Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського» Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота № 8

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування.»

Тема проекту: «26. Download Manager»

Виконала:

студентка групи ІА-34

Мушта Анна

Перевірив:

Мягкий Михайло Юрійович

Дата здачі 13.12.2025

Захищено з балом

Київ 2025

Зміст

Тема: 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)

1.1. Теоретичні відомості

Шаблон «Composite» (Компонувальник)

Призначення:
Шаблон **Composite** дозволяє об'єднувати об'єкти в деревоподібну структуру для представлення ієрархій типу «частина-ціле». Це дає змогу працювати з окремими елементами та групами елементів однаково.

Суть:
Використовується для обробки ієрархічних структур, де один елемент може містити інші елементи. Наприклад, в інтерфейсі користувача є елементи (поля, кнопки, написи), які також можуть містити інші елементи.

Переваги:

- Спрощує роботу з деревоподібними структурами.
- Дає гнучкість у додаванні та видаленні елементів.
- Полегшує рекурсивні операції.

Недоліки:

- Потребує ретельного проєктування спільного інтерфейсу.
- Складніше впровадження на початку.

Шаблон «Flyweight» (Легковаговик)

Призначення:
Шаблон **Flyweight** зменшує кількість об'єктів, розділяючи спільні дані між ними.

Суть:
Має два типи станів:

- **Внутрішній стан** — спільний для всіх екземплярів об'єкта.
- **Зовнішній стан** — індивідуальний для кожного контексту використання.

Наприклад, усі однакові літери в тексті представлені одним об'єктом, який багаторазово використовується.

Переваги:

- Економить оперативну пам'ять.

Недоліки:

- Потрібен додатковий процесорний час на пошук і відновлення контексту.
- Код стає складнішим через додаткові класи.

Шаблон «Interpreter» (Інтерпретатор)

Призначення:
Шаблон **Interpreter** використовується для створення граматики простої мови та її інтерпретатора.

Суть:
Цей шаблон формує абстрактне синтаксичне дерево, де кожен вузол є правилом або операцією. Кожен вираз інтерпретується відповідно до контексту.

Переваги:

- Легко змінювати і розширювати граматику.
- Просто додавати нові способи обчислення.

Недоліки:

- При великій кількості правил код стає громіздким.

Шаблон «Visitor» (Відвідувач)

Призначення:
Шаблон **Visitor** дозволяє виконувати операції над елементами об'єктної структури без зміни класів цих елементів.

Суть:
Виносить логіку дій (наприклад, розрахунки) в окремий клас «відвідувача». Елементи структури приймають відвідувача і викликають відповідний метод, залежно від свого типу. Це зручно для реалізації різних обчислень над однаковими об'єктами.

Переваги:

- Легко додавати нові операції (нові відвідувачі).
- Дані і логіка розділені.

Недоліки:

- Важко додавати нові типи елементів, оскільки потрібно змінювати всіх відвідувачів.

1.2. Хід роботи.

1. **Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.**

ProgressObserver (Інтерфейс):
Це інтерфейс, який використовується для спостереження за прогресом завантаження файлу. Клас, що реалізує цей інтерфейс, повинен мати метод **onProgress**, який отримує дані про прогрес завантаження: ID завдання, кількість завантажених байт, загальний розмір та статус завершення завантаження.

Command (Інтерфейс):
Інтерфейс для команд, які виконуються в рамках програми. Він має метод **execute()**, що дозволяє виконати команду, наприклад, для запуску чи скасування завдання.

Segment та SegmentIterator (Ітератор):
Клас **Segment** представляє відрізок даних для завантаження з певними початковими і кінцевими байтами. Клас **SingleSegmentIterator** імплементує інтерфейс **SegmentIterator** і забезпечує ітерацію по сегментам, повертаючи один сегмент для завантаження.

DownloadTask (Шаблон методів):
Абстрактний клас, який визначає загальний алгоритм завантаження файлу. Він містить шаблонний метод **runTask()**, що складається з кількох кроків:

- **preCheck()** – перевірка перед завантаженням.
- **openResources()** – відкриття необхідних ресурсів.
- **download()** – безпосередньо процес завантаження.
- **finalizeDownload()** – завершення процесу.

Цей клас дозволяє визначити загальну структуру алгоритму для завантаження, в той час як конкретні кроки реалізуються у підкласах (наприклад, для HTTP або P2P завантаження).

DownloadGroup (Композит):
Клас, який реалізує патерн **Composite**. Він дозволяє обробляти групи завдань як єдине ціле. Кожне завдання додається в групу, і коли виконується команда (наприклад, завантаження), вона виконується для всіх завдань в групі одночасно.

