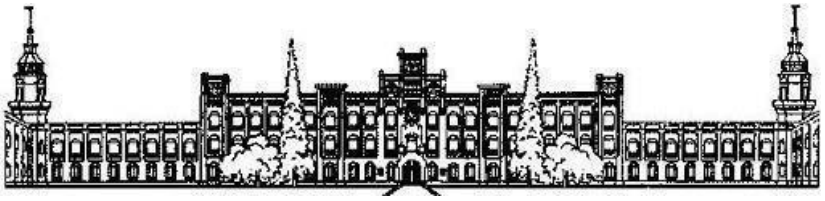


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №5

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування.»

Виконав:
студент групи ІА-32
Діденко Я.О

Перевірив:
Мягкий М.Ю

Київ 2025

Тема: Патерни проектування.
Мета: Вивчити структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи.

18. **Shell (total commander)** (state, prototype, factory method, template method, interpreter, client-server)

Оболонка повинна вміти виконувати основні дії в системі – перегляд файлів папок в файлової системі, перемикання між дисками, копіювання, видалення, переміщення об'єктів, пошук.

Теоретичні Відомості

Поняття шаблону проєктування

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях. Крім того, патерн проєктування обов'язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову. Варто підкреслити, що важливим початковим етапом при роботі з патернами є адекватне моделювання розглянутої предметної області. Це є необхідним як для отримання належним чином формалізованої постановки задачі, так і для вибору відповідних патернів проєктування.
Відповідне використання патернів проєктування дає розробнику ряд незаперечних переваг. Наведемо деякі з них. Модель системи, побудована в межах патернів проєктування, фактично є структурованим виокремленням тих елементів і зв'язків, які значимі при вирішенні поставленого завдання. Крім цього, модель, побудована з використанням патернів проєктування, більш проста і наочна у вивченні, ніж стандартна модель. Проте, не дивлячись на простоту і наочність, вона дозволяє глибоко і всебічно опрацювати архітектуру розроблюваної системи з використанням спеціальної мови.
Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи. Крім того, важко переоцінити роль використання патернів при інтеграції інформаційних систем організації. Також слід зазначити, що сукупність патернів проєктування, по суті, являє собою єдиний словник проєктування, який, будучи уніфікованим засобом, незамінний для спілкування розробників один одним.
Таким чином шаблони представляють собою, підтверджені роками розробок в різних компаніях і на різних проєктах, «ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах.

Шаблон «Prototype»

Призначення: Шаблон «Prototype» (Прототип) використовується для створення об'єктів за «шаблоном» (чи «кресленням», «ескізом») шляхом копіювання шаблонного об'єкту, який називається прототипом. Для цього визначається метод «клонувати» в об'єктах цього класу.

