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 №7
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»»**
Download manager

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

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Мета:
Ознайомитися з основними шаблонами проєктування, такими як «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD», 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

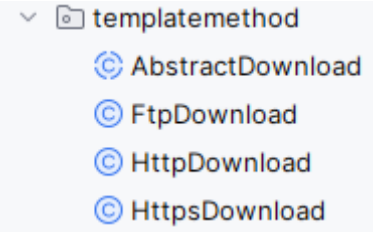

Рис. 1 — Структура проекту

Під час виконання лабораторної роботи було розроблено 4 класи, які реалізують функціонал менеджера завантажень (рис. 1). Нижче наведено детальний опис кожного з цих класів:

1. **AbstractDownload**
   Опис:
   Це базовий клас, який визначає загальний алгоритм завантаження файлу. Він використовує метод executeDownload(), що є шаблонним методом і складається з кількох етапів, деякі з яких (наприклад, автентифікація та завантаження файлу) делегуються до підкласів через абстрактні методи.
   Призначення:
   Забезпечити єдину структуру алгоритму для завантаження файлів незалежно від протоколу (HTTP, HTTPS, FTP тощо), дозволяючи підкласам реалізовувати специфічну поведінку для кожного протоколу.
2. **HttpDownload**
   Опис:
   Підклас AbstractDownload, який реалізує специфічні кроки для завантаження через протокол HTTP. Зокрема, цей клас визначає, що автентифікація для HTTP не потрібна, і реалізує завантаження файлу через HTTP.

Призначення:

Реалізувати специфічну логіку завантаження для протоколу HTTP, при цьому використовуючи загальний алгоритм, визначений у базовому класі.

3. **HttpsDownload**

Опис:

Підклас AbstractDownload, який реалізує кроки для завантаження через протокол HTTPS. Включає автентифікацію через SSL і реалізує завантаження файлу через захищений HTTPS-з'єднання.

Призначення:

Додати специфічну логіку для завантажень через HTTPS, зокрема автентифікацію через SSL, забезпечуючи безпеку даних під час передачі.

4. **FtpDownload**

Опис:

Підклас AbstractDownload, який реалізує специфічні кроки для завантаження через протокол FTP. Зокрема, цей клас визначає автентифікацію через облікові дані FTP і завантаження файлу через FTP-з'єднання.

Призначення:

Реалізувати специфічну логіку завантаження через FTP, дозволяючи інтегрувати цей протокол у загальний алгоритм, визначений у базовому класі.

2. Реалізувати один з розглянутих шаблонів за обраною темою

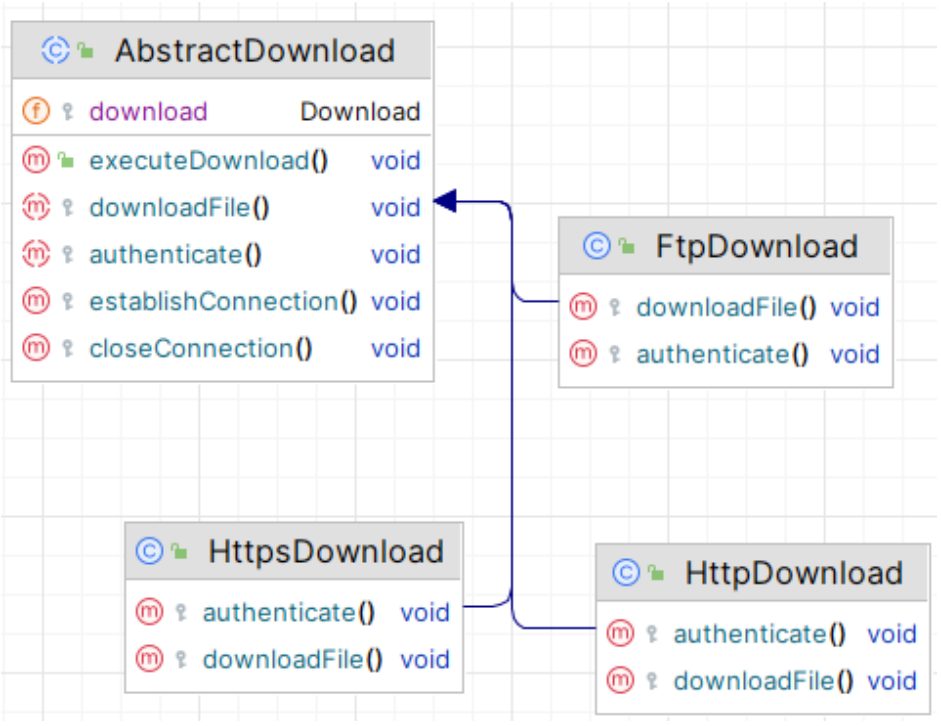

Рис. 2 — Діаграма класів

У контексті проекту менеджера завантажень, патерн Template Method використовується для організації єдиного алгоритму завантаження файлів із різних джерел (HTTP, HTTPS, FTP) із урахуванням специфіки кожного протоколу. Загальні етапи завантаження визначено в базовому класі, а специфічні дії делеговано підкласам.

