Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра інформаційних технологій та комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня *бакалавра*

Студента Хари Германа Леонідовича

академічної групи *126 – 18 – 1*

спеціальності *126 «Інформаційні системи та технології»*

на тему: *Розробка шаблонів інтерфейсів з використанням паттернів проектування*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		Рейтинговою	**Інституційною**	
кваліфікаційної роботи	*к.т.н., доц. Соколова Н.О.*			
розділів:				
Рецензент	*к.т.н., зав.каф.ПЗКС Удовик І.М.*			
Нормоконтролер	*д.т.н., проф. Коротенко Г.М.*			

Дніпро
2022

ЗАТВЕРДЖЕНО:
завідувач кафедри
Інформаційних технологій та комп'ютерної інженерії
(повна назва)
_____ *д.т.н., проф. Гнатушенко В.В.*
(підпис) (прізвище, ініціали)
«_____» _____2022року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня *бакалавра*

студенту Харі Г. Л. академічної групи *126-18-1*
спеціальності: *126 «Інформаційні системи та технології»*
на тему «
затверджену наказом ректора НТУ «Дніпровська політехніка»від *18.05.2022 р. №268-с*

Розділ	Зміст	Терміни виконання
1. Розділ 1	*1. Провести аналіз патернів проектування* *2. Розглянути приклади впливу патернів на систему* *3. Визначати найефективніші патерни для побудови шаблонів інтерфейсів*	02.05.2022 – 08.05.2022 09.05.2022 – 12.05.2022 10.05.2022 – 16.05.2022
2.Розділ 2	*1.Описати розробку шаблонів інтерфейсів* *2.Розробити додатки з використанням архітектури побудованої на патернах проектування*	17.05.2022 – 29.05.2022 30.05.2022 – 05.06.2022

Завдання видано _____ *доц. Соколова Н.О.*
(підпис) (прізвище, ініціали)

Дата видачі: *02.05.2022 р.*
Дата подання до екзаменаційної комісії:_____

Прийнято до виконання _____ _____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 65 с., 59 рис., 6 табл., 1 додаток, 16 джерел.

Об’єкт дослідження: розробка архітектури програмного забезпечення за допомогою патернів проектування.

Мета роботи полягає в аналізі підходів розробки архітектури програмного забезпечення з використанням патернів та доведені ефективності їх використання при розробці інтерфейсів програмних продуктів.

Використані методі дослідження: теоретичний метод - моделювання, аналіз, порівняння та абстрагування, а також емпіричний метод дослідження.

У вступі подано стан проблеми, сформульована мета і підхід до виконання даної кваліфікаційної роботи, а також обґрунтована її актуальність та сформовані задачі, які потрібно вирішити.

У першому розділі було визначено поняття патернів, їх класифікацію, приклади та представників а також переваги при розробці архітектури системи, були прийнято рішення розробити додатки з використанням патернів проектування для дослідження їх впливу на розробку шаблонів інтерфейсів.

У другому розділі повністю розроблені додатки за допомого за допомоги шаблонів інтерфейсів. Були розроблені шаблони інтерфейсів, створена реалізація інтерфейсів, проведена розробка архітектури додатків та проведено повне тестування всіх шаблонів.

ПАТЕРНИ ПРОЕКТВАННЯ, РОЗРОБКА АРХІТЕКТУРИ, ОПЕРЦІЙНА СИСТЕМА, МОВА ПРОГРАМУВАННЯ JAVA, КРОСПЛАТФОРМЕНІСТЬ, ІНТЕРФЕЙС.

ABSTRACT

Explanatory note: 64 pages, 59 figures, 6 tables, 1 appendix, 14 sources.

Object of research: development of software architecture with the help of design patterns.

The purpose of the work is performed in the analysis of approaches to the development of software architecture using patterns and achieving the efficiency of their use in the development of software product interfaces.

Use of research methods: theoretical method - modeling, analysis, comparison and abstraction, as well as empirical research method.

The introduction presents the state of the problem, formulates the purpose and approach to the implementation of this qualification work, as well as substantiates its relevance and the tasks to be solved.

The first section identified the concepts of patterns, their classification, examples and representatives, as well as the advantages of developing system architecture, decided to develop applications using design patterns to study their impact on the development of template interfaces.

The second section is fully developed applications to help interface templates. Interface templates were developed, interface implementation was created, application architecture was developed and all templates were fully tested.

PATTERNS DESIGN, ARCHITECTURE DEVELOPMENT, OPERATING SYSTEM, JAVA PROGRAMMING LANGUAGE, CROSSPLATFORM, INTERFACE.

ЗМІСТ

ВСТУП

Актуальність роботи. Кожного дня з’являються все більше нових розробників, а вони в свою чергу створюють все більше програмного забезпечення самих різних напрямків та різних сферах: бізнес, освіта, медицина, спорт або розваги, але після вибору напрямку та вигадки ідеї наступає найважливіший момент для створення додатків - розробка архітектури.

Розробкою архітектури додатків вже займаються багато років, а тому розробники часто зустрічали однакові проблеми при розробці, а також приходили до спільних рішень, але розумні люди зуміли систематизувати ці помилки та навчилися на них -так булі створенні патерни проектування.

Патерни проектування є найбільш поширеними рішеннями про розробці архітектури систем, тому що дозволяють вирішувати багато архітектурних проблем. Вони дозволяють вирішувати проблему розширення функціоналу при цьому зменшують код, який міг би утворитися без їх використання. Їх використання дозволяє навчатися вже на помилках інших, а не робити їх самому.

Використання патернів має позитивний вплив на кінцевий продукт, так як він стає більш мобільний, гнучкий а це приводить то того, що продукт існує довше, є ефективнішим та призводить до збільшення життєвого часу додатку.

Також патерни мають й економічний вплив на систему. Патерни зменшують час розробки архітектури так як вони вже вирішують проблеми та мають рішення, які можна використовувати знову, а не вигадувати свої рішення для якоїсь проблеми або задачі. Набір патернів доволі великий, тому вони зможуть вирішити більшу частину проблем, яка може виникнути.

Предметом дослідження є патерни та їх ефективність при розробці шаблонів інтерфейсів.

Об’єкт дослідження: розробка архітектури програмного забезпечення за допомогою патернів проектування.

Мета роботи полягає в аналізі підходів розробки архітектури програмного забезпечення з використанням патернів та доведені ефективності їх використання при розробці інтерфейсів програмних продуктів.

Для досягнення поставленої мети необхідно виконати наступний комплекс задач:

1. Визначення терміну патернів
2. Ознайомлення з класифікацію патернів та їх представниками
3. Розгляд реальних прикладів впливу патернів на систему
4. Визначення найефективніших патернів для побудови шаблонів інтерфейсів
5. Опис розробки шаблонів інтерфейсів
6. Розробка додатків з використанням архітектури побудованої на патернах проектування

Наукова новизна роботи полягає у доведенні ефективності використання патернів проектування в самих шаблонів інтерфейсів. Використання патернів проектування, як правило стосується їх використанню у бізнес логиці, проте бізнес логіка не взаємодії з користувачем напряму. В цій роботі буде показано то як патерни проектування впливають на створення ефективних, функціональних та зручних інтерфейсів користувача та показана в чому ефективність та необхідність такого підходу створення програмного забезпечення.

Наукова новизна полягає у розширенні, відтворенні, покрашення та доведенні функціональності патернів проектуванян при разробці програмного продукту.

Практичне значення. Результатом досліджень в дипломній роботі дозволяють зрозуміти як майбутнім або діючим розробникам програмних

додатків використати патерни для ефективної побудови свої продуктів з ефективним функціоналом інтерфейсів.

Інформація, яка описана в роботі може використовуватися в електронних ресурсах для розповсюдження загальних принципів проектування. Також це можуть використовувати учасники навчального процесу для більш поглибленого вивчання дисциплін, які демонструють патерни проектування. Студенти можуть брати ці наробітки для виконання лабораторних або практичних робіт. Також це можуть використовувати вже діючи розробники для того, щоб із шаблонів розробити повний функціональний продукт, який буде ефективно використовуватися в комерційній діяльності.

РОЗДІЛ 1.

АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧ

1.1. Аналіз предметної області

В перше концепція патернів була описана Крістофорем Александом в книзі «Мова патернів. Міста.Будівлі.Кострукції». Сама мова мала назву «Вічний шлях», а елементами були патерни[1].

Автор описував їх так: «Кожен шаблон описує проблему, яка виникає знову і знову в нашому середовищі, а потім описує суть рішення цієї проблеми, так що ви можете використовувати це рішення мільйон разів, ніколи не роблячи його однаково двічі»[1].

Хоча це твердження й було про архітектурні рішення, проте вони проектуються й на патерни проектування архітектурних систем.

Найчастіше при обговорені патернів звертаються до книги «Прийоми об'єктно-орієнтованого проектування. Патерни проектування» Еріх Гамма, Річард Хелм, Ральф Джонсон, Джон Вліссідес або як їх частіше називають «банда чотирьох»[2]

Вони визначають патерни проектування наступним чином:

«Тут під патернами проектування розуміється опис взаємодії об'єктів і класів, адаптованих для вирішення загального завдання проектування в конкретному контексті».[2]

Хоча ці автори були з різних напрямків своєї діяльності їх думки збігалися в тому, що патерни необхідні для вирішення проблем які виникають знову і знову.

В книзі чотирьох була також надана і структура патернів:

- Ім'я. Це можна порівняти з різними військовими планами та командами. При обговорені проектування можна використати лише ім’я патерну для того щоб колеги зрозумілі яка проблема стоїть на черзі так які кроки для її подолання.[2]

- Завдання. Опис проблему яку необхідно вирішити. Описується конкретна задача в загальному контексті. Це може бути схема, код або загальним словесний опис. Також є перелік умов за яких необхідно використовувати шаблон.[2]
- Рішення. Опис об’єктів, їх задачі, обов’язки та відношення між елементами. Треба зауважити що рішення не є єдиним розв’язком проблеми і використовуються для показу абстрактних шагів її рішення[2]
- Результати – це наслідки застосування патерну та різного роду компроміси. Патерни не є універсальним рішенням, тому перед його використанням треба зважати на його переваги та недоліки. Результати допоможуть показати ефективність патерну або його необхідність і після цього вже приймати рішення використовувати його чи ні.[2]

Тобто патерни це загальні схеми, яка вказують на проблему та спосіб її рішення, проте не треба сприймати як чітку інструкцію від якої не можна відхилятись. При проектуванні треба аналізувати вплив патерну його переваги та недоліки, а вже після цього вводити його в проект.

1.2. Класифікація патернів

Патерни проектування мають три основних класів:

- Породжувальні
- Структурні
- Поведінкові

Схематичне представлення породжувальних патернів та їх представників зображенні на рис. 1.1[3].

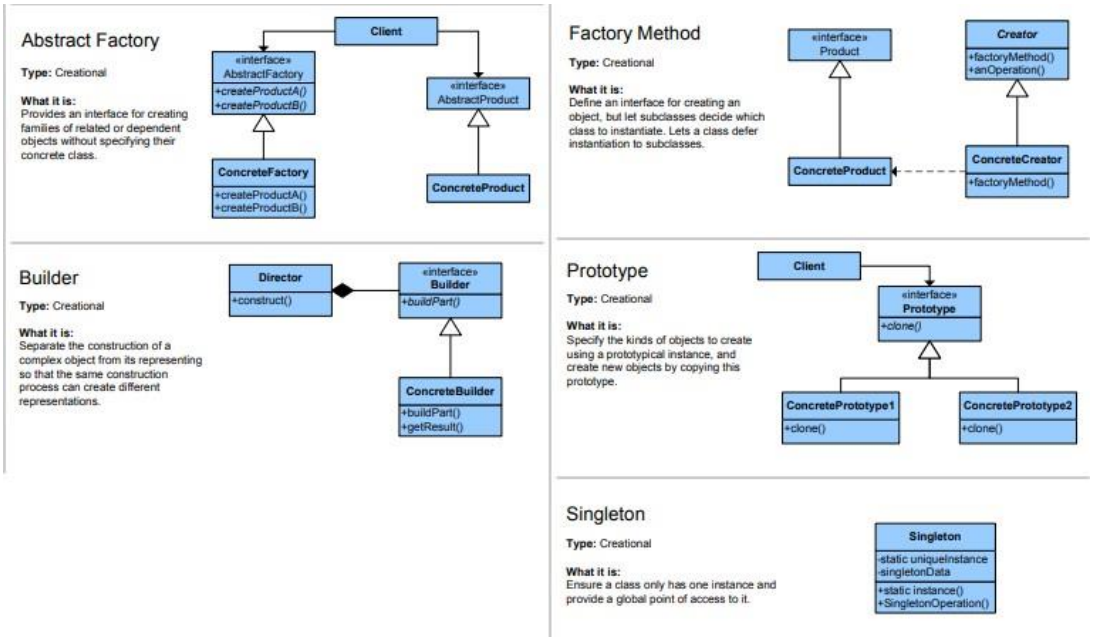


Рисунок 1.1 – Схематичне представлення породжувальних патернів

Схематичне представлення структурних патернів та їх представників зображено на рис.1.2[3].

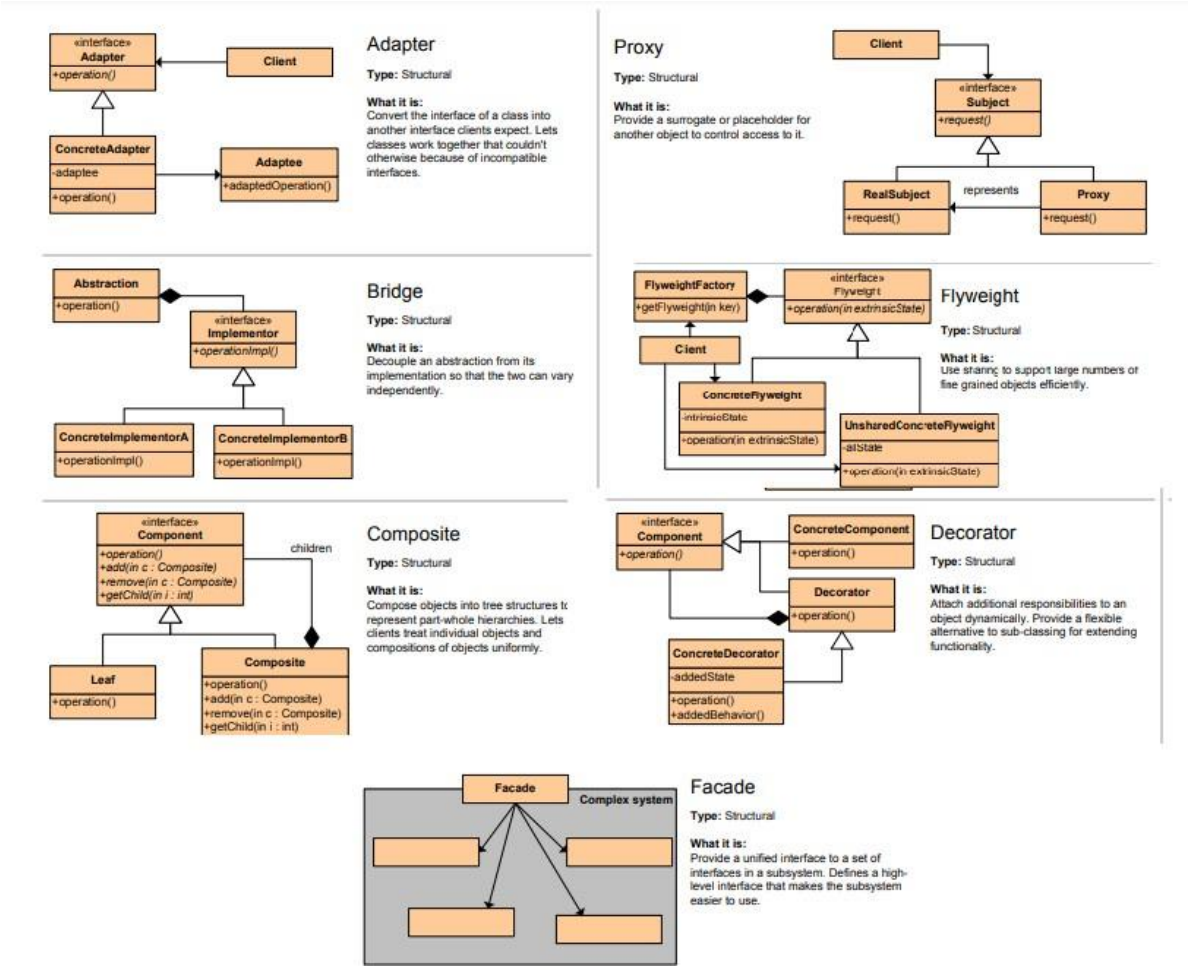


Рисунок 1.2 – Схематичне представлення структурних патернів

Схематичне представлення поведінкових патернів та їх представників зображено на рис.1.3[3].

