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 № 5

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 «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» 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 процесі виконання лабораторної роботи було реалізовано механізм **"Копіювати/Вставити" (Copy/Paste)** для елементів ментальної карти з використанням патерну **Prototype**. Це дозволяє створювати точні копії вузлів (разом із кольором, категорією, вкладеннями та дочірніми елементами) без залежності від їхнього класу.

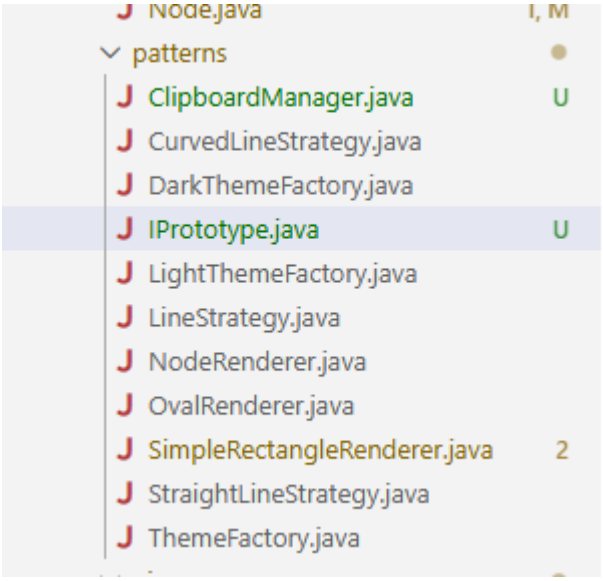


Рисунок 1 Структура проекту

Було створено та модифіковано наступні класи:

1. IPrototype (Інтерфейс) Базовий інтерфейс патерну. Він визначає контракт для всіх об'єктів, які підтримують клонування.

- **Метод:** clone(): IPrototype — повертає копію об'єкта.

2. Node (Concrete Prototype — Конкретний прототип) Клас, що представляє вузол ментальної карти. Він реалізує інтерфейс IPrototype.

- **Реалізація:** Метод clone() створює глибоку копію вузла. Він копіює всі атрибути (текст, координати, стиль) та рекурсивно клонує список дочірніх

вузлів (children), щоб скопіювати всю гілку дерева, а не лише вершину. ID при цьому скидається (null), оскільки копія є новим записом у базі даних.

3. ClipboardManager (Client / Manager — Менеджер буфера) Клас, що виконує роль клієнта/менеджера у патерні. Він відповідає за збереження прототипу та створення його копій за запитом.

- **Поля:** buffer — статичне поле для зберігання скопійованого об'єкта IPrototype.
- **Методи:**
  - copy(item): зберігає клон переданого об'єкта у буфер.
  - paste(): повертає новий клон об'єкта з буфера (дозволяє багаторазову вставку).
  - hasContent(): перевіряє, чи є щось у буфері (для активації кнопки в меню).

4. EditorForm (UI Integration) Клас інтерфейсу редактора. У контекстному меню додано пункти "Копіювати" та "Вставити", які звертаються до ClipboardManager для виконання відповідних дій над виділеним вузлом (selectedNode).

