Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 6

з дисципліни «**Технології розроблення програмного забезпечення**»

Патерни проектування.

Виконав:

Студент групи ІА-31

Шереметьєв Дмитро

Київ – 2025

Вихідний код: https://github.com/Dimon4ick68/MindMapLabs

Тема: Патерни проектування.

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

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.

Варіант:

20. **Mind-mapping software** (strategy, prototype, abstract factory, bridge, composite, SOA)

Візуальний додаток для складання "карт пам'яті" з можливістю роботи з декількома картами (у вкладках), автоматичного промальовування ліній, додавання вкладених файлів, картинок, відеофайлів (попередній перегляд); можливість додавання значків категорій / терміновості, обведення областей карти (поділ пунктирною лінією).

Хід роботи

У рамках виконання лабораторної роботи було реалізовано механізм **зміни тем оформлення** для редактора ментальних карт. Для цього використано патерн **Abstract Factory**, який дозволяє створювати сімейства взаємопов'язаних об'єктів (стиль вузлів, стиль ліній, колір фону) без прив'язки до їхніх конкретних класів.

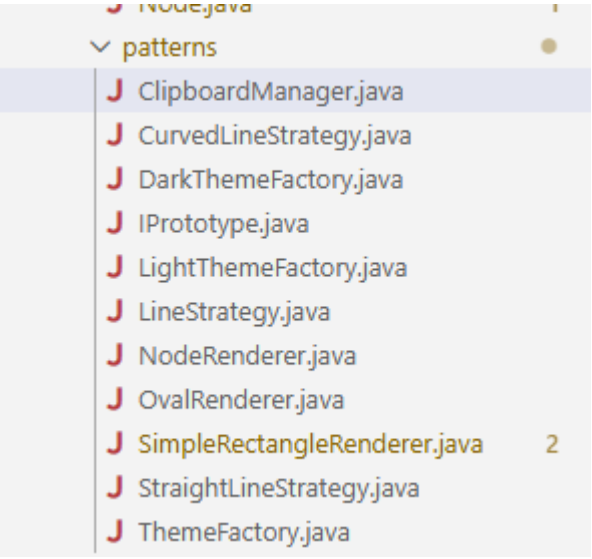


Рисунок 1 Структура проекту

Реалізовано такі компоненти патерну:

1. ThemeFactory (Abstract Factory — Абстрактна фабрика) Інтерфейс, що оголошує методи створення компонентів візуальної теми.

- **Методи:**
  - createNodeRenderer(): повертає об'єкт для малювання вузлів.
  - createLineStrategy(): повертає стратегію малювання ліній зв'язку.
  - getBackgroundColor(): повертає колір фону редактора.

2. LightThemeFactory та DarkThemeFactory (Concrete Factories — Конкретні фабрики) Класи, що реалізують інтерфейс ThemeFactory. Кожна фабрика створює узгоджений набір продуктів для певної теми:

- **Світла тема:** Використовує прямокутні вузли (SimpleRectangleRenderer), прямі лінії (StraightLineStrategy) та білий фон.
- **Темна тема:** Використовує овальні вузли (OvalRenderer), прямі лінії та темний фон.

3. EditorForm (Client — Клієнт) Клас редактора, який використовує фабрику. Він зберігає посилання на поточну фабрику (currentThemeFactory) через інтерфейс ThemeFactory. При малюванні карти редактор запитує у фабрики потрібні інструменти (рендерер, стратегію ліній, колір), не знаючи, яка саме тема зараз активна. Це дозволяє перемикати теми "на льоту" одним рядком коду (зміною об'єкта фабрики).

