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
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»»**
Download manager

Виконала:
Студентка групи ІА-22
Степанюк-Боримська А. І.

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Мета:
Ознайомитися з основними шаблонами проєктування, такими як «Singleton», «Iterator», «Proxy», «State» та «Strategy», вивчити їхні принципи роботи та навчитись застосовувати для створення гнучкого та масштабованого програмного забезпечення в загальній розробці програмних систем.

Хід роботи

1. Реалізувати не менше 3-х класів відповідно до обраної теми

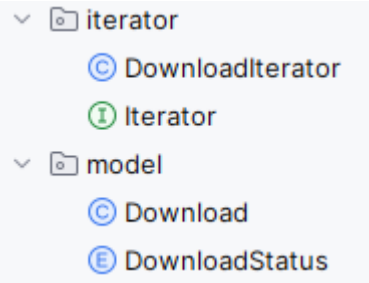

Рис. 1 — Структура проекту

Під час виконання лабораторної роботи було розроблено чотири класи, які реалізують функціонал менеджера завантажень (рис. 1). Нижче наведено детальний опис кожного з цих класів:

1. **Download**
   Опис:
   Клас, що представляє завантаження файлу з інтернету. Він містить усі основні атрибути, що описують файл, що завантажується, а саме: ім’я файлу, URL, розмір файлу, статус завантаження та прогрес завантаження.
   Призначення:
   - Зберігає інформацію про конкретне завантаження.
   - Дає можливість працювати з такими даними, як статус завантаження (наприклад, чи виконано завантаження), прогрес завантаження (відсотки) і розмір файлу.
   - Призначений для використання в інших компонентах програми, що потребують ці дані для обробки чи відображення користувачу.
2. **DownloadStatus**
   Опис:

Це клас, який містить можливі стани завантаження файлу. Кожен елемент цього переліку відображає конкретний статус, в якому може перебувати завантаження.

Призначення:

- Статуси завантаження включають: PENDING (очікує на початок), IN_PROGRESS (завантаження відбувається), PAUSED (завантаження призупинено), COMPLETED (завершено), та FAILED (не вдалося завершити).
- Використовується для чіткого визначення поточного стану завантаження в програмі, що допомагає організувати логіку виконання (наприклад, призупинити завантаження або вивести повідомлення про помилку).

3. **Iterator**

Опис:

Це загальний інтерфейс ітератора, який визначає методи для перебору елементів колекції. У нашому випадку цей інтерфейс описує базові методи для ітерації по елементах будь-якої колекції типу <T>.

Призначення:

- Надання загальної структури для ітераторів, які можуть працювати з різними типами даних.
- Визначає два основні методи: hasNext() (для перевірки наявності наступного елементу) та next() (для отримання наступного елемента в колекції).
- Цей інтерфейс дозволяє створювати ітератори для будь-якої колекції, що робить код більш універсальним і спрощує взаємодію з різними типами даних.

4. **DownloadIterator**

Опис:

Клас, що реалізує інтерфейс Iterator для колекції завантажень (Download). Цей клас надає специфічну логіку ітерації для списку об’єктів типу Download.

Призначення:

- Забезпечує можливість перебирати елементи списку завантажень, кожен з яких є об'єктом типу Download.
- Реалізує методи hasNext() та next(), які дозволяють зручно ітерувати по завантаженням, перевіряти їх на наявність наступного елемента і отримувати наступне завантаження.

- Забезпечує відокремлену від колекції логіку ітерації, що робить код чистим і зручним для використання в інших компонентах програми.

