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

Технології розробки програмного забезпечення

“ Патерни проектування.”

“Download manager”

Виконав студент групи ІА–33:
Ященко К.А.

Київ 2025

Тема: Патерни проєктування

Мета: Вивчити структуру шаблонів “Abstract Factory”, “Factory Method”, “Memento”, “Observer”, “Decorator” та навчитися застосовувати їх в реалізації програмної системи.

Зміст

Завдання

• Ознайомитись з короткими теоретичними відомостями.

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

• Реалізувати один з розглянутих шаблонів за обраною темою.

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

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Теоретичні відомості

Abstract Factory
Патерн Abstract Factory забезпечує створення сімейств пов’язаних об’єктів без вказування їхніх конкретних класів. Він визначає інтерфейс для створення різних типів продуктів, дозволяючи змінювати цілі набори об’єктів, не змінюючи

код клієнта. Зазвичай використовується тоді, коли система повинна бути незалежною від способу створення та представлення об’єктів, а також коли

потрібно гарантувати узгодженість між продуктами, що належать до однієї родини.

Factory Method
Factory Method визначає інтерфейс для створення об’єктів, але дозволяє підкласам вирішувати, який саме клас створювати. Таким чином, він делегує створення об’єктів підкласам, уникаючи прямої залежності від конкретних реалізацій. Цей патерн полегшує розширення системи новими типами об’єктів без зміни вже існуючого коду, що підвищує гнучкість і розширюваність програми.

Memento
Патерн Memento використовується для збереження та відновлення попереднього стану об’єкта без порушення інкапсуляції. Він зберігає знімок внутрішнього стану об’єкта у спеціальному об’єкті Memento, який потім може бути використаний для відновлення цього стану. Це часто застосовується у програмах із функціональністю “скасування” (Undo) або відновлення попередніх версій даних.

Observer
Observer визначає залежність “один-до-багатьох” між об’єктами, при якій зміна стану одного об’єкта автоматично повідомляє всі залежні об’єкти. Об’єкт-спостерігач (Observer) підписується на події суб’єкта (Subject) і отримує сповіщення про зміни. Цей патерн часто використовується для реалізації систем оповіщення, реактивних інтерфейсів і подій у GUI.

Decorator
Патерн Decorator дозволяє динамічно додавати нову поведінку або

функціональність об’єктам без зміни їхнього коду. Він обгортає об’єкт у спеціальний клас-декоратор, який реалізує той самий інтерфейс і делегує виклики до оригінального об’єкта, додаючи нову логіку. Це забезпечує гнучку альтернативу наслідуванню при розширенні функціональності.

Тема проєкту
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).

