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 № 4

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

Вступ до паттернів проектування.

Виконав:

Студент групи ІА-31

Шереметьєв Дмитро

Київ – 2025

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

Тема: Вступ до паттернів проектування.

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» 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).

Теоретичні відомості

Стратегія (Strategy) — це поведінковий патерн проєктування, який визначає сімейство алгоритмів, інкапсулює кожен з них і робить їх взаємозамінними. Цей патерн дозволяє змінювати алгоритми незалежно від клієнтів, які їх використовують.

У проєкті MindApp цей патерн використано для малювання ліній (зв'язків) між вузлами. Користувач може динамічно перемикати вигляд ліній (прямі або вигнуті), не змінюючи логіку самого редактора.

Реалізація класів

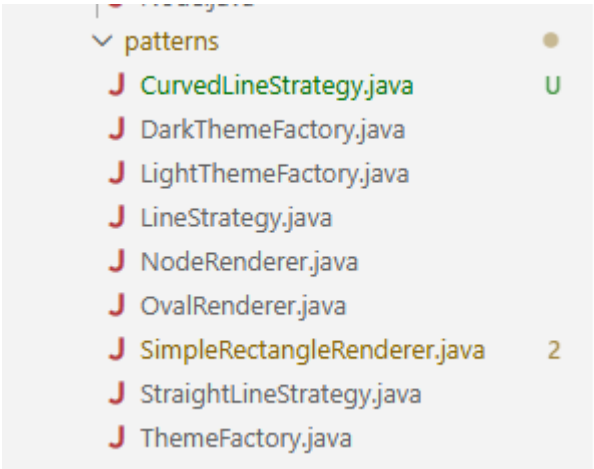


Рисунок 1 Структура проекта

Було створено та модифіковано наступні класи:

1. **LineStrategy (Інтерфейс)** Визначає спільний інтерфейс для всіх алгоритмів малювання ліній.

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

import com.mindapp.client.models.Node;

import javafx.scene.canvas.GraphicsContext;

You, 2 days ago | 1 author (You)
public interface LineStrategy {
    // Перевірте, чи є цей метод і чи всі аргументи на місці
    void drawLine(GraphicsContext gc, Node parent, Node child, NodeRenderer renderer);
}
```

2. **StraightLineStrategy (Конкретна стратегія)** Реалізує малювання класичних прямих ліній між центрами вузлів.

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

import com.mindapp.client.models.Node;

import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

You, 2 days ago | 1 author (You)
// Патерн Strategy: Конкретна стратегія (Пряма лінія)
You, 2 days ago | 1 author (You)
public class StraightLineStrategy implements LineStrategy {
    @Override
    public void drawLine(GraphicsContext gc, Node parent, Node child, NodeRenderer renderer) {
        double startX = parent.getX() + renderer.getWidth(parent) / 2;
        double startY = parent.getY() + renderer.getHeight(parent) / 2;

        double endX = child.getX() + renderer.getWidth(child) / 2;
        double endY = child.getY() + renderer.getHeight(child) / 2;

        gc.setStroke(Color.BLACK);
        gc.setLineWidth(lw: 1);
        gc.strokeLine(startX, startY, endX, endY);
    }
}       You, 2 days ago • init commit, 1-3 lab
```

3. **CurvedLineStrategy (Конкретна стратегія)** Реалізує малювання плавних ліній (кривих Безьє), що покращує візуальне сприйняття складних карт.

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

import com.mindapp.client.models.Node;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

// [Concrete Strategy B]
public class CurvedLineStrategy implements LineStrategy {
    @Override
    public void drawLine(GraphicsContext gc, Node parent, Node child, NodeRenderer renderer) {
        // Координати центрів вузлів
        double startX = parent.getX() + renderer.getWidth(parent) / 2;
        double startY = parent.getY() + renderer.getHeight(parent) / 2;

        double endX = child.getX() + renderer.getWidth(child) / 2;
        double endY = child.getY() + renderer.getHeight(child) / 2;

        gc.setStroke(Color.DARKGRAY);
        gc.setLineWidth(lw: 2);

        // Малюємо криву Безьє
        gc.beginPath();
        gc.moveTo(startX, startY);

        // Контрольні точки для вигину (створюємо ефект "дерева")
        // Вигин залежить від відстані по Y
        double controlY = startY + (endY - startY) / 2;

        // curveTo(controlX1, controlY1, controlX2, controlY2, endX, endY)
        gc.bezierCurveTo(startX, controlY, endX, controlY, endX, endY);

        gc.stroke();
        gc.closePath();
    }
}
```

4. **EditorForm (Контекст)** Виступає в ролі клієнта/контексту. Зберігає посилання на поточну стратегію (private LineStrategy lineStrategy) і делегує їй роботу при відмальовуванні дерева. Дозволяє змінювати стратегію під час виконання програми (Runtime).