2. Реалізувати один з розглянутих шаблонів за обраною темою

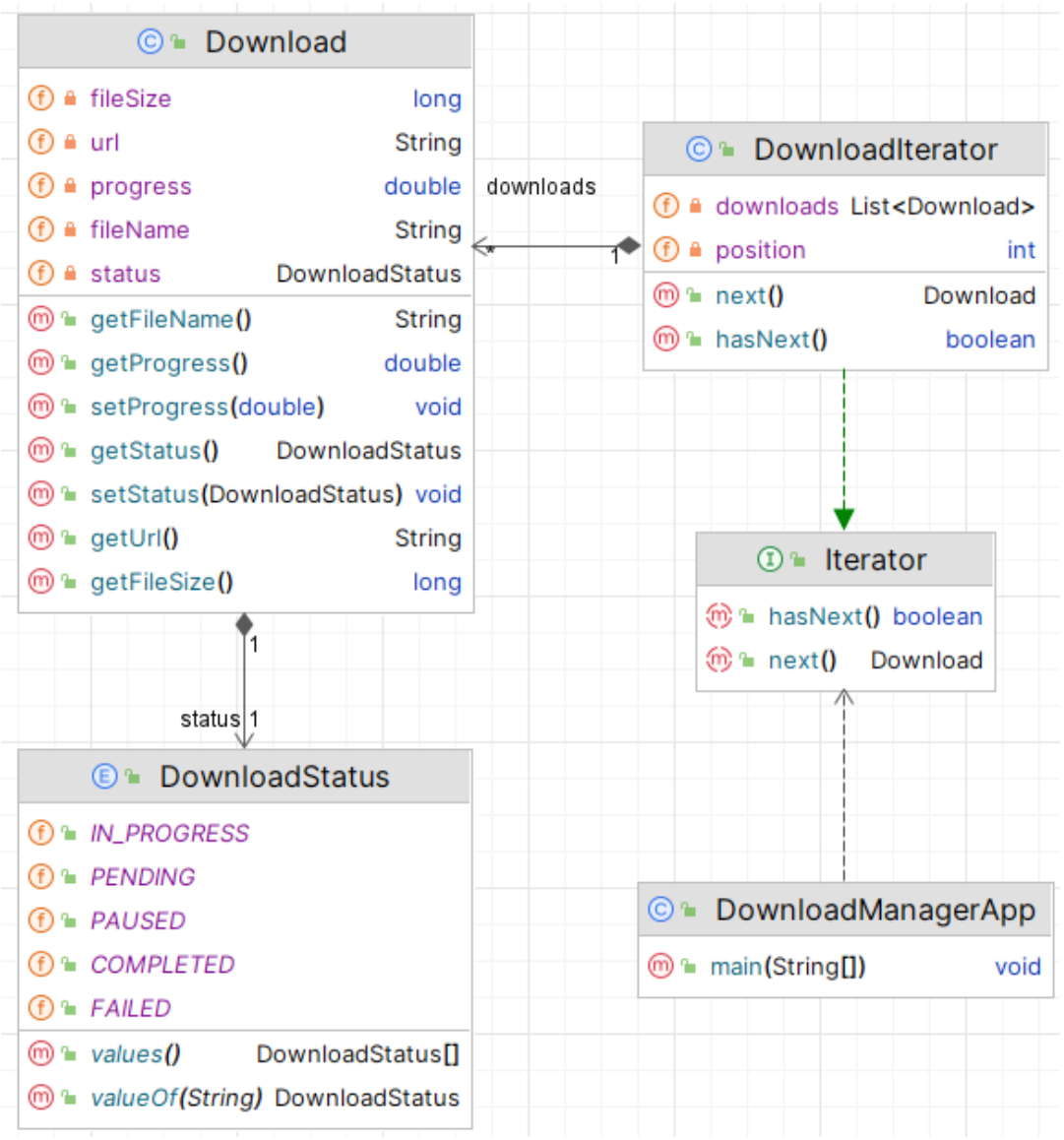

Рис. 2 — Діаграма класів

В нашому проекті менеджера завантажень патерн Iterator був реалізований для ефективного перебору списку завантажень файлів. Це дозволяє зручно працювати з колекцією завантажень, що зберігає різні стани (наприклад, завантаження в процесі, завершені або призупинені), не розкриваючи внутрішню структуру цієї колекції.

Як реалізований патерн Iterator

1. Ітератор працює з колекцією завантажень: Ми створили клас DownloadIterator, який реалізує інтерфейс Iterator. Цей ітератор перебирає список завантажень, що зберігаються в колекції, і дає змогу отримувати по черзі елементи (об’єкти типу Download).
2. Використання інтерфейсу Iterator: Ітератор визначає два основних методи:
   - hasNext(), який перевіряє, чи є ще елементи в колекції.
   - next(), який повертає наступний елемент у колекції.

