. Саме для реалізації цієї можливості вибору алгоритму архівації було застосовано шаблон **«Стратегія»**.

Ключові класи системи:

1. IArchiver – це інтерфейс Стратегії. Він визначає загальний метод Compress, який повинні реалізувати всі конкретні алгоритми архівації.
2. ZipAdapter – Конкретна Стратегія. Клас, що реалізує інтерфейс IArchiver та інкапсулює логіку для створення ZIP-архівів.
3. TarAdapter – Конкретна Стратегія. Клас, що реалізує інтерфейс IArchiver та інкапсулює логіку для створення TAR-архівів.
4. MainForm – клас, що представляє головне вікно програми і виступає в ролі Контексту. Він містить посилання на об'єкт стратегії (IArchiver) і використовує його для виконання операції архівації, не вдаючись у деталі конкретного алгоритму. Користувач через інтерфейс обирає потрібний формат, і MainForm встановлює відповідну конкретну стратегію.

Таким чином, MainForm делегує завдання архівації обраному об'єкту-стратегії. Якщо в майбутньому знадобиться додати підтримку нового формату (наприклад, 7-Zip), достатньо буде створити новий клас, що реалізує інтерфейс IArchiver, без необхідності змінювати існуючий код MainForm.

Діаграма Класів реалізованого шаблону



- MainForm (Контекст) має асоціацію з інтерфейсом IArchiver (Стратегія).
- ZipAdapter та TarAdapter (Конкретні Стратегії) реалізують (імплементують) інтерфейс IArchiver.

Код реалізованого шаблону

```
namespace ArchiverApp.Forms
{
    1 usage  PriMerro23
    public partial class MainForm : Form
    {
        private readonly ArchiveManager _manager = new ArchiveManager();
        private string? _currentArchive;

        1 usage  PriMerro23
        public MainForm(){...}

        1 usage  PriMerro23
        private void btnOpen_Click(object? sender, EventArgs e){...}

        1 usage  PriMerro23
        private void btnAdd_Click(object? sender, EventArgs e){...}

        1 usage  PriMerro23
        private void btnRemove_Click(object? sender, EventArgs e){...}

        1 usage  PriMerro23
        private void btnChecksum_Click(object? sender, EventArgs e){...}

        1 usage  PriMerro23
        private void btnTest_Click(object? sender, EventArgs e){...}

        1 usage  PriMerro23
        private void btnSplit_Click(object? sender, EventArgs e){...}

        1 usage  PriMerro23
        private void btnCreateZip_Click(object? sender, EventArgs e){...}

        1 usage  PriMerro23
        private void btnCreateTar_Click(object? sender, EventArgs e){...}

        1 usage  PriMerro23
        private void btnExtract_Click(object? sender, EventArgs e){...}

        2 usages  PriMerro23
        private void btnSettings_Click(object sender, EventArgs e){...}
```

Рис. 1 – MainForm.cs

```
1 usage  PriMerro23
private void settingsToolStripMenuItem_Click(object sender, EventArgs e){...}

2 usages  PriMerro23
private void ShowSettingsForm(){...}

2 usages  PriMerro23
private void btnHistory_Click(object sender, EventArgs e){...}

1 usage  PriMerro23
private void historyToolStripMenuItem_Click(object sender, EventArgs e){...}

2 usages  PriMerro23
private void ShowHistoryForm(){...}

5 usages  PriMerro23
private void ReloadEntries(){...}
```

Рис. 2 – MainForm.cs

```
namespace ArchiverApp.Core
{
    2 usages  3 inheritors  PriMerro23  1 exposing API
    public interface IArchiveStrategy
    {
        1 usage  2 implementations  PriMerro23
        void CreateEmpty(string archivePath);
        6 usages  2 implementations  PriMerro23
        IEnumerable<string> ListEntries(string archivePath);
        1 usage  2 implementations  PriMerro23
        void AddFiles(string archivePath, IEnumerable<string> filePaths);
        1 usage  2 implementations  PriMerro23
        void RemoveEntries(string archivePath, IEnumerable<string> entryNames);
        3 usages  2 implementations  PriMerro23
        byte[] ReadEntry(string archivePath, string entryName);
    }

    2 usages  2 inheritors  PriMerro23
    public interface IArchiveAdapter : IArchiveStrategy { }

    2 usages  2 inheritors  PriMerro23
    public interface IArchiveVisitor
    {
        2 implementations  PriMerro23
        void Visit(string entryName, byte[] content);
    }
}
```