Peer та P2PManager (Імітація P2P):

- **Peer** – клас для представлення окремого піринга в мережі, який має локальну директорію з файлами для обміну.
- **P2PManager** – менеджер P2P, що керує списком пирів і намагається знайти ресурс, якого потребує користувач (віддаючи його з локальних директорій пирів).

CancelledException:
Спеціальний клас виключення для сигналізації про те, що завантаження було скасоване.

HttpDownloadTask (Реалізація завантаження HTTP):
Конкретна реалізація завантаження через HTTP. Тут реалізовано логіку перевірки доступності ресурсу через HTTP-запити, завантаження файлів з підтримкою продовження (Resume) та завершення завантаження.

P2PDownloadTask (Реалізація завантаження P2P):
Клас, що відповідає за завантаження через P2P-систему. Він імітує пошук файлів у локальній мережі пирів (через **P2PManager**) та завантаження цих файлів.

StartCommand та CancelCommand (Команди):

- **StartCommand** – команда для запуску завантаження. Вона передає завдання на виконання в **ExecutorService**, забезпечуючи асинхронне виконання.
- **CancelCommand** – команда для скасування завантаження. Вона викликає метод **cancel()** у завданні для скасування його виконання.

DownloadRegistry (Менеджер завдань):
Клас для реєстрації завдань завантаження, зберігає інформацію про статус кожного завдання, кількість завантажених байтів, а також загальний розмір. Клас також обробляє прогрес завантаження, оновлюючи інформацію про завдання через спостерігачів.

DownloadManager (HTTP-сервер та менеджер завантажень):
Головний клас програми, що містить сервер для керування завантаженнями через веб-інтерфейс. Клас налаштовує HTTP сервер на порту 8000, дозволяючи користувачам через браузер додавати нові

завдання на завантаження, перевіряти статуси завантажених файлів та відображати список поточних завдань.

- **handleRoot** – обробляє запити до кореневого каталогу.
- **handleStart** – обробляє запуск нових завдань на завантаження.
- **handleStatus** – обробляє запити про статус завантажень.
- **handleList** – обробляє запити про список завантажених файлів.

1. Опис реалізації шаблону Template Method у класі DownloadTask

1. У класі DownloadTask реалізовано шаблон Template Method, що дозволяє визначити основну послідовність операцій для завантаження файлів. Ми визначаємо загальний процес завантаження у методі runTask(), що включає кілька етапів:

   preCheck() – перевірка перед завантаженням.

   openResources() – підготовка ресурсів для завантаження.

   download() – власне завантаження.

   finalizeDownload() – завершення завантаження.

2. Ці методи абстрактні, тому реалізуються у підкласах. Для кожного конкретного типу завдання, такого як HttpDownloadTask чи P2PDownloadTask, реалізуються специфічні дії для кожного етапу завантаження.

```java
abstract class DownloadTask {  14 usages  3 inheritors
    final String id = UUID.randomUUID().toString();  4 usages
    protected volatile boolean cancelled = false;  2 usages
    protected final List<ProgressObserver> observers = new CopyOnWriteArrayList<>();
    protected final AtomicLong downloaded = new AtomicLong( initialValue: 0);  6 usages
    protected volatile long totalSize = -1;  11 usages
    protected final String sourceUrl;  1 usage
    protected final String targetFile;  4 usages

    DownloadTask(String sourceUrl, String targetFile) {  3 usages
        this.sourceUrl = sourceUrl;
        this.targetFile = targetFile;
    }

    public String getId() { return id; }  5 usages
    public void addObserver(ProgressObserver o){ observers.add(o); }  1 usage
    public void removeObserver(ProgressObserver o){ observers.remove(o); }  no usages

    public final void runTask() {  2 usages
        try {
            preCheck();
            openResources();
            download();
            finalizeDownload();
            notifyAllObservers( finished: true);
        } catch (CancelledException e) {
            notifyAllObservers( finished: false);
            System.out.println("Download cancelled: " + id);
        } catch (Exception e) {
            notifyAllObservers( finished: false);
            System.err.println("Error in download " + id + ": " + e.getMessage());
            e.printStackTrace();
        }
    }
    protected abstract void preCheck() throws Exception;  1 usage  3 implementations
    protected abstract void openResources() throws Exception;  1 usage  3 implementations
    protected abstract void download() throws Exception;  1 usage  3 implementations
    protected abstract void finalizeDownload() throws Exception;  1 usage  3 implementations
```

```
    protected void notifyAllObservers(boolean finished) {  6 usages
        for (ProgressObserver o : observers) {
            o.onProgress(id, downloaded.get(), totalSize, finished);
        }
    }

    public void cancel() { cancelled = true; }  1 usage

    protected void checkCancelled() throws CancelledException {  3 usages
        if (cancelled) throw new CancelledException();
    }
}
```

Метод **runTask()** є шаблонним методом, який визначає основну логіку завантаження, зокрема перевірку ресурсів, їх завантаження і завершення. Він викликає абстрактні методи, залишаючи конкретні деталі для підкласів.