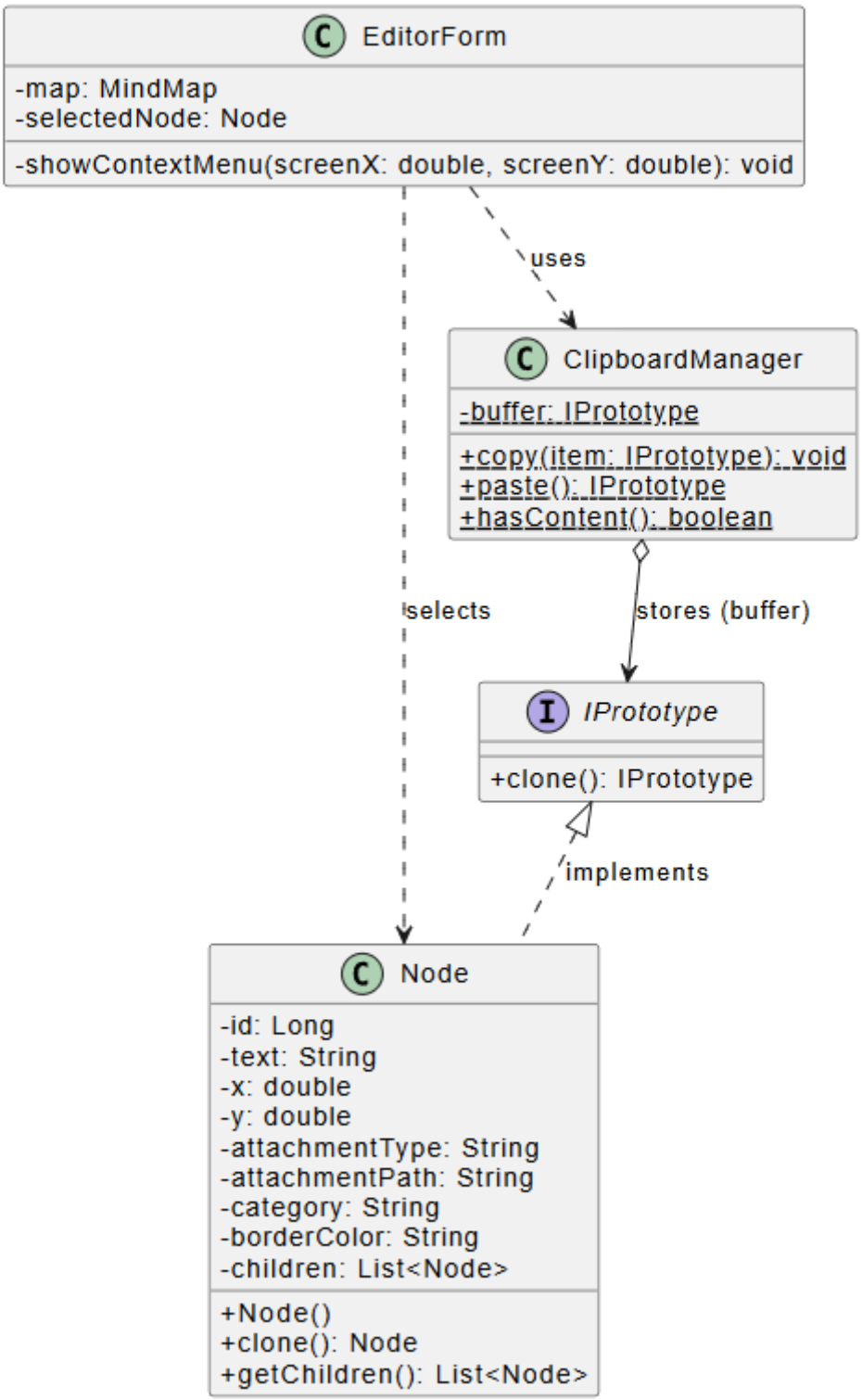


Рисунок 2 Діаграма класів

Реалізація

У рамках проєкту патерн **Prototype** було реалізовано для забезпечення механізму копіювання та вставки (Copy/Paste) елементів ментальної карти. Основна ідея

полягала у створенні інтерфейсу для клонування об'єктів та менеджеру буфера обміну, який керує цим процесом.

- **Інтерфейс (IPrototype)** визначає єдиний контракт для всіх об'єктів системи, які можуть бути скопійовані. Він містить метод clone(), що гарантує отримання точної копії об'єкта.
- **Конкретний прототип (Node)** реалізує цей інтерфейс. У методі клонування закладено логіку глибокого копіювання: створюється новий екземпляр вузла, копіюються всі його візуальні та змістові атрибути (колір, текст, вкладення), а також рекурсивно клонується весь список дочірніх вузлів. Це дозволяє копіювати не просто один елемент, а цілі гілки діаграми.
- **Менеджер (ClipboardManager)** виступає в ролі клієнта, який керує життєвим циклом копії. Він зберігає посилання на клонований об'єкт у буфері та надає його нові копії за запитом "Вставити". Це ізолює логіку буфера обміну від решти програми.
- **Клієнтський код (EditorForm)** ініціює процес через контекстне меню, звертаючись до менеджера, і не залежить від деталей створення нових об'єктів.

