Міністерство освіти і науки України Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського” Факультет
інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 4

Технології розробки програмного забезпечення

“ Вступ до паттернів проектування.”

“Download manager”

Виконав студент групи ІА–33:
Ященко К.А.

Київ 2025

Тема: Вступ до паттернів проектування.

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

Зміст

Завдання 2
Теоретичні відомості 2
Тема проєкту 3
Хід роботи 3
2. Зв’язки та взаємодія між елементами 4
3. Залежність (використання) 4
Взаємодія 4
Частина коду програми з Iterator 5
Висновок 11

Завдання

- Ознайомитись з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- Реалізувати один з розглянутих шаблонів за обраною темою.
- Реалізувати не менше 3-х класів відповідно до обраної теми.
- 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.

Теоретичні відомості

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях. Крім того, патерн проєктування обов’язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову. Варто підкреслити, що важливим початковим етапом при роботі з патернами є адекватне моделювання розглянутої
предметної області. Це є необхідним як для отримання належним чином формалізованої постановки задачі, так і для вибору відповідних патернів проєктування.

Відповідне використання патернів проєктування дає розробнику ряд незаперечних переваг. Наведемо деякі з них. Модель системи, побудована в межах патернів проєктування, фактично є структурованим виокремленням тих елементів і зв’язків, які значимі при вирішенні поставленого завдання. Крім цього, модель, побудована з використанням патернів проєктування, більш проста і наочна у вивченні, ніж стандартна модель. Проте, не дивлячись на простоту і наочність, вона дозволяє глибоко і всебічно опрацювати архітектуру розроблюваної системи з використанням спеціальної мови.

Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи. Крім того, важко переоцінити роль використання патернів при інтеграції інформаційних систем організації. Також слід зазначити, що сукупність патернів проєктування, по суті, являє собою єдиний словник проєктування, який, будучи уніфікованим засобом, незамінний для спілкування розробників один одним.

Таким чином шаблони представляють собою, підтверджені роками розробок в різних компаніях і на різних проєктах, «ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах.

Тема проєкту

26. Download manager (iterator, command, observer, template method, composite, p2p) Інструмент для скачування файлів з інтернету по протоколах http або https з можливістю продовження завантаження в зупиненому місці, розподілу швидкостей активним завантаженням, ведення статистики завантажень, інтеграції в основні браузери (firefox, opera, internet explorer, chrome).

Паттерн: Iterator
Посилання на GitHub: https://github.com/fl1ckyexe/trpz

Хід роботи

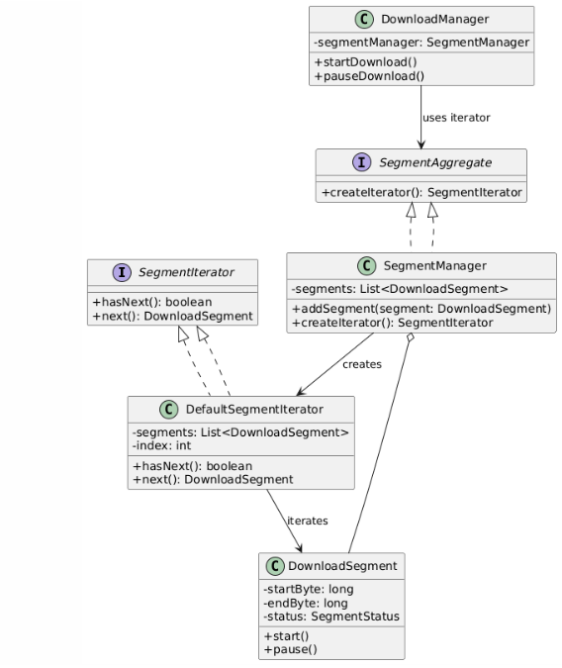


Рис. 1 – Діаграма класів для Iterator

Діаграма класів відображає частину системи менеджера завантажень, у якій реалізовано шаблон проєктування Iterator для перебору сегментів файлу під час процесу завантаження. Метою діаграми є демонстрація відокремлення логіки зберігання сегментів від логіки їх обробки, що підвищує гнучкість та розширюваність системи.