Рис. 3 –IArchiveStrategy.cs

```
namespace ArchiverApp.Core
{
    1 usage  PriMerro23
    public class TarAdapter : IArchiveAdapter
    {
        0+1 usages  PriMerro23
        public void CreateEmpty(string archivePath){...}

        0+6 usages  PriMerro23
        public IEnumerable<string> ListEntries(string archivePath){...}

        0+1 usages  PriMerro23
        public void AddFiles(string archivePath, IEnumerable<string> filePaths){...}

        0+1 usages  PriMerro23
        public void RemoveEntries(string archivePath, IEnumerable<string> entryNames)
        {
            var temp :string  = Path.GetTempFileName();
            var keep :List<(name,path)>  = ExtractAllToTemp(archivePath).Where(e :(name,path)  => !entryNames.Contains(e.name)
            using (var outFs :FileStream  = File.Create(temp))
            using (var tar = new TarWriter(outFs))
            {...}
            File.Copy( sourceFileName: temp,  destFileName: archivePath, overwrite: true);
            File.Delete(temp);
        }

        0+3 usages  PriMerro23
        public byte[] ReadEntry(string archivePath, string entryName){...}

        2 usages  PriMerro23
        private List<(string name, string path)> ExtractAllToTemp(string archivePath){...}
    }
}
```