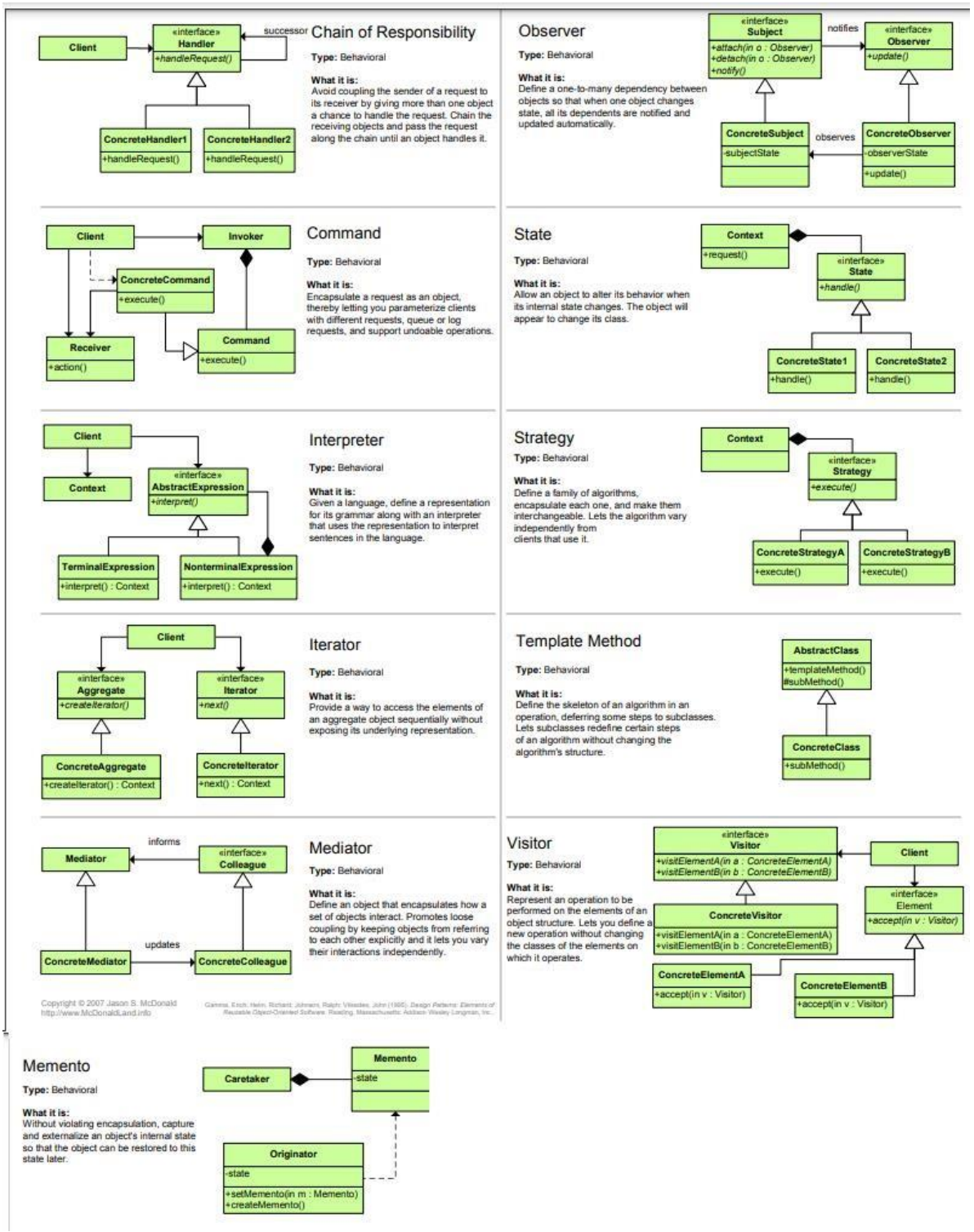


Рисунок 1.3 – Схематичне представлення поведінкових патернів

1.2.1. Породжувальні патерни

Породжувальні патерни абстрагують процес створення об’єктів.

Вони роблять систему незалежну від реалізацію, композиції та представлення об’єктів. Породжувальні патерни делегують процес створення об’єктів іншим класам. Основна задача полягає на створенні фундаментальних поведінок а на основі їх створювати вже велику кількість більш вузьких.

Ці патерни приховують як саме створюються об’єкт, а система лише відомі інтерфейси створенні за допомогою абстрактних класів.

Породжувальні патерни дають гнучкість створення об’єктів, як вони створюються, ким створюються і коли створюються. Це дає можливість створювати систему с готових об’єктів з різної реалізацією під різні системи.

Породжувальні патерни можна використовувати як по одному так і комбінувати між собою.

Породжувальні патерни стають необхідними, оскільки системи розвиваються, і їй необхідно більше залежати від композиції об’єкту, ніж успадкування класів. Коли це відбувається, акцент зміщується з жорсткого кодування фіксованого набору поведінок на визначення меншого набору основних поведінки, які можна поєднати з будь-якої кількості більш складних. Таким чином створюючи об'єкти з певною поведінкою вимагає більше, ніж просто створення екземпляра класу[4].

У цих шаблонах є дві повторювані теми. По-перше, вони інкасують знання про те, які конкретні класи використовує система. По-друге, вони приховують, як екземпляри цих класів створюються та об’єднуються. Вся система в цілому знає тільки про об'єкти та їх інтерфейси, визначені абстрактними класами[4].

Отже, породжувальні патерни дають вам велику гнучкість у тому, що створюється, хто створює, як і коли створюється. Вони дозволяють вам налаштувати система з об'єктами "продукту", які сильно відрізняються за структурою та функціональністю.

Конфігурація може бути статичною (тобто вказана під час компіляції) або динамічною (під час виконання). Іноді породжувальні патерни є

конкурентами. Наприклад, бувають випадки, коли тільки Прототип або тільки Абстрактна фабрика можна було б використовувати для ефективного подолання проблеми . В іншому раз вони є доповнюваними: Будівельник може використовувати один з інших шаблонів для реалізування. Прототип може використовувати Одинака в своїй в своїй реалізації[4].

Яскравим прикладом для демонстарвції цього класу патернів є патерн Одинак.

Патерн проектування Одинак — це породжувальний патерн, мета якого — створити лише один екземпляр класу та забезпечити лише одну глобальну точку доступу до цього об’єкта. Одним із часто використовуваних прикладів такого класу в Java є Calendar, де ви не можете створити екземпляр цього класу. Він також використовує власний метод getInstance() для отримання об’єкта, який буде використовуватися [5].

На рис. 1.4 зображено структуру класу Одинак[5].

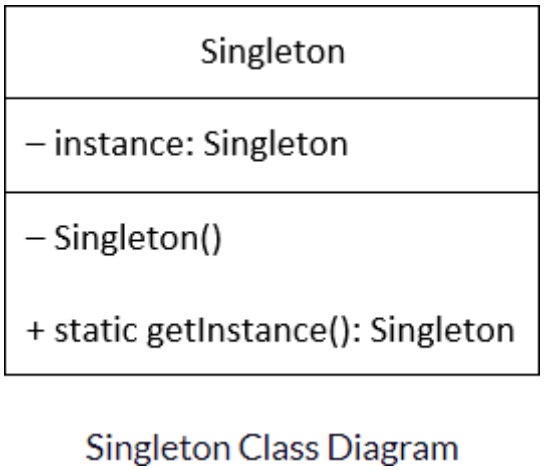


Рисунок 1.4 Схема класу Одинак

Приватна статична змінна, що містить єдиний екземпляр класу. Приватний конструктор, тому його екземпляр не можна створити в іншому місці. Загальнодоступний статичний метод для повернення окремого екземпляра класу[5].

Існує багато різних реалізацій патерну Одинак:

- Справедливий примірник
- Лінива інстанція
- Потокобезпечний екземпляр

Справедливий примірник - тип створення екземпляра відбувається під час завантаження класу, оскільки створення екземпляра змінної відбувається поза будь-яким методом. Це створює серйозний недолік, якщо цей клас взагалі не використовується клієнтською програмою. План на випадок надзвичайних ситуацій, якщо цей клас не використовується, є Лінива інстанція[5].

Реалізацію справедливого примірника можна переглянути на рис. 1.5[5].

```
public class EagerSingleton {
    // create an instance of the class.
    private static EagerSingleton instance = new EagerSingleton();

    // private constructor, so it cannot be instantiated outside this class.
    private EagerSingleton() {  }

    // get the only instance of the object created.
    public static EagerSingleton getInstance() {
        return instance;
    }
}
```

Рисунок 1.5 – Реалізація справедливого примірника

Лінива інстанція немає великої різниці від наведеної вище реалізації. Основні відмінності полягають у тому, що статична змінна спочатку оголошується нульовою і створюється в методі getInstance() лише тоді, коли - і тільки тоді - змінна екземпляра залишається нульовою на момент перевірки[5].

Нижче зображено код реалізації лінивої інстанції на рис. 1.6 [5].

```
public class LazySingleton {
    // initialize the instance as null.
    private static LazySingleton instance = null;

    // private constructor, so it cannot be instantiated outside this class.
    private LazySingleton() {  }

    // check if the instance is null, and if so, create the object.
    public static LazySingleton getInstance() {
        if (instance == null) {
            instance = new LazySingleton();
        }
        return instance;
    }
}
```

Рисунок 1.6 – Код реалізації лінивої інстанції

Це вирішує одну проблему, але інша все ще існує. Що робити, якщо два різні клієнти отримують доступ до класу Singleton одночасно, до мілісекунди? Що ж, вони перевірять, чи є екземпляр нульовим одночасно, і знайдуть його істинним, і таким чином створять два екземпляри класу для кожного запиту двох клієнтів. Щоб виправити це, має бути реалізовано потокобезпечний екземпляр [5].

Потокопезпечний екземпляр. У Java ключове слово synchronized використовується в методах або об’єктах для реалізації потокової безпеки, так що тільки один потік буде одночасно звертатися до певного ресурсу. Екземпляр класу поміщається в синхронізований блок, тому метод може бути доступний лише одному клієнту в даний момент часу [5].

Реалізацію цього методу можна переглянути на рис.1.7 [5].

```
public class ThreadSafeSingleton {
    // initialize the instance as null.
    private static ThreadSafeSingleton instance = null;

    // private constructor, so it cannot be instantiated outside this class.
    private ThreadSafeSingleton() {  }

    // check if the instance is null, within a synchronized block. If so, create the object
    public static ThreadSafeSingleton getInstance() {
        synchronized (ThreadSafeSingleton.class) {
            if (instance == null) {
                instance = new ThreadSafeSingleton();
            }
        }
        return instance;
    }
}
```

Рисунок 1.7 – Реалізація блоку синхронізації

Накладні витрати для синхронізованого методу високі і зменшують продуктивність усієї операції. Наприклад, якщо змінна екземпляра вже створена, то кожен раз, коли будь-який клієнт звертається до методу getInstance(), запускається synchronized метод і продуктивність падає. Це відбувається лише для того, щоб перевірити, чи значення змінних екземпляра є нульовим. Якщо він виявляє, що це так, він залишає метод [5].

Щоб зменшити ці накладні витрати, використовується подвійне блокування. Перевірка також використовується перед synchronized методом, і якщо значення є лише нульовим, то запускається synchronized метод [5].

Для вирішення цієї проблеми був розроблений код на рис.1.7 [5].

```
// double locking is used to reduce the overhead of the synchronized method
public static ThreadSafeSingleton getInstanceDoubleLocking() {
    if (instance == null) {
        synchronized (ThreadSafeSingleton.class) {
            if (instance == null) {
                instance = new ThreadSafeSingleton();
            }
        }
    }
    return instance;
}
```

Рисунок 1.7 – Реалізацію подвійного блокування.

1.2.2. Структурні патерни

Структурні патерни вирішують питання як з менших об’єктів створюються великі структури. Вони допомагають створити універсальні класи за допомогою наслідування з використанням інтерфейсів та реалізацій. Таким чином створюється один клас який має можливості всіх своїх батьків. Це допомагає створювати об’єднану структуру з класів які мають зовсім різні напрямки реалізації.

Як простий приклад розглянемо, як множинна спадковість змішує два або більше класів в один. Результатом є клас, який поєднує властивості своїх батьківських класів. Цей шаблон особливо корисний для того, щоб самостійно розроблені бібліотеки класів працювали разом. Інший приклад є формою класу шаблону Адаптер. Загалом, адаптер робить один інтерфейс адаптований іншому, забезпечуючи тим самим уніфікацію абстракція різних інтерфейсів. Адаптер класу виконує це за допомогою успадкування приватно від адаптованого класу. Потім адаптер виражає його інтерфейс з точки зору адаптованого[4].

Замість створення інтерфейсів або реалізацій, структурні шаблони об’єктів описують способи компонування об'єктів для реалізації нової функціональності. Гнучкість композиції об'єкта походить від можливості змінювати композицію під час виконання, що неможливо зі статичною композицією класів. Перевага цих патернів полягає у тому, що замість реалізації вони роблять акцент на створення нового функціоналу[4].

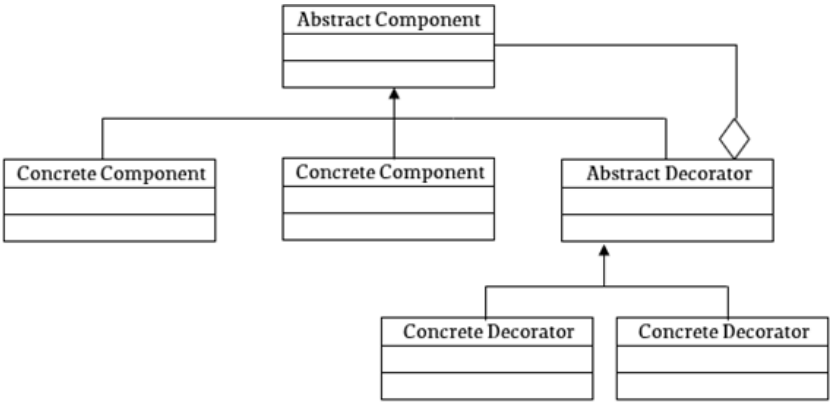
Для відображення структурних патернів, був обраний патерн – Декоратор.

Шаблон дизайну декоратора відноситься до структурної категорії, яка має справу з фактичною структурою класу, незалежно від того, чи є вона спадковістю, композицією або обома. Мета цього дизайну — змінити функціональність об’єктів під час виконання. Це один із багатьох інших шаблонів проектування, які використовують абстрактні класи та інтерфейси з композицією, щоб отримати бажаний результат [5].

Для доказу ефективності цього шаблону буде моделюватися ситуація виготовлення програмного продукту для кав'ярні.