Як реалізований патерн Template Method

Базовий клас AbstractDownload визначає шаблонний метод executeDownload(), який складається з таких етапів:

1. Встановлення з'єднання (загальний для всіх протоколів).
2. Автентифікація (реалізується підкласами, залежно від протоколу).
3. Завантаження файлу (реалізується підкласами).
4. Закриття з'єднання (загальний для всіх протоколів).

Конкретні протоколи (HTTP, HTTPS, FTP) реалізують лише специфічні кроки, що стосуються їхньої логіки, а весь процес завантаження залишається стандартизованим.

Проблеми, які вирішує патерн Template Method

1. Дублювання коду: Загальні частини алгоритму (наприклад, встановлення і закриття з'єднання) винесені в базовий клас, що усуває дублювання в підкласах.
2. Уніфікація алгоритму: Незалежно від типу протоколу, завантаження завжди виконується через метод executeDownload(), що забезпечує єдину структуру для всіх операцій.
3. Гнучкість та розширюваність: Додавання підтримки нового протоколу (наприклад, SFTP) потребує лише створення нового підкласу з реалізацією специфічних кроків, не зачіпаючи існуючу структуру.
4. Легкість підтримки: Завдяки чіткій структурі шаблонного методу легко змінювати або оновлювати загальні етапи алгоритму, не змінюючи підкласи.
5. Прозорість використання: Користувачеві не потрібно знати специфіку протоколу, оскільки викликається лише загальний метод executeDownload().

Переваги використання Template Method

1. Повторне використання коду: Загальні кроки алгоритму визначені один раз у базовому класі, що зменшує розмір і складність коду підкласів.

2. Модульність: Логіка завантаження розбита на чіткі частини: спільні для всіх протоколів (у базовому класі) та унікальні для кожного протоколу (у підкласах).
3. Розширюваність: Легко додавати нові протоколи завантаження, створюючи нові підкласи, не змінюючи існуючий код.
4. Стандартизація: Завдяки шаблонному методу всі завантаження виконуються за єдиним алгоритмом, що спрощує інтеграцію й тестування системи.

Перевірка роботи

```
public class DownloadManagerApp {
    public static void main(String[] args) {
        Download httpFile = new Download( fileName: "file1.txt",  url: "http://example.com/file1",  fileSize: 1024, DownloadStatus.PENDING,  progress: 0.0);
        Download httpsFile = new Download( fileName: "file2.txt",  url: "https://secure.com/file2",  fileSize: 2048, DownloadStatus.PENDING,  progress: 0.0);
        Download ftpFile = new Download( fileName: "file3.txt",  url: "ftp://ftpserver.com/file3",  fileSize: 4096, DownloadStatus.PENDING,  progress: 0.0);

        AbstractDownload httpDownload = new HttpDownload(httpFile);
        AbstractDownload httpsDownload = new HttpsDownload(httpsFile);
        AbstractDownload ftpDownload = new FtpDownload(ftpFile);

        httpDownload.executeDownload();
        httpsDownload.executeDownload();
        ftpDownload.executeDownload();
    }
}
```

Рис. 3 — Перевірка роботи

У цьому прикладі ми створили три об’єкти Download для файлів, що завантажуються через різні протоколи (HTTP, HTTPS, FTP). Потім для кожного з них ми створили відповідний підклас AbstractDownload (HttpDownload, HttpsDownload, FtpDownload), який реалізує специфіку роботи з цими протоколами. Викликавши метод executeDownload() для кожного об'єкта, ми перевірили, як шаблонний метод керує процесом завантаження.

```
Connecting to http://example.com/file1
HTTP download does not require authentication.
Downloading file via HTTP from http://example.com/file1
Closing connection for http://example.com/file1
Connecting to https://secure.com/file2
Authenticating via SSL for HTTPS connection.
Downloading file via HTTPS from https://secure.com/file2
Closing connection for https://secure.com/file2
Connecting to ftp://ftpserver.com/file3
Authenticating with FTP credentials.
Downloading file via FTP from ftp://ftpserver.com/file3
Closing connection for ftp://ftpserver.com/file3
```

Рис. 4 — Результат роботи

У результаті виконання ми бачимо послідовний виклик загальних кроків (встановлення та закриття з'єднання) і специфічних для кожного протоколу (автентифікація, завантаження). Кожен файл успішно завантажується через свій протокол, що підтверджує правильну роботу шаблонного методу і розділення логіки між базовим класом і підкласами.

Висновки:

Реалізація патерну Template Method дозволила уніфікувати процес завантаження для різних протоколів, зробивши систему гнучкою, модульною та легкою в розширенні. Це рішення не лише усунуло дублювання коду, а й забезпечило стандартизацію алгоритму завантаження, що спрощує підтримку та розвиток проекту.