Рис. 4 –TarAdapter.cs

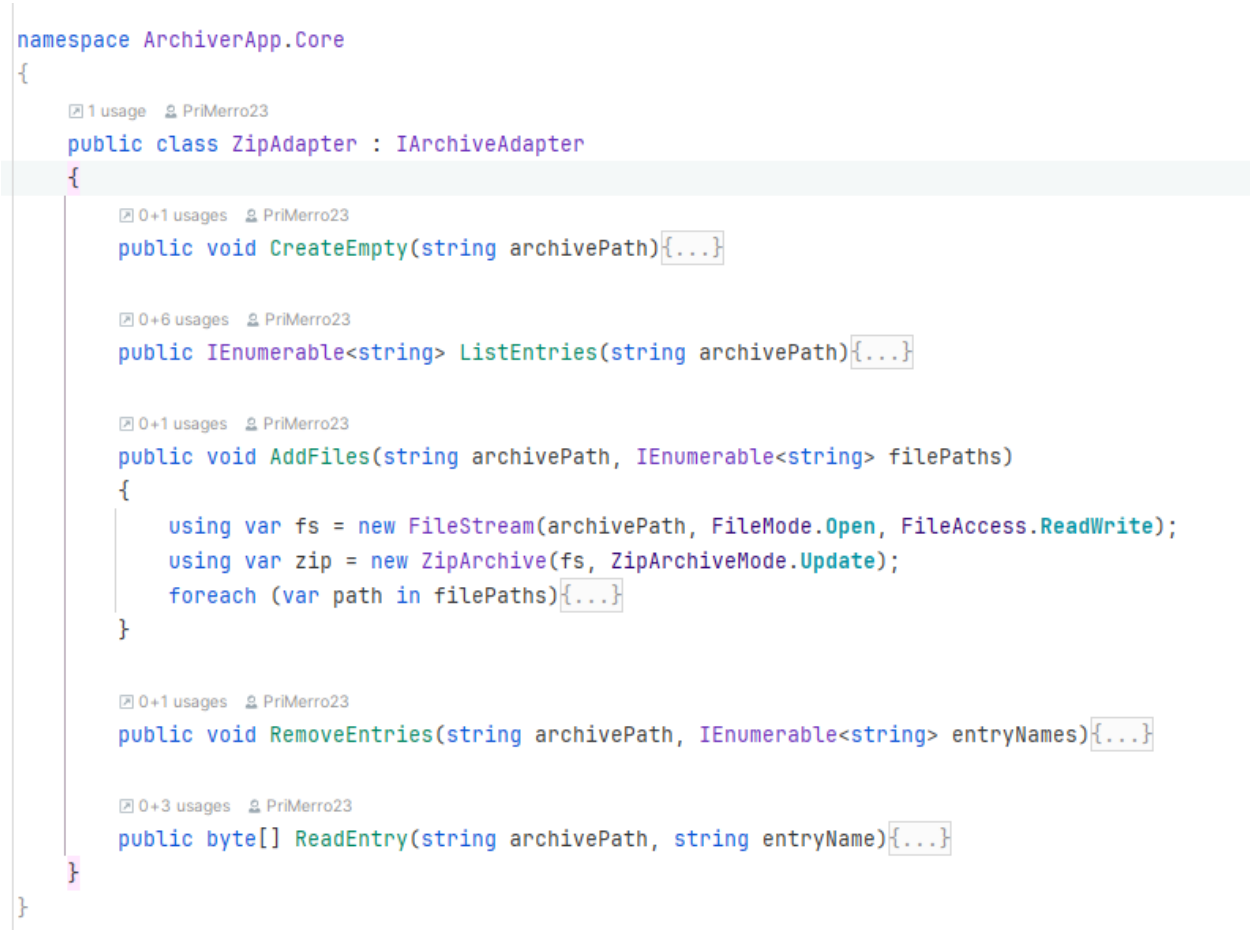


Рис. 5 –ZipAdapter.cs

Висновок: В рамках лабораторної роботи, я ознайомився з теоретичними основами шаблонів проєктування та успішно застосував патерн «Стратегія» для розробки програми-архіватора. Реалізація цього шаблону дозволила створити гнучку та розширювану архітектуру. Основний клас MainForm не залежить від конкретних методів архівації, що відповідає принципу відкритості/закритості (Open/Closed Principle). Додавання нових алгоритмів стиснення не вимагатиме внесення змін до існуючого коду, що значно спрощує підтримку та розвиток програми. Таким чином, мета лабораторної роботи була повністю досягнута.

Питання до лабораторної роботи

1. Що таке шаблон проєктування?

Шаблон проєктування — це іменоване, багаторазове рішення типових задач архітектури ПЗ у заданому контексті, що описує структури взаємодії об’єктів без прив’язки до конкретної мови чи реалізації.

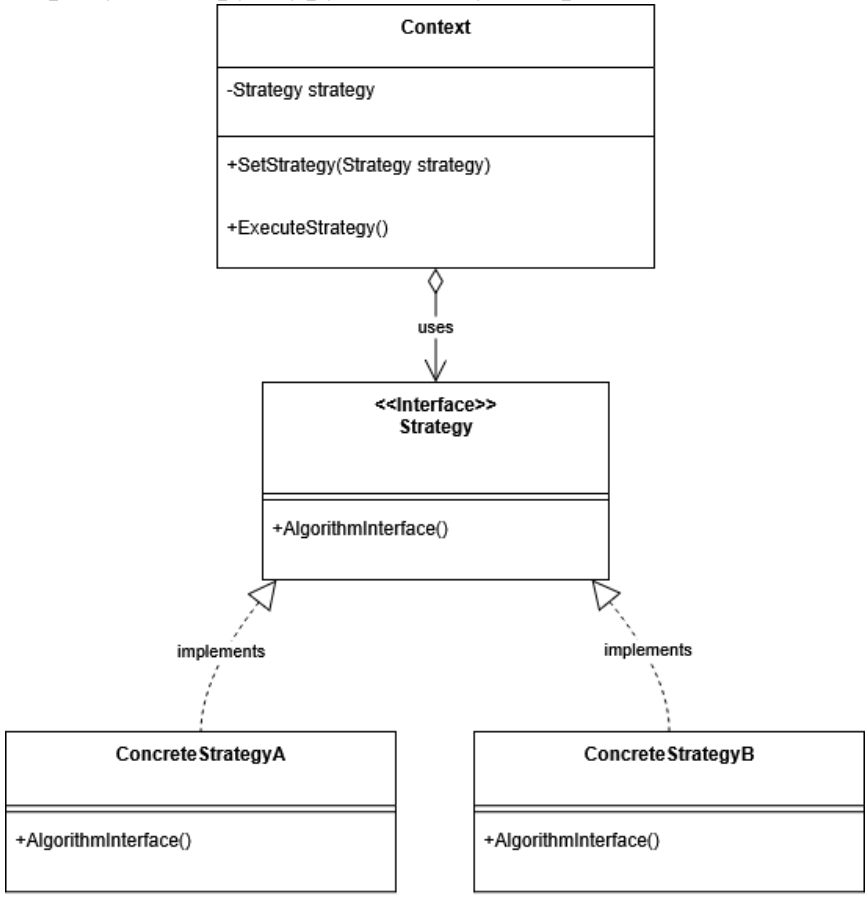
2. Навіщо використовувати шаблони проєктування?

Шаблони використовують для прискорення розробки, покращення читабельності та підтримуваності, зменшення технічного боргу й помилок, а також для спільної мови між інженерами та узгоджених архітектурних рішень.