Проблеми, які вирішує патерн

- **Інкапсуляція логіки створення копій.** Клієнтський код (форма редактора) не повинен знати, які саме поля має вузол Node і як правильно скопіювати вкладені списки дітей. Об'єкт сам відповідає за своє коректне клонування.
- **Копіювання складних ієрархічних структур.** Вузли ментальної карти утворюють деревоподібну структуру. Створення копії гілки вручну вимагало б складного рекурсивного коду в контролері UI. Патерн Prototype приховує цю складність всередині методу clone() самого вузла.
- **Незалежність від конкретних класів.** Система буфера обміну (ClipboardManager) працює з абстракцією IPrototype. Це означає, що в майбутньому можна буде додати можливість копіювання інших типів об'єктів (наприклад, зв'язків або груп), не змінюючи код менеджера.
- **Ізоляція станів.** При копіюванні в буфер створюється новий об'єкт (клон), а не посилання на існуючий. Це запобігає ситуації, коли зміна оригінального вузла після копіювання впливає на те, що буде вставлено з буфера.

Переваги використання

- **Гнучкість:** Дозволяє додавати та видаляти об'єкти під час виконання програми (Runtime) шляхом клонування прототипів, що значно спрощує код ініціалізації нових об'єктів зі складними налаштуваннями.
- **Зменшення кількості підкласів:** Замість створення фабрик для кожного варіанту об'єкта, ми можемо мати набір попередньо налаштованих прототипів і клонувати їх.
- **Продуктивність:** Клонування об'єкта часто є ефективнішим за його створення через конструктор, особливо якщо ініціалізація вимагає складних обчислень або завантаження ресурсів (наприклад, налаштування стилів).

Висновок

У ході виконання лабораторної роботи було успішно реалізовано патерн "Прототип". Це дозволило додати у редактор функцію буфера обміну. Реалізація через інтерфейс IPrototype робить систему гнучкою: у майбутньому можна буде легко додати можливість копіювання інших елементів системи, просто реалізувавши цей інтерфейс.

Код

IPrototype.java

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

public interface IPrototype {
    IPrototype clone();
}
```

ClipboardManager.java

client > demo > src > main > java > com > mindapp > client > patterns > ClipboardManager.java > Java > Clipboard

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

public class ClipboardManager {
    // Зберігаємо скопійований об'єкт
    private static IPrototype buffer = null;

    // Метод копіювання (зберігає клон, щоб відв'язатися від оригіналу)
    public static void copy(IPrototype item) {
        if (item != null) {
            buffer = item.clone();
            System.out.println("Copied to clipboard: " + item);
        }
    }

    // Метод вставки (повертає новий клон з буфера)
    public static IPrototype paste() {
        if (buffer != null) {
            return buffer.clone();
        }
        return null;
    }

    // Перевірка, чи є щось у буфері (для активації кнопки "Вставити")
    public static boolean hasContent() {
        return buffer != null;
    }
}
```

Node.java

```
You, 17 minutes ago | 1 author (You)
1  package com.mindapp.client.models;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import com.mindapp.client.patterns.IPrototype;
7
   You, 17 minutes ago | 1 author (You)
8  public class Node implements Cloneable, IPrototype {
9      private Long id;
10     private String text;
11     private double x;
12     private double y;
13
14     // Вкладення та категорії
15     private String attachmentType = "NONE"; // "IMAGE", "VIDEO", "FILE"
16     private String attachmentPath;
17     private String category = "NORMAL";
18
19     private String borderColor;
20     // ------------------------------
21
22     private List<Node> children = new ArrayList<>();
23
24     public Node() {}
25
26     public Node(String text, double x, double y) {
27         this.text = text;
28         this.x = x;
29         this.y = y;
30     }
31
32     @Override
33     public Node clone() {
34         try {
```

```
33      public Node clone() {
36          cloned.setId(id: null); // Скидаємо ID, бо це новий об'єкт
37
38          // Копіювання полів (String і так immutable, тому просто присвоюємо)
39          cloned.setText(this.getText());
40          cloned.setX(this.getX());
41          cloned.setY(this.getY());
42          cloned.setAttachmentType(this.getAttachmentType());
43          cloned.setAttachmentPath(this.getAttachmentPath());
44          cloned.setCategory(this.getCategory());
45          cloned.setBorderColor(this.getBorderColor());
46
47          // Глибоке копіювання дітей (щоб скопіювати всю гілку)
48          List<Node> clonedChildren = new ArrayList<>();
49          if (this.getChildren() != null) {
50              for (Node child : this.getChildren()) {
51                  clonedChildren.add(child.clone()); // Рекурсивний виклик
52              }
53          }
54          cloned.setChildren(clonedChildren);
55
56          return cloned;
57      } catch (CloneNotSupportedException e) {
58          throw new RuntimeException(message: "Cloning not supported", e);
59      }
60  }
61
62  // Гетери та Сетери       You, 2 days ago • init commit, 1-3 lab
63  public Long getId() { return id; }
64  public void setId(Long id) { this.id = id; }
65
66  public String getText() { return text; }
67  public void setText(String text) { this.text = text; }
68
69  public double getX() { return x; }
70  public void setX(double x) { this.x = x; }
71
72  public double getY() { return y; }
73  public void setY(double y) { this.y = y; }
74
75  public String getAttachmentType() { return attachmentType; }
76  public void setAttachmentType(String attachmentType) { this.attachmentType = attachmentType; }
77
78  public String getAttachmentPath() { return attachmentPath; }
79  public void setAttachmentPath(String attachmentPath) { this.attachmentPath = attachmentPath; }
80
81  public String getCategory() { return category; }
```

