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»»

Варіант №26

Виконав:
студент групи ІА-23
Мозоль В.О

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема. 3
Мета. 3
Завдання. 3
Обрана тема. 3
Короткі теоретичні відомості. 4
Хід роботи. 7
Робота паттерну. 9
Висновки. 10
Додаток А. 11

Тема.
Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Мета.
Вивчити та застосувати на практиці шаблони проектування "Abstract Factory", "Factory Method", "Memento", "Observer" та "Decorator", навчитися використовувати їх для розробки програмного забезпечення. Отримати практичні навички реалізації патерну Observer на прикладі розробки системи Download Manager.

Завдання.
1 . Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Обрана тема.
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).

Короткі теоретичні відомості.

Принципи SOLID

1. S (Single Responsibility Principle – Принцип єдиного обов'язку)
    - Суть: Клас повинен виконувати лише одну задачу і мати одну причину для зміни.
    - Переваги:
        - Зменшення складності коду.
        - Спрощення тестування, налагодження та підтримки.
    - Приклад порушення: Клас, що одночасно відповідає за логіку збереження даних і відображення інтерфейсу.
    - Рішення: Розділити відповідальності між двома окремими класами.
2. O (Open/Closed Principle – Принцип відкритості/закритості)
    - Суть: Код повинен бути відкритий для розширення, але закритий для модифікації.
    - Переваги:
        - Запобігання помилкам у вже протестованому коді.
        - Легкість додавання нової функціональності.
    - Приклад реалізації: Використання інтерфейсів і абстрактних класів для визначення поведінки, яку можна розширити.
3. L (Liskov Substitution Principle – Принцип підстановки Лісков)
    - Суть: Кожен об'єкт базового класу повинен бути замінний об'єктом його підкласу без порушення працездатності системи.
    - Порушення: Якщо підклас змінює поведінку методу базового класу.
    - Приклад: Метод у підкласі не повинен кидати виняток, якщо базовий метод цього не робить.
4. I (Interface Segregation Principle – Принцип розділення інтерфейсу)
    - Суть: Краще створювати декілька вузькоспеціалізованих інтерфейсів, ніж один великий.
    - Переваги:

        - Клієнти працюють тільки з тими методами, які їм дійсно потрібні.
    - Приклад: Поділ інтерфейсу "Printer" на "Scanner", "Copier", "Printer" для багатофункціональних пристроїв.
5. D (Dependency Inversion Principle – Принцип інверсії залежностей)
    - Суть: Модулі верхнього рівня не повинні залежати від модулів нижнього рівня. Обидва повинні залежати від абстракцій.
    - Переваги:
        - Зменшення зв’язності системи.
        - Збільшення можливостей для тестування.
    - Реалізація: Інверсія залежностей через інтерфейси або фабричні методи.

Патерни проектування

1. Abstract Factory (Абстрактна фабрика)

- Суть: Дозволяє створювати сімейства пов’язаних об’єктів без вказівки їхніх конкретних класів.
- Переваги:
    - Забезпечує узгодженість створюваних об'єктів.
    - Полегшує заміну наборів об'єктів.
- Приклад використання: Різні графічні бібліотеки для Windows і macOS, де фабрика створює взаємопов’язані елементи інтерфейсу.

2. Factory Method (Фабричний метод)

- Суть: Визначає інтерфейс для створення об'єктів, але дозволяє підкласам вибирати конкретний тип об'єкта.
- Переваги:
    - Виносить логіку створення об’єктів за межі клієнтського коду.
    - Полегшує підтримку та розширення коду.
- Приклад: Створення об'єктів у різних форматах, наприклад, текстових чи XML-файлів.

3. Memento (Знімок)

- Суть: Зберігає внутрішній стан об'єкта, щоб можна було повернути його до цього стану.
- Переваги:
  - Забезпечує інкапсуляцію стану.
  - Зручність реалізації функції "Скасувати" (Undo).
- Приклад: Текстовий редактор, де зберігаються проміжні стани документа.

4. Observer (Спостерігач)

- Суть: Встановлює залежність "один-до-багатьох" між об'єктами, де один об'єкт (спостерігач) автоматично оновлюється при зміні іншого.
- Переваги:
  - Зручність для роботи з подіями.
  - Зменшення зв’язності компонентів.
- Приклад: Система новин, де всі підписники отримують повідомлення про нові статті.

5. Decorator (Декоратор)

- Суть: Динамічно додає нову поведінку до об'єкта, залишаючи незмінним його інтерфейс.
- Переваги:
  - Гнучка альтернатива наслідуванню.
  - Можливість багаторазового комбінування поведінок.
- Приклад: Декоратори для графічних елементів, таких як рамки, тіні тощо.

Хід роботи.