Паттерн: Observer
Посилання на GitHub: https://github.com/fl1ckyexe/trpz

Хід роботи

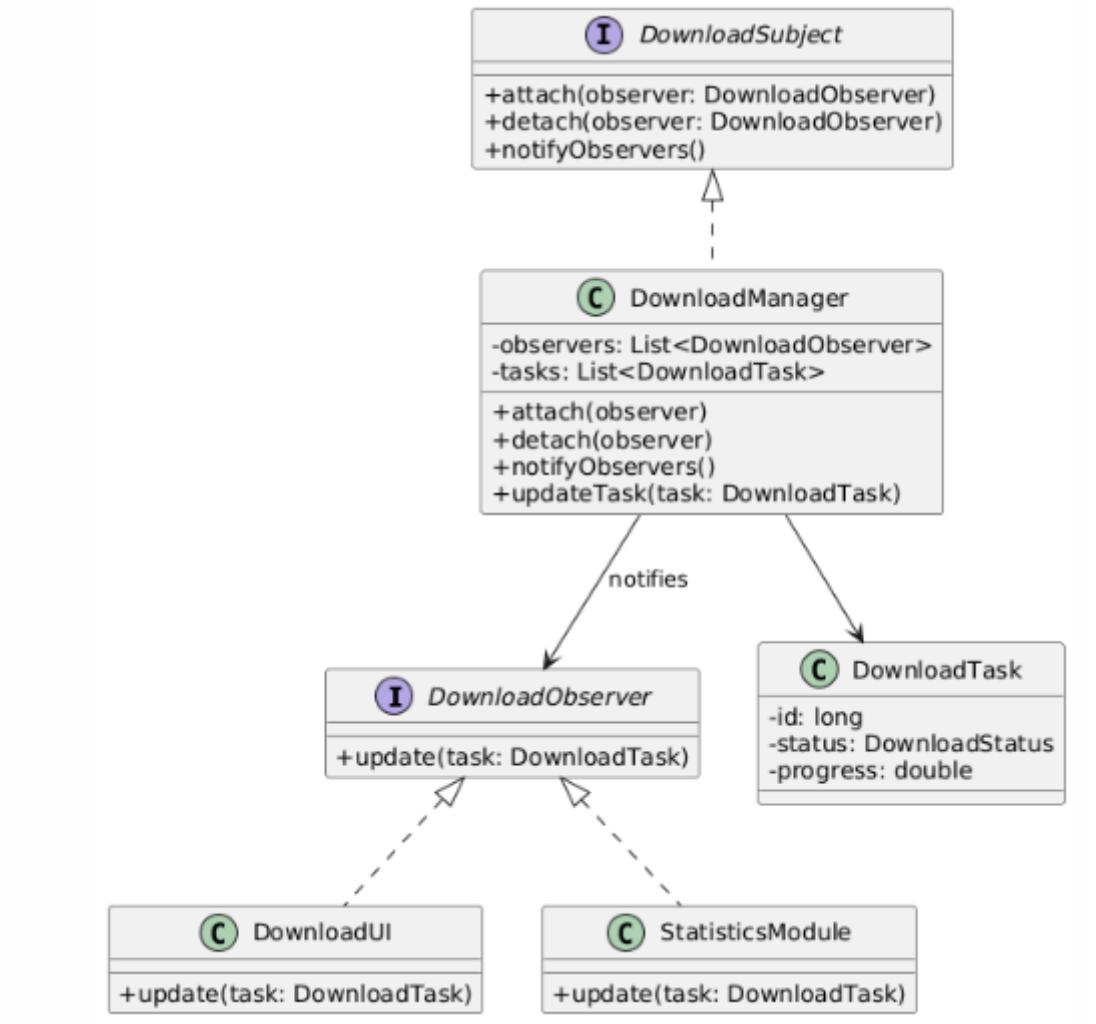


Рис. 1 – Діаграма класів для Observer

Діаграма класів відображає використання шаблону проєктування Observer у десктопному менеджері завантажень. Основна ідея полягає в тому, що об’єкти користувацького інтерфейсу та модуля статистики автоматично отримують повідомлення про зміни стану завантаження без жорсткого зв’язування з логікою керування завантаженнями.

Основні елементи діаграми

Центральним елементом є клас DownloadManager, який виступає у ролі суб’єкта (Subject). Він зберігає список активних завдань завантаження та список підписників-спостерігачів.

Інтерфейс DownloadSubject визначає базові операції для керування спостерігачами: підписку, відписку та сповіщення.

Інтерфейс DownloadObserver визначає метод update, який викликається при зміні стану завдання завантаження.

Клас DownloadTask представляє модель завантаження та містить інформацію про статус і прогрес.

Класи DownloadUI та StatisticsModule реалізують інтерфейс DownloadObserver і реагують на зміни стану завантаження відповідно, оновлюючи інтерфейс користувача та статистичні дані.

Зв’язки та взаємодія між елементами

Клас DownloadManager реалізує інтерфейс DownloadSubject та зберігає колекцію об’єктів типу DownloadObserver.

Об’єкти DownloadUI та StatisticsModule підписуються на DownloadManager і реєструються як спостерігачі.

При зміні стану об’єкта DownloadTask (наприклад, оновлення прогресу, пауза або завершення завантаження) DownloadManager викликає метод notifyObservers.

Кожен спостерігач отримує актуальний стан завдання через метод update та виконує власну логіку обробки.

Переваги використання шаблону Observer у проєкті

Використання шаблону Observer дозволяє досягти слабкого зв’язування між логікою завантаження та компонентами інтерфейсу й статистики.

Система стає легко розширюваною, оскільки нові спостерігачі можуть бути додані без змін у коді DownloadManager.