Візьмемо 4 кавові суміші та 10 добавок. Якщо ми дотримуємося створення підкласів для кожної різної комбінації всіх доповнень для одного типу кави. Це: $(10–1)^2 = 9^2 = 81$ підклас. А якщо використовувати комбінування доповнень з чотирма видами кави, то виходить 324 класи. Патерн декоратор дозволяє зменшити це число до 16 класів [5].

Загальна структура патерну та конкретна схема класів зображені на рис. 1.8 та рис. 1.9[5].



Decorator Design Pattern Class diagram

Рисунок 1.8 –Загальна схема класів патерну Декоратор

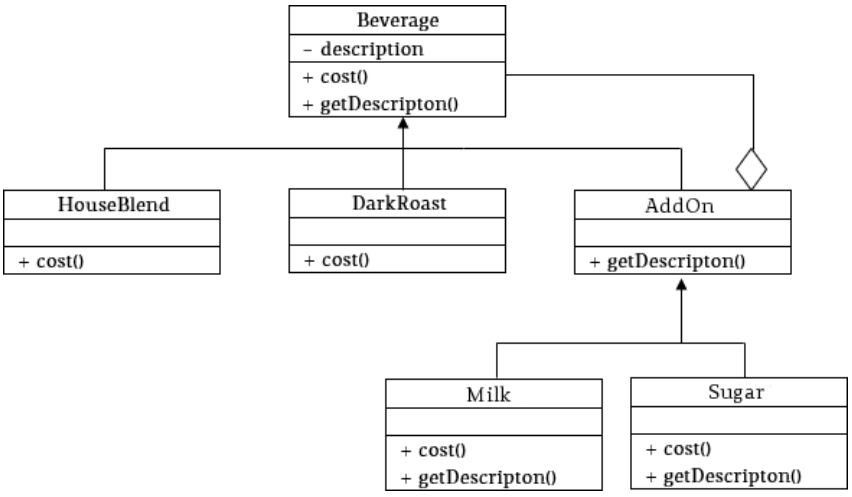


Рисунок 1.9 – Структура класів додатку для кофейні

Якщо ми розробимо наш сценарій відповідно до діаграми класів вище, ми отримаємо 4 класи для 4 кавових сумішей, 10 для кожного доповнення і 1 для абстрактного компонента і ще 1 для абстрактного декоратора [5].

Як ви можете бачити, так само як конкретні кавові суміші є підкласами абстрактного класу напою, абстрактний клас Add On також успадковує від нього свої методи. Додатки, які є його підкласами, у свою чергу успадковують будь-які нові методи для додавання функціональності базовому об’єкту, коли це необхідно[5].

Для реалізації необхідно наступне:

Спершу створимо абстрактний клас напоїв , який успадкуватимуть усі різні кавові суміші, код класу зображено на рис.1.10[5].

```
public abstract class Beverage {
    private String description;

    public Beverage(String description) {
        super();
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public abstract double cost();
}
```

Рисунок 1.10 – Реалізація класу напоїв

Потім необхідно додати обидва класи конкретних кавових сумішей. На рис. 1.11 зображено реалізацію цих класів [5].

```
public class HouseBlend extends Beverage {
    public HouseBlend() {
        super("House blend");
    }

    @Override
    public double cost() {
        return 250;
    }
}

public class DarkRoast extends Beverage {
    public DarkRoast() {
        super("Dark roast");
    }

    @Override
    public double cost() {
        return 300;
    }
}
```

Рисунок 1.11 – Реалізація класів для кавових сумішей

Абстрактний клас AddOn також успадковується від абстрактного класу Beverage. Це можна побачити на рис. 1.12 [5].

```
public abstract class AddOn extends Beverage {
    protected Beverage beverage;

    public AddOn(String description, Beverage bev) {
        super(description);
        this.beverage = bev;
    }

    public abstract String getDescription();
}
```

Рисунок 1.12 – Реалізація класу AddOn

А тепер конкретні реалізації цього абстрактного класу можна переглянути на рис. 1.13 [5].

```
public class Sugar extends AddOn {
    public Sugar(Beverage bev) {
        super("Sugar", bev);
    }

    @Override
    public String getDescription() {
        return beverage.getDescription() + " with Mocha";
    }

    @Override
    public double cost() {
        return beverage.cost() + 50;
    }
}

public class Milk extends AddOn {
    public Milk(Beverage bev) {
        super("Milk", bev);
    }

    @Override
    public String getDescription() {
        return beverage.getDescription() + " with Milk";
    }

    @Override  public double cost() {
        return beverage.cost() + 100;
    }
}
```

Рисунок 1.13 – Реалізація класу AddOn в класах нащадках

Як ви бачите вище, ми можемо передати будь-який підклас Beverage в будь-який підклас AddOn і отримати додаткову вартість, а також оновлений опис. І оскільки клас AddOn по суті має тип Beverage, ми можемо передати AddOn в інший AddOn. Таким чином, ми можемо додати будь-яку кількість добавок до конкретної суміші кави [5].

На рис. 1.14 зображено реалізацію клієнтського коду[5].

```
public class CoffeeShop {
    public static void main(String[] args) {
        HouseBlend houseblend = new HouseBlend();
        System.out.println(houseblend.getDescription() + ": " + houseblend.cost());

        Milk milkAddOn = new Milk(houseblend);
        System.out.println(milkAddOn.getDescription() + ": " + milkAddOn.cost());

        Sugar sugarAddOn = new Sugar(milkAddOn);
        System.out.println(sugarAddOn.getDescription() + ": " + sugarAddOn.cost());
    }
}
```

Рисунок 1.15 – Реалізація клієнтського коду

Також ми можемо переглянути результат виконання цієї програми на рис. 1.16 [5].

```
<terminated> CoffeeShop [Java Application] C:\Program File
House blend: Rs. 250.0
House blend with Milk: Rs. 350.0
House blend with Milk with Sugar: Rs. 400.0
```

Рисунок 1.16 – Результат виконання програми

Можемо переконатися, що з додаванням кожної добавки ціна збільшується, це і означає, що програма працює вірно.

1.2.3. Поведінкові патерни

Поведінкові патерни займаються розподіленням обов’язків між об’єктами. Вони характеризують складний потік керування командами, який важко відслідкувати під час виконання програми. Реалізація йде не на потік управління, а на взаємодію об’єктів.

Вони описують як за допомогою взаємодії рівноправних об’єктів виконати якусь задачу, яку вони не можуть виконати окремо. Проте це приводить до збільшення залежності класів. Саме поведінкові патерни допомагають розробити ефективну взаємодію між класами та не дозволяють створити залежність.

Поведінкові патерни пов'язані з алгоритмами та призначенням відповідальність між об'єктами. Ці шаблони характеризують складний потік

керування, який важко дотримуватися під час виконання. Вони переміщують ваш фокус від потоку контролю, щоб ви могли зосередитися лише на тому, як об’єкти взаємопов’язані. Шаблони поведінкових класів використовують успадкування для розподілу поведінки між класами[4].

Приклад поведінкового патерна є патерн команда

З’єднання – це спосіб взаємодії двох (або більше) класів, які взаємодіють один з одним. Ідеальний сценарій взаємодії цих класів полягає в тому, що вони не залежать один від одного. Це слабке з’єднання. Отже, кращим визначенням слабкого зв’язку було б класи, які взаємопов’язані, що найменше використовують один одного[5].

Потреба в цьому шаблоні виникла, коли запити потрібно було відправити, не знаючи свідомо, що ви просите або хто одержувач[5].

У цьому шаблоні клас, що викликає, відокремлений від класу, який фактично виконує дію. У класі виклику є тільки викликаний метод execute, який запускає необхідну команду, коли її запитує клієнт[5].

Більш детально цей патерн буде розглянутий в виборі проектних рішень для створення шаблонів інтерфейсів

1.3. Переваги та недоліки використання патернів

Багато розробників, особливо початківців, використовують патерни самі того не розуміючи , але краще знати де їх використовувати та для чого.

Використання патернів має ряд переваг:

– При розробці використовується вже готові рішення, які зможуть суттєво зменшити час розробки, а це в свою чергу зменшить її вартість.

– Перш за все, патерн це вирішення проблем, а отже вони допомагають покращити ПЗ. Більшість патернів допомагають вирішити проблему розширення функціоналу, який в свою чергу збільшує цикл життя програми

– Стандартизація. При використанні вже прийнятих патернів, розробники з легкістю можуть комунікувати між собою при обоготворенні, а також це створює можливість легкого переходу від одного проекту до

іншого, бо код використовую прийняті шаблони які допомагають швидше розібратися з новим кодом.

Патрни проектування набули своєї популярності з книги чотирьох, і звісно за цей час підвергались неодноразової критиці.

Недоліки використання патернів:

– Адаптивність. Особливо початківці використовують птерни без адаптації під свою конкретну проблему, а це в свою чергу збільшую заплутаність коду а не вирішує проблему

– Ускладнення коду. Часто для вирішення проблем створюються додаткові класси які збільшують програми, тобто треба чітку аналізувати ситуацію та використовувати шаблони, якщо це необхідно. Безрозумне використання патернів може призвести для ускладнення розуміння коду.

Виходячи з цього треба розуміти що патерни це в перш за все інструмент який допомагає вирішите конкретні проблеми, але це не догма від якої неможна відхилятися під час розробки програми.

1.4. Огляд патернів проектування для розробки шаблонів інтерфейсів

Існує велика кількість різних патрнів проектування. В цьому розділі будуть описані рішення для проектування шаблонів інтерфейсів. Будуть описані їх переваги, недоліки, проблеми які вони вирішують, способи їх реалізацію та необхідні дії для цього.

1.4.1. Фабричний метод

Фабричний метод — це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів[6].

Фабричний метод застосовується:

– Коли заздалегідь невідомо тип об’єктів які створюються. Фабричний метод відокремлює створення об’єкту від коду який його використовує.

– Коли необхідно зробити розширення деяким продуктом. Користувач може використати готову бібліотеку, витягнути в деякий клас продукту метод який він хоче переписати. А фабричний метод допоможе йому використати новий метод при створенні об’єкту.

– Якщо необхідно використовувати об’єкти декілька разів, а не створювати нові. Фабричний метод виконує функцію сховища и надає як нові об’єкти так і ті що булі вже створенні

Кроки реалізації:

– Приведіть усі створювані продукти до загального інтерфейсу[6].

– Створіть порожній фабричний метод у класі, який виробляє продукти. В якості типу, що повертається, вкажіть загальний інтерфейс продукту[6].

– Пройдіться по коду класу й знайдіть усі ділянки, що створюють продукти. По черзі замініть ці ділянки викликами фабричного методу, переносячи в нього код створення різних продуктів[6].

– Можливо, доведеться додати до фабричного методу декілька параметрів, що контролюють, який з продуктів потрібно створити[6].

– Імовірніше за все, фабричний метод виглядатиме гнітюче на цьому етапі. В ньому житиме великий умовний оператор, який вибирає клас створюваного продукту. Але не хвилюйтеся, ми ось-ось все це виправимо[6].

– Для кожного типу продуктів заведіть підклас і перевизначте в ньому фабричний метод. З суперкласу перемістіть туди код створення відповідного продукту[6].

– Якщо створюваних продуктів занадто багато для існуючих підкласів творця, ви можете подумати про введення параметрів до фабричного методу, аби повертати різні продукти в межах одного підкласу[6].

– Якщо після цих всіх переміщень фабричний метод став порожнім, можете зробити його абстрактним. Якщо ж у ньому щось

залишилося — не страшно, це буде його типовою реалізацією (за замовчуванням) [6].

Структура патерну зображена схемою на рис. 1.17

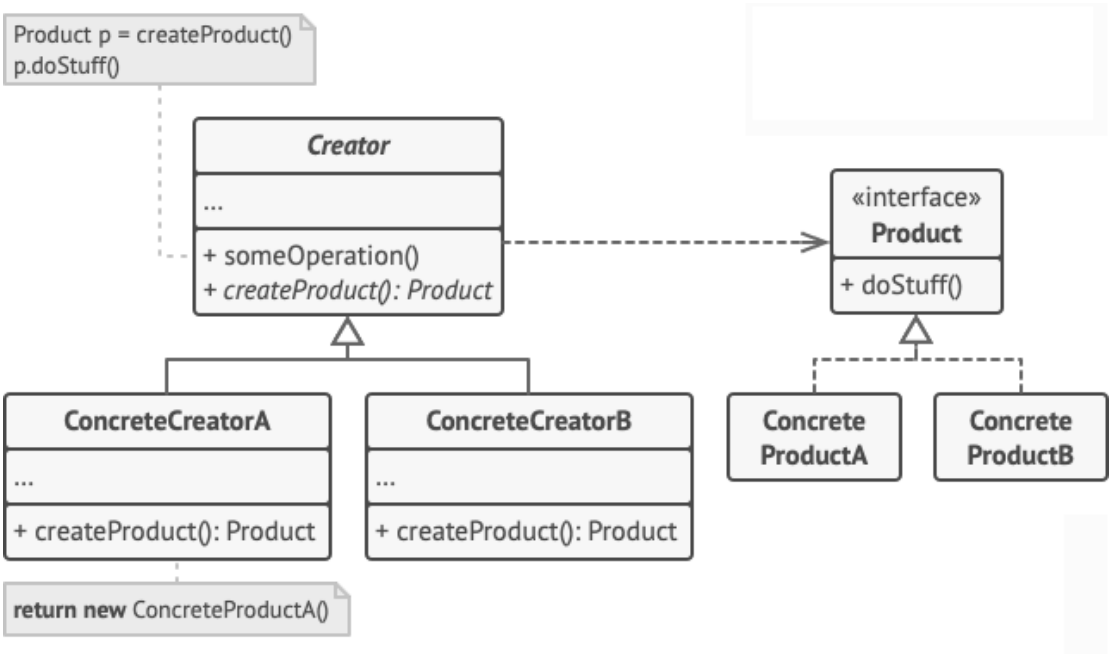


Рисунок 1.17 – Структура класів патерну Фабричний метод

Інтерфейс продуктів визначає спосіб створення продуктів- його нащадків.

Конкретні продукти вже реалізують метод спосіб створення себе. Вони відрізняються способом реалізацію, але для того щоб їх можна було використовувати однаково вони мають спільний інтерфейс.

Клас творець має в собі абстрактний фабричний метод, який повинен повертати нові продукти. Сам метод створюється абстрактним, щоб класи нащадки обов’язково его реалізували за своїми потребами.

А вже конкретні класи творці групують в собі ріні варіацію продуктів, потім в клієнтському коді вони визиваються в залежності від того який саме вид продуктів необхідний системі.

Переваги:

– Позбавляє клас від прив’язки до класів продуктів.

– Спрощує розширення коду новими продуктами.

– Реалізує принцип Open/Close.

Недоліки:

– Іноді призводить до створення подібних класів.

– Збільшує програму додатковими класами

1.4.2. Абстрактна фабрика

Абстрактна фабрика — це породжувальний патерн проектування, що дає змогу створювати сімейства пов’язаних об’єктів, не прив’язуючись до конкретних класів створюваних об’єктів[15].

Застосування:

– Система неповинна залежати від реалізації об’єктів.

– Об’єкти що входять в сімейство повинні використовуватися разом.

– Система повинна реалізуватися одним з сімейства.

– Необхідно представити бібліотеку об’єктів без їх реалізації.

Кроки реалізації:

– Створіть таблицю співвідношень типів продуктів до варіацій сімейств продуктів[15].

– Зведіть усі варіації продуктів до загальних інтерфейсів[15].

– Визначте інтерфейс абстрактної фабрики. Він повинен мати фабричні методи для створення кожного типу продуктів[15].

– Створіть класи конкретних фабрик, реалізувавши інтерфейс абстрактної фабрики. Цих класів має бути стільки ж, скільки й варіацій сімейств продуктів[15].

– Змініть код ініціалізації програми так, щоб вона створювала певну фабрику й передавала її до клієнтського коду[15].

– Замініть у клієнтському коді ділянки створення продуктів через конструктор на виклики відповідних методів фабрики[15].