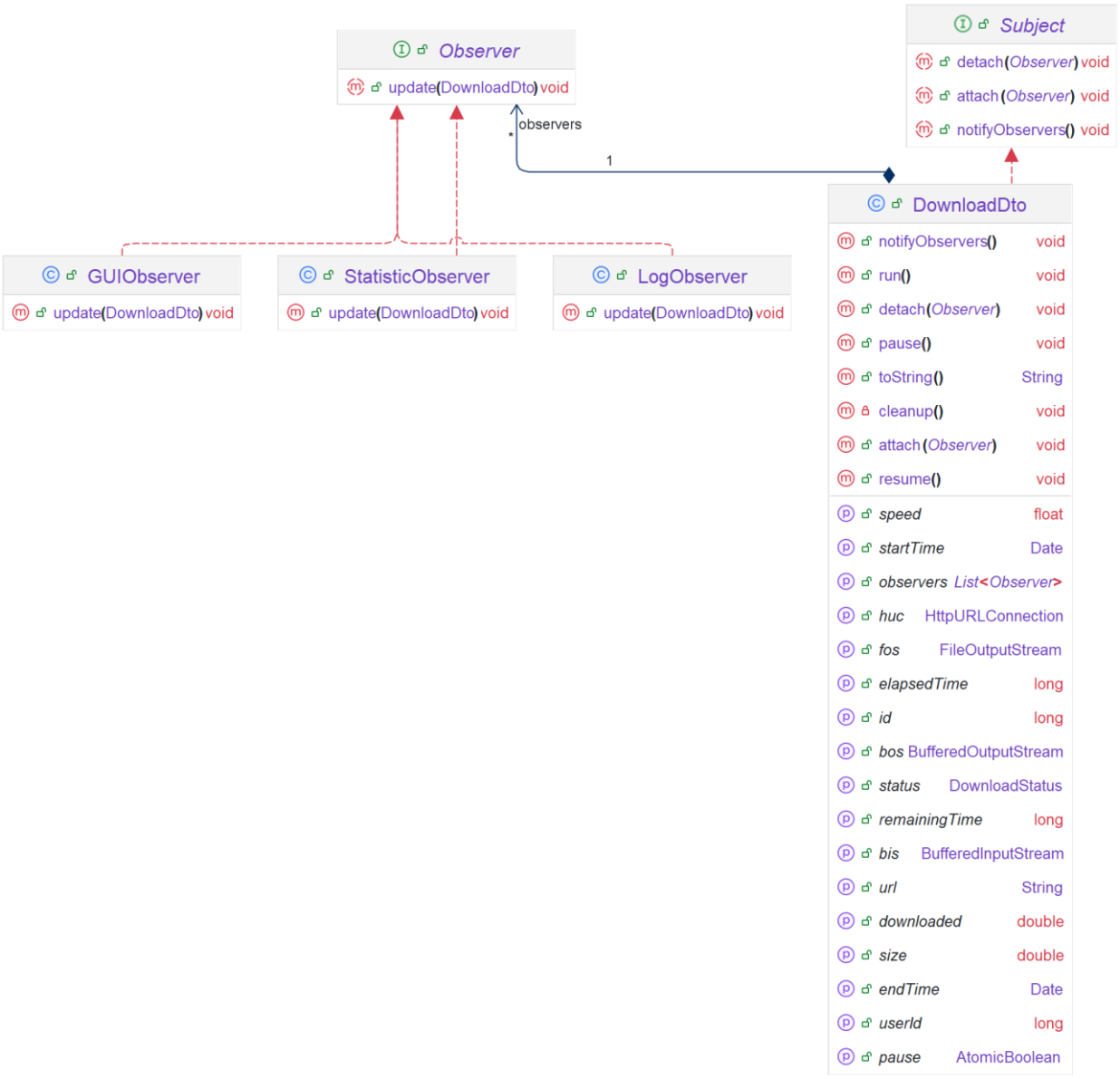

Рисунок №1 – Діаграма класів , згенерована IDE, реалізації шаблону Observer

На наданій діаграмі реалізація патерну складається з таких компонентів:

Основні елементи:

1. Subject (Суб’єкт):
   - Інтерфейс або абстрактний клас, який визначає методи для роботи з спостерігачами:
     - attach(Observer observer): додає спостерігача до списку.
     - detach(Observer observer): видаляє спостерігача зі списку.
     - notifyObservers(): сповіщає всіх зареєстрованих спостерігачів про зміну стану.

У цьому випадку DownloadDto виконує роль суб'єкта, забезпечуючи можливість відстежувати стан завантаження.