Observer забезпечує автоматичне та синхронізоване оновлення стану програми, що є критично важливим для менеджера завантажень з реальним відображенням прогресу.

Архітектура стає більш зрозумілою та відповідає принципам SOLID, зокрема принципу відкритості/закритості.

Частина коду програми з Observer

Observer pattern implementation for Download Manager

```
// DownloadObserver.java
public interface DownloadObserver {
    void update(DownloadTask task);
}

// DownloadSubject.java
public interface DownloadSubject {
    void attach(DownloadObserver observer);
```

```
    void detach(DownloadObserver observer);
    void notifyObservers();
}

// DownloadStatus.java
public enum DownloadStatus {
    CREATED,
    RUNNING,
    PAUSED,
    COMPLETED,
    FAILED
}

// DownloadTask.java
public class DownloadTask {
    private long id;
    private DownloadStatus status;
    private double progress;

    public DownloadTask(long id) {
        this.id = id;
        this.status = DownloadStatus.CREATED;
        this.progress = 0.0;
    }

    public long getId() {
        return id;
    }

    public DownloadStatus getStatus() {
        return status;
    }

    public double getProgress() {
        return progress;
    }

    public void setStatus(DownloadStatus status) {
        this.status = status;
    }

    public void setProgress(double progress) {
        this.progress = progress;
    }
}

// DownloadManager.java
import java.util.ArrayList;
import java.util.List;
```

```
public class DownloadManager implements DownloadSubject {

    private List<DownloadObserver> observers = new ArrayList<>();
    private List<DownloadTask> tasks = new ArrayList<>();

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

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

    @Override
    public void notifyObservers() {
        for (DownloadObserver observer : observers) {
            for (DownloadTask task : tasks) {
                observer.update(task);
            }
        }
    }

    public void addTask(DownloadTask task) {
        tasks.add(task);
        notifyObservers();
    }

    public void updateTask(long taskId, DownloadStatus status, double progress) {
        for (DownloadTask task : tasks) {
            if (task.getId() == taskId) {
                task.setStatus(status);
                task.setProgress(progress);
                notifyObservers();
                break;
            }
        }
    }
}

// DownloadUI.java
public class DownloadUI implements DownloadObserver {

    @Override
    public void update(DownloadTask task) {
        System.out.println(
            "[UI] Task #" + task.getId() +
```

```
            " | Status: " + task.getStatus() +
            " | Progress: " + task.getProgress() + "%"
        );
    }
}

// StatisticsObserver.java
public class StatisticsObserver implements DownloadObserver {

    @Override
    public void update(DownloadTask task) {
        if (task.getStatus() == DownloadStatus.COMPLETED) {
            System.out.println(
                "[Statistics] Task #" + task.getId() + " completed"
            );
        }
    }
}


public class Main {
    public static void main(String[] args) {

        DownloadManager manager = new DownloadManager();

        DownloadObserver uiObserver = new DownloadUI();
        DownloadObserver statsObserver = new StatisticsObserver();

        manager.attach(uiObserver);
        manager.attach(statsObserver);

        DownloadTask task = new DownloadTask(1);
        manager.addTask(task);

        manager.updateTask(1, DownloadStatus.RUNNING, 25.0);
        manager.updateTask(1, DownloadStatus.RUNNING, 70.0);
        manager.updateTask(1, DownloadStatus.COMPLETED, 100.0);
    }
}
```

Опис програмного коду

Поданий програмний код реалізує шаблон проєктування Observer у межах десктопного менеджера завантажень файлів. Шаблон використовується для автоматичного сповіщення різних компонентів системи про зміни стану завантаження.

Клас DownloadManager виступає у ролі суб’єкта спостереження та реалізує інтерфейс DownloadSubject. Він зберігає список завдань завантаження та колекцію

підписаних спостерігачів. При будь-якій зміні стану завантаження менеджер викликає метод notifyObservers, який інформує всі зацікавлені компоненти.

Інтерфейс DownloadObserver визначає єдиний метод update, що дозволяє реалізувати різну поведінку реакції на зміну стану завантаження.

Клас DownloadUI реалізує роль спостерігача та відповідає за оновлення користувацького інтерфейсу. Клас StatisticsObserver використовується для збору статистики та фіксації завершених завантажень.