Структура патерну зображена на рис. 1.18[15].

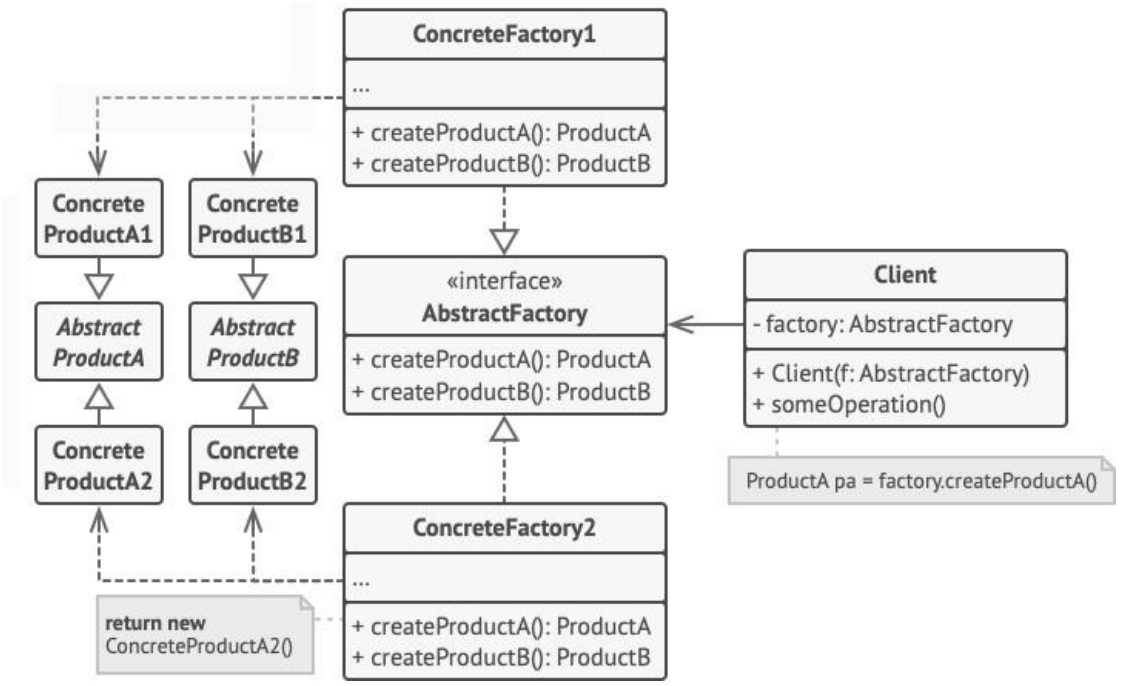


Рисунок 1.18 – Схема класів патерну Абстрактна фабрика

Абстрактні продукти оголошують спільний інтерфейс для конкретних продуктів , що мають різну реалізацію.

Конкретні продукти – набір класів, що належать до різних абстрактних класів класів, але мають однакові варіації.

Абстрактна фабркиа оголошує методи за допомогою яких будуть створюватися продукти.

Конкретні фабрики реалізують методи створення та групують собі різни набори продуктів одного сімейства. Вони повинні повертати абстрактний тип продукту для того, щоб клієнтський код міг працювати з будь-який варіаціями продуктів не прив’язуючись до їх реалізацій.

Переваги:

– Позбавляє клас від прив’язки до класів продуктів.

– Спрощує розширення коду новими продуктами.

– Реалізує принцип Open/Close[15].

– Виносить код створення продукту в окремий клас, що робить код більш зручним.

Недоліки:

– Програма ускладняється через додавання великої кількості класів.

– Вимагає наявності всіх типів продукту в кожній варіації[15].

1.4.3. Команда

Команда — це поведінковий патерн проектування, який перетворює запити на об’єкти, дозволяючи передавати їх як аргументи під час виклику методів, ставити запити в чергу, логувати їх, а також підтримувати скасування операцій[15].

Застосування:

– Є необхідність параметризувати об’єкти, які виконують дію

– Потрібно використовувати відкладний виклик команд або визивати їх за розкладом

– При розробці програми виникла потреба в створення механізму відміни.

Кроки реалізації:

– Створіть загальний інтерфейс команд і визначте в ньому метод запуску[15].

– Один за одним створіть класи конкретних команд. У кожному класі має бути поле для зберігання посилання на один або декілька об’єктів-одержувачів, яким команда перенаправлятиме основну роботу[15].

– Крім цього, команда повинна мати поля для зберігання параметрів, потрібних під час виклику методів одержувача. Значення всіх цих полів команда повинна отримувати через конструктор[15].

– І, нарешті, реалізуйте основний метод команди, викликаючи в ньому ті чи інші методи одержувача[15].

– Додайте до класів відправників поля для зберігання команд. Зазвичай об’єкти-відправники приймають готові об’єкти команд ззовні — через конструктор або через сетер поля команди[15].

– Змініть основний код відправників так, щоб вони делегували виконання дії команді[15].

Порядок ініціалізації об’єктів повинен виглядати так:

– Створюємо об’єкти одержувачів[15].

– Створюємо об’єкти команд, зв’язавши їх з одержувачами[15].

– Створюємо об’єкти відправників, зв’язавши їх з командами[15].

Структура патерну зображено на схемі рис. 1.19.

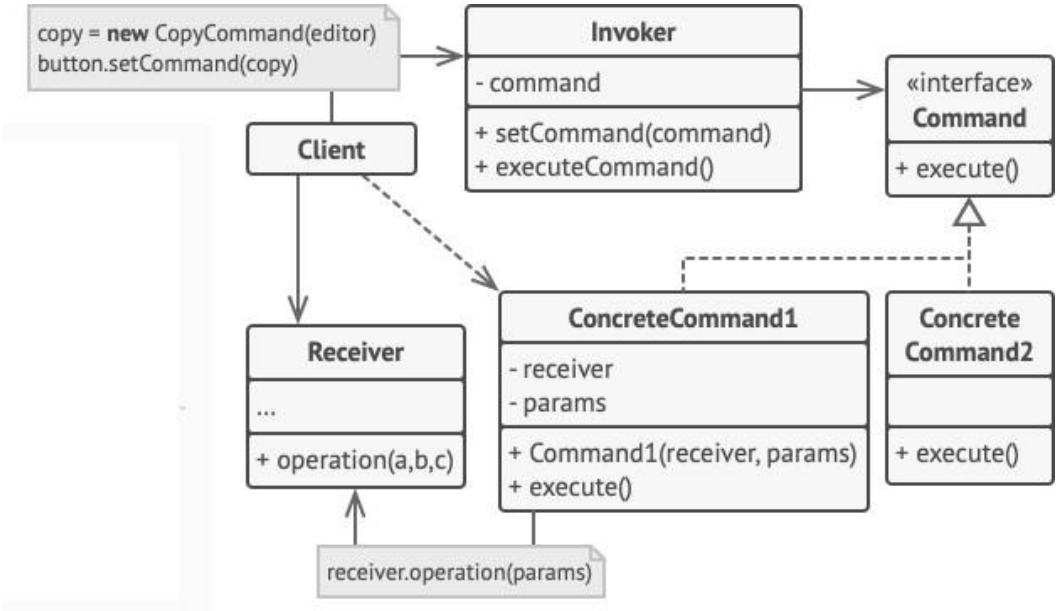


Рисунок 1.19 – Схема класів патерну абстрактна фабрика

Відправник має в своєму коді об’єкт команди та посилається на нього, коли є необхідність виконати якусь дію. У відправника нема потреби, щоб знати яку саме команду він використовує так як він отримує вже контрений об’єкт конкретної команди.

Клас команди описує метод запуску самих команд.

Конкретні команди вже реалізують метод запуску, але саму дію як правило відправляють на опрацювання бізнес логіки.

Саме опрацювання відбувається в класі отримувачі, він реалізує логіку команд та що вони повинні робити. Іноді цим класам нехтують та передають реалізацію на самі класи команд.

Клієнт вже працює з командами через створення їх об’єктів та внесення необхідних значень в поля цих об’єктів.

Переваги

– Прибирає пряму залежність між об’єктами, що викликають операції, та об’єктами, які їх безпосередньо виконують[15].

– Дозволяє реалізувати просте скасування і повтор операцій[15].

– Дозволяє реалізувати відкладений запуск операцій[15].

– Дозволяє збирати складні команди з простих[15].

– Реалізує принцип відкритості/закритості[15].

Недоліки

Вводить велику кількість допоміжних класів через що ускладняється програма[15].

1.5. Висновки до першого розділу

Дослідивши поняття патернів, їх класифікацію, приклади та представників а також переваги при розробці архітектури системи, були прийнято рішення розробити додатки з використанням патернів проектування для дослідження їх впливу на розробку шаблонів інтерфейсів.

РОЗДІЛ 2.

ПРОЕКТНІ РІШЕННЯ

2.1. Розробка шаблонів інтерфейсів з використанням патернів

Для показу переваг використання патернів було розроблено три шаблони:

– Кросплатформна програма з надзвичайною інформацією

– Рекламний додаток

– Текстовий редактор

2.2. Огляд технологічних рішень

2.2.1. Вибір мови програмування

При створенні шаблонів інтерфейсів були використання два напрямки створення десктопний та для «Кросплатформна програма з надзвичайною інформацією» була розроблена веб версію.

Для програмування десктопних додатків була обрана мова програмування JAVA з використанням бібліотеки SWING. При розробці веб версії була використана комбінація з HTML, CSS та JS. Далі будуть описані переваги цих технологій через, які вони були обрані для розробки шаблонів інтерфейсів.

Технології десктопних додатків:

Java - це мова програмування та обчислювальна платформа, вперше випущена Sun Microsystems у 1995 році. Вона розвинулась із скромних початків, щоб забезпечити живлення значної частини сучасного цифрового світу, забезпечуючи надійну платформу, на якій побудовано багато послуг і програм. Нові, інноваційні продукти та цифрові послуги, розроблені для майбутнього, також продовжують покладатися на Java[7].

Переваги Java:

У мові Java є багато переваг перед іншими мовами програмування, що дозволяє вирішувати за його допомогою практично будь-які завдання.

Нижче перераховані основні переваги Java:

– Мова Java простий для вивчення. При розробці Java було приділено велику увагу простоті мови, тому програми Java, в порівнянні з програмами іншими мовами, простіше писати, компілювати, налагоджувати і вивчати[8]. Це сприяє тому, що розроблені шаблони будуть зрозуміли для більшості розробників, а також допоможе додаткам розроблених в цьому розділі.

– Java – це об'єктно-орієнтована мова. Це дозволяє створювати модульні програми, вихідний код яких можна використовувати багаторазово. А так як більшість патернів складаються з взаємодії класів, то це просто необхідність при їх використанні[8].

– Мова Java не залежить від платформи. Однією з основних переваг мови Java є можливість перенесення програм із однієї системи до іншої. Оскільки програми Java не залежать від платформи як на рівні вихідного коду, так і на двійковому рівні, їх можна запускати в різних системах, що особливо важливо для програм, призначених для World Wide Web. Це дозволяє створювати кросплатформні додатки[8].

– Широкі можливості Java, простота застосування, незалежність від платформи та вбудовані функції захисту роблять цю мову програмування однією з найкращих для створення програм для Internet[8].

Swing в Java – це легкий інструментарій з графічним інтерфейсом, який має широкий спектр віджетів для створення оптимізованих віконних додатків. Це частина JFC (Java Foundation Classes). Він побудований на основі AWT API і повністю написаний на Java. Він не залежить від платформи від відмінності від AWT і має легкі компоненти[9].

Переваги Swing :

– Компоненти Swing не залежать від платформи[10].

– Компоненти Swing мають можливість змінювати свій дизайн[10].

– Компоненти Swing використовують парадигму Model-View-Controller (MVC) і, таким чином, можуть забезпечити набагато більш гнучкий інтерфейс користувача[10].

– Компоненти Swing мають легку вагу, тобто вони не вимагають багато ресурсів для свого створення та обробки[10].

– Swing забезпечує вбудовану подвійну буферизацію[10].

– Swing надає підтримку налагодження вигляду, коли ви створюєте власні компоненти[10].

Технологія для створення веб додатків.

Для створення веб версії додатку « Кросплатформна програма з надзвичайною інформацією» була обрана комбінація з технологій: HTML, CSS та JS.

HTML (Hypertext Markup Language) - це код, який використовується для структурування та відображення веб-страниці та її вмісту. Наприклад, вміст може бути структурований всередині множини параграфів, маркованих списків або з використанням зображень і таблиць даних[11].

Переваги HTML:

– HTML підтримується всіма браузерами. Практично всі браузери по всьому світу підтримуються HTML. Тому не потрібно турбуватися про веб-сайт, написаний в HTML для підтримки браузера, оскільки веб-сайт легко відображатиметься у всіх браузерах, якщо програма має на увазі оптимізувати веб-сайт для різних браузерів. HTML надає простий спосіб оптимізувати веб-сайт у HTML відповідно до браузерів веб-розробникам[12].

– HTML простий для редагування HTML дуже легко редагувати, оскільки для його редагування немає необхідності мати спеціальний інтерфейс або платформу. Він написаний простим блокнотом, а тому може бути просто відредагований у будь-якому текстовому редакторі, наприклад блокнот, блокнот ++ тощо[12].

– HTML може легко інтегруватися з іншими мовами HTML можна легко інтегрувати з декількома мовами і не створювати в ньому жодних проблем. Наприклад, у Javascript, Php, node.js, CSS та багатьох інших, ми пишемо код цих мов між HTML, і він змішується з ними дуже легко[12].

– HTML легкий HTML - це легка мова. Він має високе співвідношення сигнал / шум порівняно з іншими формами зв'язку. Також швидше завантажувати HTML-код, а це означає, що він також дуже стислий[12].

CSS (Cascading Style Sheets) – це код, який ви використовуєте для стилізації вашої веб-сторінки. Це не мова розмітки – це мова таблиці стилів. Це означає, що він дозволяє використовувати стилі вибірково до елементів у документах HTML[11].

Переваги CSS:

– Кожен веб-сайт має вміст і має заздалегідь задану структуру чи макет. Цей макет використовується для представлення вмісту. Це головна мета CSS; він відокремлює структуру документа від представлення документа. Якщо розробник хоче змінити будь-який колір дизайну, він просто повинен виправити єдиний рядок коду, який відображатиметься в будь-якій кількості веб-сторінок[14].

– Менше використання кодування підвищить ефективність сторінки та скоротить час завантаження. При загрузці браузером код сторінки переглядається двічі: По-перше, щоб зрозуміти основну структуру таблиці, а другий раз фактично відобразити вміст у таблиці. Ця подвійна робота може уповільнити завантаження веб-сайту. Тепер переваги CSS, оскільки ми використовували CSS в окремому файлі, вбудований у формат CSS код буде кешований після першого початкового запиту, і немає необхідності. щоб знову завантажити сторінку. Це одна з важливих поведінки CSS над HTML[14].

– Послідовний підхід - це те, що CSS пропонує своїм користувачам, вносячи невеликі зміни на ваш веб-сайт, такі ж зміни відображаються і в інших частинах веб-сайту. Чим більший вміст вашого веб-сайту та макетів, тим більше часу заощаджує CSS для змін. Він також перевіряє та забезпечує кожну сторінку послідовною, що також є основними перевагами CSS[14].

– Пробіл, вирівнювання та позиціонування: означає контроль над CSS та його вищою версією надзвичайний над візуальними місцями розташування, такими як поля, поплавці, відступ тексту. Це збільшує кількість зображень і інформативність сторінки[14].

2.2.2. Вибір методу тестування

Для перевірки коректності роботи додатків був обраний метод ручного тестування.