2. Observer (Спостерігач):
    - Інтерфейс або абстрактний клас, який має метод update(DownloadDto dto), що викликається суб’єктом при оновленні його стану.
3. Конкретні Спостерігачі:
    - GUIObserver: оновлює графічний інтерфейс відповідно до змін стану завантаження.
    - StatisticObserver: зберігає або обробляє статистику про завантаження.
    - LogObserver: веде журнал подій, що відбуваються під час завантаження.
4. DownloadDto:
    - Реалізує інтерфейс Subject.
    - Має список спостерігачів (observers), яких сповіщає про зміни стану через notifyObservers().
    - Інкапсулює дані завантаження, такі як швидкість, статус, URL, розмір, і методи для керування завантаженням (run(), pause(), resume() тощо).

Процес роботи:

1. Спостерігачі (наприклад, GUIObserver, StatisticObserver, LogObserver) реєструються у суб'єкта (DownloadDto) через метод attach().
2. Після зміни стану суб’єкт викликає notifyObservers(), що запускає метод update() у всіх зареєстрованих спостерігачів.
3. Кожен спостерігач реагує на зміну стану відповідно до своєї ролі.

Переваги:

- Слабка зв'язаність: суб'єкт не знає деталей реалізації спостерігачів.
- Гнучкість: легко додавати нові спостерігачі без зміни коду суб’єкта.
- Розширюваність: можна динамічно додавати або видаляти спостерігачів.

Недоліки:

- Може бути важко відстежувати порядок сповіщення спостерігачів у складних системах.
- Якщо є багато спостерігачів, це може вплинути на продуктивність.

Код реалізації паттерну можна переглянути у GitHub репозиторії у папці DownloadManager або у Додатку А.

Робота паттерну.

Для демонстрації роботи паттерну будемо симулювати завантаження файлу і виводи інформацію про логування, імітацію GUI та статистику в консоль. (Код для тестування є у GitHub репозиторії або у Додатку А)
Результат виконання коду:

```
Starting download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Download started
LOG -> Download https://sabnzbd.org/tests/internetspeed/50MB.bin status: DOWNLOADING
GUI -> Downloaded: NaN%, Speed: 0,00 KB/s, Remaining: 0 sec
STATISTICS -> DOWNLOADING download: https://sabnzbd.org/tests/internetspeed/50MB.bin
```

Рисунок №2 – Початок завантаження

```
LOG -> Download https://sabnzbd.org/tests/internetspeed/50MB.bin status: DOWNLOADING
GUI -> Downloaded: 11,82%, Speed: 10386,39 KB/s, Remaining: 4 sec
STATISTICS -> DOWNLOADING download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Download paused
LOG -> Download https://sabnzbd.org/tests/internetspeed/50MB.bin status: PAUSED
GUI -> Downloaded: 11,82%, Speed: 10386,39 KB/s, Remaining: 4 sec
STATISTICS -> PAUSED download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Resuming download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Download resumed
LOG -> Download https://sabnzbd.org/tests/internetspeed/50MB.bin status: DOWNLOADING
GUI -> Downloaded: 11,82%, Speed: 10386,39 KB/s, Remaining: 4 sec
STATISTICS -> DOWNLOADING download: https://sabnzbd.org/tests/internetspeed/50MB.bin
LOG -> Download https://sabnzbd.org/tests/internetspeed/50MB.bin status: DOWNLOADING
GUI -> Downloaded: 11,82%, Speed: Infinity KB/s, Remaining: 0 sec
```

Рисунок №3 – Зупинка та продовження завантаження

```
STATISTICS -> DOWNLOADING download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Cancelling download: https://sabnzbd.org/tests/internetspeed/50MB.bin
Download deleted!
LOG -> Download https://sabnzbd.org/tests/internetspeed/50MB.bin status: CANCELLED
GUI -> Downloaded: 17,96%, Speed: 26721,56 KB/s, Remaining: 1 sec
STATISTICS -> CANCELLED download: https://sabnzbd.org/tests/internetspeed/50MB.bin
```

Рисунок №4 – Видалення завантаження

Висновки.

В ході виконання лабораторної роботи було досліджено основні принципи SOLID та їх важливість у розробці програмного забезпечення. Вивчено теоретичні основи шаблонів проектування "Abstract Factory", "Factory Method", "Memento", "Observer" та "Decorator", їх призначення та випадки застосування. Реалізовано патерн Observer на прикладі системи Download Manager, Практично продемонстровано роботу патерну через реалізацію трьох типів спостерігачів.