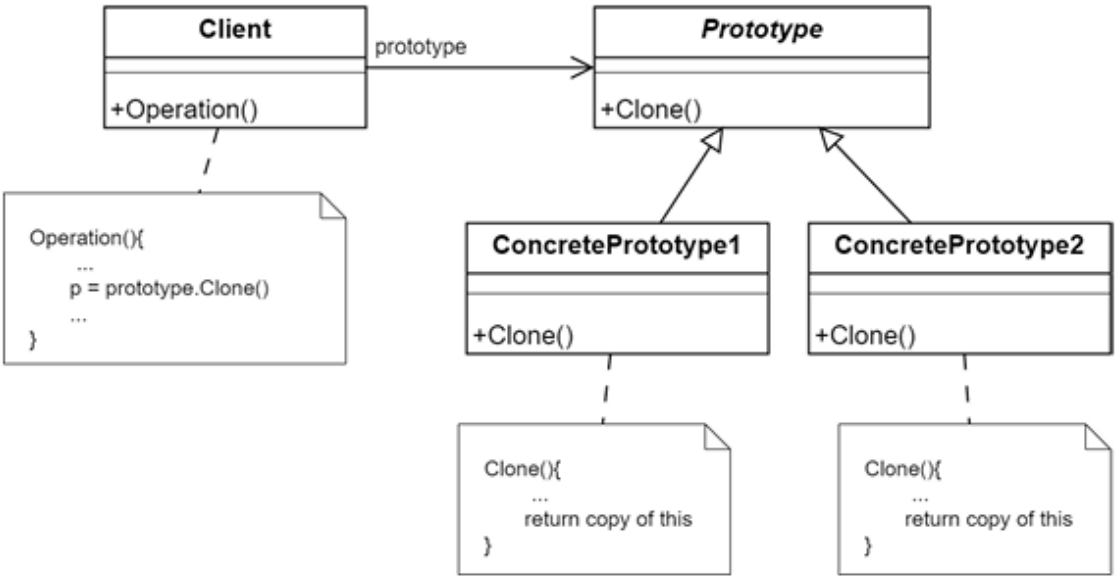


рис 1. Структура патерну Prototype

Цей шаблон зручно використати, коли заздалегідь відомо як виглядатиме кінцевий об'єкт (мінімізується кількість змін до об'єкту шляхом створення шаблону), а також для видалення необхідності створення об'єкту - створення відбувається за рахунок клонування, і самій програмі немає необхідності знати, як створювати об'єкт. Також, це дозволяє маніпулювати об'єктами під час виконання програми шляхом настроювання відповідних прототипів; значно зменшується ієрархія наслідування (оскільки в іншому випадку це були б не прототипи, а вкладені класи, що наслідують).

Проблема: Ви розробляєте редактор рівнів для 2D гри на основі спрайтів. В панелі інструментів ви маєте багато кнопок для різних елементів, які можна розташовувати на екрані, такі як сходи, стіни, підлога, оздоблення та інші. Ці елементи у вас об'єднані в ієрархію з базовим класом GameObject. Під кожен елемент можна зробити свій тип кнопки, але тоді ми отримаємо паралельну ієрархію кнопок і при додаванні нового типу ігрового об’єкту потрібно буде додавати і новий тип кнопки.

Рішення: Використовуючи патерн прототип, додаємо до базового об'єкта GameObject метод Clone(), а кнопки будуть зберігати посилання на об'єкт базового типу GameObject. При натисканні на кнопку об'єкт який потрібно добавити на ігровому полі отримуємо не створенням нового, а клонуванням прототипу, який прив'язаний до кнопки. Таким чином, коли ми будемо додавати 66 нові типи ігрових об'єктів, то логіка роботи з кнопками не буде змінюватися, тому що не має прив'язки до конкретних типів.

Також слід відзначити, що при копіюванні об'єкту, навіть приватні поля будуть скопійовані, тому що реалізація методу копіювання знаходиться в цьому класі, що копіюється і таким чином є доступ для копіювання і до відкритих і до закритих полів.

Переваги:
+ За рахунок клонування складних об’єктів замість їх створення, підвищується продуктивність.
+ Різні варіації об’єктів можна отримувати за рахунок клонування, а не розширення ієрархії класів.
+ Вища гнучкість, тому що клоновані об’єкти можна модифікувати незалежно, не впливаючи на об’єкт з якого була зроблена копія.

Недоліки:
- Реалізація глибокого клонування досить проблематична, коли об’єкт, що клонується, містить складну внутрішню структуру та посилання на інші об’єкти.
- Надмірне використання патерну Прототип може привести до ускладнення коду та проблем із супроводом такого коду.

Хід роботи

1. Загальний опис виконаної роботи

В рамках даної лабораторної роботи було реалізовано шаблон проектування **«Прототип» (Prototype)**.

Основна мета полягала в тому, щоб створити гнучку систему створення об'єктів шляхом копіювання (клонування) вже існуючих екземплярів, які

називаються прототипами. Такий підхід дозволяє уникнути прямої залежності клієнтського коду від класів об'єктів, що створюються, та значно спрощує процес ініціалізації складних, багаторівневих структур.

Програмна система складається з наступних ключових компонентів:

- **Інтерфейс прототипу (FileSystemObject та Cloneable):** Абстрактний клас FileSystemObject, що реалізує стандартний інтерфейс Cloneable. Він оголошує метод clone(), який повинні реалізувати всі конкретні класи-прототипи, та визначає загальну поведінку для всіх об'єктів файлової системи.
- **Конкретні прототипи (File, Directory):** Класи, що реалізують логіку клонування.
    - File реалізує **поверхневе копіювання**, оскільки не містить посилань на інші складні об'єкти.
    - Directory реалізує **глибоке копіювання**, рекурсивно клонуючи всі дочірні елементи. Це гарантує, що копія теки є повністю незалежною від оригіналу.
- **Клієнт / Реєстр Прототипів (FileManager):** Клас, який відповідає за керування прототипами. Він зберігає набір шаблонів (прототипів) і на запит клієнта створює нові об'єкти, клонуючи відповідний шаблон. Shell виступає кінцевим клієнтом, який ініціює створення об'єктів через FileManager.