```java
private void toggleLineStrategy() {
if (lineStrategy instanceof StraightLineStrategy) {
    lineStrategy = new CurvedLineStrategy(); // Перемикаємо на криві
} else {
    lineStrategy = new StraightLineStrategy(); // Перемикаємо на прямі
}
draw(); // Перемальовуємо полотно
    }
```

```java
// Кнопка перемикання стратегії ліній (Strategy Pattern Demo)
    Button btnLineStyle = new Button(text: "〰 Лінії");
    btnLineStyle.setOnAction(e -> toggleLineStrategy());
```

```
ToolBar toolbar = new ToolBar(
    new Label(text: "Назва:"), titleField, btnSave,
    new Separator(),
    btnAddChild, btnAddImg, btnAddVid, btnUrgent, btnArea,
    new Separator(),
    btnExport,
    btnTheme,
    btnLineStyle
);
```

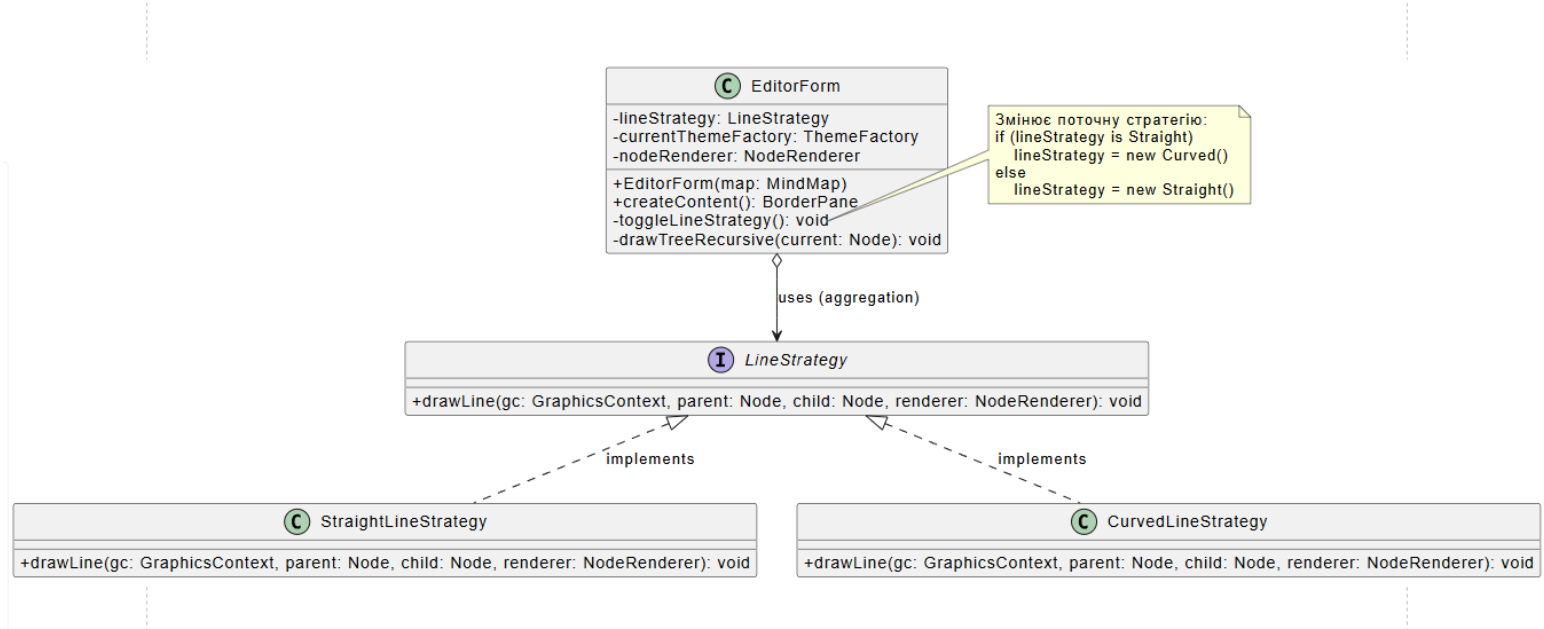


Рисунок 2 Діаграма класів

На діаграмі класів зображено структуру реалізації патерну **Strategy (Стратегія)** для зміни алгоритму малювання ліній зв'язку між вузлами ментальної карти.

Основні елементи діаграми:

1. EditorForm (Context — Контекст) Клас, що виступає в ролі контексту. Він відповідає за візуалізацію редактора та керування процесом малювання.

- **Атрибути:** Містить приватне поле lineStrategy типу LineStrategy, яке зберігає посилання на поточний алгоритм малювання ліній .
- **Методи:**
    - toggleLineStrategy(): метод, який динамічно змінює об'єкт стратегії в полі lineStrategy (перемикає між прямими та кривими лініями).

- drawTreeRecursive(): метод, який делегує виконання малювання лінії обраній стратегії, викликаючи її метод drawLine.

2. LineStrategy (Strategy — Стратегія) Загальний інтерфейс для всіх варіацій алгоритмів малювання ліній.

- **Методи:** Оголошує абстрактний метод drawLine(GraphicsContext gc, Node parent, Node child, NodeRenderer renderer), який повинні реалізувати всі конкретні стратегії .

3. StraightLineStrategy (ConcreteStrategy — Конкретна стратегія) Клас, що реалізує інтерфейс LineStrategy.

