Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №4

Технології розроблення програмного забезпечення

«Вступ до патернів проектування. Паттерн «Iterator»»

«System activity monitor»

Виконав
студент групи ІА–32:
Феклістов Д.С.

Київ 2025

Зміст

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. Будувати звіти про використання комп'ютера за різними критеріями. Правильно поводитися з «простоюванням» системи.

Теоретичні відомості

Паттерн Ітератор (Iterator) — це поведінковий шаблон проєктування, що надає спосіб **послідовного доступу** до елементів **колекції**, **не розкриваючи** її внутрішньої структури.

Призначення:

Відокремлення логіки обходу колекції від самої колекції. Це забезпечує **інкапсуляцію** структури даних та дозволяє клієнтському коду працювати з різними колекціями через **єдиний інтерфейс**, підвищуючи **гнучкість** системи.

Учасники паттерну:

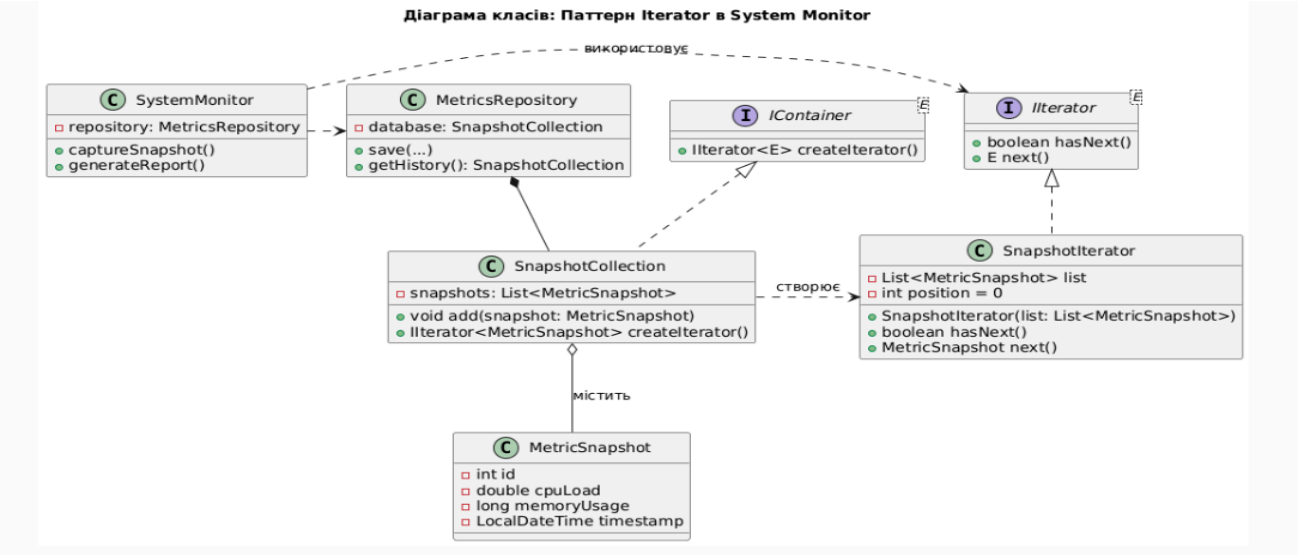
- **IIterator**: Інтерфейс обходу.
- **SnapshotIterator**: Конкретна реалізація обходу колекції метрик.
- **IContainer**: Інтерфейс для створення ітератора.
- **SnapshotCollection**: Клас колекції, що зберігає MetricSnapshot і повертає SnapshotIterator.

Хід роботи

Метою роботи було застосувати паттерн **Ітератор** до компоненту зберігання даних (MetricsRepository) для забезпечення уніфікованого доступу до історії системних метрик (MetricSnapshot) у системі **"System Activity Monitor"**.

Діаграма класів (UML Class Diagram)

Діаграма відображає статичну структуру класів, які реалізують паттерн «Ітератор».



