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»

Варіант - 17

Виконав
Студент групи ІА-31:
Лисенко В. Є

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. 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.

Для підсистеми відображення історії активності (логів ресурсів CPU/RAM) було обрано паттерн Iterator. Система оперує колекцією записів ResourceLog, які зберігаються в базі даних та завантажуються в пам'ять для відображення на UI. Використання стандартних списків (List<T>) напряму в клієнтському коді (UI) створює сильну зв'язність і розкриває внутрішню структуру збереження даних.

Паттерн Iterator дозволяє:

1. Абстрагувати процес обходу колекції логів. Клієнт (MainWindow) не знає, як саме зберігаються дані (масив, список, стек), він лише використовує методи Next() та CurrentItem().
2. Забезпечити єдиний інтерфейс для доступу до елементів, що дозволяє в майбутньому змінювати структуру колекції LogCollection без змін у коді відображення.
3. Реалізувати "ручний" контроль над обходом (наприклад, для покрокового аналізу інцидентів), не покладаючись на вбудовані цикли foreach.

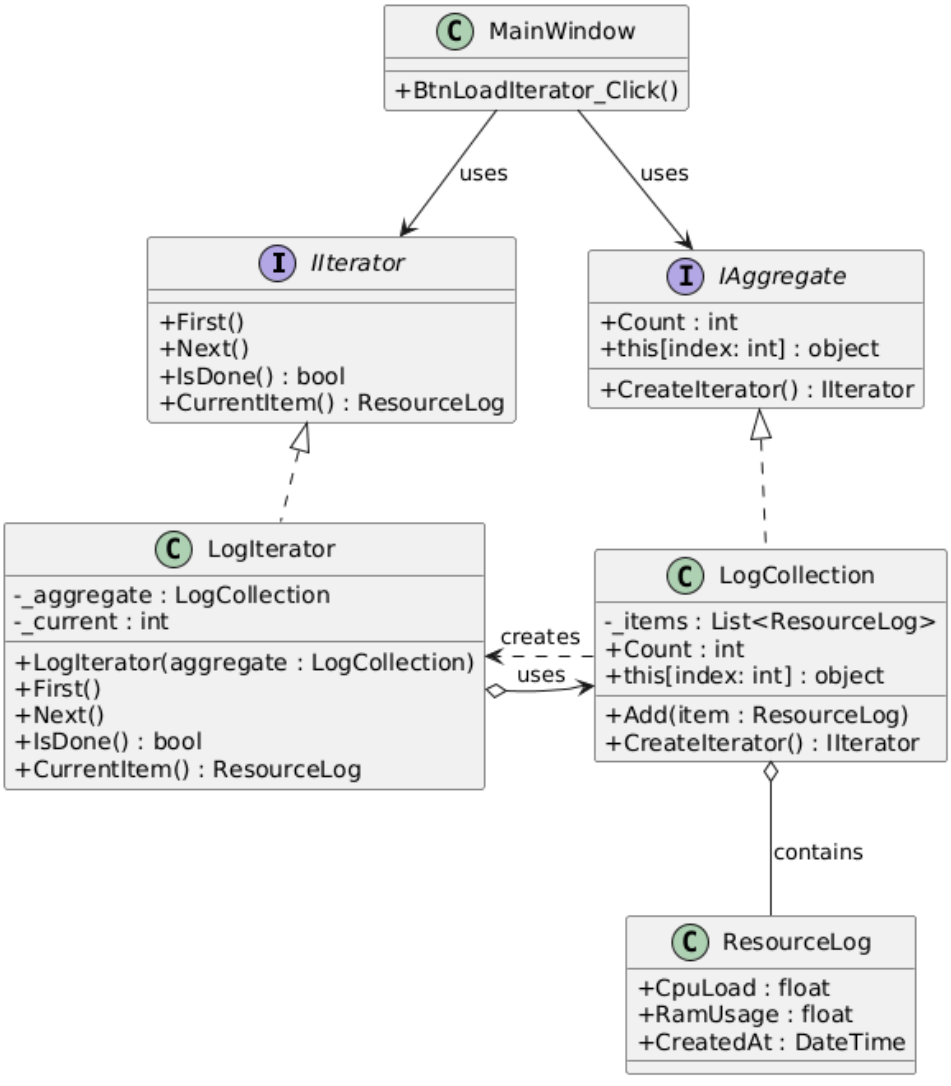


Рисунок 1 - Структура патерну Iterator

IAggregate	Інтерфейс агрегата	Описує метод CreateIterator(), який має реалізувати будь-яка колекція логів у системі.
LogCollection	Конкретний агрегат	Реальна колекція, що зберігає список об'єктів ResourceLog (отриманих з бази даних). Вона реалізує інтерфейс IAggregate і повертає екземпляр LogIterator.
IIterator	Інтерфейс ітератора	Визначає стандартні методи для навігації: First() (на початок), Next() (наступний), IsDone() (перевірка кінця), CurrentItem() (отримання даних).
LogIterator	Конкретний ітератор	Відповідає за покроковий обхід конкретного екземпляра LogCollection. Він зберігає поточну позицію курсору (_current).
MainWindow	Клієнт	Використовує ітератор для виводу даних на екран. Клієнт не працює з індексами масиву напряму, а делегує це ітератору

Програмна реалізація:

Iiterator.cs:

```
using SystemActivityMonitor.Data.Entities;

namespace SystemActivityMonitor.Data.Patterns.Iterator

{

    public interface IIterator

    {

        void First();

        void Next();

        bool IsDone();

        ResourceLog CurrentItem();

    }

}
```

Iaggregate.cs:

```
namespace SystemActivityMonitor.Data.Patterns.Iterator

{

    public interface IAggregate

    {

        IIterator CreateIterator();

        int Count { get; }

        object this[int index] { get; set; }

    }

}
```

LogCollection.cs:

```
using System.Collections.Generic;

using SystemActivityMonitor.Data.Entities;
```

```
namespace SystemActivityMonitor.Data.Patterns.Iterator
{
    public class LogCollection : IAggregate