1. Основні елементи діаграми

- DownloadManager
  Клієнтський клас, який керує процесом завантаження файлу. Він використовує ітератор для послідовного доступу до сегментів без знання деталей їх зберігання.
- SegmentAggregate
  Інтерфейс агрегату, що оголошує метод createIterator() для створення ітератора сегментів.
- SegmentManager
  Конкретний агрегат, який зберігає колекцію об’єктів DownloadSegment та реалізує інтерфейс SegmentAggregate.
- SegmentIterator
  Інтерфейс ітератора, який визначає стандартні операції перебору: hasNext() та next().
- DefaultSegmentIterator
  Конкретна реалізація ітератора, яка здійснює послідовний перебір сегментів, що зберігаються у SegmentManager.
- DownloadSegment
  Клас предметної області, який представляє окремий сегмент файлу з інформацією про діапазон байтів та стан сегмента.

2. Зв’язки та взаємодія між елементами

- Реалізація інтерфейсів

SegmentManager реалізує інтерфейс SegmentAggregate.
DefaultSegmentIterator реалізує інтерфейс SegmentIterator.

- Агрегація
  SegmentManager агрегує множину об’єктів DownloadSegment, оскільки сегменти існують як частина завдання завантаження, але можуть керуватися незалежно.

3. Залежність (використання)

DownloadManager використовує SegmentAggregate для отримання ітератора та перебору сегментів.

DefaultSegmentIterator працює з об’єктами DownloadSegment, послідовно повертаючи їх клієнту.

Взаємодія

Під час запуску або керування завантаженням DownloadManager отримує ітератор від SegmentManager і за його допомогою послідовно обробляє всі сегменти файлу, не звертаючись безпосередньо до внутрішньої колекції сегментів.

Частина коду програми з Iterator

```
package model;


public enum SegmentStatus {
    PENDING,
    DOWNLOADING,
    COMPLETED
}


package model;


public class DownloadSegment {


    private final long startByte;
    private final long endByte;
    private SegmentStatus status;


    public DownloadSegment(long startByte, long endByte) {
        this.startByte = startByte;
        this.endByte = endByte;
        this.status = SegmentStatus.PENDING;
    }
```

```
    public long getStartByte() {
        return startByte;
    }

    public long getEndByte() {
        return endByte;
    }

    public SegmentStatus getStatus() {
        return status;
    }

    public void setStatus(SegmentStatus status) {
        this.status = status;
    }
}


package segment.iterator;

import model.DownloadSegment;

public interface SegmentIterator {
    boolean hasNext();
    DownloadSegment next();
}
```

```
package segment.iterator;

public interface SegmentAggregate {
    SegmentIterator createIterator();
}

package segment;

import model.DownloadSegment;
import segment.iterator.SegmentIterator;

import java.util.List;

public class SegmentIteratorImpl implements SegmentIterator {

    private final List<DownloadSegment> segments;
    private int index = 0;

    public SegmentIteratorImpl(List<DownloadSegment> segments) {
        this.segments = segments;
    }

    @Override
    public boolean hasNext() {
        return index < segments.size();
```

```
    }

    @Override
    public DownloadSegment next() {
        return segments.get(index++);
    }
}
```

```
package segment;

import model.DownloadSegment;
import segment.iterator.SegmentAggregate;
import segment.iterator.SegmentIterator;

import java.util.ArrayList;
import java.util.List;

public class SegmentManager implements SegmentAggregate {

    private final List<DownloadSegment> segments = new ArrayList<>();

    public void splitFile(long fileSize, int parts) {
        long partSize = fileSize / parts;
        long start = 0;
```

```
        for (int i = 0; i < parts; i++) {
            long end = (i == parts - 1) ? fileSize : start + partSize - 1;
            segments.add(new DownloadSegment(start, end));
            start = end + 1;
        }
    }

    @Override
    public SegmentIterator createIterator() {
        return new SegmentIteratorImpl(segments);
    }
}
```

```
package core;

import model.DownloadSegment;
import model.SegmentStatus;
import segment.SegmentManager;
import segment.iterator.SegmentIterator;

public class DownloadManager {

    private final SegmentManager segmentManager;

    public DownloadManager(SegmentManager segmentManager) {
        this.segmentManager = segmentManager;
```

```
}

public void startDownload() {
    SegmentIterator iterator = segmentManager.createIterator();

    while (iterator.hasNext()) {
        DownloadSegment segment = iterator.next();
        downloadSegment(segment);
    }
}

private void downloadSegment(DownloadSegment segment) {
    segment.setStatus(SegmentStatus.DOWNLOADING);

    System.out.println(
        "Downloading segment: " +
        segment.getStartByte() + "-" + segment.getEndByte()
    );

    // imitation of download
    try {
        Thread.sleep(300);
    } catch (InterruptedException ignored) {}

    segment.setStatus(SegmentStatus.COMPLETED);
}
```

```
}
```

```
package ui;

import core.DownloadManager;
import segment.SegmentManager;

public class Main {

    public static void main(String[] args) {

        SegmentManager segmentManager = new SegmentManager();
        segmentManager.splitFile(10000, 4);

        DownloadManager manager = new DownloadManager(segmentManager);
        manager.startDownload();
    }
}
```