Ручне тестування — це процес тестування програмного забезпечення, в якому тестові випадки виконуються вручну без використання будь-яких автоматизованих інструментів. Усі тестові випадки виконуються тестером вручну відповідно до точки зору кінцевого користувача. Він гарантує, чи працює програма, як зазначено в документі з вимогами, чи ні. Тестові випадки плануються та впроваджуються, щоб завершити майже 100 відсотків програмного додатка. Звіти про тестові випадки також створюються вручну[13].

Ручне тестування є одним з найбільш фундаментальних процесів тестування, оскільки воно може виявити як видимі, так і приховані дефекти програмного забезпечення. Різниця між очікуваним результатом і результатом, заданою програмним забезпеченням, визначається як дефект. Розробник усунув дефекти та передав тестувальнику на повторне тестування[13].

Ручне тестування є обов’язковим для кожного нещодавно розробленого програмного забезпечення перед автоматизованим тестуванням. Це

тестування вимагає великих зусиль і часу, але воно дає впевненість у відсутності помилок. Ручне тестування вимагає знання методів ручного тестування, але не будь-яких автоматизованих інструментів тестування[13].

2.3. Розробка шаблону інтерфейсу для кросплатформної програми з надзвичайною інформацією

2.3.1. Формулювання задачі

Для формулювання задачі була використана діаграма варіантів використання програми на рис. 2.1.

Рисунок 2.1- Схема варіантів використання

2.3.2. Вибір принципів проектування

При проектуванні системи був використаний патерн «Фабричний метод», який дозволяє визначити загальний інтерфейс для створення об’єктів у класі батькові, а це в свою чергу дозволяє нащадкам змінювати реалізацію створюваних об’єктів.

Такий підхід дозволяє створювати систему, яка заздалегідь не знає який продукт їй необхідно створити.

Така ситуація може виникати коли систему необхідно розширити об’єктами, які різні за своєю реалізацією. Наприклад створення системи, яка дозволяє створювати програму для декількох операційних систем.

На цій основі була розроблено програму, яка дозволяє переглядати надзвичайну інформацію в операційний системі Windows 10 або використовувати веб версію.

2.3.3. Ієрархія та взаємодія класів

Взаємодія та структура системи зображена на рис. 2.2

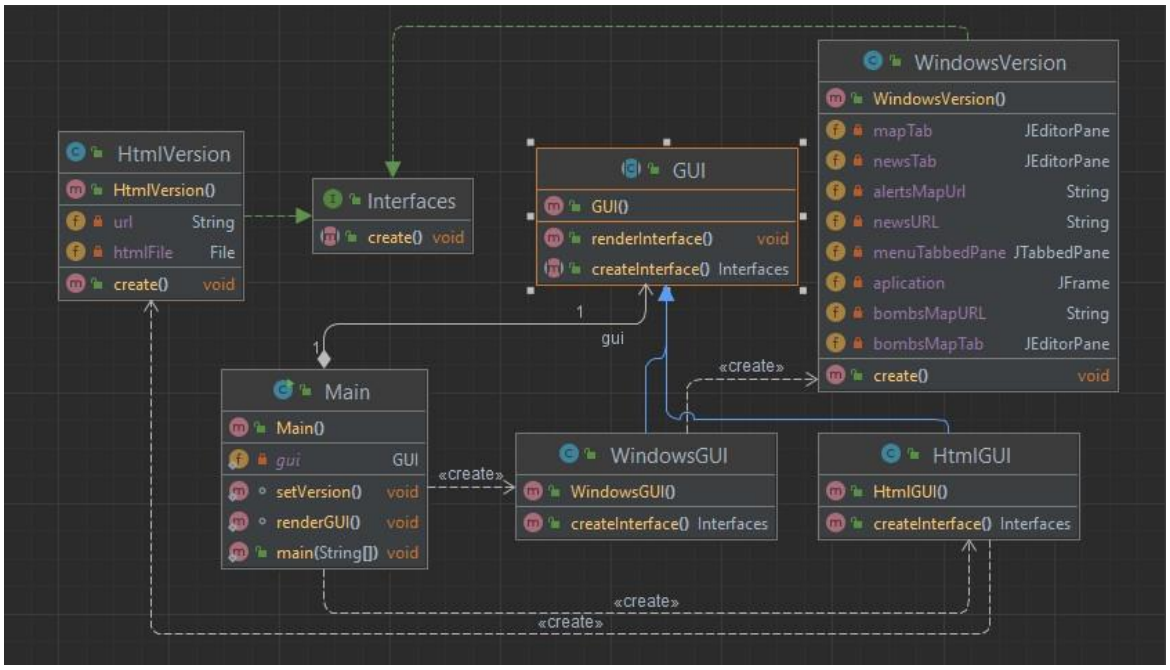


Рисунок 2.2- Схема ієрархії класів

Клас GUI має в своєму коду фабричний метод createInterface(), який є абстрактним. Цей метод необхідно реалізувати так, щоб тип який він повертає співпадал із загальним інтерфейсом продукту. В нашому випадку метод crateInterface() повертає тип загального інтерфейсу для конкретних продуктів. Його необхідно зробити абстрактним, для того щоб кожний конкретний вид фабричного методу реалізував його за своєю задачею.

Конкретні фабричні класи WindowsGUI та HtmlGUI створюють та групують реалізації продуктів, які описані в класах HtmlVersion та WindowsVersion,

Ці два класи реалізують різні продукти, тобто різні елементи інтерфейсу проте сам метод і створення в них спільний бо є реалізацією спільного інтерфейсу.

Клієнтський код перевіряє, яка система на машині користувача , а потім за допомогою умовного оператора if викликає екземпляр класу с конкретною реалізацією інтерфейсу та створює інтерфейс з реалізацію на Java або на Html.

2.3.4. Розробка інтерфейсу

Створені шаблони інтерфейсу показані на рис.2.3-рис.2.5.

Рисунок 2.3 – Шаблон першої вкладки

Перша вкладка повина показувати мапу тривог України.

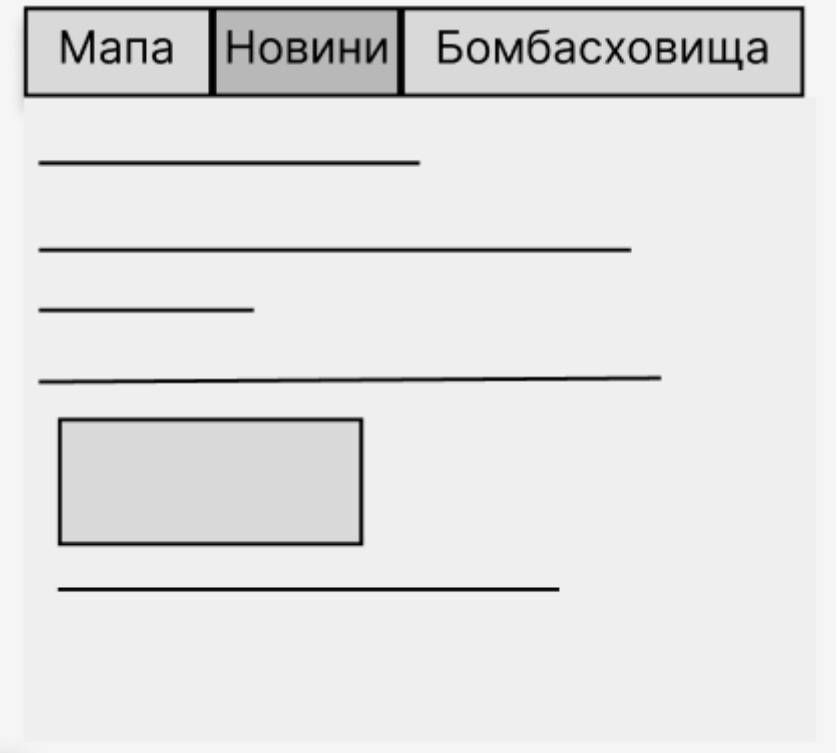


Рисунок 2.4 – Шаблон другої вкладки

Друга вкладка показує новини

Рисунок 2.5 – Шаблон третьої вкладки

На останній вкладці необхідно показати мапу бомбосховищ.

Інтерфейс на JAVA

Для розробки інтерфейсу використовувалась Java SWING. Ця бібліотека дає можливість створювати програми з графічним інтерфейсом. Приклади створених інтерфейсів будуть показані на рис 2.6- 2.8.

Використовувались наступні елементи:

– JTabbedPane - використовується для створення вкладок

– JEditorPane – використовується для показу контенту с з сайтів з необхідною інформацією

– JFrame – використовується як основа для всього додатку.

Інтерфейс на HTML буде показаний на рис. 2.9-2.11:

Для створення вкладок використовувались , які переключались за допомогою <button> з використання JS та CSS.

Інтерфейс на Java:

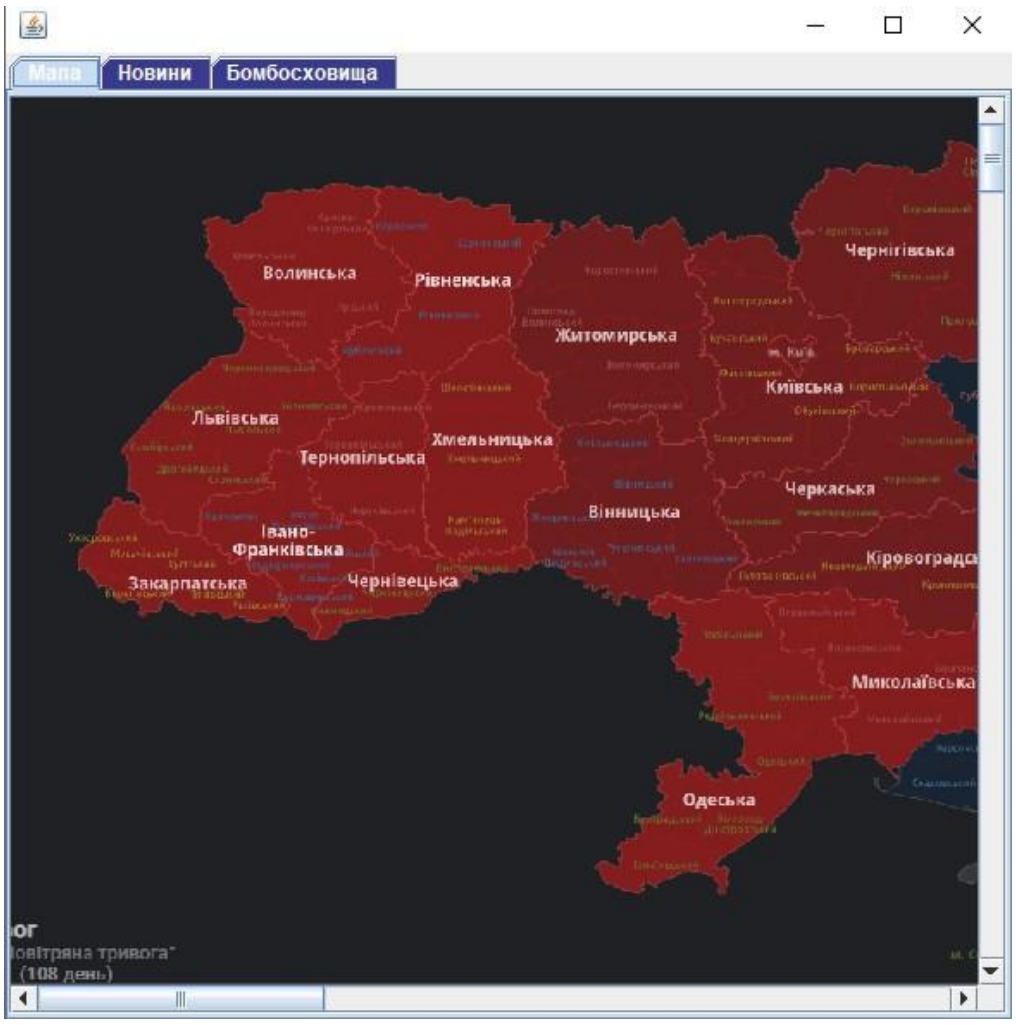


Рисунок 2.6 -Вкладка з мапою тривог

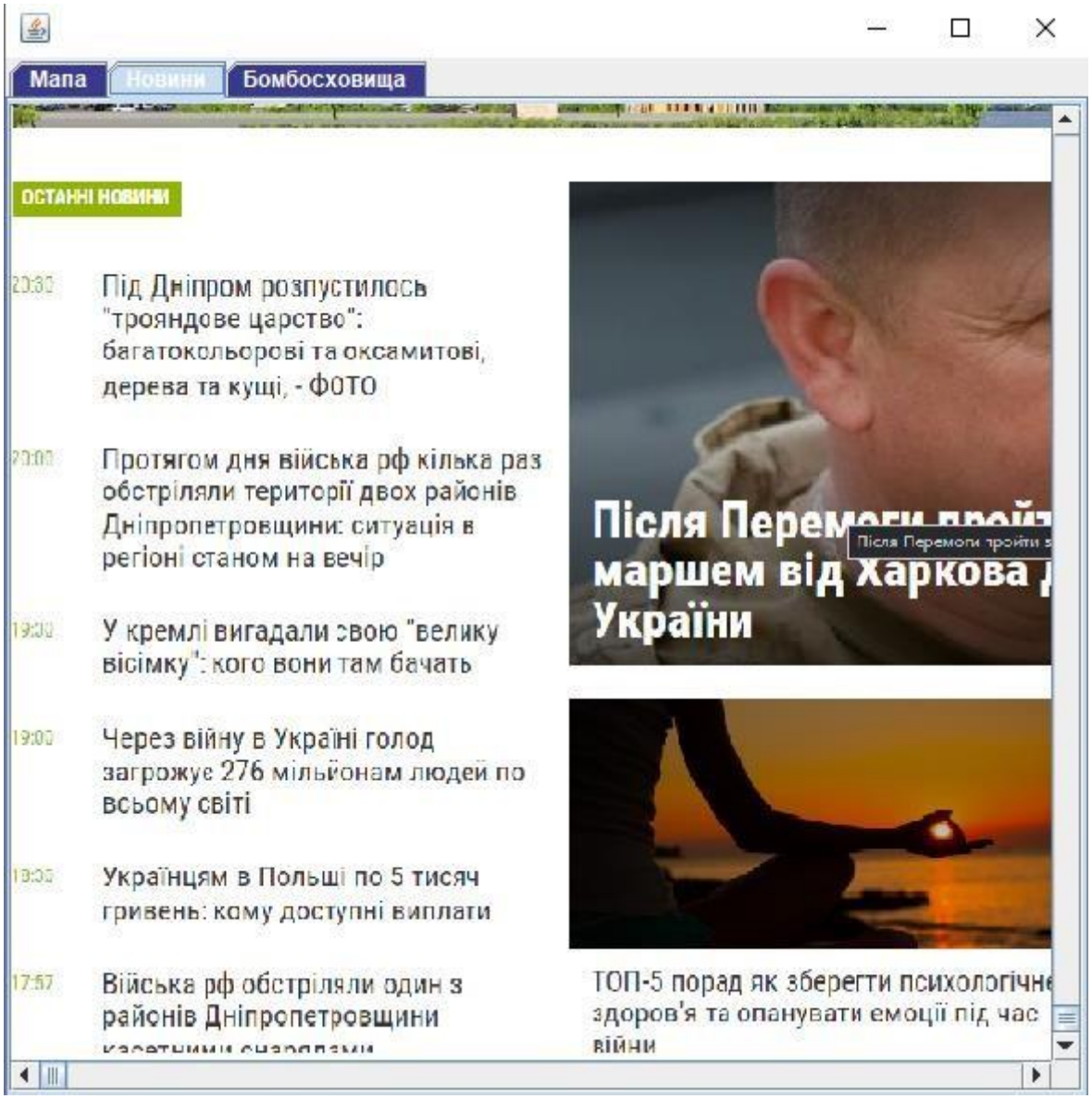


Рисунок 2.7 -Вкладка з новинами

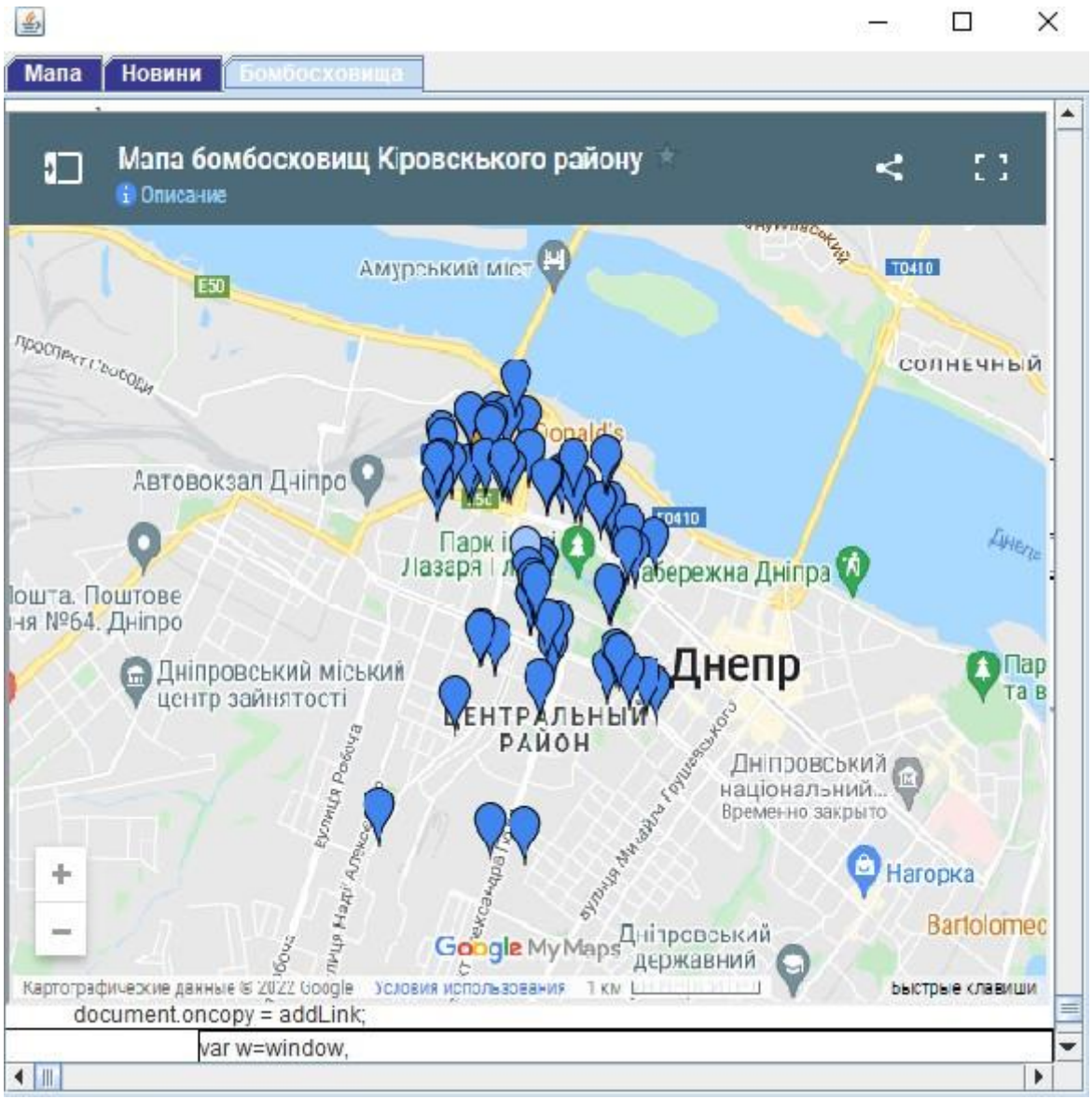


Рисунок 2.8 -Вкладка с мапою бомбосховищ

Інтерфейс на HTML:

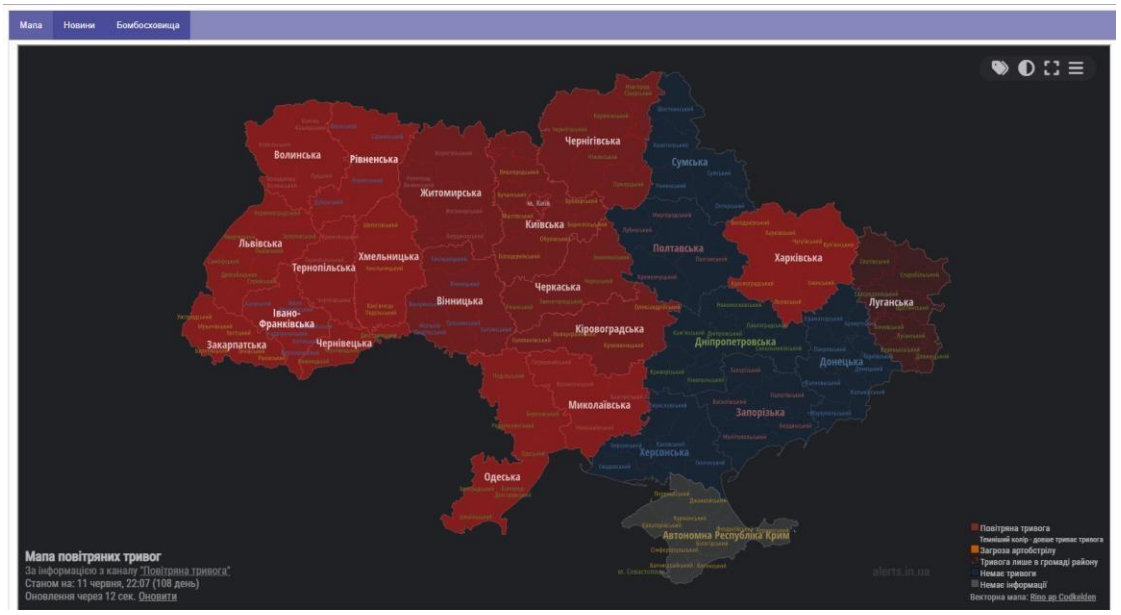


Рисунок 2.9 -Вкладка з мапою тривог

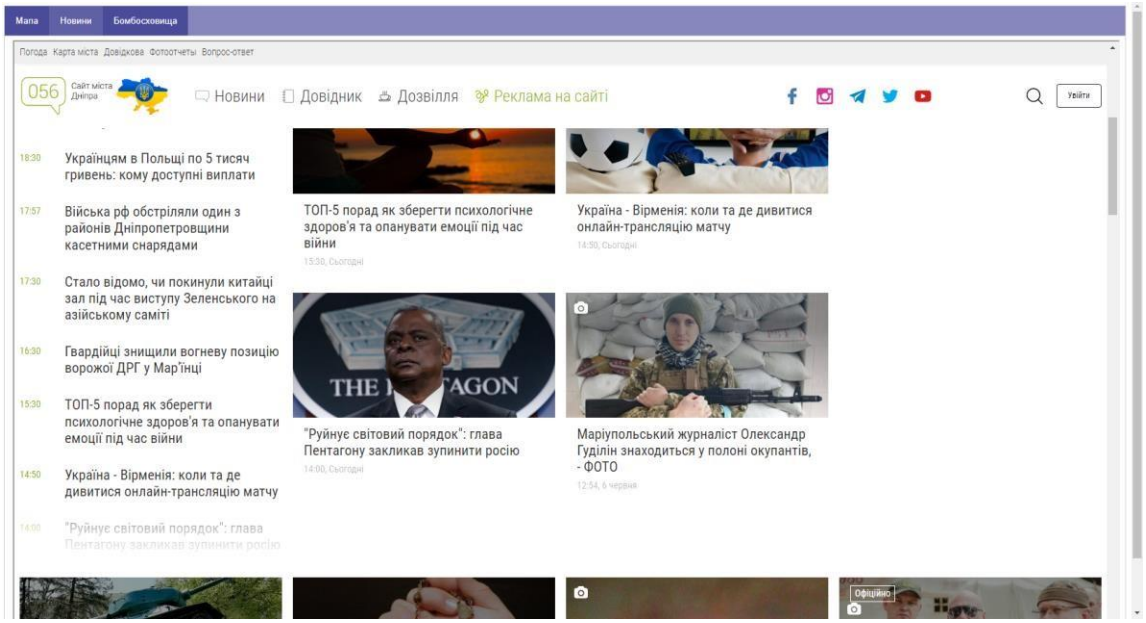


Рисунок 2.10 -Вкладка з новинами

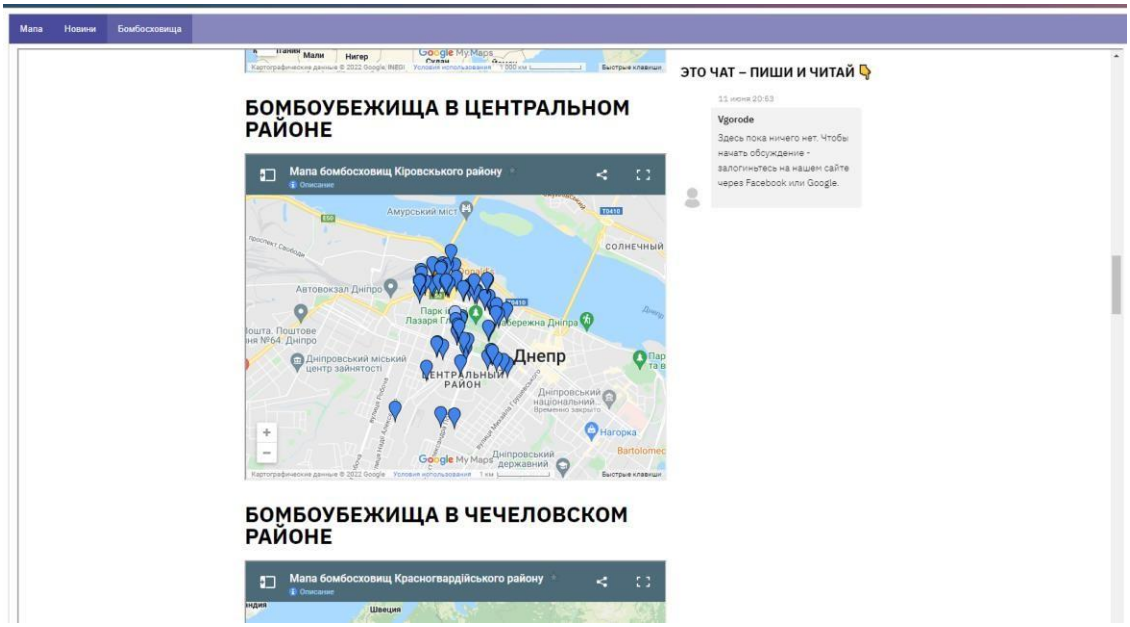


Рисунок 2.11 -Вкладка с мапою бомбосховищ

2.3.5. Тестування додатку

Додаток був протестований ручним способом. Створився теск кейс с шагами, які програма повинна була пройти, щоб тест був позитивний тест показаний в таб.2.1.

Таблиця 2.1

Тест кейс для перевірки додатку з надзвичайною інформацією

Шаги	**Очікуваний результат**	**Пройдено/ провалено**
Запуск додатку	Відкриється перша вкладка с мапою повітряної тривоги України	Пройдено
Вибір другої вкладки	Відкриється друга вкладка з новиними міста Дніпро	Пройдено
Вибір третьої вкладки	Відкриється третя вкладка з мапо бомбосховища міста Дніпра	Пройдено

***Результат тесту*: Пройдено**

Додаток має повністю функціональний інтерфейс и пройшов перевірку.

2.4. Розробка шаблону інтерфейсу для рекламного додатку

2.4.1. Формулювання задачі

Для формулювання задачі була використана діаграма варіантів використання програми на рис. 2.12.

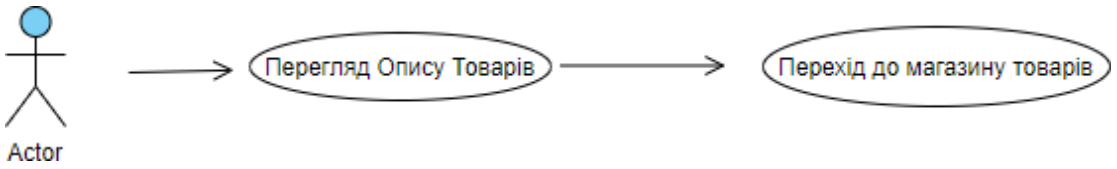


Рисунок 2.12- Схема варіантів використання

2.4.2. Вибір принципів проектування

Для розробки цієї системи був обраний патерн «Абстрактна фабрика»

Абстрактна фабрика – це патерн, що дає змогу створювати пов’язані об’єкти, не прив’язуючись до реалізації цих об’єктів. Якщо відділити загальні характеристики продуктів однієї суті їх можна перенести в окремий інтерфейс й описати спільні риси. А те чим вони будуть відрізнятися вже винести в клас реалізацій. Саме такий інтерфейс и буде фабрикою.

Така система дозволяє створювати продукти, які різноманітні за своєю реалізацією, але мають спільні риси.

За допомогою цього патерну є можливість розробляти кросплатформені додатки, які будуть створюватися наче конструктор з елементів інтерфейсу.

Для демонстрацію було розроблено програму яка показує рекламу та перенаправляє користувача до інтернет магазину.

Додаток демонструє різну рекламу для різних операційних систем, з можливістю перейти до покупки продукту саме для операційної системи користувача.

2.4.3. Ієрархія та взаємодія класів

Взаємодія та структура системи зображена на рис. 2.13

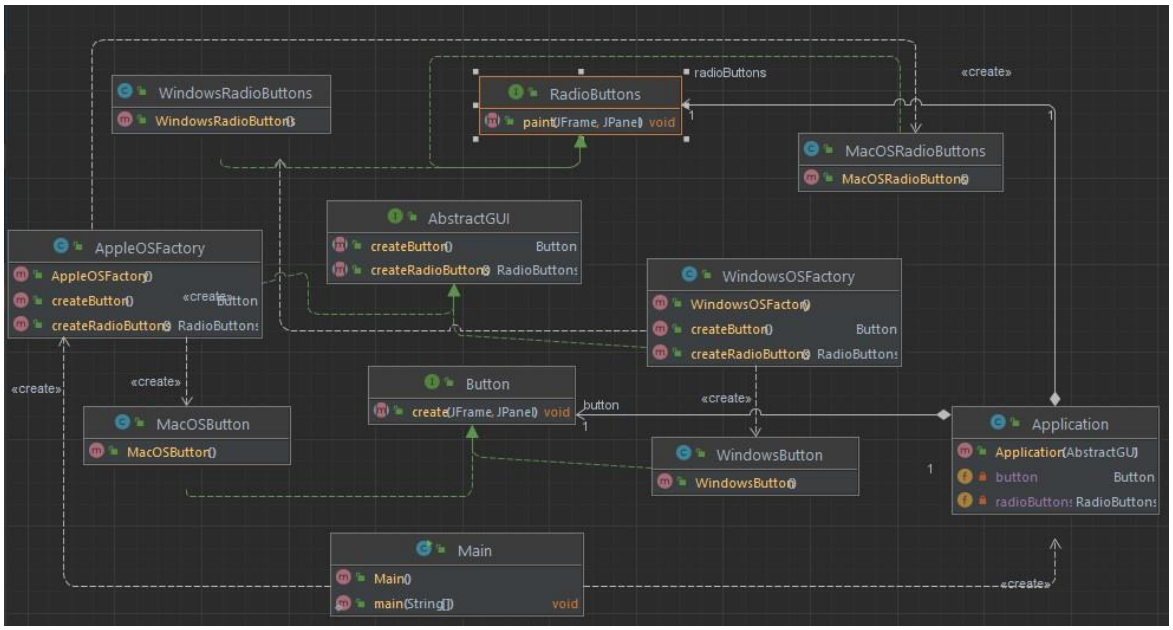


Рисунок 2.13- Схема ієрархії класів

Інтерфейс AbstractGUI виступає асбтрактною фабрикою з двома методами createButton() та createRadioButton() вони є абстрактними для того , щоб під різні операційні системи або потреби реалізувались свої фабрики.