2. Реалізація шаблону Composite у класі DownloadTaskGroup

Шаблон **Composite** дозволяє обробляти групи завдань як єдині об'єкти. Клас **DownloadTaskGroup** реалізує цей шаблон. Він містить колекцію завдань і дозволяє виконувати операції для всієї групи одночасно (наприклад, запуск, пауза, скасування).

```
class DownloadGroup extends DownloadTask {  no usages
    private final List<DownloadTask> children = new ArrayList<>();  2 usages
    DownloadGroup(String name) { super( sourceUrl: "group://" + name, name); }
    public void add(DownloadTask t) { children.add(t); }
    @Override protected void preCheck() throws Exception {}  1 usage
    @Override protected void openResources() throws Exception {}  1 usage
    @Override protected void download() throws Exception {  1 usage
        for (DownloadTask t : children) {
            checkCancelled();
            t.runTask();
        }
    }
    @Override protected void finalizeDownload() throws Exception {}  1 usage
}
```

Клас **DownloadTaskGroup** дозволяє додавати завдання до групи та керувати їх виконанням одночасно. Кожне завдання в групі виконується в

межах методу **download()**, де для кожного завдання викликається метод **runTask()**.

3. Реалізація шаблону Command

Шаблон **Command** дозволяє інкапсулювати операції над завданнями. Наприклад, команди для запуску, паузи або скасування завдання є незалежними об'єктами, які реалізують інтерфейс **Command**.

```
class StartCommand implements Command {  1 usage
    private final DownloadTask task;  2 usages
    private final ExecutorService exec;  2 usages
    StartCommand(DownloadTask task, ExecutorService exec) {  1 usage
        this.task = task; this.exec = exec; }
    @Override public void execute() { exec.submit(task::runTask); }  1 usage
}

class CancelCommand implements Command {  no usages
    private final DownloadTask task;  2 usages
    CancelCommand(DownloadTask task) { this.task = task; }  no usages
    @Override public void execute() { task.cancel(); }  1 usage
}
```

Клас **StartCommand** дозволяє запускати завдання за допомогою команди **execute()**, а **CancelCommand** дозволяє скасовувати завдання. Оскільки ці команди інкапсулюють операції, їх можна зберігати, повторно виконувати або скасовувати в майбутньому.

1.3. Висновки.

Шаблон Composite дозволяє створювати складні структури з окремих елементів, які можна обробляти як єдине ціле. У нашому випадку, використання шаблону Composite в класі DownloadTaskGroup дозволяє обробляти групи завдань одночасно, що значно спрощує управління великими наборами завдань. Цей підхід також забезпечує гнучкість у виконанні операцій на всіх елементах групи (наприклад, пауза або відновлення завантаження для всіх завдань у групі). Замість того, щоб викликати операцію на кожному завданні окремо, ми можемо скористатися методом групи, який виконає всі необхідні операції одночасно.

Використання шаблону Composite дозволяє нам уникнути дублювання коду і зробити структуру більш чистою і зрозумілою. Ми можемо додавати

нові завдання в групу, не змінюючи основну логіку програми, оскільки група автоматично опрацьовує всі операції над елементами. Такий підхід дозволяє краще управляти великими проектами та автоматизувати обробку завдань.

Шаблон Template Method є потужним інструментом для визначення основної структури алгоритму, де специфічні кроки можуть бути реалізовані в підкласах. У нашій програмі клас DownloadTask використовує шаблон Template Method для визначення основного алгоритму завантаження, де реалізовані загальні кроки, такі як підготовка ресурсів та завершення завантаження. Специфічні для типу завантаження кроки, такі як перевірка, завантаження і фіналізація, реалізуються в підкласах, таких як HttpDownloadTask чи P2PDownloadTask.

Шаблон Template Method дозволяє нам стандартизувати основні операції завантаження та залишити можливість налаштувати або змінити конкретні етапи для різних типів завантажень. Таким чином, загальний процес залишаються стабільним, а реалізація специфічних кроків може бути змінена без впливу на загальну структуру. Це значно підвищує гнучкість та зручність модифікацій програми.