Висновок

У ході виконання даної лабораторної роботи було досліджено та реалізовано шаблон проєктування Iterator на прикладі десктопного менеджера завантажень файлів. Шаблон було застосовано для організації послідовного доступу до сегментів файлу під час процесу сегментного завантаження без розкриття внутрішньої структури їх зберігання.

У процесі роботи було спроєктовано та реалізовано класи агрегату, ітератора та клієнта, що відповідають класичній структурі шаблону Iterator. Клас SegmentManager виступає у ролі агрегату, який зберігає сегменти, тоді як SegmentIteratorImpl забезпечує їх послідовний перебір. Клас DownloadManager використовує ітератор для керування сегментами, не звертаючись безпосередньо до колекції.

Застосування шаблону Iterator дозволило підвищити гнучкість та розширюваність системи, спростити логіку керування сегментами та забезпечити слабке зв’язування між компонентами. Реалізований підхід є зручним для подальшого розвитку системи, зокрема для інтеграції інших шаблонів проєктування та розширення функціональності менеджера завантажень.

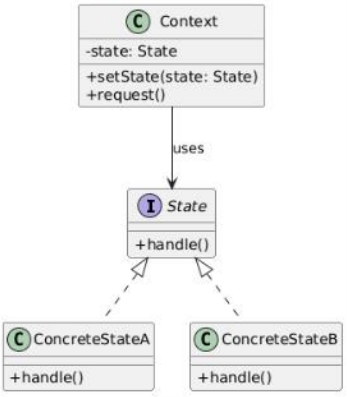
Відповіді на контрольні питання

1. Що таке шаблон проєктування?
Шаблон проєктування — це узагальнене, перевірене на практиці рішення типової проблеми, що виникає під час проєктування програмного забезпечення. Він описує структуру взаємодії класів і об’єктів, не прив’язуючись до конкретної реалізації.

2. Навіщо використовувати шаблони проєктування?
Шаблони проєктування використовуються для спрощення розробки складних систем, підвищення гнучкості та розширюваності коду, зменшення зв’язності між компонентами та застосування перевірених архітектурних рішень, що полегшує підтримку і розвиток програмного продукту.

3. Яке призначення шаблону «Стратегія»?
Призначення шаблону «Стратегія» полягає в інкапсуляції сімейства алгоритмів і наданні можливості динамічно змінювати алгоритм роботи об’єкта без зміни клієнтського коду. Це дозволяє відокремити алгоритм від контексту, в якому він використовується.