Фабрики AppleOSFactory та WindowsOSFactory в свою чергу мають методи для створення різних типів продуктів одного сімейства, вони використовують різни єлементи наче конструктор для свторення єдиного та цілльного інтерфейсу. В методах createButton() та createRadioButtons() в залежності від реалізаціє повертаються продукти різних систем. Навіть якщо фабрика створює продукти різноє реалізацій вони всеодно повині повертати продукти одного типу класу Button або RadioButtons

Класи абстрактних продуктів Button та RadioButtons мають методи paint() для реалізацій своїх єлементів. Конкретні класи продуктів WindowsRadioButtons, MacOSRadioButtons, WindowsButton, MacOSButton реалізують вже ці методи аби задовольнити потреби вони реалізуться з різним дизайном та різніими функціональними можливостями.

Вибір яку саме фабрику використати передається клієнствокму коду, а той в залежності від потреб може викликати одну чи іншу фабрику.

2.4.4. Розробка інтерфейсу

Створений шаблон інтерфейсу показаний на рис.2.14.

Рисунок 2.14 – Шаблон інтерфейсу

Інтерфейс на JAVA

Для створення цього інтерфейсу необхідно використати наступні елементи:

– JRadioButton – для переходу між продуктами та його опису.

– JTextArea – рекламний банер, який показує інформацію для користувача

– JButton. – для переходу на сайт продавця де можна придбати продукт з оголошення.

Інтерфейс для WindowsOS буде показаний на рис.2.15 та рис.2.16.

Інтерфейс для MacOS буде показаний на рис.2.17 та рис.2.18.

Інтерфейс для Windows

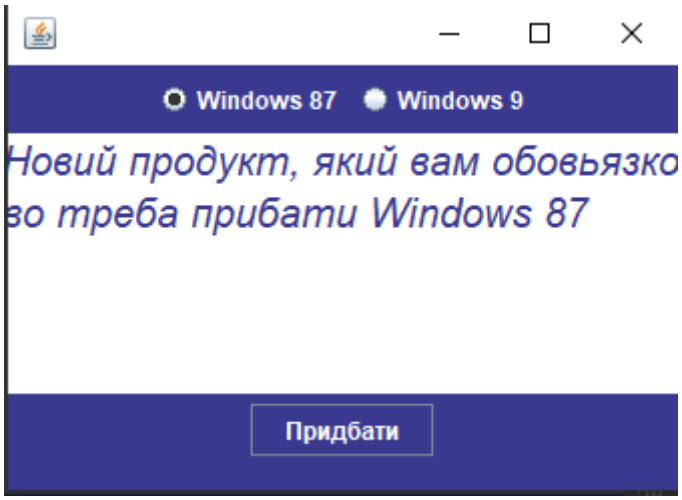


Рисунок 2.15 – Інтерфейс для першого продукту

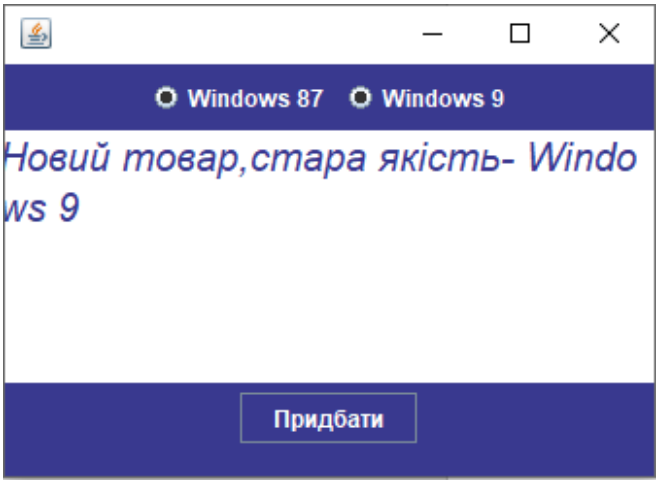


Рисунок 2.16 – Інтерфейс для другого продукту

Інтерфейс на MacOS

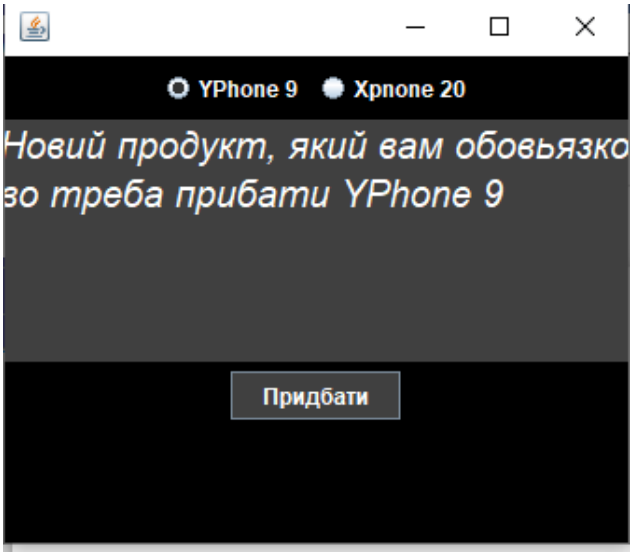


Рисунок 2.17 – Інтерфейс для першого продукту

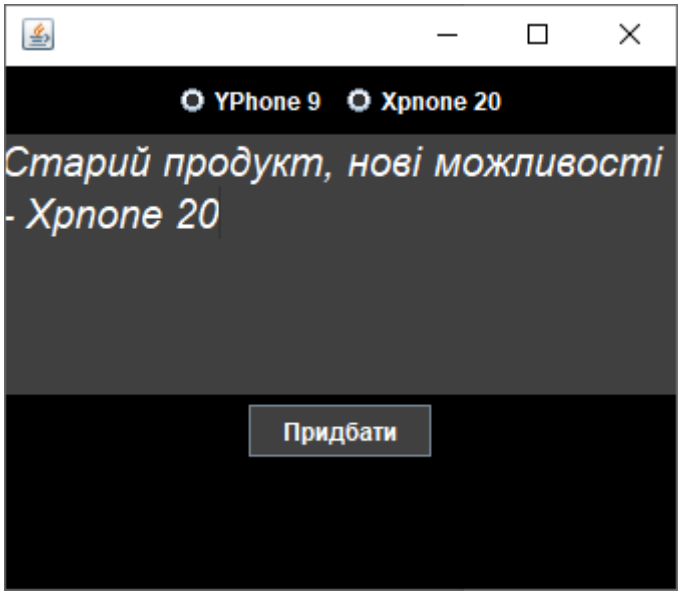


Рисунок 2.18 – Інтерфейс для другого продукту

2.4.5. Тестування продукту

Додаток був протестований ручним способом. Створився теск кейс с шагами, які програма повинна була пройти, щоб тест був позитивний тест показаний в таб.2.2 та таб.2.3.

Таблиця 2.2

Тестування на Windows OS

Шаги	Очікуваний результат	Пройдено/провалено
Запустити додаток та пристрої с ОС Windows	Відкриється меню з товаром «Windows 87» товаром та його рекламним текстом	Пройдено
Вибрати товар «Windows 9»	Відкриється текст рекламного оголошення «Windows 87»	Пройдено
Натиснути кнопку «Придбати»	Відкрється браузер зі сторінкою інтернет магазину	Пройдено

***Результат тесту*: Пройдено**

Таблиця 2.3

Тестування на MacOS

Шаги	Очікуваний результат	Пройдено/провалено
Запустити додаток та пристрої с ОС Mac OS	Відкриється меню з товаром «YPhone 9» товаром та його рекламним текстом	Пройдено
Вибрати товар «Windows 9»	Відкриється текст рекламного оголошення «YPhone 20»	Пройдено
Натиснути кнопку «Придбати»	Відкрється браузер зі сторінкою інтернет магазину	Пройдено

***Результат тесту*: Пройдено**

Тестування показало що додатки працюють без помилок.

2.5. Розробка шаблону інтерфейсу для редактору файлів

2.5.1. Формулювання задачі

Для формулювання задачі була використана діаграма варіантів використання програми на рис. 2.19.

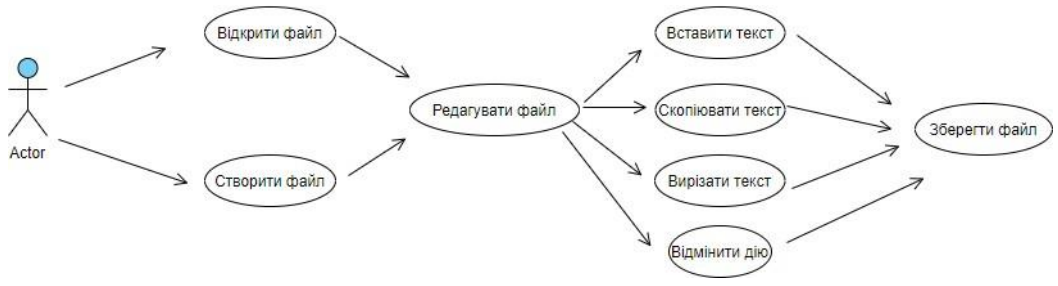


Рисунок 2.19- Схема варіантів використання

2.5.2 Вибір принципів проектування

Для створення редактору файлів був використаний патерн Команда.

Цей патерн дозволяє запроси як аргументи при визові методів, це в свою чергу дозволяє ставити команди в чергу. Найбільша користь такого підходу, що можна робити відкат дії.

Створивши таку команду її можна визивати з любої частини коду.

Найбільш наглядно це можна показати на прикладі текстового редактуору, коли одну и ту саму дію треба використовувати з різними елемантами.

Наприклад ви можете вставити текст через спеціальну кнопку «Вставка». Реалізацію цієї функціє можна побачити на рис. 2.20.

```
past.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) { executeCommand(new Paste(application)); }
});
```

Рисунок 2.20 – Реалізація функції вставки тексту через кнопку

Цей код показує як команда визивається через спеціальну кнопку.

Або можна використати комбінацію клавіш Cntr + C. Реалізацію цієї функціє можна побачити на рис. 2.21.

```
@Override
public void keyPressed(KeyEvent e) {
    if (e.isControlDown() && e.getKeyCode() == KeyEvent.VK_C) {
        executeCommand(new Copy(application));
```

Рисунок 2.21 – Реалізація функції вставки тексту через комбінацію клавиш

На цих рисунка можна побачити, що хоч використовуються різні елементи сам запрос однаковий.

2.5.2. Ієрархія та взаємодія класів

Взаємодія та структура системи зображена на рис. 2.22

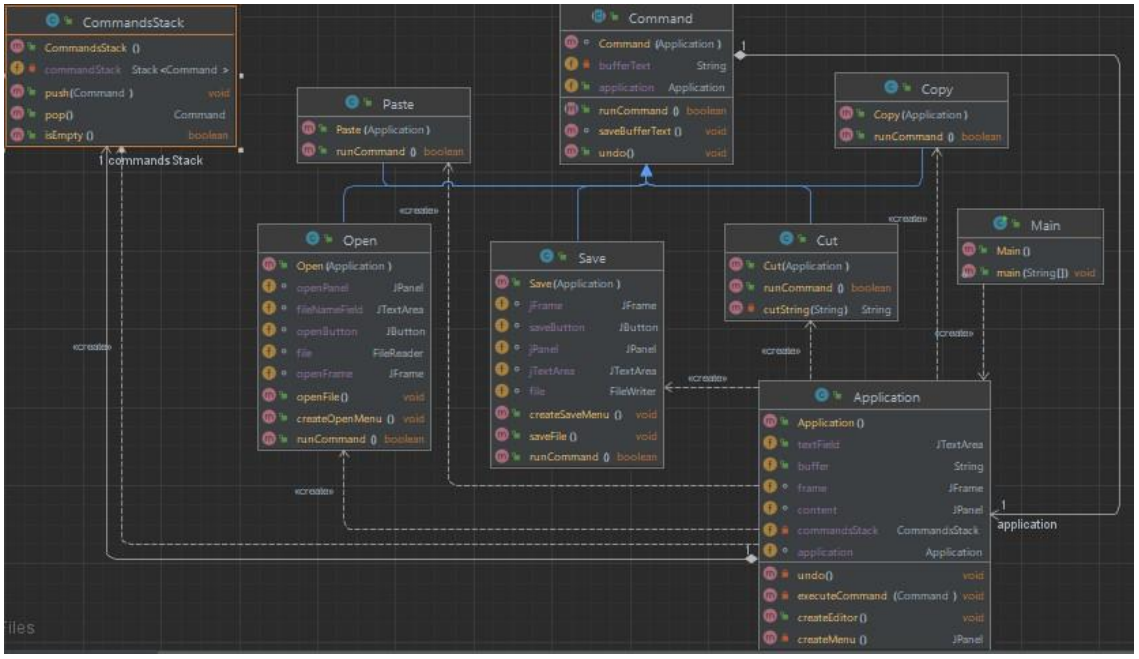


Рисунок 2.22- Схема ієрархії класів

Клас Command має загальний інтерфейс для конкретних команд також він має шаблону реалізацію відмін команд. Найголовніший метод це runCommand() він відповідає за запуск відповідних дій.

Класи команд Copy Cut Past Save Open реалізують логіку своїх дій через метод runCommand(); Класи команд, що змінюють панель редактору коду, а це Cut та Past, вони повинні зберігати стан редактору до своєї дій. Якщо команду повертають зі стеку команд, то вона повинна відновити стан редактору до свого виклику.

Історія команд реалізується за допомоги структури даних Stack, це означає шо команди можна записувати за допомогою методів push та undo. Використання Stack в свою чергу означає що є можливість витягувати клманди с кінця, що я приємним варіантом для відміни команд редагування тексту.

Сам виклик команд не передбачає нічого складного. Якщо нам необхідно зробити виклик команди, мі створюємо якусь обробку дій графічного інтерфейсу. А в самій обробці дій визиваємо визиваємо метод runCommand() та об’єкт класу команди. Таким чином ми можемо реалізувати

для різних елементів або комбінацій клавіш виклик команд який не засмічує код та допомагає вирішенню проблеми.

2.5.3. Розробка інтерфейсу

Інтерфейсу необхідно мати меню, яке буде складатися с команд редагування файлу, а також потрібно вікно з можливістю відкривати та зберігати файли.

На першій вкладці необхідно створити меню яке викликало вікно відкриття або зберігання файлу, а також саме поле для редагування файлу.

За допомогою другої вкладки користувач необхідно мати можливість вставляти, копіювати, вирізати текст або відміняти команди.

Створений шаблон інтерфейсу показаний на рис.2.23- рис.2.26.