EditorForm.java

```
        itemAddChild.setOnAction(e -> addChildNode());

        MenuItem itemCopy = new MenuItem(text: "📄 Копіювати");
        itemCopy.setOnAction(e -> {
            if (selectedNode != null) {
                ClipboardManager.copy(selectedNode);
            }
        });
        MenuItem itemPaste = new MenuItem(text: "📋 Вставити");
        // Робимо кнопку активною, тільки якщо в буфері щось є
        itemPaste.setDisable(!ClipboardManager.hasContent());
        itemPaste.setOnAction(e -> {
            if (selectedNode != null) {
                IPrototype pastedItem = ClipboardManager.paste();
                if (pastedItem instanceof Node) {
                    Node newNode = (Node) pastedItem;
                    // Трохи зсуваємо, щоб було видно, що це новий об'єкт
                    newNode.setX(selectedNode.getX() + 50);
                    newNode.setY(selectedNode.getY() + 50);

                    selectedNode.getChildren().add(newNode);
                    draw(); // Оновлюємо малюнок
                }
            }
        });
```

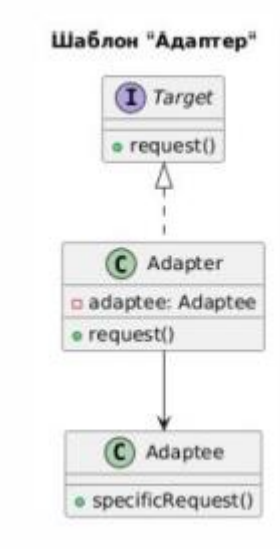
```
    itemClear.setOnAction(e -> clearAttachment());

    menuAttach.getItems().addAll(itemImg, itemVid, itemFile, new SeparatorMenuItem(), itemClear);
You, 2 days ago • init commit, 1-3 lab
    menu.getItems().addAll(itemEdit, itemAddChild, new SeparatorMenuItem(), itemCopy, itemPaste, itemImportant, itemArea, menuAttach,
            new SeparatorMenuItem(), itemDelete);
    menu.show(canvas, screenX, screenY);
}
```

Відповіді на контрольні питання:

1. Яке призначення шаблону «Адаптер»?

Шаблон «Адаптер» дозволяє об’єктам з несумісними інтерфейсами працювати разом, “обгортаючи” один клас іншим, щоб привести інтерфейс до потрібного виду.

2. Нарисуйте структуру шаблону «Адаптер».



3. Які класи входять в шаблон «Адаптер», та яка між ними взаємодія?

- Target - інтерфейс, який очікує клієнт.
- Adaptee - існуючий клас з іншим інтерфейсом.
- Adapter - “перекладач”, що перетворює виклики Target у виклики Adaptee.

Взаємодія: Клієнт працює з Adapter через інтерфейс Target, а Adapter викликає методи Adaptee.