4. Нарисуйте структуру шаблону «Стратегія».



5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?
У шаблоні «Стратегія» взаємодіють класи Strategy, ConcreteStrategy та Context. Context зберігає посилання на об’єкт Strategy та делегує йому виконання алгоритму, тоді як ConcreteStrategy реалізують конкретні варіанти поведінки.

6. Яке призначення шаблону «Стан»?
Призначення шаблону «Стан» полягає в тому, щоб дозволити об’єкту змінювати свою поведінку залежно від внутрішнього стану, при цьому зміна поведінки

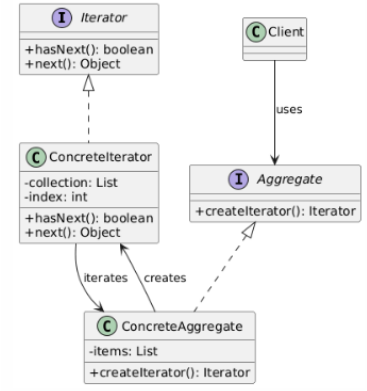
відбувається без використання великої кількості умовних операторів.

7. Нарисуйте структуру шаблону «Стан».
Структура шаблону «Стан» включає інтерфейс State, що описує поведінку, набір конкретних станів ConcreteState та клас Context, який зберігає поточний стан і делегує йому виконання операцій.

8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?
У шаблоні «Стан» клас Context взаємодіє з об’єктами ConcreteState через інтерфейс State. Кожен конкретний стан реалізує власну поведінку та може змінювати стан контексту під час виконання операцій.

9. Яке призначення шаблону «Ітератор»?
Призначення шаблону «Ітератор» полягає в забезпеченні послідовного доступу до елементів колекції без розкриття її внутрішньої структури, що дозволяє працювати з різними типами колекцій уніфікованим способом.

10. Нарисуйте структуру шаблону «Ітератор».



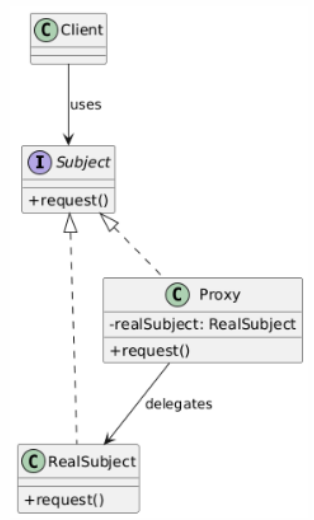
11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?
У шаблоні «Ітератор» Aggregate відповідає за створення ітератора, ConcreteAggregate зберігає колекцію елементів, Iterator визначає інтерфейс перебору, ConcreteIterator реалізує алгоритм обходу, а Client використовує ітератор для доступу до елементів.

12. В чому полягає ідея шаблона «Одинак»?
Ідея шаблону «Одинак» полягає в гарантуванні існування лише одного екземпляра класу та наданні глобальної точки доступу до цього екземпляра в межах програми.

13. Чому шаблон «Одинак» вважають «анти-шаблоном»?
Шаблон «Одинак» вважають анти-шаблоном, оскільки він створює приховані глобальні залежності, ускладнює модульне тестування та порушує принципи об’єктно-орієнтованого проєктування, зокрема принцип інверсії залежностей.

14. Яке призначення шаблону «Проксі»?
Призначення шаблону «Проксі» полягає в контролі доступу до об’єкта шляхом використання замісника, який може виконувати додаткові дії перед або після звернення до реального об’єкта.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?
У шаблоні «Проксі» клієнт взаємодіє з об’єктом через інтерфейс Subject, Proxy перехоплює виклики клієнта та за необхідності делегує їх RealSubject, забезпечуючи контроль доступу, кешування або інші додаткові функції.