Процес створення нових файлів чи тек делегується самим об'єктам-прототипам. Замість того, щоб конструювати новий об'єкт з нуля (new Directory(...)), клієнт просить FileManager клонувати вже існуючий шаблон, що робить систему гнучкою та легко розширюваною.

2. Опис класів програмної системи.

Клас FileSystemObject.java

Цей абстрактний клас є **базовим прототипом** і служить спільною основою для всіх об'єктів у віртуальній файловій системі (File та Directory). Він реалізує маркерний інтерфейс Cloneable і перевизначає метод clone(), що є центральним елементом шаблону «Прототип».

Ключові обов'язки класу:

- **Визначення спільного стану:** Інкапсулює загальні для всіх об'єктів поля, такі як name та path.
- **Оголошення спільного інтерфейсу:** Визначає набір методів, які повинні мати всі нащадки. Метод display(int depth) є абстрактним, що змушує дочірні класи надавати власну реалізацію для візуалізації.
- **Надання базової функціональності клонування:** Його реалізація методу clone() викликає super.clone(), забезпечуючи механізм **поверхневого копіювання**, який є достатнім для простих нащадків (як File) і служить відправною точкою для складних (як Directory).

```
1   public abstract class FileSystemObject implements Cloneable {  13 usages  2 inheritors
2       protected String name;
3       protected String path;  3 usages
4
5       public FileSystemObject(String name, String path) {  2 usages
6           this.name = name;
7           this.path = path;
8       }
9
10      public void setName(String name) {  1 usage
11          this.name = name;
12      }
13
14      public void setPath(String path) {  1 usage
15          this.path = path;
16      }
17
18      public String getName() {  no usages
19          return name;
20      }
21
22      public String getPath() {  no usages
23          return path;
24      }
25
26      public abstract void display(int depth);  3 usages  2 implementations
27
28      @Override  2 overrides
29      public Object clone() throws CloneNotSupportedException {
30          return super.clone();
31      }
32  }
```

рис 2.1 - код класу FileSystemObject.java

Клас File.java

Цей клас є **конкретним прототипом**, що представляє простий об'єкт - файл. Він успадковує FileSystemObject і додає специфічну для файлу властивість size.

Ключові обов'язки класу:

- **Розширення базового стану:** Додає поле size для зберігання розміру файлу.
- **Конкретна реалізація поведінки:** Перевизначає метод display(int depth) для відображення інформації про файл, включаючи його ім'я та розмір.
- **Реалізація поверхневого клонування:** Клас успадковує реалізацію методу clone() від батьківського класу FileSystemObject. Оскільки File не містить посилань на інші об'єкти, стандартного **поверхневого копіювання** цілком достатньо для створення повної та незалежної копії.

```
1   public class File extends FileSystemObject {  6 usages
2       private long size;  2 usages
3
4       public File(String name, String path, long size) {  3 usages
5           super(name, path);
6           this.size = size;
7       }
8
9       @Override  3 usages
10      public void display(int depth) {
11          System.out.println(" ".repeat(depth) + "- " + name + " (File, " + size + " bytes)");
12      }
13
14      @Override
15      public Object clone() throws CloneNotSupportedException {
16          return super.clone();
17      }
18  }
```

рис 2.2 - код класу File.java

Клас Directory.java

Цей клас є **конкретним прототипом**, що представляє складний, композитний об'єкт - теку. Вона може містити колекцію інших об'єктів FileSystemObject (файлів та інших тек), що робить її ієрарханічною структурою.