Програмна реалізація (Фрагменти коду)

Клас `SystemMonitor` (Клієнтський код):

```
package monitor;
import pattern.IContainer;
import pattern.IIterator;
import data.MetricSnapshot;
import service.MetricsRepository;
import service.OsAdapter;

public class SystemMonitor {
  private OsAdapter adapter = new OsAdapter();
  private MetricsRepository repository = new MetricsRepository();

  public void captureSnapshot() {
     double cpu = adapter.getCpuLoad();
     long ram = adapter.getMemoryUsage();
     repository.save(cpu, ram);
  }

  // Генерація звіту з використанням паттерну Iterator
  public void generateReport() {
     System.out.println("\n--- REPORT GENERATION (Iterator Pattern) ---");

     // 1. Отримуємо контейнер (інтерфейс IContainer)
     IContainer container = repository.getHistory();

     // 2. Створюємо ітератор (інтерфейс IIterator)
     IIterator iterator = container.createIterator();

     // 3. Послідовний обхід
     while (iterator.hasNext()) {
        MetricSnapshot item = iterator.next();
        System.out.println("LOG >> " + item.toString());
     }
     System.out.println("--- END OF REPORT ---\n");
  }
}
```

Клас `SnapshotIterator` (Конкретний Ітератор):

```
package pattern;
import data.MetricSnapshot;
import java.util.List;

public class SnapshotIterator implements IIterator {
  private List<MetricSnapshot> list;
  private int position = 0;

  public SnapshotIterator(List<MetricSnapshot> list) {
     this.list = list;
  }

  @Override
  public boolean hasNext() {
     return position < list.size();
  }
```

```
  @Override
  public MetricSnapshot next() {
     if (hasNext()) {
        return list.get(position++);
     }
     return null;
  }
}
```

Клас SnapshotCollection:

```
package pattern;

import data.MetricSnapshot;

import java.util.ArrayList;

import java.util.List;


public class SnapshotCollection implements IContainer {

    // Прихований список, до якого клієнт не має прямого
доступу

    private List<MetricSnapshot> snapshots = new
ArrayList<>();


    public void add(MetricSnapshot snapshot) {

        this.snapshots.add(snapshot);

    }


    @Override

    public IIterator createIterator() {

        return new SnapshotIterator(this.snapshots);

    }

}
```

Результати роботи програми

```
=== ЗАПУСК SYSTEM ACTIVITY MONITOR ===

-> [Repo] Saved: [ID:1 | 01:12:18] CPU: 53,2%, RAM: 2089 MB
-> [Repo] Saved: [ID:2 | 01:12:18] CPU: 51,5%, RAM: 3217 MB
-> [Repo] Saved: [ID:3 | 01:12:18] CPU: 59,0%, RAM: 5241 MB

--- REPORT GENERATION (Iterator Pattern) ---
LOG >> [ID:1 | 01:12:18] CPU: 53,2%, RAM: 2089 MB
LOG >> [ID:2 | 01:12:18] CPU: 51,5%, RAM: 3217 MB
LOG >> [ID:3 | 01:12:18] CPU: 59,0%, RAM: 5241 MB
--- END OF REPORT ---
```

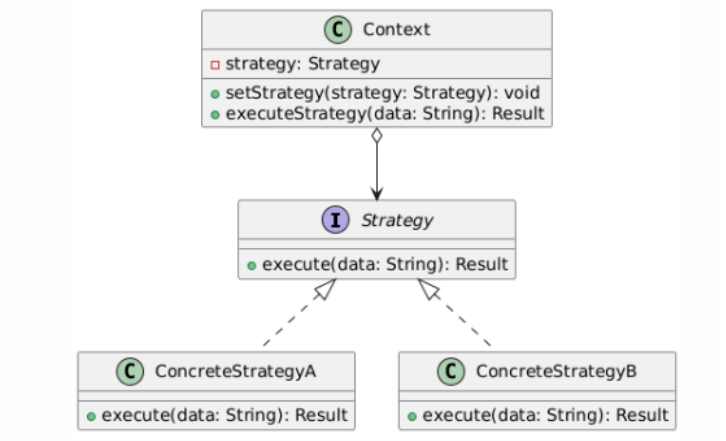
Висновок

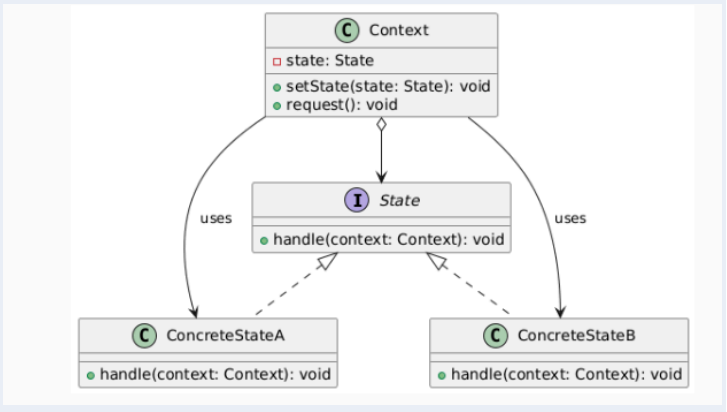
В ході виконання лабораторної роботи №4 було успішно застосовано поведінковий паттерн **Ітератор (Iterator)** до системи **"System Activity Monitor"**.