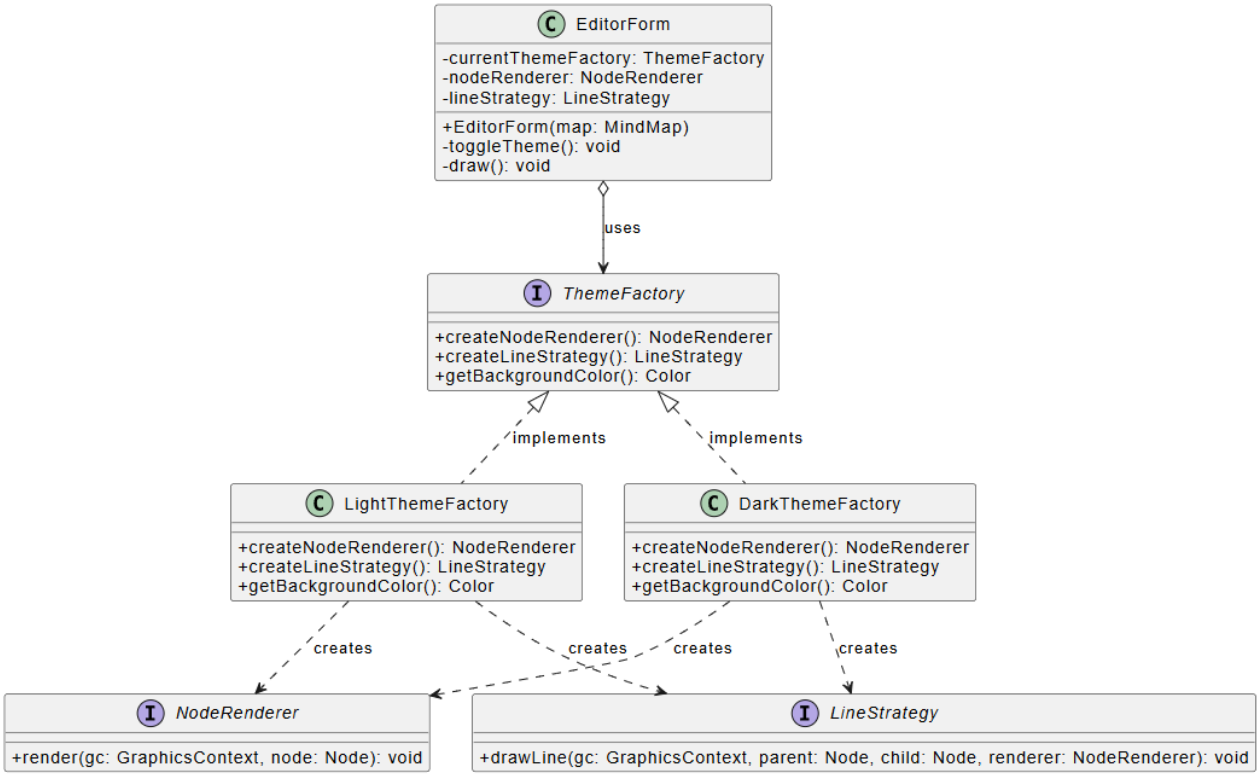


Рисунок 2 Діаграма класів

Реалізація

У рамках проєкту "MindApp" патерн **Abstract Factory** було використано для створення системи візуальних тем оформлення. Це дозволило об'єднати різні візуальні компоненти (вигляд вузлів, стиль ліній, кольори) у узгоджені сімейства та змінювати їх комплексно.

- **Абстрактна фабрика (ThemeFactory)** — це інтерфейс, який оголошує методи для створення абстрактних продуктів: рендерера вузлів (createNodeRenderer), стратегії малювання ліній (createLineStrategy) та отримання кольору фону (getBackgroundColor).
- **Конкретні фабрики (LightThemeFactory, DarkThemeFactory)** реалізують цей інтерфейс. Кожна фабрика відповідає за створення продуктів, що належать до однієї теми:
  - *Світла тема* використовує прямокутні вузли (SimpleRectangleRenderer) та білий фон.
  - *Темна тема* використовує овальні вузли (OvalRenderer) та темний фон.
- **Абстрактні продукти (NodeRenderer, LineStrategy)** — це інтерфейси для компонентів, які створюються фабрикою. Вони визначають контракти для малювання елементів.
- **Клієнт (EditorForm)** працює з фабрикою через абстрактний інтерфейс ThemeFactory. При ініціалізації або зміні теми редактор отримує конкретну реалізацію фабрики і використовує її для створення необхідних інструментів малювання, не вдаючись у деталі їхньої реалізації.

Проблеми, які вирішує патерн

- **Залежність від конкретних класів.** Без використання фабрики код редактора (EditorForm) був би наповнений жорсткими прив'язками до конкретних класів візуалізації (наприклад, new SimpleRectangleRenderer()). Це ускладнило б заміну стилю малювання.
- **Узгодженість сімейств об'єктів.** При використанні тем важливо, щоб стиль вузлів відповідав стилю ліній та фону (наприклад, не можна малювати білі

лінії на білому фоні або використовувати "футуристичні" вузли у "класичній" темі). Абстрактна фабрика гарантує, що всі створені об'єкти належатимуть до однієї теми, оскільки вони створюються єдиним об'єктом-фабрикою.
- **Складність налаштування.** Замість того, щоб налаштовувати кожен компонент окремо (фон, вузли, лінії), клієнтський код просто перемикає фабрику, і вся візуальна частина оновлюється автоматично.

