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 №6
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»»**
Download manager

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

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Мета:
Ознайомитися з основними шаблонами проєктування, такими як «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator», 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) та оновлено попередній клас завантаження. Нижче наведено детальний опис кожного з цих класів:

1. **Observer**
   Опис:
   Інтерфейс, який визначає метод update(). Він реалізується всіма класами, які хочуть отримувати повідомлення про зміни в об’єктах Download.
   Призначення:
   Забезпечує єдину точку взаємодії між спостережуваними об’єктами (Download) та їх спостерігачами. Гарантує, що всі спостерігачі реалізують однакову логіку отримання оновлень.
2. **Observable**
   Опис:
   Абстрактний клас, який додає механізм для управління списком спостерігачів (Observer) та сповіщення їх про зміни. Він надає методи addObserver, removeObserver та notifyObservers.

Призначення:

Дозволяє будь-якому класу, який його успадковує, стати спостережуваним об'єктом. Цей клас є базою для Download, щоб забезпечити можливість реєстрації спостерігачів і надсилання їм повідомлень про зміни.

3. **ProgressDisplay**

   Опис:

   Клас, що реалізує інтерфейс Observer. Він відповідає за відображення інформації про стан завантаження (ім’я файлу, статус, прогрес) в реальному часі, наприклад, у консолі.

   Призначення:

   Реагує на зміни об'єкта Download і виводить оновлені дані для користувача. Це може бути частиною інтерфейсу користувача для відображення прогресу завантажень.

4. **EventLogger**

   Опис:

   Ще один спостерігач, який реалізує інтерфейс Observer. Він веде журнал подій, зберігаючи інформацію про зміни завантаження у вигляді логів.

   Призначення:

   Забезпечує запис інформації про зміни у файл чи консоль для подальшого аналізу або відстеження історії дій у системі.

5. **Download**

   Клас Download тепер успадковує функціонал від абстрактного класу Observable, що дозволяє йому працювати зі списком спостерігачів та сповіщати їх про зміни. Додано виклики методу notifyObservers() у таких методах:

   - setProgress(double progress): Сповіщає спостерігачів про зміну прогресу завантаження.
   - setStatus(DownloadStatus status): Сповіщає спостерігачів про зміну статусу завантаження.