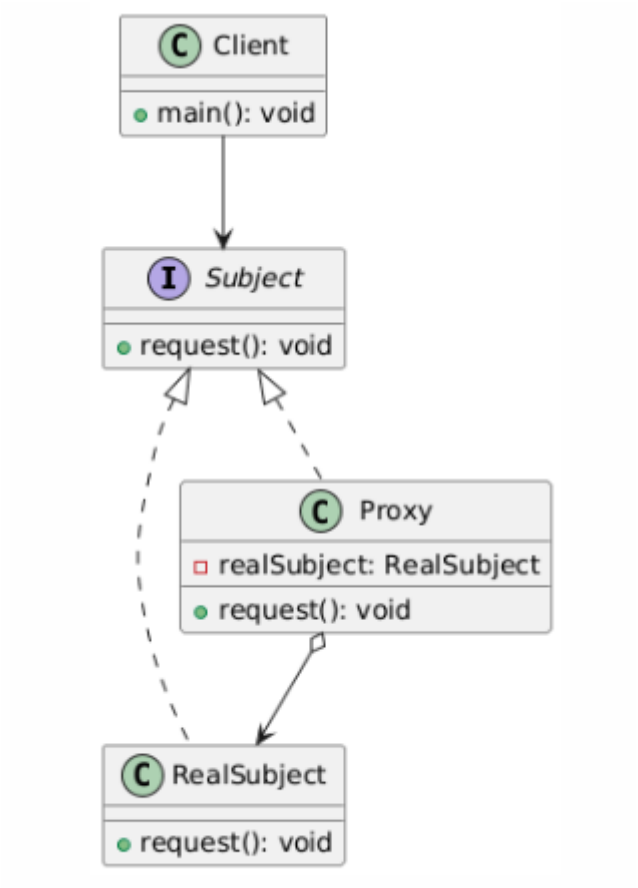
Було досягнуто:

- **Інкапсуляція:** Логіка обходу відокремлена від логіки зберігання даних.
- **Слабка зв'язаність:** Модуль генерації звіту взаємодіє з колекцією лише через універсальні інтерфейси `IIterator` та `IContainer`, що дозволяє змінювати внутрішню структуру зберігання даних без зміни клієнтського коду.

Відповіді на питання

Питання	Відповідь
Що таке шаблон проєктування?	Типове, перевірене рішення, що вирішує часто виникаючу проблему при проектуванні програмного забезпечення.
Навіщо використовувати шаблони проєктування?	Підвищують гнучкість, уніфікують код та спрощують підтримку архітектури системи.
Яке призначення шаблону «Стратегія»?	Визначити сімейство алгоритмів, інкапсулювати їх в окремі класи та зробити взаємозамінними.
Нарисуйте структуру шаблону «Стратегія».	
Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?	**Strategy** (інтерфейс), **ConcreteStrategy** (реалізація алгоритму), **Context** (використовує Strategy).

Яке призначення шаблону «Стан»?	Дозволяє об'єкту змінювати свою поведінку, коли змінюється його внутрішній стан, створюючи ілюзію зміни класу.
Нарисуйте структуру шаблону «Стан».	
Які класи входять в шаблон «Стан»?	**Context**, **State** (інтерфейс), **ConcreteState** (конкретний стан).
Яке призначення шаблону «Ітератор»?	Надати уніфікований спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури.
Нарисуйте структуру шаблону «Ітератор».	*(Див. Діаграму класів у розділі Хід роботи).*
Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?	**Iterator**, **ConcreteIterator**, **Aggregate** (інтерфейс колекції), **ConcreteAggregate**.

В чому полягає ідея шаблона «Одинак»?	Гарантувати, що клас має лише один екземпляр, і надати глобальну точку доступу до нього.
Чому шаблон «Одинак» вважають «анти-шаблоном»?	Вводить глобальний стан, ускладнює тестування, порушує принцип єдиної відповідальності (SRP).
Яке призначення шаблону «Проксі»?	Надати об'єкт-замінник для контролю доступу до іншого об'єкта (для кешування, захисту, лінивої ініціалізації).
Нарисуйте структуру шаблону «Проксі».	

Які класи входять в шаблон «Проксі»?	**Subject** (інтерфейс), **RealSubject** (реальний об'єкт), **Proxy** (замісник).