- **Реалізація:** У методі drawLine реалізує алгоритм малювання класичної **прямої лінії** між центрами батьківського та дочірнього вузлів .

4. CurvedLineStrategy (ConcreteStrategy — Конкретна стратегія) Клас, що реалізує інтерфейс LineStrategy.

- **Реалізація:** У методі drawLine реалізує алгоритм малювання **кривої Безьє**, що створює плавний, природний вигин лінії зв'язку .

Взаємодія між класами: Клас EditorForm (Клієнт) не знає деталей реалізації алгоритмів малювання. Він взаємодіє з ними виключно через інтерфейс LineStrategy (відношення агрегації). Це дозволяє змінювати поведінку малювання ліній "на льоту" (Runtime) без необхідності змінювати код самого редактора, просто підставляючи інший об'єкт конкретної стратегії.

Відповіді на контрольні питання:

1. Що таке шаблон проєктування?

Шаблон проєктування - це типове, перевірене рішення певної задачі проєктування програмного забезпечення, яке можна повторно використовувати у різних проєктах.

2. Навіщо використовувати шаблони проєктування?

Вони допомагають уникати типових помилок, спрощують розробку, покращують читабельність і підтримку коду, а також забезпечують гнучкість і повторне використання рішень.

3. Яке призначення шаблону «Стратегія»?

Шаблон «Стратегія» дозволяє визначити сімейство алгоритмів, інкапсулювати кожен з них і робити їх взаємозамінними без зміни клієнтського коду.

4. Нарисуйте структуру шаблону «Стратегія».

5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

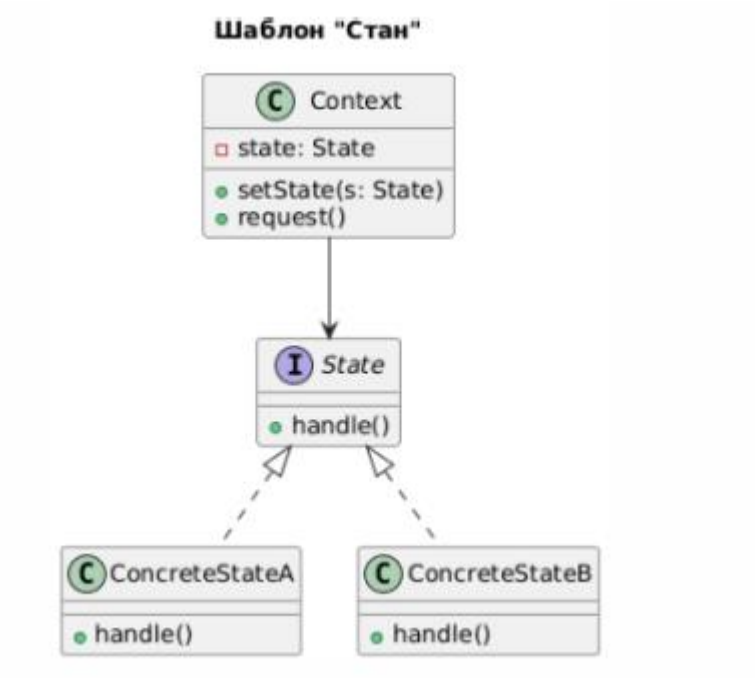
- Context - зберігає посилання на об’єкт стратегії.
- Strategy (інтерфейс) - оголошує метод, який реалізують стратегії.
- ConcreteStrategy - конкретні реалізації алгоритму.

Взаємодія: Context викликає метод Strategy, не знаючи, який саме алгоритм використовується.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» дозволяє об’єкту змінювати свою поведінку при зміні внутрішнього стану, створюючи ілюзію зміни його класу.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