Додаток А.
DownloadDto.java (Основний код для роботи патерну)

```
private final List<Observer> observers = new ArrayList<>();


@Override
public void run() {
    try {
        setStatus(DownloadStatus.DOWNLOADING);
        setStartTime(new Date());
        notifyObservers();

        URL urlObj = new URL(getUrl());
        String downloadDirectory = ConfigLoader.getDownloadDirectory();
        String fileName = new File(urlObj.getPath()).getName();
        String filePath = downloadDirectory + File.separator + fileName;

        File directory = new File(downloadDirectory);
        if (!directory.exists()) {
            directory.mkdirs();
        }

        // If download is starting fresh
        if (downloaded == 0) {
            huc = (HttpURLConnection) urlObj.openConnection();
            setSize((double) huc.getContentLengthLong());
            bis = new BufferedInputStream(huc.getInputStream());
            fos = new FileOutputStream(filePath);
            bos = new BufferedOutputStream(fos);
        } else {
            // Resume download
            huc = (HttpURLConnection) urlObj.openConnection();
            huc.setRequestProperty("Range", "bytes=" + (long)downloaded + "-");
            bis = new BufferedInputStream(huc.getInputStream());
            fos = new FileOutputStream(filePath, true);  // append mode
            bos = new BufferedOutputStream(fos);
        }

        byte[] buffer = new byte[1024];
        int read;
        long startTimeMillis = System.currentTimeMillis();

        while ((read = bis.read(buffer, 0, buffer.length)) >= 0) {
            if (status == DownloadStatus.CANCELLED) {
                cleanup();
                notifyObservers();
                return;
            }

            if (pause.get()) {
                cleanup();
                setStatus(DownloadStatus.PAUSED);
                notifyObservers();
                return;
            }

            bos.write(buffer, 0, read);
            setDownloaded(getDownloaded() + read);

            long elapsedMillis = System.currentTimeMillis() - startTimeMillis;
            setSpeed((float) (getDownloaded() / (elapsedMillis / 1000.0)));
            setRemainingTime((long) ((getSize() - getDownloaded()) / getSpeed()));
            notifyObservers();
        }

        cleanup();
        setEndTime(new Date());
        setStatus(DownloadStatus.COMPLETED);
        notifyObservers();
```

```
    } catch (Exception e) {
        setStatus(DownloadStatus.ERROR);
        notifyObservers();
        e.printStackTrace();
    }
}


@Override
public void attach(Observer observer) {
    observers.add(observer);
}

@Override
public void detach(Observer observer) {
    observers.remove(observer);
}

@Override
public void notifyObservers() {
    for(Observer observer : observers) {
        observer.update(this);
    }
}
```

Subject.java

```
package com.project.downloadmanager.util.observer;

public interface Subject {
    void attach(Observer observer);
    void detach(Observer observer);
    void notifyObservers();
}
```

Observer.java

```
package com.project.downloadmanager.util.observer;


import com.project.downloadmanager.model.DownloadDto;

public interface Observer {

    void update(DownloadDto dto);

}
```

GUIObserver.java

```
package com.project.downloadmanager.util.observer.impl;

import com.project.downloadmanager.model.DownloadDto;
import com.project.downloadmanager.util.observer.Observer;

public class GUIObserver implements Observer {


    @Override
    public void update(DownloadDto dto) {
        System.out.printf("GUI -> Downloaded: %.2f%%, Speed: %.2f KB/s, Remaining: %d
sec\n",
                          (dto.getDownloaded() / dto.getSize()) * 100,
                          dto.getSpeed() / 1024,
                          dto.getRemainingTime());
    }
}
```

LogObserver.java

```
package com.project.downloadmanager.util.observer.impl;


import com.project.downloadmanager.model.DownloadDto;
import com.project.downloadmanager.util.observer.Observer;

public class LogObserver implements Observer {

    @Override
    public void update(DownloadDto dto) {
        System.out.println("LOG -> Download " + dto.getUrl() + " status: " +
dto.getStatus());
    }
}
```

StatisticObserver.java

```
package com.project.downloadmanager.util.observer.impl;

import com.project.downloadmanager.model.DownloadDto;
import com.project.downloadmanager.model.enums.DownloadStatus;
import com.project.downloadmanager.util.observer.Observer;

public class StatisticObserver implements Observer {

    @Override
    public void update(DownloadDto dto) {
        for (DownloadStatus status : DownloadStatus.values()) {
            if (status == dto.getStatus()) {
                System.out.println("STATISTICS -> " + status + " download: " +
dto.getUrl());
            }
        }
    }
}
```

DownloadManager.java (Основний код для виконання)

```
public void downloadStart(String url) {
    if (downloads.containsKey(url)) {
        System.out.println("Download already in progress or completed for: " + url);
        return;
    }

    System.out.println("Starting download: " + url);
    DownloadDto download = new DownloadDto(url);

    download.attach(new LogObserver());
    download.attach(new GUIObserver());
    download.attach(new StatisticObserver());

    downloads.put(url, download);
    executorService.submit(download);
}
```