2. Реалізувати один з розглянутих шаблонів за обраною темою

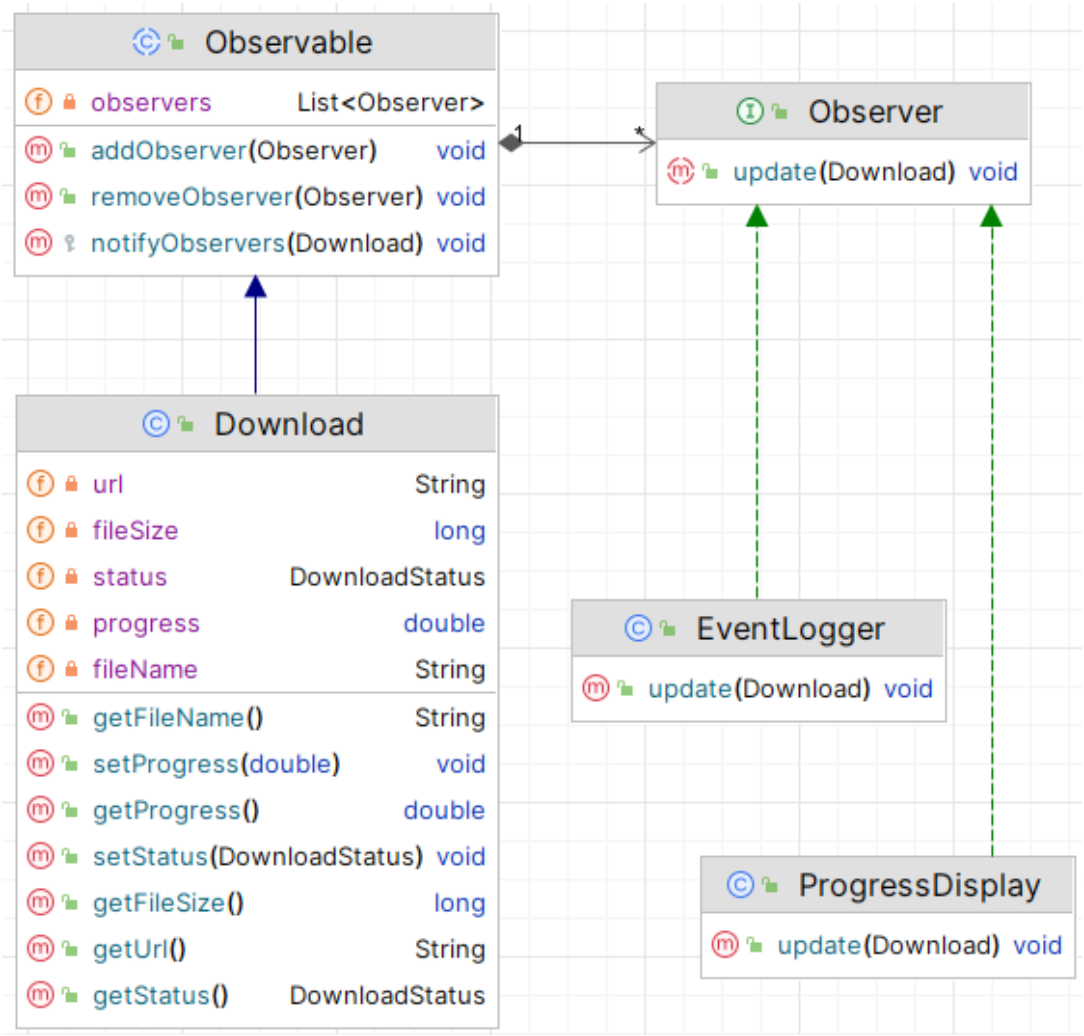

Рис. 2 — Діаграма класів

У контексті проекту менеджера завантажень патерн Observer був реалізований для автоматичного сповіщення зацікавлених компонентів про зміни в завантаженнях. Основна ідея полягала в тому, щоб зробити об’єкти Download "спостережуваними", дозволивши іншим компонентам (спостерігачам) підписуватися на їхні зміни, отримуючи актуальну інформацію без потреби в ручному опитуванні стану завантажень.

Як реалізований патерн Observer

1. Зв’язок між об'єктами: Клас Download було оновлено для успадкування функціоналу Observable. Це дало можливість реєструвати спостерігачів і автоматично сповіщати їх про зміни, такі як оновлення прогресу або зміна статусу завантаження.
2. Сповіщення спостерігачів: При кожній зміні прогресу чи статусу в Download викликається метод notifyObservers, який надсилає оновлення всім зареєстрованим спостерігачам (наприклад, ProgressDisplay або EventLogger). Це дозволяє забезпечити актуальність даних у реальному часі.
3. Декларативний підхід: Спостерігачі (компоненти, які реалізують інтерфейс Observer) отримують оновлення через єдиний метод update, який абстрагує логіку обробки змін від логіки роботи об’єкта Download.

Проблеми, які вирішує патерн Observer

1. Актуальність даних: Завдяки автоматичному сповіщенню, інтерфейс користувача, логери або інші компоненти завжди отримують актуальну інформацію про стан завантажень.
2. Зменшення зв’язності: Патерн дозволяє уникнути жорсткого зв’язку між об'єктами. Клас Download не потребує знань про те, хто саме підписаний на його зміни, а спостерігачі не залежать від деталей реалізації завантаження.
3. Зручність масштабування: Легко додати нові спостерігачі (наприклад, віджет для прогресу або систему сповіщень), не змінюючи існуючий код об’єкта Download.

Переваги використання

1. Автоматизація: Компоненти автоматично синхронізуються зі змінами завантажень, що знижує ймовірність помилок і забезпечує правильну реакцію системи на події.
2. Розділення відповідальностей: Download відповідає лише за управління завантаженням, а відображення прогресу, логування та інші дії делегуються окремим спостерігачам.
3. Гнучкість: Завдяки незалежності компонентів система легко адаптується до змін або розширюється новими функціями.
4. Актуальність : Сповіщення про зміни відбувається миттєво, що особливо важливо для динамічних додатків, таких як менеджер завантажень.

Перевірка роботи

```
public class DownloadManagerApp {
    public static void main(String[] args) {
        Download download = new Download( fileName: "file1.zip",  url: "http://example.com/file1.zip",  fileSize: 1000, DownloadStatus.PENDING,  progress: 0);

        ProgressDisplay progressDisplay = new ProgressDisplay();
        EventLogger eventLogger = new EventLogger();

        download.addObserver(progressDisplay);
        download.addObserver(eventLogger);

        download.setStatus(DownloadStatus.IN_PROGRESS);
        download.setProgress(25.0);
        download.setProgress(50.0);
        download.setStatus(DownloadStatus.COMPLETED);
    }
}
```

Рис. 3 — Перевірка роботи

Ми створили клас DownloadManagerApp, в якому ініціалізували завантаження файлу з початковими параметрами (ім'я файлу, URL, розмір, статус та прогрес). Потім створили два спостерігачі: ProgressDisplay (для відображення прогресу завантаження) та EventLogger (для логування подій). Останній крок включав в себе зміну статусу та прогресу завантаження, при цьому ці зміни автоматично повідомляли обох спостерігачів, що є реалізацією патерну Observer.

```
Download updated: file1.zip | Status: IN_PROGRESS | Progress: 0.0%
[LOG] File: file1.zip | Status: IN_PROGRESS | Progress: 0.0%
Download updated: file1.zip | Status: IN_PROGRESS | Progress: 25.0%
[LOG] File: file1.zip | Status: IN_PROGRESS | Progress: 25.0%
Download updated: file1.zip | Status: IN_PROGRESS | Progress: 50.0%
[LOG] File: file1.zip | Status: IN_PROGRESS | Progress: 50.0%
Download updated: file1.zip | Status: COMPLETED | Progress: 50.0%
[LOG] File: file1.zip | Status: COMPLETED | Progress: 50.0%
```

Рис. 4 — Результат роботи

Результат показує, що коли ми змінюємо статус і прогрес завантаження, обидва спостерігачі отримують ці оновлення. ProgressDisplay виводить повідомлення про поточний прогрес завантаження, а EventLogger зберігає лог з оновленими даними. Це демонструє коректну реалізацію патерну Observer, де спостерігачі оновлюються автоматично на основі змін у спостережуваному об'єкті, що підтверджує успішну роботу патерну.

Висновки:

У ході лабораторної роботи було реалізовано патерн Observer для автоматичного сповіщення компонентів про зміни в завантаженнях у менеджері завантажень. Це забезпечило актуальність даних, зменшило зв’язаність між об’єктами та спростило масштабування системи.