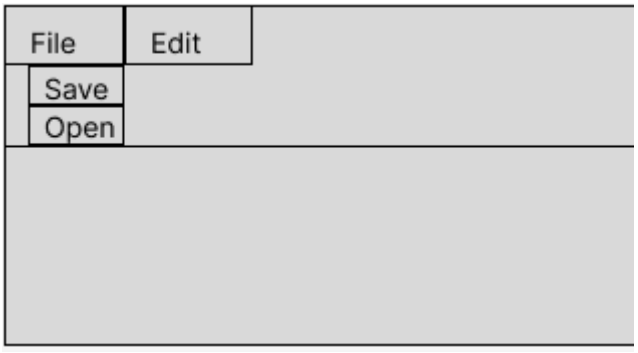


Рисунок 2.23 – Шаблон першого меню

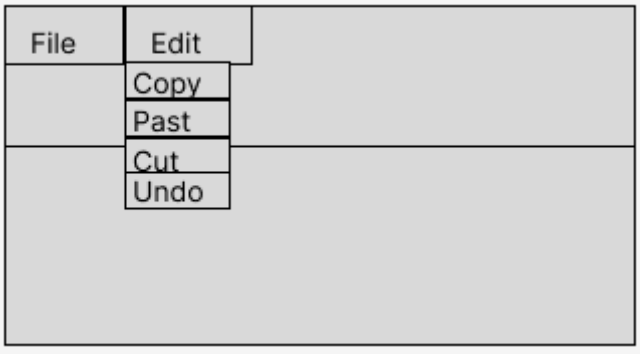


Рисунок 2.24 – Шаблон другого меню

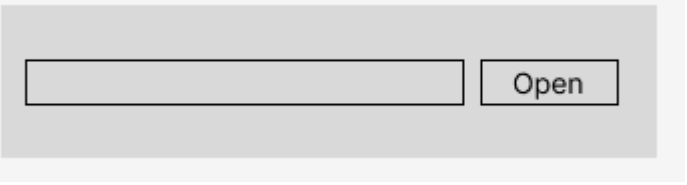


Рисунок 2.25 – Шаблон меню відкриття файлу

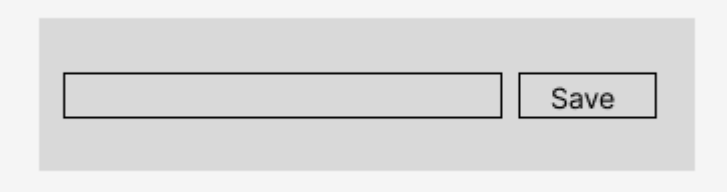


Рисунок 2.26 – Шаблон меню зберігання файлу

Для створення такого інтерфейсу необхідно використати наступні елементи:

– JTextArea – для поля редагування файлів та вводу імені файлів
– JButton – виклику команд збереження та відкривання
– JMenuBar- для створення меню з редагування файлів

Реалізований інтерфейс буде показаний на рис.2.27- рис.2.30рис.

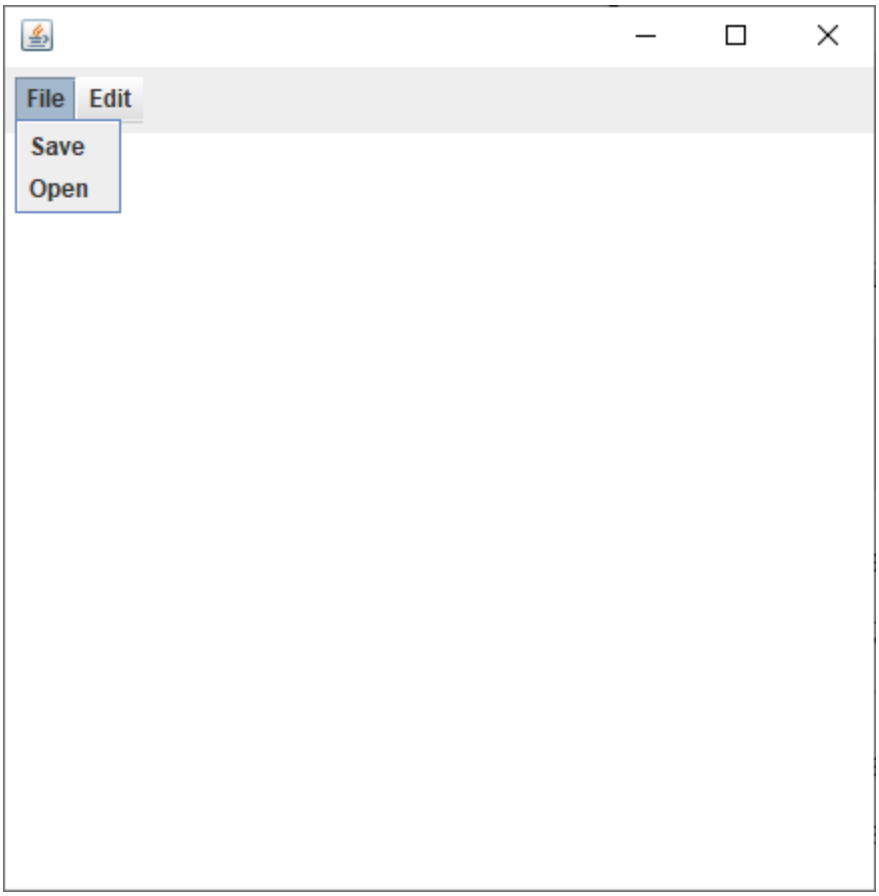


Рисунок 2.27 – Меню взаємодії з файлом

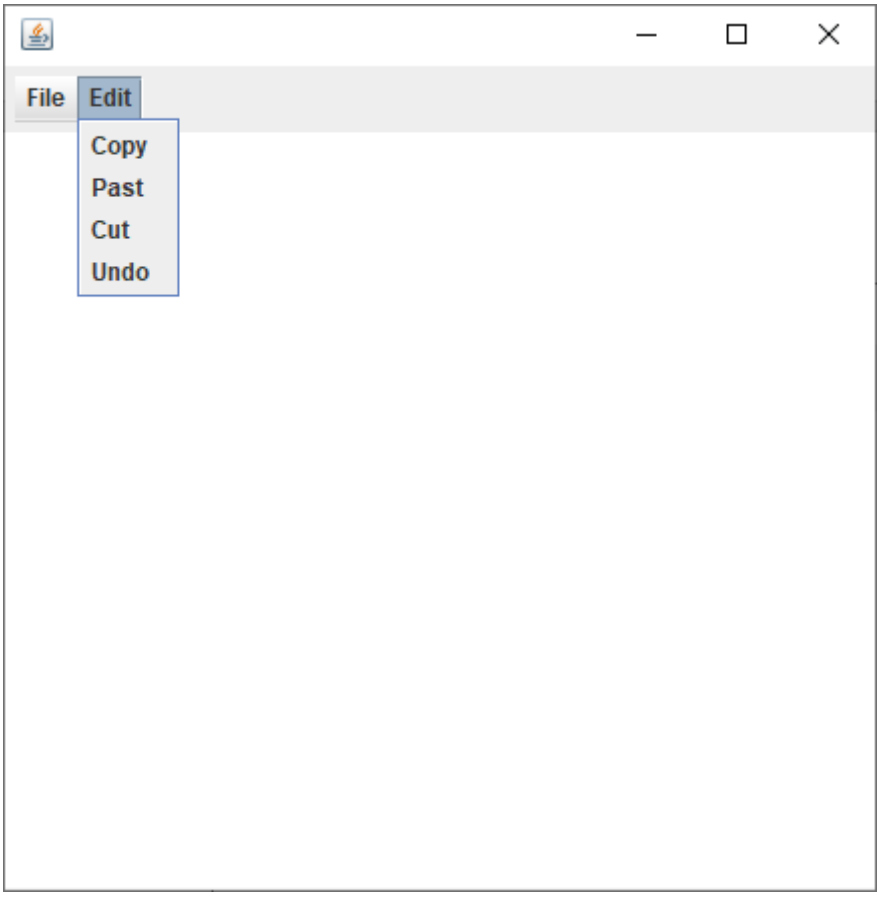


Рисунок 2.28 – Меню редагування файлу

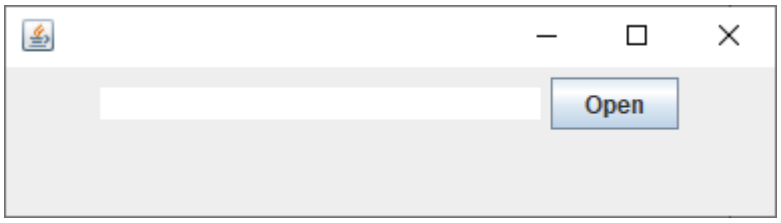


Рисунок 2.29 – Меню відкриття файлу

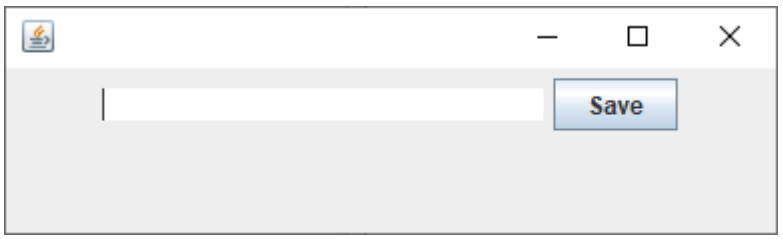


Рисунок 2.30 – Меню збереження файлу

2.5.4. Тестування продукту

Додаток був протестований ручним способом. Створився тест кейс с шагами, які програма повинна була пройти, щоб тест був позитивний тест показаний в таб.2.4 таб.2.6.

Таблиця 2.4

Перевірка функціоналу редагування тексту

Шаги	Очікуваний результат	Пройдено/провалено
Запустити додаток	Відкривається меню с з редагуванням файлу	Пройдено
Ввести текст «Хара Г.Л.» в полі редагування файлу	В полі редагування файлу з’явиться текст	Пройдено
Виділити текст Натиснути кнопку Edit Натиснути кнопку Copy Натиснути кнопку Paste	В полі редагування з’явиться дублікат тексту	Пройдено
Виділити текст Натиснути кнопку Edit Натиснути кнопку Cut	Текст буде видалений з поля	Пройдено

***Результат тесту*: Пройдено**

Таблиця 2.5

Перевірка функціоналу редагування тексту з комбінацією клавиш

Шаги	Очікуваний результат	Пройдено/провалено
Запустити додаток	Відкривається меню с з редагуванням файлу	Пройдено
Ввести текст «Хара Г.Л.» в полі редагування файлу	В полі редагування файлу з’явиться текст	Пройдено
Виділити текст Натиснути кнопку Cntr + C Натиснути кнопку Cntr + V	В полі редагування з’явиться дублікат тексту	Пройдено
Виділити текст Натиснути кнопку Cntr + X	Текст буде видалений з поля	Пройдено

***Результат тесту*: Пройдено**

Таблиця 2.6

Перевірка функціоналу зберігання та відкриття файлу

Шаги	Очікуваний результат	Пройдено/провалено
Запустити додаток	Відкривається меню с з редагуванням файлу	Пройдено
Ввести текст «Хара Г.Л.» в полі редагування файлу	В полі редагування файлу з’явиться текст	Пройдено
Натиснути кнопку File Натиснути кнопку Save	З’явиться вікно с полем вводом для імені файлу	Пройдено
Ввести ім’я файлу text.txt Натиснути кнопку Save/cntr + S	Створюється файл text.txt	Пройдено
Натиснути кнопку File Натиснути кнопку Open	З’явиться вікно с полем вводом для імені файлу	Пройдено
Ввести ім’я файлу text.txt Натиснути кнопку Open	Відкривається файл В полі редагування файлу з’явитися текст «Хара Г.Л.»	Пройдено

***Результат тесту*: Пройдено**

2.6. Висновки до другого розділу

В цьому розділі були повністю розроблені додатки за допомого за допомоги шаблонів інтерфейсів.

Були зроблені наступне: шаблони інтерфейсів, реалізація інтерфейсів, розробка архітектури додатків та проведено повне тестування всіх шаблонів.

Найкраща якість всіх додатків це їх великий потенціал для розширення функціоналу. Вони зберігають принцип відкритості/закритості, що дозволяють розширювати інтерфейс без зміни початкових сутностей.

Система розбивається на велику кількість модулів, які є простими та не перевантажують систему, а це в свою чергу відповідає принципу KISS.

Також систему відповідає ще одному принципу SOLID, а саме Барбари Лісков, це означає, що нащадків можна використовувати в основній логіці програми при цьому ефективність та коректність програми не постраждає.

Та один із принципів ООП- поліморфізм. Завдяки якому програма розширюється новим функціоналом, створюючи декілька реалізацій одного інтерфейсу та дозволяючи при реалізацій клієнтського коду використовувати абстракції замість конкретної реалізацій.

ВИСНОВКИ

Проводячи дослідження, було визначено поняття патерни проектування та вплив їх використання при побудові програмних продуктів, їх ефективність при розробки архітектури на реальних прикладах та вплив їх на можливість розширення та спрощення побудови системи.

Була визначена класифікація патернів так огляд їх представників: приклади структурних та породжувальних патернів, а саме Декоратор та Одинак. Ці приклади повністю описують структуру патернів: описують проблему та ситуацію в якій стало необхідно використовувати патерн, кроки необхідні для її подолання, структуру та взаємодія класів та результат позитивного впливу на систему .

Були визначенні патерни для створення шаблонів інтерфейсів. Патерни Фабричний метод, Абстрактна фабрика та Команда є найактивнішими для створення шаблонів інтерфейсів, тому що мають велику кількість переваг, а саме: дозволяють створювати систему с можливість к розширенню функціоналу без роздуття величезною кількістю класів. Фабричний метод та Абстрактна фабрика дозволяють створювати кросплатформені продуктів, що призводить до збільшення потенційних користувачів. А патерн Команда допомагає створювати функціональний інтерфейс з мінімальними зусиллями, що створює більш ефективний продукт.

В результаті було розроблено три додатки:

- Кросплатформна програма з надзвичайною інформацією
- Рекламний додаток
- Текстовий редактор

Для їх створення був здійснений: аналіз та вибір мови програмування, опис взаємодії класів та методів, розроблені моделі інтерфейсів, опис взаємодії з користувач та шляхи для реалізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крістофер, А. Мова шаблонів. Міста. Будівлі. Будівництво / А. Крістофер. — Нью-Йорк : OXFORD UNIVERSITY PRESS, 1977-313с.
2. Мова шаблонів. Міста. Будівлі. Будівництво / Г. Еріх, Р.Хелм,Р.Джонсон,Д.Влессілес— Санкт-Петербург : Бібліотека програміста, 2015.-368с.
3. Швидка довідка з патернів [Електронний ресурс] – Режим доступу до ресурсу: http://www.mcdonaldland.info/2007/11/28/40/
4. Жанг, К. Патерни дизайну.Елементи багаторазового об’єктно-орієнтованого програмного забезпечення./ Жанг К. – 431с.
5. 3 типи шаблонів проектування, які повинні знати всі розробники (з прикладами коду кожного) [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/the-basic-design-patterns-all-developers-need-to-know/
6. Фабричний метод [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/factory-method
7. Що таке технологія Java і яково її застосування? [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/ru/download/help/whatis_java.html
8. Переваги Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/ru/aix/7.2?topic=monitoring-advantages-java
9. Як створити графічний інтерфейс із прикладами Swing на Java[Електронний ресурс] – Режим доступу до ресурсу: https://hr-vector.com/java/swing-graficheskij-interfejs
10. Вступ до Swing на Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.c-sharpcorner.com/UploadFile/fd0172/introduction-of-swing-in-java/

11. Документування веб-технологій, включаючи CSS, HTML і JavaScript, [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org
12. Переваги HTML Електронний ресурс] – Режим доступу до ресурсу: https://uk.education-wiki.com/1711441-advantages-of-html
13. Ручне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/manual-testing
14. Переваги CSS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.education-wiki.com/6587081-advantages-of-css
15. Абстрактна фабрика [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/abstract-factory
16. Команда [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/command

ДОДАТОК А

Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Найменування	Кільк. аркушів	Примітки
1				
2		Документація		
3				
4	**ІТКІ.КР.22.11.ДА.ПЗ**	Пояснювальна записка	65	Формат А4
5				
6		Презентація		
7				
8		Диск CD-R з презентацією	1	Диск CD-R
9				

					ІТКІ.КР.22.11.ДА.ПЗ.			
Зм.	Ар-куш	№ докум.	Підпис	Дата				
Розроб.		Хара Г.Л.			**Матеріали кваліфікаційної роботи**	Літ.	Аркуш	Арку-шів
Керівник		Соколова Н.О.				Н	1	1
Рецензент		Удовик І.М.				НТУ «ДП», 12; 126-18-1		
Н.контр.		Коротенко Г.М.						
Зав. каф.		Гнатушенко В.В.						