Ключові обов'язки класу:

- **Зберігання дочірніх елементів:** Має поле children (список List<FileSystemObject>) для зберігання вкладених об'єктів.
- **Керування структурою:** Надає метод add() для додавання нових елементів до теки.
- **Рекурсивна візуалізація:** Перевизначає метод display(int depth), який не тільки відображає назву самої теки, але й рекурсивно викликає метод display() для кожного дочірнього елемента, створюючи ієрархічний вивід.
- **Реалізація глибокого клонування:** Це найважливіша відповідальність класу. Метод clone() перевизначено для виконання **глибокого копіювання**: він спочатку створює поверхневу копію самої теки, а потім рекурсивно клонує кожен елемент зі списку children і додає його до нового, окремого списку. Це гарантує, що скопійована тека є повністю незалежною від оригіналу.

```java
public class Directory extends FileSystemObject {  8 usages
    private List<FileSystemObject> children = new ArrayList<>();  4 usages

    public Directory(String name, String path) {  3 usages
        super(name, path);
    }

    public void add(FileSystemObject child) {
        children.add(child);
    }

    @Override  3 usages
    public void display(int depth) {
        System.out.println(" ".repeat(depth) + "[" + name + "] (Directory)");
        for (FileSystemObject child : children) {
            child.display( depth: depth + 2);
        }
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        Directory clone = (Directory) super.clone();

        clone.children = new ArrayList<>();

        for (FileSystemObject child : this.children) {
            clone.add((FileSystemObject) child.clone());
        }

        return clone;
    }
}
```

рис 2.3 - код інтерфейсу Directory.java

Клас FileManager.java

Цей клас реалізує патерн **Реєстр Прототипів (Prototype Registry)** і виступає в ролі централізованого менеджера шаблонів. Він приховує від клієнта (Shell) логіку вибору та клонування конкретних прототипів, надаючи простий інтерфейс для створення об'єктів за ключем.

Ключові обов'язки класу:

- **Зберігання прототипів:** Використовує Map для зберігання набору попередньо створених та налаштованих об'єктів-прототипів, асоціюючи кожен з них з унікальним рядковим ключем (наприклад, "document", "project").

- **Реєстрація прототипів:** Надає публічний метод registerTemplate(), який дозволяє клієнту додавати нові шаблони до реєстру під час виконання програми.
- **Фабрика об'єктів:** Метод createFromTemplate() діє як фабричний метод. Він знаходить потрібний прототип за ключем, викликає його метод clone() для створення копії, а потім налаштовує властивості (name, path) нового об'єкта відповідно до переданих параметрів.

```java
import java.util.HashMap;
import java.util.Map;

public class FileManager {  2 usages
    private Map<String, FileSystemObject> templates = new HashMap<>();  2 usages

    public void registerTemplate(String key, FileSystemObject prototype) { templates.put(key, prototype); }

    public FileSystemObject createFromTemplate(String key, String newPath, String newName)  2 usages
            throws CloneNotSupportedException {
        FileSystemObject prototype = templates.get(key);
        if (prototype == null) {
            throw new IllegalArgumentException("Template with key '" + key + "' not found.");
        }

        FileSystemObject copy = (FileSystemObject) prototype.clone();
        copy.setPath(newPath);
        copy.setName(newName);
        return copy;
    }
}
```

рис 2.4 - код інтерфейсу FileManager.java

Клас Shell.java

Цей клас є **клієнтом** і служить точкою входу в програму (main). Його основна мета - продемонструвати практичне застосування патерну «Прототип» та роботу FileManager. Він не створює об'єкти напряму, а делегує це завдання менеджеру.

Ключові обов'язки класу:

- **Ініціалізація системи:** Створює екземпляр FileManager.
- **Створення та конфігурація прототипів:** Готує початкові об'єкти-шаблони (простий файл docTemplate та складну структуру теки projectTemplate).
- **Реєстрація прототипів:** Передає створені шаблони до FileManager за допомогою методу registerTemplate(), асоціюючи їх з логічними іменами.

- **Демонстрація використання:** Викликає метод createFromTemplate() для створення нових, повністю функціональних об'єктів на основі зареєстрованих шаблонів. Результати роботи (створені об'єкти) виводяться в консоль, що наочно показує, як складні структури були відтворені за одну операцію клонування.

```
public class Shell {
    public static void main(String[] args) {

        System.out.println("--- Реєстрація шаблонів ---");

        File docTemplate = new File( name: "template.docx",  path: "",  size: 1024);
        manager.registerTemplate( key: "document", docTemplate);
        System.out.println("Шаблон 'document' зареєстровано.");

        Directory projectTemplate = new Directory( name: "NewProject",  path: "");
        projectTemplate.add(new Directory( name: "src",  path: ""));
        projectTemplate.add(new Directory( name: "docs",  path: ""));
        projectTemplate.add(new File( name: "README.md",  path: "",  size: 512));
        manager.registerTemplate( key: "project", projectTemplate);
        System.out.println("Шаблон 'project' зареєстровано.\n");

        System.out.println("--- Створення об'єктів з шаблонів ---");

        File financialReport = (File) manager.createFromTemplate(
                key: "document",  newPath: "C:/Reports/",  newName: "FinancialReport_Q3.docx");

        Directory myApp = (Directory) manager.createFromTemplate(
                key: "project",  newPath: "C:/Development/",  newName: "MyAwesomeApp");

        System.out.println("Створено новий документ:");
        financialReport.display( depth: 0);

        System.out.println("\nСтворено новий проект:");
        myApp.display( depth: 0);

    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
```