Переваги використання

- **Ізоляція конкретних класів.** Клієнтський код оперує лише інтерфейсами, що дозволяє змінювати реалізацію продуктів (наприклад, змінити алгоритм малювання овалу) без впливу на основний код програми.
- **Легка заміна сімейств продуктів.** Зміна візуальної теми відбувається в одному місці — шляхом заміни об'єкта конкретної фабрики в полі currentThemeFactory. Вся решта логіки малювання підлаштовується автоматично.
- **Дотримання принципу відкритості/закритості (Open/Closed Principle).** Додавання нової теми (наприклад, "High Contrast Theme") не потребує зміни існуючого коду редактора. Достатньо створити новий клас фабрики, що реалізує ThemeFactory, та відповідні класи продуктів.

Висновок

У ході виконання лабораторної роботи було вдосконалено проєкт системи для створення ментальних карт ("MindApp") із використанням породжувального патерну **Abstract Factory** (Абстрактна фабрика).

Реалізовано основні компоненти патерну:

- **ThemeFactory** — інтерфейс абстрактної фабрики, що визначає контракт для створення сімейств взаємопов'язаних об'єктів візуалізації (рендерерів вузлів та стратегій малювання ліній).

- **LightThemeFactory** та **DarkThemeFactory** — конкретні фабрики, що забезпечують створення узгоджених наборів компонентів для світлої та темної тем оформлення відповідно.
- **EditorForm** — клієнтський клас, який використовує фабрику для побудови інтерфейсу, не залежачи від конкретних класів продуктів.

Застосування патерну **Abstract Factory** дозволило досягти гнучкості в налаштуванні зовнішнього вигляду програми, забезпечило гарантовану сумісність візуальних елементів в межах однієї теми та спростило додавання нових стилів оформлення в майбутньому без модифікації основного коду редактора. Також було розроблено відповідну діаграму класів, що відображає архітектуру реалізованого рішення.

Код

DarkThemeFactory.java

```
You, 2 days ago | 1 author (You)
1   package com.mindapp.client.patterns;
2
3   import javafx.scene.paint.Color;
4
    You, 2 days ago | 1 author (You)
5   public class DarkThemeFactory implements ThemeFactory {
6       @Override
7       public NodeRenderer createNodeRenderer() {
8           return new OvalRenderer(); // Інший вигляд вузлів
9       }
10
11      @Override
12      public LineStrategy createLineStrategy() {
13          return new StraightLineStrategy(); // Можна лишити прямі або додати криві
14      }
15
16      @Override
17      public Color getBackgroundColor() {
18          return Color.rgb(red: 30, green: 30, blue: 30); // Темний фон
19      }
20  }       You, 2 days ago • init commit, 1-3 lab
```

LightThemeFactory.java

```java
package com.mindapp.client.patterns;

import javafx.scene.paint.Color;

public class LightThemeFactory implements ThemeFactory {
    @Override
    public NodeRenderer createNodeRenderer() {
        return new SimpleRectangleRenderer();
    }

    @Override
    public LineStrategy createLineStrategy() {
        return new StraightLineStrategy();
    }

    @Override
    public Color getBackgroundColor() {
        return Color.WHITE;
    }
}
```

ThemeFactory.java

```java
You, 2 days ago | 1 author (You)
package com.mindapp.client.patterns;

import javafx.scene.paint.Color;

You, 2 days ago | 1 author (You)
public interface ThemeFactory {
    // Створює сімейство об'єктів:
    NodeRenderer createNodeRenderer();
    LineStrategy createLineStrategy();
    Color getBackgroundColor();
}
```

EditorForm.java

```
    }

    private void toggleTheme() {
        if (currentThemeFactory instanceof LightThemeFactory)
            currentThemeFactory = new DarkThemeFactory();
        else
            currentThemeFactory = new LightThemeFactory();
        nodeRenderer = currentThemeFactory.createNodeRenderer();
        lineStrategy = currentThemeFactory.createLineStrategy();
        draw();
    }
```