Шаблон Command дозволяє інкапсулювати операції, що дає змогу керувати ними за допомогою команд, які можуть бути виконані, скасовані або повторені. У нашій програмі клас StartCommand використовується для інкапсуляції операції запуску завдання, а CancelCommand — для скасування. Ці команди виконують операції в окремих об'єктах і дозволяють легко додавати нову поведінку, таку як скасування або повторне виконання команд. Цей підхід спрощує код і дозволяє зберігати операції в окремих об'єктах, що значно підвищує можливості для керування завданнями та їх станами.

Шаблон Command допомагає нам створити гнучку систему, де кожна операція є окремим об'єктом, який може бути виконаний в будь-який час, що зручне для реалізації складних сценаріїв. Це дозволяє застосовувати такі операції, як скасування або повторення, що підвищує стабільність і зручність програмної логіки.

Загалом, використання патернів проектування в програмі для Download Manager значно покращує структуру і підтримку коду, знижує його складність і дозволяє легше додавати нові функціональні можливості.

Патерни забезпечують високу гнучкість і дозволяють ефективно керувати завданнями та їхнім виконанням. Шаблони Composite, Template Method і Command забезпечують хорошу основу для розширення програми та її адаптації під нові вимоги.

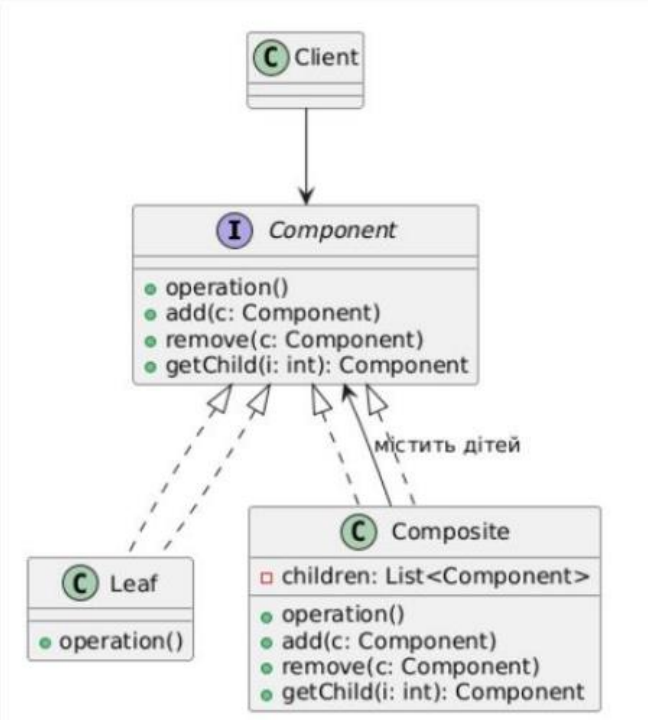
Також варто зазначити, що застосування таких патернів дозволяє оптимізувати та стандартизувати розробку програмних рішень, що особливо важливо при роботі з великими і складними проектами. У майбутньому це забезпечить легшу інтеграцію нових можливостей без необхідності переписувати основну структуру програми.

1.4 Контрольні запитання

1. Яке призначення шаблону «Композит»?

Шаблон **Композит** використовується для представлення ієрархічних структур, які мають типи "ціле–частина". Цей шаблон дозволяє об'єднувати окремі об'єкти в деревоподібні структури та працювати з ними однаково, незалежно від того, чи це єдине завдання, чи ціла група завдань. Клієнт може викликати однакові методи для окремих елементів чи для контейнерів, що містять ці елементи.

2. Нарисуйте структуру шаблону «Композит».



3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

Component — це спільний інтерфейс для листків (Leaf) та вузлів (Composite). Клієнт взаємодіє лише з цим інтерфейсом, не зважаючи на тип елемента.

Leaf — це кінцевий елемент без дітей. Він реалізує реальну роботу у методі operation().

Composite — це контейнер, що містить колекцію інших Component. Він викликає методи для своїх дочірніх елементів і делегує їм роботу.