    {
        private List<ResourceLog> _items = new List<ResourceLog>();
        public void Add(ResourceLog item)
        {
            _items.Add(item);
        }
        public IIterator CreateIterator()
        {
            return new LogIterator(this);
        }
        public int Count => _items.Count;
        public object this[int index]
        {
            get => _items[index];
            set => _items[index] = (ResourceLog)value;
        }
    }
}
```

LogIterator.cs:

```
using SystemActivityMonitor.Data.Entities;
namespace SystemActivityMonitor.Data.Patterns.Iterator
{
    public class LogIterator : IIterator
    {
        private readonly LogCollection _aggregate;
        private int _current = 0;
        public LogIterator(LogCollection aggregate)
        {
            _aggregate = aggregate;
        }
        public void First()
        {
            _current = 0;
        }
        public void Next()
        {
            _current++;
        }
        public bool IsDone()
```

```
        {
            return _current >= _aggregate.Count;
        }
        public ResourceLog CurrentItem()
        {
            if (IsDone())
                return null;
            return _aggregate[_current] as ResourceLog;
        }
    }
}
```

MainWindow.xaml.cs:

```
private void BtnLoadIterator_Click(object sender, RoutedEventArgs e)
        {
            lstLogs.Items.Clear();
            LogCollection collection = new LogCollection();

            using (var db = new MonitorDbContext())
            {
                var logsFromDb = db.ResourceLogs.OrderByDescending(l => l.CreatedAt).ToList();
                foreach (var log in logsFromDb)
                {
                    collection.Add(log);
                }
            }
            IIterator iterator = collection.CreateIterator();
            iterator.First();
            while (!iterator.IsDone())
            {
                var item = iterator.CurrentItem();
                if (item != null)
                {
                    string displayText = $"[{item.CreatedAt.ToLongTimeString()}] CPU: {item.CpuLoad}% | RAM:
{item.RamUsage} MB";
                    lstLogs.Items.Add(displayText);
                }
                iterator.Next();
            }
        }
```

Рисунок 2 – Результат роботи ітератора

5. Висновок

У ході лабораторної роботи вивчено та реалізовано базові шаблони проєктування — Singleton, Iterator, Proxy, State і Strategy — для створення гнучкої та масштабованої архітектури програмних систем. На прикладі розробки системи моніторингу «System Activity Monitor» було практично застосовано шаблон Iterator, реалізувавши його «вручну» без використання стандартних інтерфейсів .NET.