Завдяки використанню шаблону Observer система забезпечує слабке зв’язування між логікою завантаження та компонентами відображення і статистики, що підвищує розширюваність та підтримуваність програмного забезпечення.

Висновок

У ході виконання лабораторної роботи було розглянуто та реалізовано шаблон проєктування Observer у межах десктопного менеджера завантажень файлів. Даний шаблон дозволив організувати механізм автоматичного сповіщення компонентів системи про зміни стану завантажень без жорсткого зв’язування між ними.

У процесі реалізації було визначено суб’єкт спостереження у вигляді менеджера завантажень, а також декілька спостерігачів, що відповідають за оновлення користувацького інтерфейсу та збір статистичних даних. Це дало змогу продемонструвати принцип роботи шаблону та його практичну доцільність у системах з динамічними змінами стану.

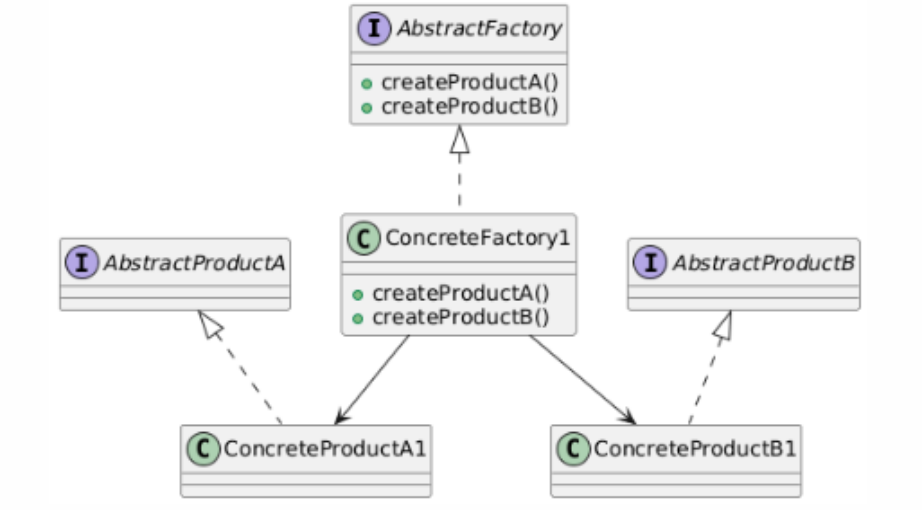
Застосування шаблону Observer підвищило гнучкість архітектури, спростило розширення функціональності та покращило підтримуваність програмного коду. Отримані результати підтверджують ефективність використання даного шаблону для побудови подійно-орієнтованих програмних систем.

Відповіді на контрольні питання

Питання та відповіді до лабораторної роботи

1. Яке призначення шаблону «Абстрактна фабрика»?
Шаблон проєктування «Абстрактна фабрика» призначений для створення сімейств взаємопов’язаних або взаємозалежних об’єктів без прив’язки до їх конкретних класів. Він дозволяє інкапсулювати логіку створення об’єктів та забезпечує узгодженість між створюваними продуктами.

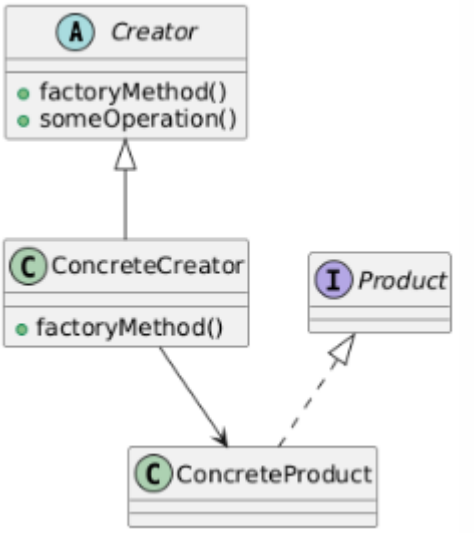
2. Нарисуйте структуру шаблону «Абстрактна фабрика».



3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?
У шаблоні беруть участь інтерфейс AbstractFactory, конкретні фабрики ConcreteFactory, абстрактні продукти AbstractProduct та конкретні реалізації продуктів. Клієнт використовує лише абстрактні інтерфейси, а конкретна фабрика відповідає за створення сумісних продуктів.