   Завдяки цьому користувач чи програма може отримувати доступ до кожного завантаження без необхідності знати, як саме організована колекція чи список завантажень.
3. Колекція завантажень ітератора: Для зберігання завантажень використовувалася звичайна колекція List<Download>, а сам ітератор надавав доступ до її елементів по черзі. Це дозволяє користувачам програми зручно обробляти всі активні завантаження, не турбуючись про те, як саме зберігаються ці дані.

Проблеми, які вирішує патерн Iterator

1. Абстрагування логіки перебору: Патерн Iterator приховує внутрішню реалізацію колекції, дозволяючи зручно працювати з її елементами без необхідності знати про конкретну структуру даних (чи це список, масив чи інший тип колекції).
2. Покращення гнучкості: Завдяки ітератору, ми можемо змінювати колекцію (наприклад, додавати нові елементи або змінювати структуру даних), не змінюючи коду, що взаємодіє з цією колекцією. Ітератор забезпечує стандартизований доступ до елементів, що робить програму більш гнучкою.
3. Легкість обробки великих наборів даних: Оскільки ітератор дозволяє по черзі отримувати елементи, програма може ефективно працювати з великими колекціями завантажень без необхідності завантажувати всю колекцію в пам'ять або виконувати складні операції для доступу до окремих елементів.

Переваги використання

1. Простота коду: Ітератор спрощує доступ до елементів колекції, дозволяючи зосередитися на логіці обробки завантажень, а не на способах

доступу до елементів колекції. Це робить код більш чистим і легким для розуміння.

2. Модульність: Логіка перебору елементів ізолюється від решти програми. Це означає, що якщо ми змінюємо структуру даних (наприклад, замінюємо список на інший тип колекції), нам не потрібно змінювати код, що використовує ітератор.
3. Покращення підтримки та розширюваності: Додавання нових типів колекцій або зміна поточної структури не потребує великих змін у коді. Потрібно лише оновити ітератор, якщо структура колекції змінюється. Це дозволяє розширювати програму без втручання в основну логіку.
4. Краще управління пам'яттю: Патерн Iterator дозволяє працювати з елементами колекції по одному, що може бути особливо корисно для великих списків завантажень, де потрібно мінімізувати використання пам'яті.

Перевірка роботи

```
public class DownloadManagerApp {
    public static void main(String[] args) {
        List<Download> downloads = new ArrayList<>();
        downloads.add(new Download( fileName: "file1.zip",  url: "http://example.com/file1.zip",  fileSize: 1000, DownloadStatus.IN_PROGRESS,  progress: 50.0));
        downloads.add(new Download( fileName: "file2.zip",  url: "http://example.com/file2.zip",  fileSize: 2000, DownloadStatus.COMPLETED,  progress: 100.0));
        downloads.add(new Download( fileName: "file3.zip",  url: "http://example.com/file3.zip",  fileSize: 1500, DownloadStatus.PAUSED,  progress: 25.0));

        Iterator iterator = new DownloadIterator(downloads);

        System.out.println("Downloads:");
        while (iterator.hasNext()) {
            Download download = iterator.next();
            System.out.println("File: " + download.getFileName() +
                    ", Status: " + download.getStatus() +
                    ", Progress: " + download.getProgress() + "%");
        }
    }
}
```

Рис. 3 — Перевірка роботи

Ми створили список завантажень із різними статусами (IN_PROGRESS, COMPLETED, PAUSED) і використали ітератор (DownloadIterator) для поетапного доступу до кожного елемента списку. Вивели деталі кожного завантаження (назва файлу, статус, прогрес) за допомогою ітератора.

```
Downloads:
File: file1.zip, Status: IN_PROGRESS, Progress: 50.0%
File: file2.zip, Status: COMPLETED, Progress: 100.0%
File: file3.zip, Status: PAUSED, Progress: 25.0%
```

Рис. 4 — Результат роботи

Результат показує, що ітератор успішно перебирає всі завантаження у списку й надає доступ до їхніх даних. Це демонструє коректну реалізацію патерну Iterator, який забезпечує послідовний і зручний доступ до елементів колекції без залежності від її внутрішньої структури.

Висновки:

У ході лабораторної роботи було реалізовано патерн Iterator для ефективного перебору колекції завантажень у проекті менеджера завантажень. Патерн дозволяє абстрагувати логіку доступу до елементів колекції, спрощує обробку великих наборів даних, покращує гнучкість і підтримку коду, а також робить програму модульною та зручною для розширення.