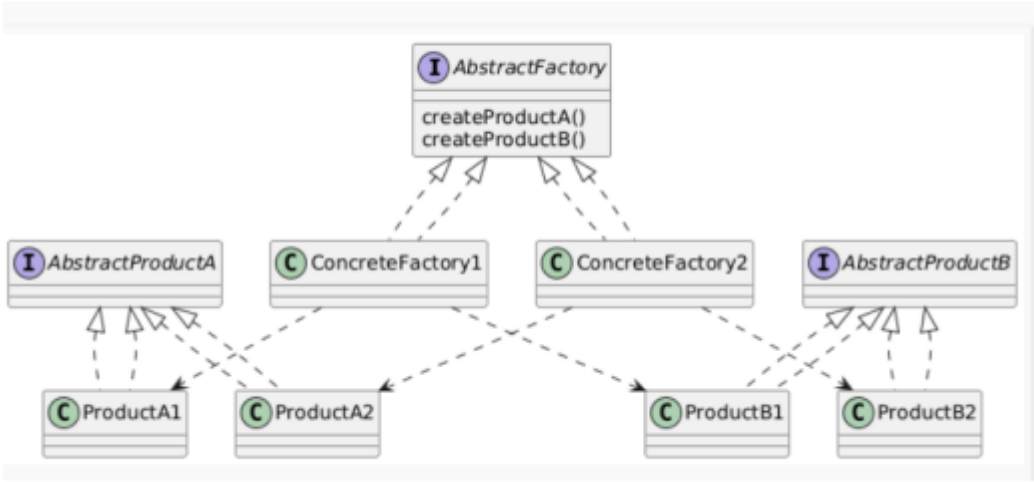
```
    // --- ПАТЕРНИ ---
    // Abstract Factory: Фабрика тем
    private ThemeFactory currentThemeFactory = new LightThemeFactory();      You, 2
```

Відповіді на контрольні питання:

1. Призначення шаблону «Абстрактна фабрика»

Шаблон Абстрактна фабрика використовується для створення сімейств пов’язаних об’єктів без прив’язки до конкретних класів, забезпечуючи узгодженість продуктів між собою. 10

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними

взаємодія?

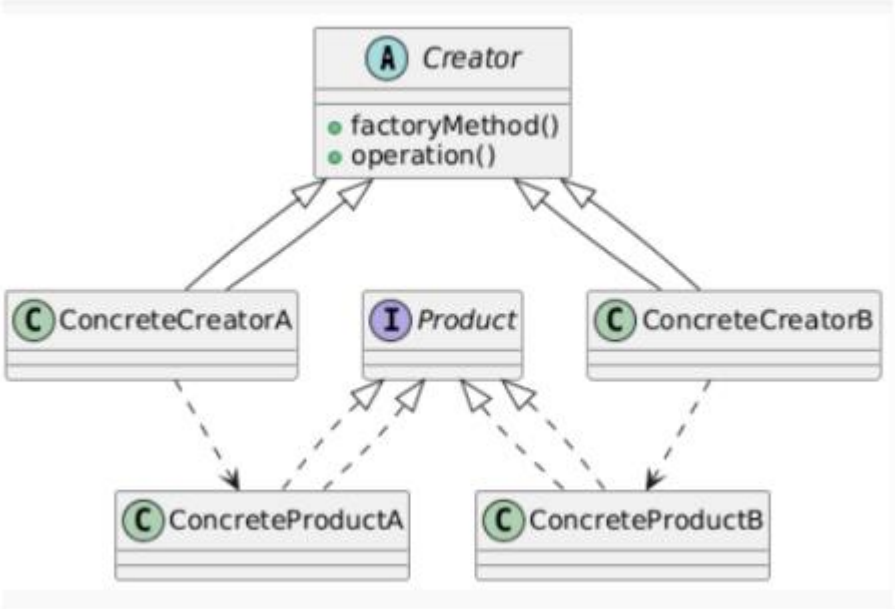
- AbstractFactory - інтерфейс фабрики
- ConcreteFactory - конкретні фабрики
- AbstractProduct - загальні інтерфейси продуктів
- ConcreteProduct - реалізації продуктів

Клієнт викликає методи абстрактної фабрики, яка повертає відповідні узгоджені об'єкти.

4. Яке призначення шаблону «Фабричний метод»?

Створює об’єкти через делегування створення підкласам, дозволяючи змінювати тип об’єктів без зміни коду клієнта.