4. Яке призначення шаблону «Фабричний метод»?
Шаблон «Фабричний метод» призначений для визначення інтерфейсу створення об’єкта, дозволяючи підкласам вирішувати, який саме клас створювати. Він переносить відповідальність за створення об’єктів у підкласи.

5. Нарисуйте структуру шаблону «Фабричний метод».



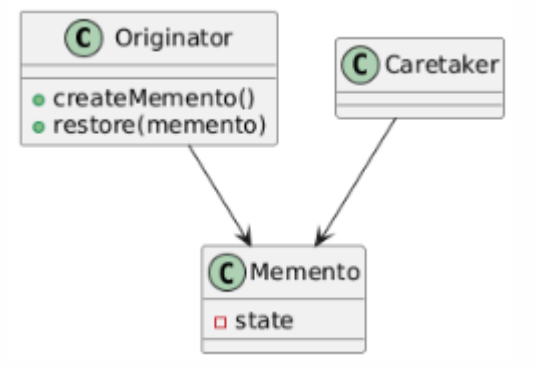
6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?
Шаблон складається з класу Creator, який оголошує фабричний метод, конкретного класу ConcreteCreator, що реалізує створення продукту, інтерфейсу Product та конкретного продукту ConcreteProduct.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?
Фабричний метод створює один продукт і використовує наслідування для вибору реалізації, тоді як Абстрактна фабрика створює цілі сімейства пов’язаних продуктів і використовує композицію.

8. Яке призначення шаблону «Знімок»?
Шаблон «Знімок» (Memento) призначений для збереження та відновлення

попереднього стану об’єкта без порушення принципу інкапсуляції.

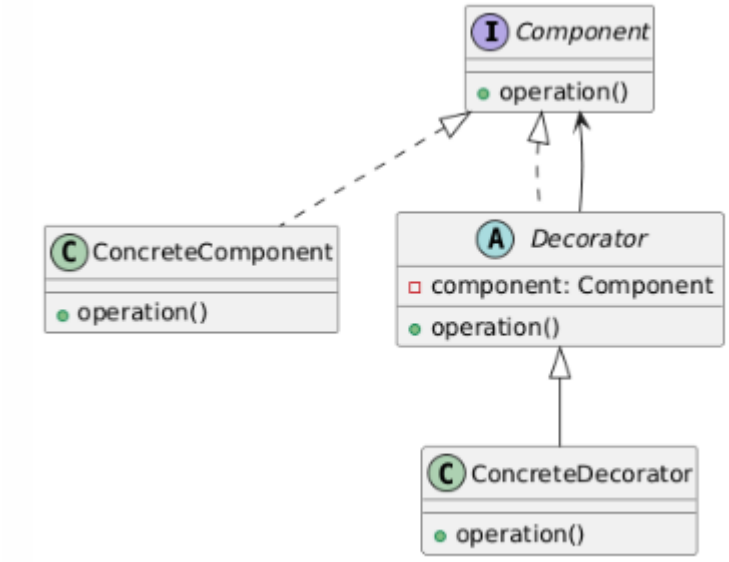
9. Нарисуйте структуру шаблону «Знімок».



10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?
Шаблон включає класи Originator, Memento та Caretaker. Originator створює знімок свого стану, Caretaker зберігає його, а відновлення стану відбувається через Memento без прямого доступу до внутрішніх даних.

11. Яке призначення шаблону «Декоратор»?
Шаблон «Декоратор» дозволяє динамічно додавати об’єктам нову поведінку без зміни їхнього класу шляхом обгортання об’єкта в інший об’єкт-декоратор.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?
У шаблоні беруть участь інтерфейс Component, базовий компонент ConcreteComponent, абстрактний клас Decorator та конкретні декоратори. Декоратор містить посилання на Component і делегує виклик, додаючи власну поведінку.

14. Які є обмеження використання шаблону «Декоратор»?
Основними обмеженнями є ускладнення структури програми через велику кількість дрібних класів, складність налагодження та можливе зниження читабельності коду при використанні багатьох вкладених декораторів.