Iterator використовувався для послідовного обходу колекції логів системних ресурсів (ResourceLog). Логіку перебору винесено в окремий клас LogIterator, який реалізує інтерфейс IIterator. Це дозволило відокремити алгоритм обходу від структури даних у LogCollection, спростити клієнтський код (MainWindow), приховавши деталі реалізації списку, а також забезпечити можливість змінювати спосіб обходу (наприклад, додати фільтрацію або зворотний порядок) без модифікації самої колекції чи інтерфейсу користувача. Робота підтвердила ефективність використання патернів проєктування для зменшення зв’язності коду та підвищення його гнучкості для подальшого розвитку.

6. 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?

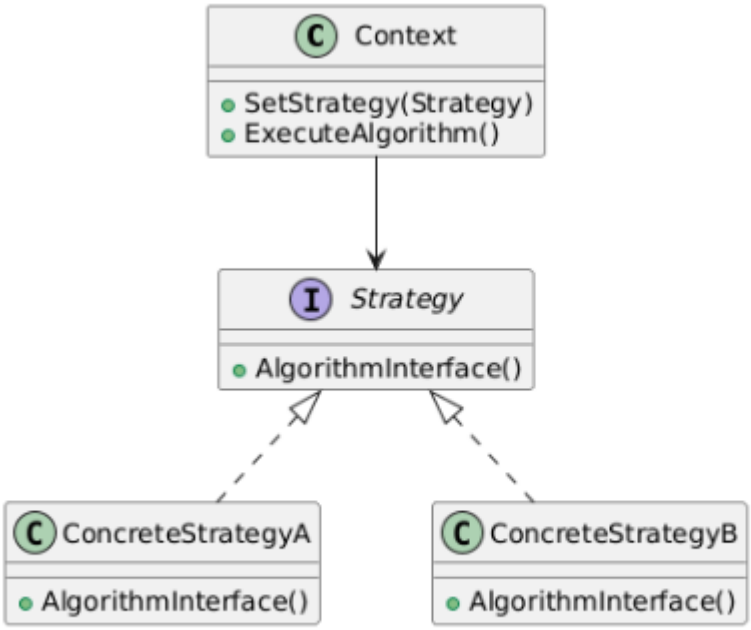
- Спрощують розробку.
- Покращують читабельність і масштабування коду.

- 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.

4. Нарисуйте структуру шаблону «Стратегія».



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

Класи:

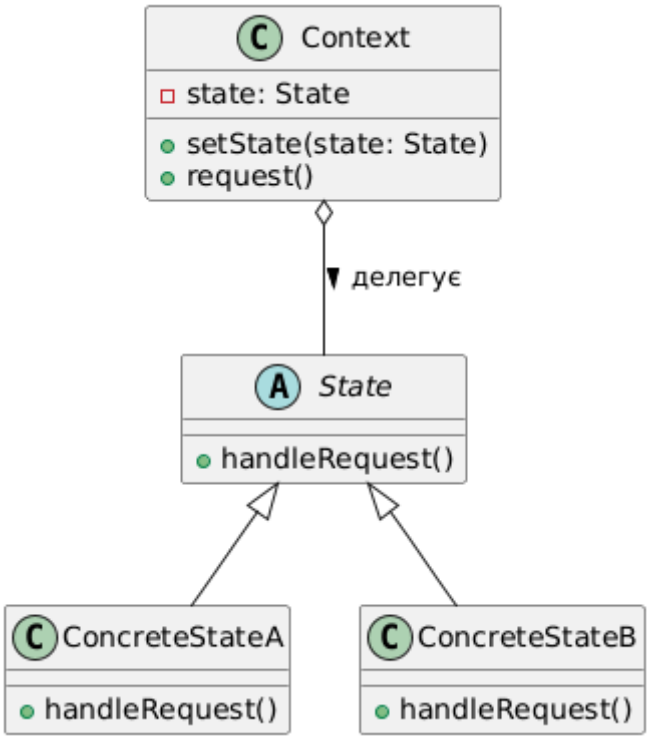
- Strategy: Інтерфейс із методом algorithm().
- ConcreteStrategy: Реалізації алгоритмів.
- Context: Використовує Strategy через setStrategy() і викликає algorithm().