4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

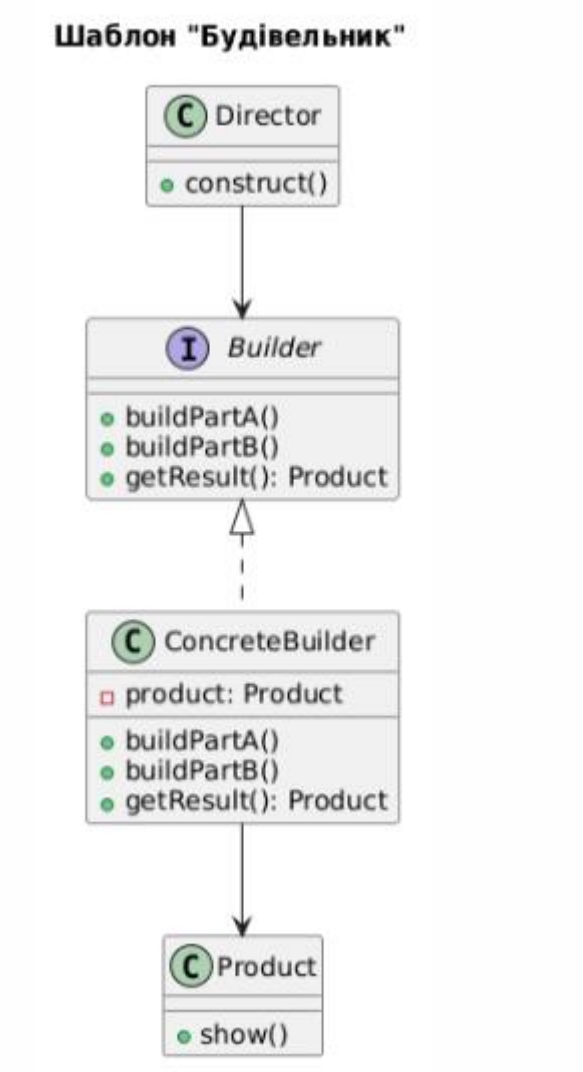
- Об’єктний адаптер - використовує композицію (Adapter має посилання на 13
- Класовий адаптер - використовує наслідування (Adapter наслідує Adaptee).

Об’єктний підхід більш гнучкий, класовий - простіший, але менш універсальний.

5. Яке призначення шаблону «Будівельник»?

Шаблон «Будівельник» відокремлює процес створення складного об’єкта від його представлення, щоб один і той самий процес міг створювати різні об’єкти.

6. Нарисуйте структуру шаблону «Будівельник».



7. Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

- Director - керує порядком виклику методів будівельника.
- Builder - інтерфейс для створення частин об’єкта.
- ConcreteBuilder - реалізує побудову частин конкретного продукту.
- Product - кінцевий об’єкт.

Взаємодія: Director викликає методи Builder, а ConcreteBuilder поступово створює Product.

8. У яких випадках варто застосовувати шаблон «Будівельник»?

- Коли потрібно створювати складні об’єкти поетапно.
- Коли процес створення повинен бути незалежним від деталей

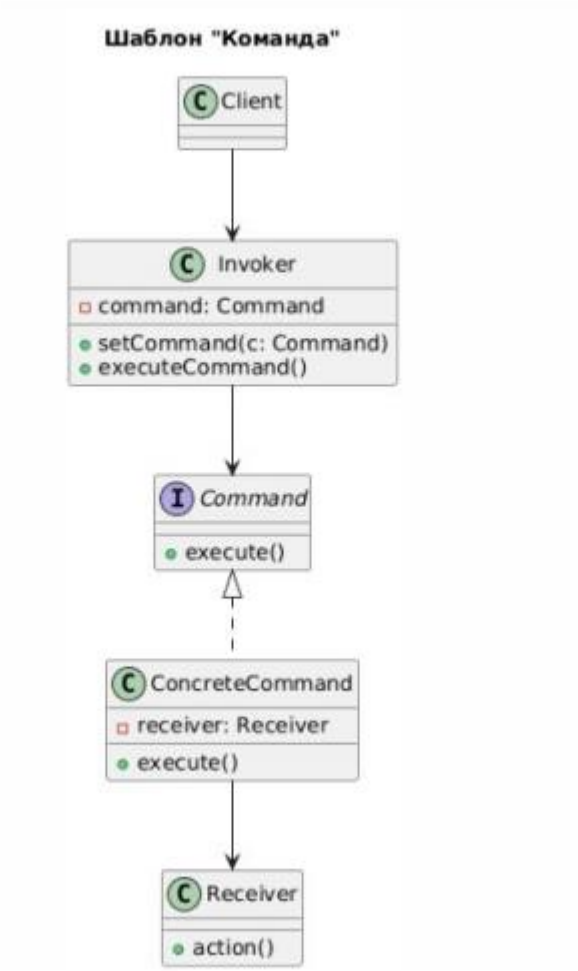
представлення.