3. Яке призначення шаблону «Стратегія»?

«Стратегія» визначає сімейство взаємозамінних алгоритмів, інкапсулює кожен у власному класі та дозволяє перемикати алгоритми під час виконання без зміни клієнтського коду.

4. Нарисуйте структуру шаблону «Стратегія».



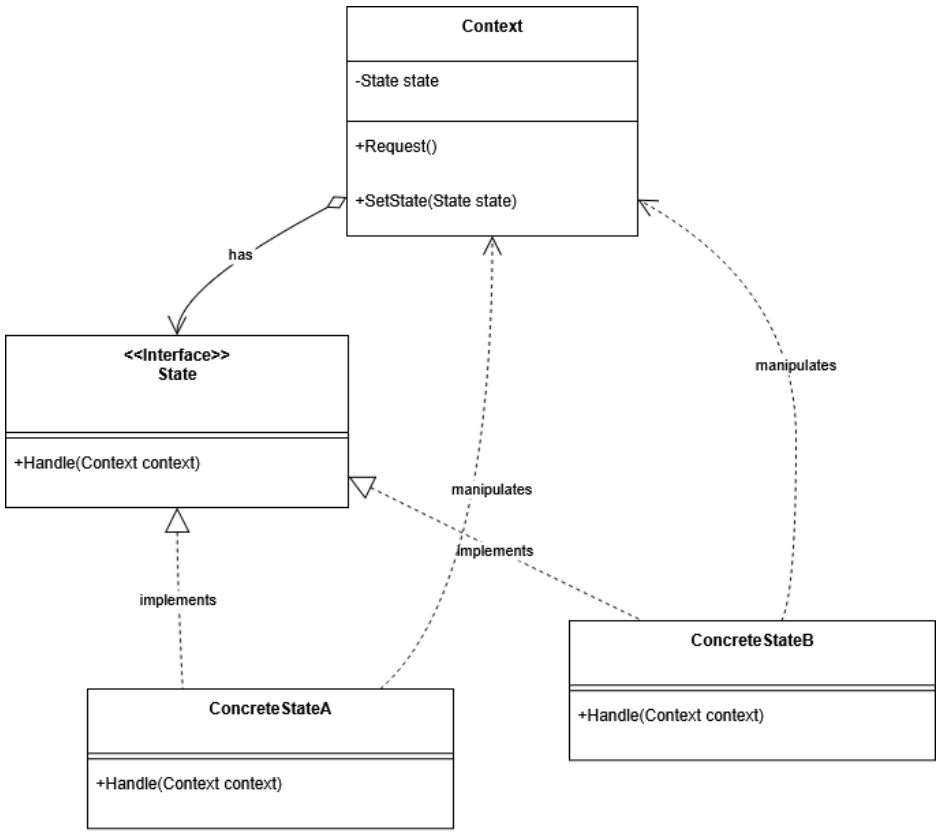
5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

Context зберігає посилання на Strategy, викликаючи її методи; Strategy визначає контракт; ConcreteStrategy реалізують конкретні варіанти; заміна об’єкта Strategy змінює поведінку без модифікації Context.

6. Яке призначення шаблону «Стан»?

«Стан» дає змогу об’єктові змінювати поведінку залежно від внутрішнього стану, створюючи враження зміни класу об’єкта під час виконання.

7. Нарисуйте структуру шаблону «Стан».



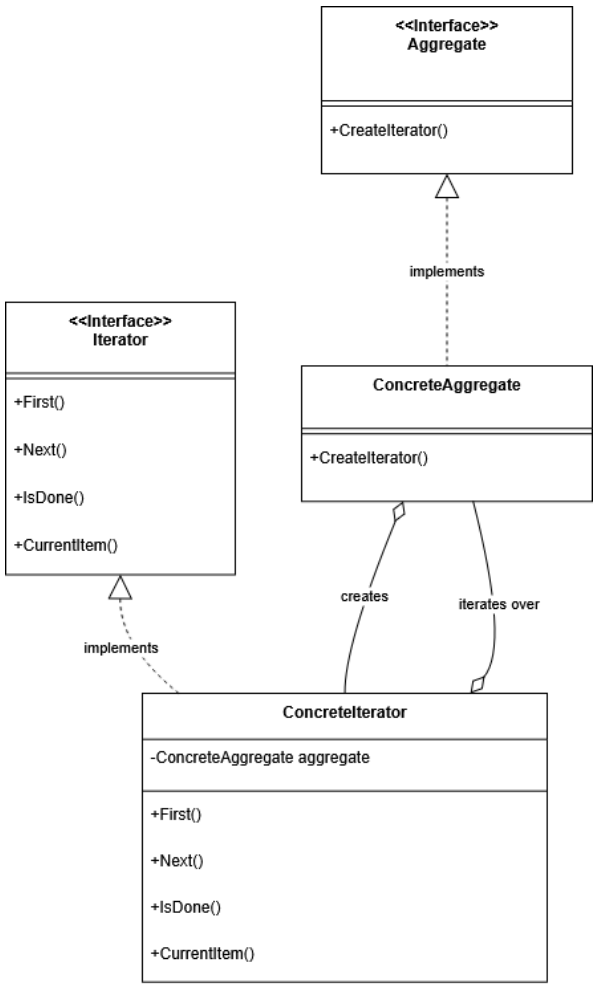
8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