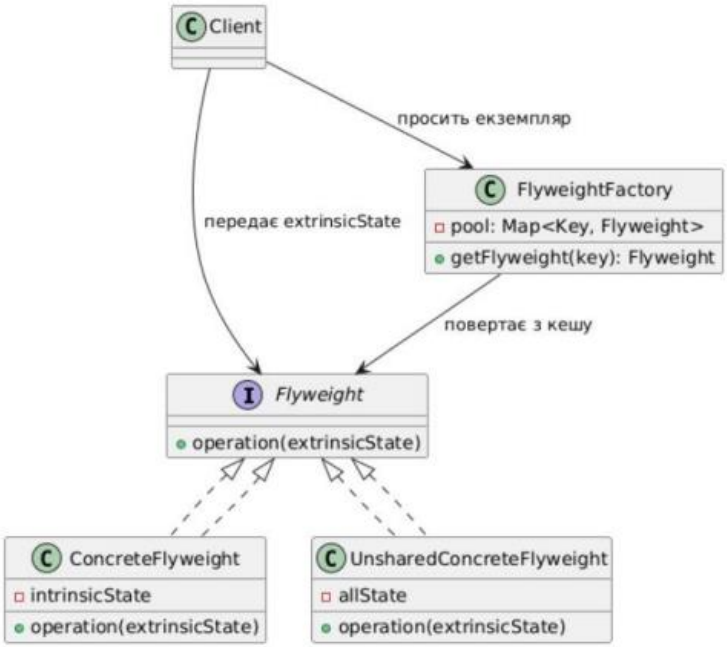
Client — це об'єкт, який викликає методи через інтерфейс Component без розрізнення між листком або вузлом.

4. Яке призначення шаблону «Легковаговик»?

Шаблон **Легковаговик** сприяє оптимізації пам'яті за допомогою використання пулу об'єктів. Патерн допомагає зберігати стан лише для спільних, незмінних частин об'єктів, мінімізуючи дублювання однотипних елементів. Він розділяє стан об'єкта на два типи:

- **Intrinsic** — внутрішній, незмінний, спільний стан, що зберігається в об'єкті **Flyweight**.
- **Extrinsic** — зовнішній, змінний стан, який передається клієнтом під час виклику.

5. Нарисуйте структуру шаблону «Легковаговик».



6. Які класи входять в шаблон «Легковаговик», та яка між ними взаємодія?

Flyweight — інтерфейс, що визначає операцію, яка приймає extrinsic стан.

ConcreteFlyweight — клас, який містить intrinsic стан і реалізує логіку в методі operation().

UnsharedConcreteFlyweight — варіант без спільного використання, коли екземпляр об'єкта унікальний.

FlyweightFactory — управляє пулом об'єктів і повертає вже створені екземпляри за певним ключем.

Client — не зберігає intrinsic стан у Flyweight, але при кожному виклику передає extrinsic стан.

7. Яке призначення шаблону «Інтерпретатор»?

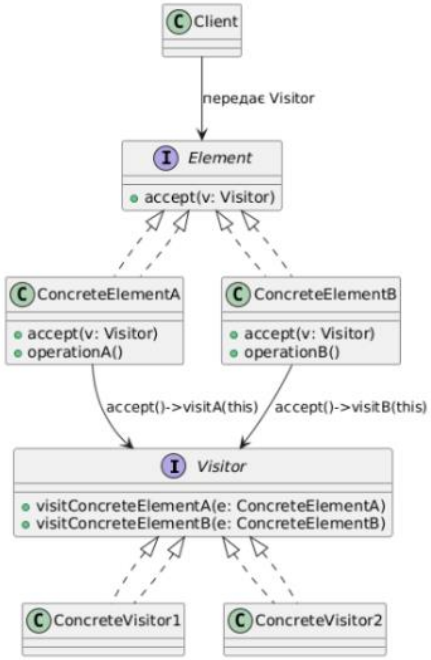
Шаблон **Інтерпретатор** надає спосіб визначити граматику предметної мови та інтерпретувати вирази цієї мови через набір класів, що представляють правила. Цей шаблон корисний, коли потрібно часто обчислювати вирази, наприклад, логічні вирази чи арифметичні формули, і забезпечує гнучкість у побудові та обчисленні виразів

8. Яке призначення шаблону «Відвідувач»?

Шаблон **Відвідувач** дозволяє додавати нові операції до існуючих класів без змін у їхній структурі. Елементи класів приймають відвідувача через

метод **accept(visitor)**, який виконує певну операцію, використовуючи перевантажені методи **visitXxx(...)** для різних типів елементів. Це зручно для виконання операцій на складних структурах, наприклад, при обході дерева з різними видами «обробки».

9. Нарисуйте структуру шаблону «Відвідувач».



10.Які класи входять в шаблон «Відвідувач», та яка між ними взаємодія?

Element — інтерфейс для елементів, який містить метод accept(Visitor).
ConcreteElementA/B — конкретні елементи, які в методі accept() викликають відповідний метод visitXxx(this).
Visitor — інтерфейс для відвідувача, що містить методи visit... для кожного типу елементів.
ConcreteVisitor1/2 — реалізації різних операцій, таких як аналіз, підрахунок, експорт тощо.
Client — формує колекцію елементів і передає їх через відповідного відвідувача для виконання операцій.