• Коли потрібна можливість створення різних варіантів об’єкта одним способом.

9. Яке призначення шаблону «Команда»?

Шаблон «Команда» інкапсулює запит у вигляді об’єкта, дозволяючи передавати, зберігати, виконувати або скасовувати дії.

10. Нарисуйте структуру шаблону «Команда».



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

• Command - інтерфейс із методом execute().

• ConcreteCommand - реалізує команду, викликаючи методи Receiver.

- Receiver - об’єкт, який виконує фактичну дію.
- Invoker - зберігає команду і викликає її.
- Client - створює конкретні команди.

Взаємодія: Client створює команду, Invoker її викликає, Receiver виконує дію.

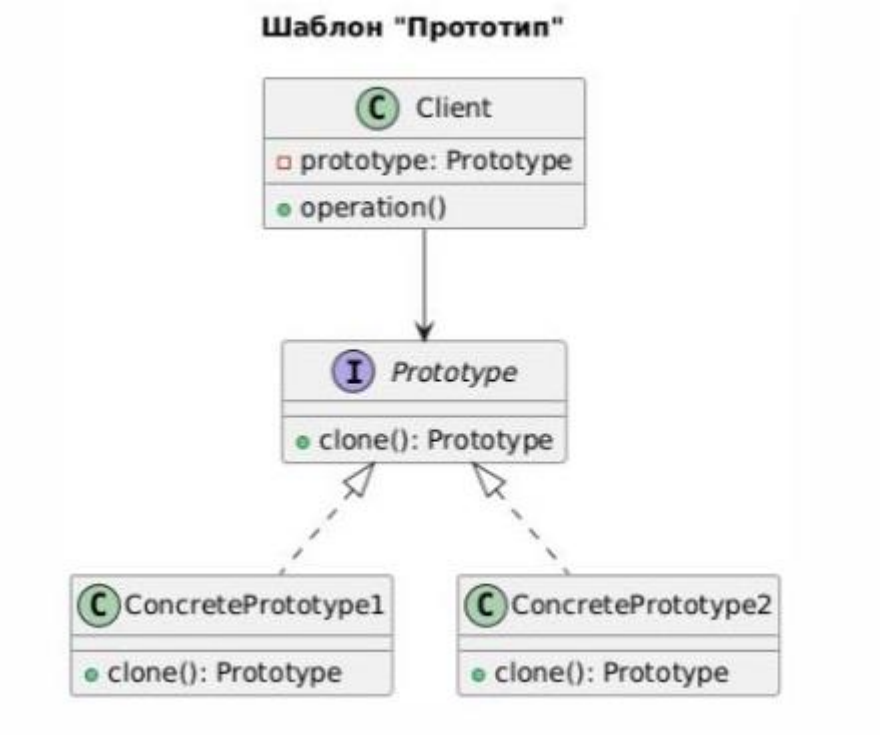
12. Розкажіть як працює шаблон «Команда».

Кожна дія програми представлена як об’єкт-команда. Коли користувач або 16 система хоче виконати дію, Invoker викликає метод execute() у команді, а вона - виконує потрібну операцію через Receiver. Це дозволяє легко додавати нові дії, скасовувати або повторювати команди.

13. Яке призначення шаблону «Прототип»?

Шаблон «Прототип» дозволяє створювати нові об’єкти шляхом копіювання вже існуючого екземпляра (прототипу), замість створення через конструктор.

14. Нарисуйте структуру шаблону «Прототип».



15. Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

- Prototype - інтерфейс із методом clone().

- ConcretePrototype - реалізує копіювання самого себе.
- Client - створює нові об’єкти через клонування прототипу.

Взаємодія: Client зберігає посилання на Prototype і створює нові об’єкти методом clone().. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?

- Обробка HTTP-запитів (middleware у вебфреймворках).
- Система техпідтримки (запит проходить від оператора до менеджера).
- Система логування (повідомлення передаються кільком логерам).
- GUI-події - натискання кнопки може оброблятись компонентом або передаватись вгору по ієрархії.