МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТУ “ЛЬВІВСЬКА ПОЛІТЕХНІКА“

Кафедра систем штучного інтелекту

Лабораторна робота №3

з дисципліни

«Системний аналіз»

Виконав:

Студент групи КН-214

Добрій Назарій

Викладач:

Зімоха.І.О.

Львів – 2021р.

Застосування :

Якщо вам потрібно додавати об’єктам нові обов’язки «на льоту», непомітно для коду, який їх використовує. Об’єкти вкладаються в обгортки, які мають додаткові поведінки. Обгортки і самі об’єкти мають однаковий інтерфейс, тому клієнтам не важливо, з чим працювати — зі звичайним об’єктом чи з загорнутим. Якщо не можна розширити обов’язки об’єкта за допомогою спадкування.

У багатьох мовах програмування є ключове слово final, яке може заблокувати спадкування класу. Розширити такі класи можна тільки за допомогою Декоратора.

Кроки реалізації

1. Переконайтеся, що у вашому завданні присутні основний компонент і декілька опціональних доповнень-надбудов над ним.

2. Створіть інтерфейс компонента, який описував би загальні методи як для основного компонента, так і для його доповнень.

3. Створіть клас конкретного компонента й помістіть в нього основну бізнеслогіку.

4. Створіть базовий клас декораторів. Він повинен мати поле для зберігання посилань на вкладений об’єкт-компонент. Усі методи базового декоратора повинні делегувати роботу вкладеному об’єкту.

5. Конкретний компонент, як і базовий декоратор, повинні дотримуватися одного і того самого інтерфейсу компонента.

6. Створіть класи конкретних декораторів, успадковуючи їх від базового декоратора. Конкретний декоратор повинен виконувати свою додаткову функціональність, а потім (або перед цим) викликати цю ж операцію загорнутого об’єкта.

7. Клієнт бере на себе відповідальність за конфігурацію і порядок загортання об’єктів.

Виконання

Декоратор — це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки».

Функція та сервіс :