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:

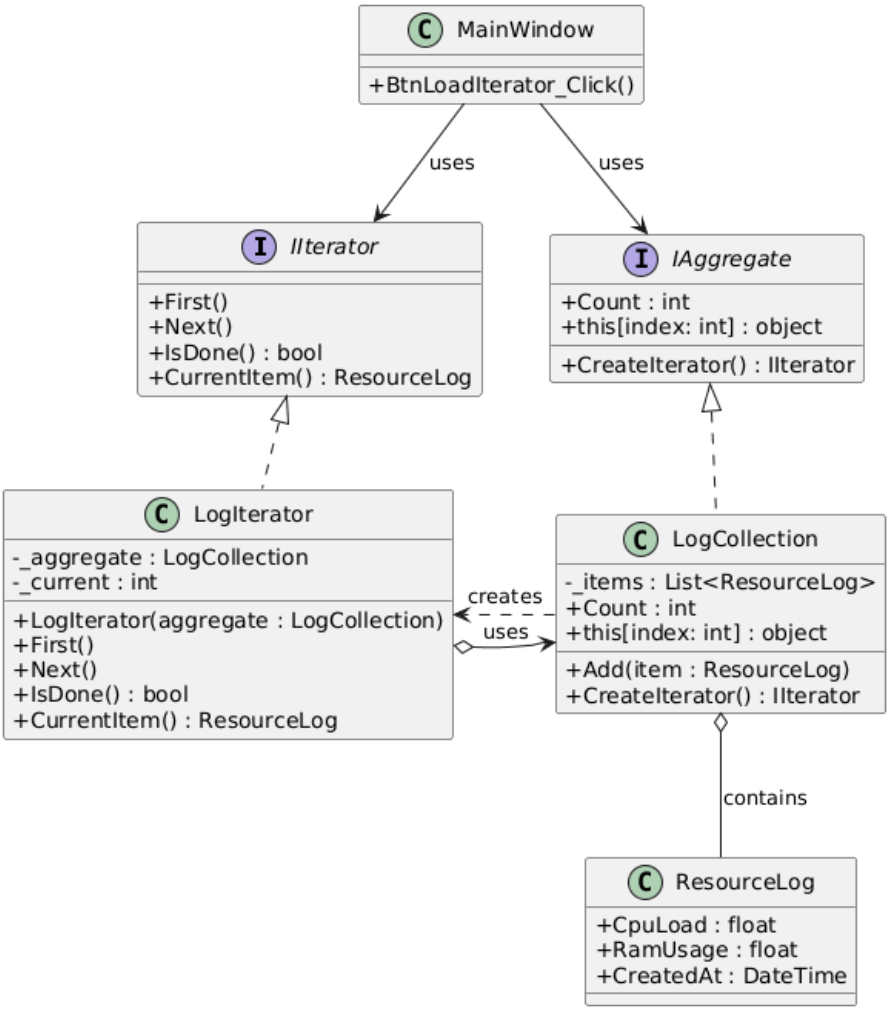
- State: Інтерфейс із методом handle().
- ConcreteState: Реалізації поведінки для стану.
- Context: Зберігає стан, делегує виконання handle().

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.

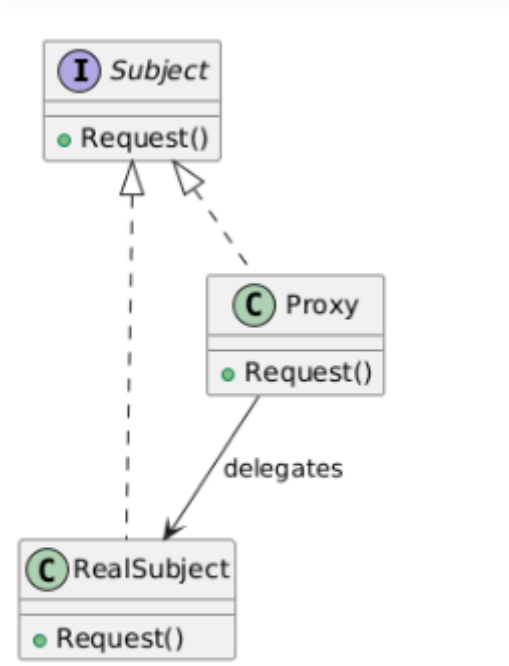
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

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.