Context делегує обробку поточному State; State визначає операції поведінки; ConcreteState реалізують поведінку та за потреби перевстановлюють стан Context для переходів між станами.

9. Яке призначення шаблону «Ітератор»?

«Ітератор» забезпечує послідовний обхід елементів колекції, не розкриваючи внутрішню структуру, та уніфікує доступ до різних типів колекцій через спільний інтерфейс обходу.

10. Нарисуйте структуру шаблону «Ітератор».



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

Client використовує Iterator для навігації; ConcreteIterator знає структуру ConcreteAggregate та позицію обходу; Aggregate/ConcreteAggregate інкапсулюють зберігання й створюють відповідні ітератори.

12.В чому полягає ідея шаблона «Одинак»?

«Одинак» гарантує існування лише одного екземпляра класу та надає глобальну точку доступу до нього, зазвичай через статичний метод або поле з керованим створенням.

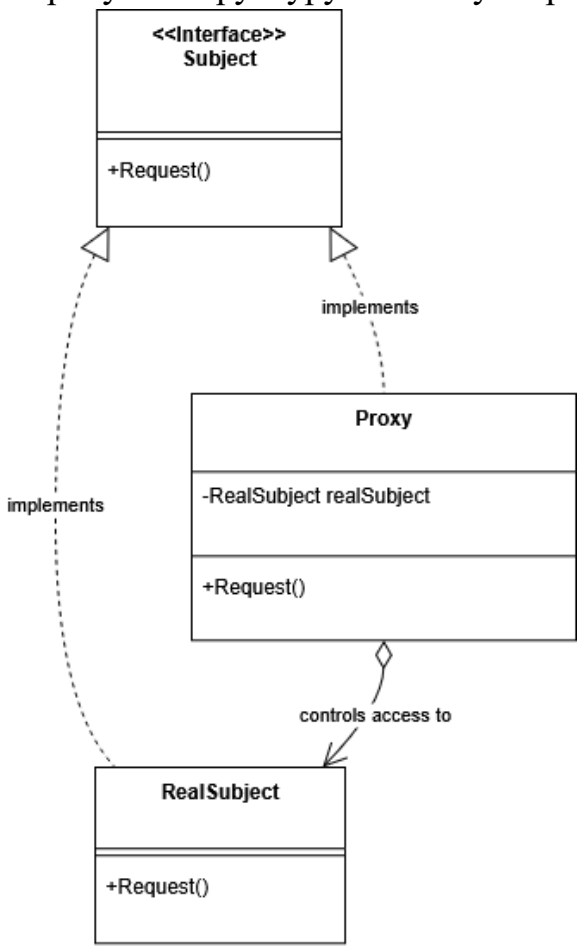
13.Чому шаблон «Одинак» вважають «анти-шаблоном»?

«Одинак» часто вважають антишаблоном, оскільки він подібний до глобального стану, ускладнює тестування, приховує залежності та порушує принцип єдиної відповідальності, що призводить до тісного зчеплення і проблем з підтримкою.

14.Яке призначення шаблону «Проксі»?

«Проксі» (Замісник) підставляє об’єкт-замінник замість реального об’єкта й контролює доступ до нього, дозволяючи виконувати додаткові дії до або після делегування викликів оригіналу.

15.Нарисуйте структуру шаблону «Проксі».



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

Client викликає методи через Subject; Proxy перевіряє доступ, ліниво ініціалізує, кешує або маршалізує виклики й делегує RealSubject; поширені різновиди — віддалений, віртуальний, захисний і кешуючий проксі.