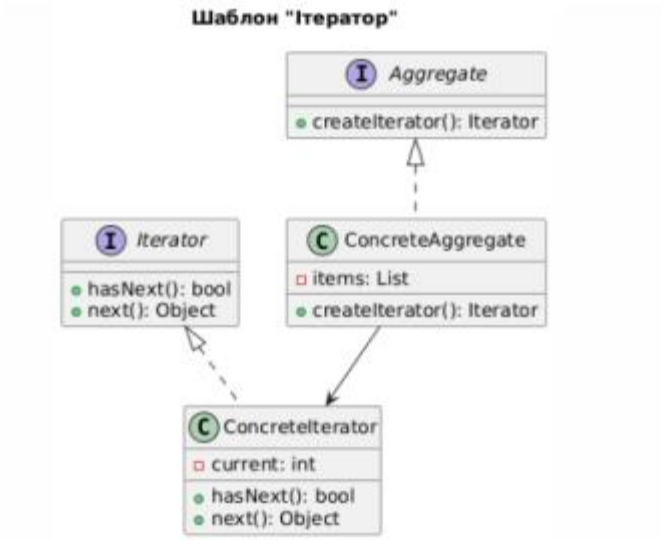
- Context - зберігає поточний стан і делегує йому поведінку.
- State (інтерфейс) - визначає інтерфейс для станів.
- ConcreteState - реалізує конкретні варіанти поведінки.

Взаємодія: Context викликає метод поточного стану, а стан може змінювати Context на інший стан.

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» дає змогу послідовно отримувати доступ до елементів колекції без розкриття її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

- Iterator (інтерфейс) - визначає інтерфейс для обходу елементів.
- ConcreteIterator - реалізує цей інтерфейс для конкретної колекції.
- Aggregate (інтерфейс) - створює об’єкт ітератора.
- ConcreteAggregate - реалізує Aggregate і зберігає колекцію.

Взаємодія: Клієнт отримує ітератор від колекції та використовує його для послідовного доступу до елементів.

12. В чому полягає ідея шаблона «Одинак»?

Ідея полягає в тому, щоб клас мав лише один екземпляр (instance) у програмі та забезпечував глобальну точку доступу до нього.

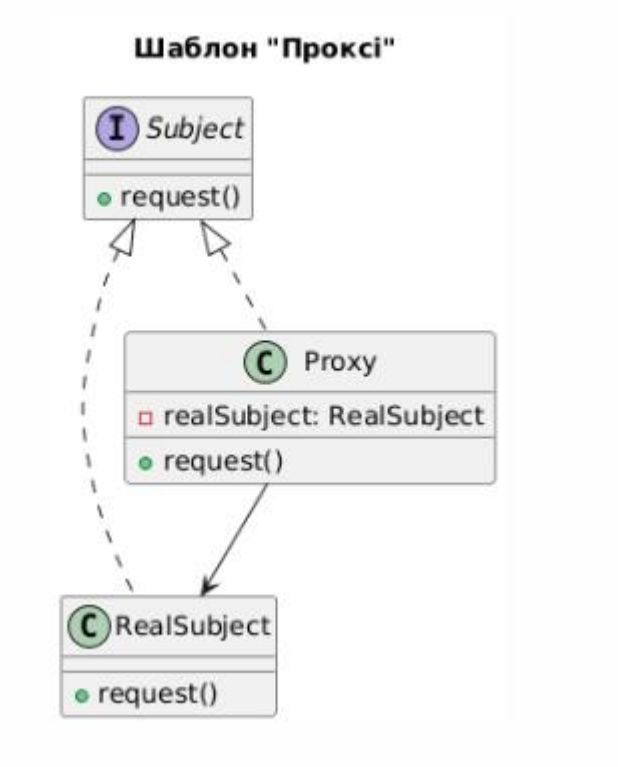
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Тому що він порушує принципи чистої архітектури — створює глобальний стан, ускладнює тестування, приховано зв’язує частини програми та знижує гнучкість.

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» використовується для контролю доступу до об’єкта, наприклад, для відкладеного створення, кешування або безпеки.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

- Subject (інтерфейс) - оголошує спільний інтерфейс для Proxy і RealSubject.
- RealSubject - реальний об’єкт, до якого здійснюється доступ.
- Proxy - зберігає посилання на RealSubject і контролює звернення до нього.

Взаємодія: Client викликає метод Proxy, Proxy вирішує — передати запит

RealSubject чи виконати додаткову логіку.