5. Нарисуйте структуру шаблону «Фабричний метод».



6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

- Creator - містить фабричний метод
- ConcreteCreator - реалізує фабричний метод
- Product - базовий інтерфейс продукту

• ConcreteProduct - конкретні продукти

Creator викликає factoryMethod(), який визначає підклас.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Шаблони Абстрактна фабрика та Фабричний метод обидва належать до породжуючих шаблонів, але застосовуються в різних ситуаціях та відрізняються підходом до створення об'єктів.

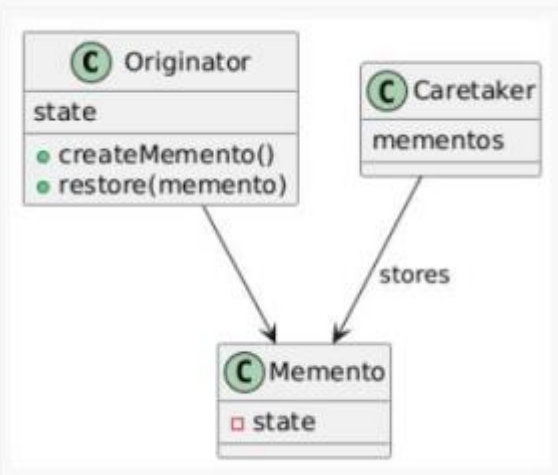
Абстрактна фабрика використовується тоді, коли потрібно створювати групу взаємопов’язаних або взаємосумісних об’єктів. Головна ідея - об’єкти створюються фабрикою, яка повертає цілу “лінійку” продуктів, що логічно 12 працюють разом. Клієнт не знає конкретних класів і працює лише з інтерфейсами. При зміні фабрики автоматично змінюються всі продукти, які вона створює.

Фабричний метод, на відміну від цього, зосереджений на створенні одного типу продукту, і сам процес створення делегується підкласам через перевизначення методу. Основна ідея -дати можливість змінювати тип створюваного об’єкта без зміни базового класу, використовуючи спадкування.

8. Яке призначення шаблону «Знімок»?

Зберігає стан об'єкта без порушення інкапсуляції, дозволяючи відкотитися назад.

9. Нарисуйте структуру шаблону «Знімок».



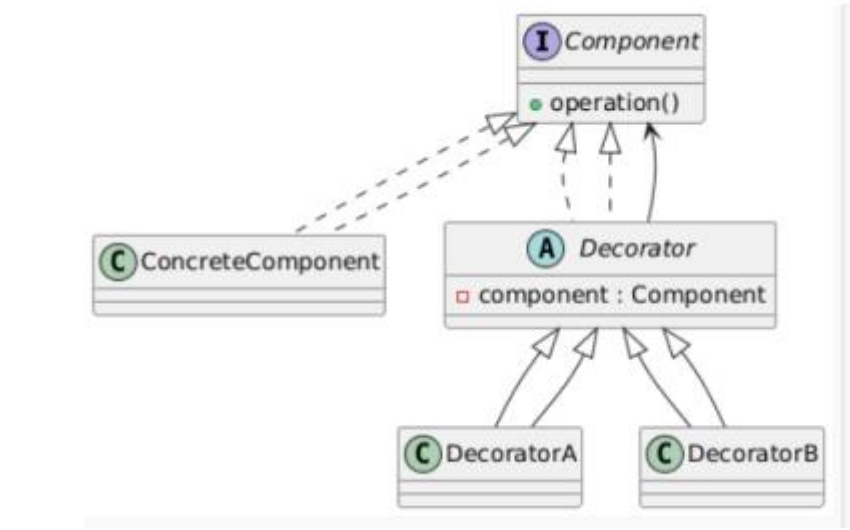
10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

- Originator - створює та відновлює знімки
- Memento - зберігає стан
- Caretaker - керує історією станів

11. Яке призначення шаблону «Декоратор»?

Додає нову функціональність об'єктам без зміни їхнього коду та без наслідування.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

- Component -базовий інтерфейс
- ConcreteComponent - основний об'єкт
- Decorator - містить посилання на Component
- ConcreteDecorator - додають поведінку

14. Які є обмеження використання шаблону «декоратор»?

- Багато декораторів → складна структура
- Важко дебажити через ланцюг обгорток
- Не підходить, якщо треба змінювати поведінку всім об'єктам одразу