№ вар	Предмета область
5	Видання книг

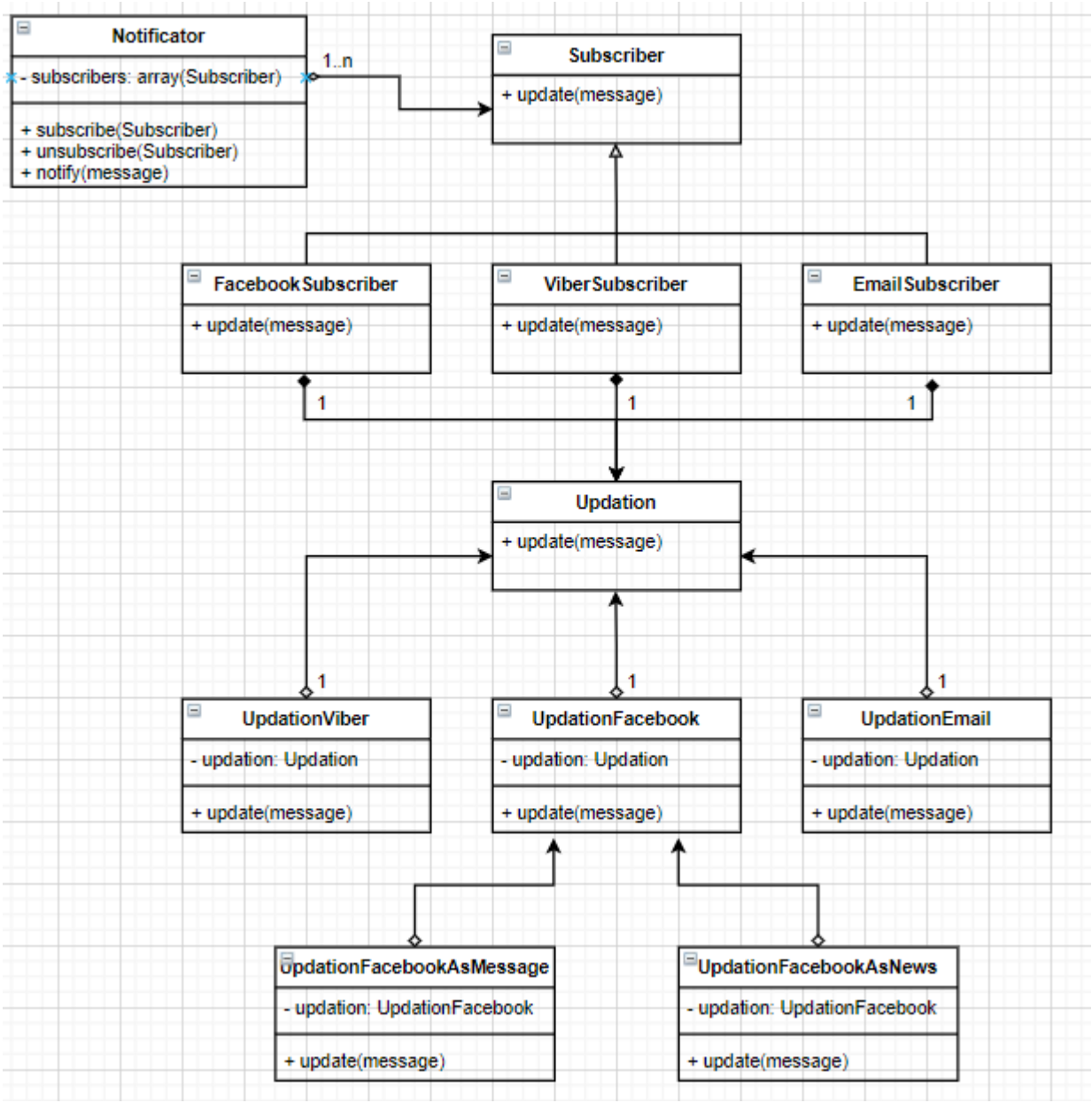

Рис. 1. UML-діаграма патерну “Декоратор”

Код програми на мові Python:

```
from abc import ABC, abstractmethod


class Updation():
    def __init__(self, message, name):
        self.message = message
        self.name = name

    def update(self):
        return 'Користувач {} отримав повідомлення "{}" '.format(self.name,
self.message)


class UpdationFacebook():
    def __init__(self, updation):
        self.updation = updation
```

```
    def update(self):
        return self.updation.update() + " через Facebook"


class UpdationFacebookAsNews():
    def __init__(self, updation):
        self.updation = updation

    def update(self):
        return self.updation.update() + ", як новину"


class UpdationFacebookAsMessage():

    def __init__(self, updation):
        self.updation = updation

    def update(self):
        return self.updation.update() + ", як повідомлення"


class UpdationViber():
    def __init__(self, updation):
        self.updation = updation

    def update(self):
        return self.updation.update() + " через Вайбер"


class UpdationEmail():
    def __init__(self, updation):
        self.updation = updation

    def update(self):
        return self.updation.update() + " через електронну пошту"


class Subscriber(ABC):
    @abstractmethod
    def update(self, message):
        pass


class FacebookSubscriber(Subscriber):
    def __init__(self, name, type):
        self.name = name
        self.type = type

    def update(self, message):
        updation = UpdationFacebook(Updation(message, self.name))

        if self.type == "повідомлення":
            updation = UpdationFacebookAsMessage(updation)
        elif self.type == 'новина':
            updation = UpdationFacebookAsNews(updation)

        print(updation.update())


class ViberSubscriber(Subscriber):
    def __init__(self, name):
        self.name = name
```

```
    def update(self, message):
        updation =  UpdationViber(Updation(message, self.name))

        print(updation.update())


class EmailSubscriber(Subscriber):
    def __init__(self, name):
        self.name = name

    def update(self, message):
        updation = UpdationEmail(Updation(message, self.name))

        print(updation.update())




class Notificator:
    def __init__(self):
        self.subscribers = set()

    def subscribe(self, sub: Subscriber):
        self.subscribers.add(sub)
        print('{} підписався'.format(sub.name))
    def unsubscribe(self, sub):
        self.subscribers.discard(sub)
        print('{} відписався'.format(sub.name))

    def notify(self, message):
        for subs in self.subscribers:
            subs.update(message)


notifiactor = Notificator()

sub1 = EmailSubscriber("nazardobriy@gmail.com")
sub2 = FacebookSubscriber("Nazar Dobriy", 'новина')
sub3 = ViberSubscriber("Nazar Dobriy")

notifiactor.subscribe(sub1)
notifiactor.subscribe(sub2)
notifiactor.subscribe(sub3)

notifiactor.notify("Виконання замовлення №5 продовжено на день!")

notifiactor.unsubscribe(sub3)

notifiactor.notify("Замовлення №5 виконано!")
```

Рис. 2.1. Код реалізації програми.

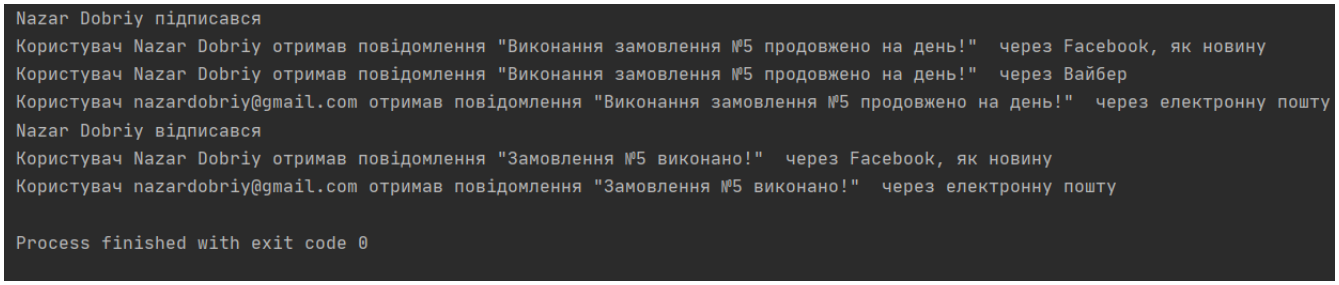

Рис. 2.2. Виконання програми.

Висновок: я навчився використовувати патерн декоратор. Реалізував його програмно на основі предметної області «Видання книг».