рис 2.5 - код класу Shell.java

Опис результатів коду

```
--- Реєстрація шаблонів ---
Шаблон 'document' зареєстровано.
Шаблон 'project' зареєстровано.

--- Створення об'єктів з шаблонів ---
Створено новий документ:
- FinancialReport_Q3.docx (File, 1024 bytes)

Створено новий проект:
[MyAwesomeApp] (Directory)
  [src] (Directory)
  [docs] (Directory)
  - README.md (File, 512 bytes)
```

рис 2.6 - результати виконання коду

Наданий скріншот демонструє консольний вивід, який є результатом роботи програми та наочно ілюструє всі ключові етапи та переваги використання патерну проектування «Прототип» у поєднанні з Реєстром Прототипів (FileManager).

Етап 1: Підготовча фаза - Реєстрація шаблонів

Перший блок виводу в консолі відповідає початковій фазі роботи програми. На цьому етапі клієнт (Shell) створює та конфігурує базові об'єкти-прототипи, які потім будуть використовуватися як шаблони для створення нових екземплярів. Було зареєстровано два типи прототипів:

- **Шаблон 'document':** Створено екземпляр класу File, що є простим (атомарним) прототипом. Він представляє собою один об'єкт без вкладених залежностей.
- **Шаблон 'project':** Створено екземпляр класу Directory, який є складним (композитним) прототипом. Він містить у собі інші об'єкти (Directory та File), утворюючи ієрархічну структуру.

Ці прототипи були передані до FileManager, який зберіг їх у внутрішньому сховищі, асоціювавши кожен з унікальним ключем. Цей етап є критично важливим, оскільки він "наповнює" наш реєстр шаблонами, готовими до багаторазового клонування.

Етап 2: Основна логіка - Створення об'єктів з шаблонів

Друга частина виводу демонструє практичне застосування патерну. Клієнт більше не конструює об'єкти вручну. Замість цього він звертається до FileManager з проханням створити об'єкт на основі одного з раніше зареєстрованих шаблонів.

- **Створення простого об'єкта:**
  Програма успішно створила новий файл FinancialReport_Q3.docx. Важливо відзначити, що його внутрішній стан (розмір 1024 bytes) був точно скопійований з прототипу document. При цьому його зовнішні, ідентифікуючі властивості (ім'я та шлях) були змінені після клонування. Це демонструє, як патерн дозволяє створювати об'єкти з однаковою базовою конфігурацією, але різними ідентифікаторами.
- **Створення складного об'єкта (демонстрація глибокого копіювання):**
  Найбільш показовим є створення нового проєкту MyAwesomeApp. Вивід чітко показує, що був клонований не лише батьківський об'єкт-тека, а й **вся його внутрішня ієрархія**: підтеки src і docs та файл README.md. Це є візуальним підтвердженням того, що в класі Directory було коректно реалізовано механізм **глибокого копіювання**. Кожен вкладений елемент був рекурсивно скопійований, завдяки чому новостворений проєкт є повністю незалежною копією, і будь-які подальші зміни в ньому не вплинуть на оригінальний прототип project.

Вивід програми повністю підтверджує успішну реалізацію патерну «Прототип». Він демонструє, що система здатна ефективно створювати як прості, так і складні композитні об'єкти шляхом клонування, приховуючи від клієнта деталі цього процесу. Це підкреслює головні переваги патерну: зменшення складності коду, підвищення продуктивності при створенні "важких" об'єктів та гнучкість у конфігурації системи.

3. Діаграма класів

Для візуалізації архітектури розробленої системи та взаємозв'язків між її компонентами була побудована UML-діаграма класів. Вона наочно демонструє структуру, що відповідає шаблону проектування «Prototype».

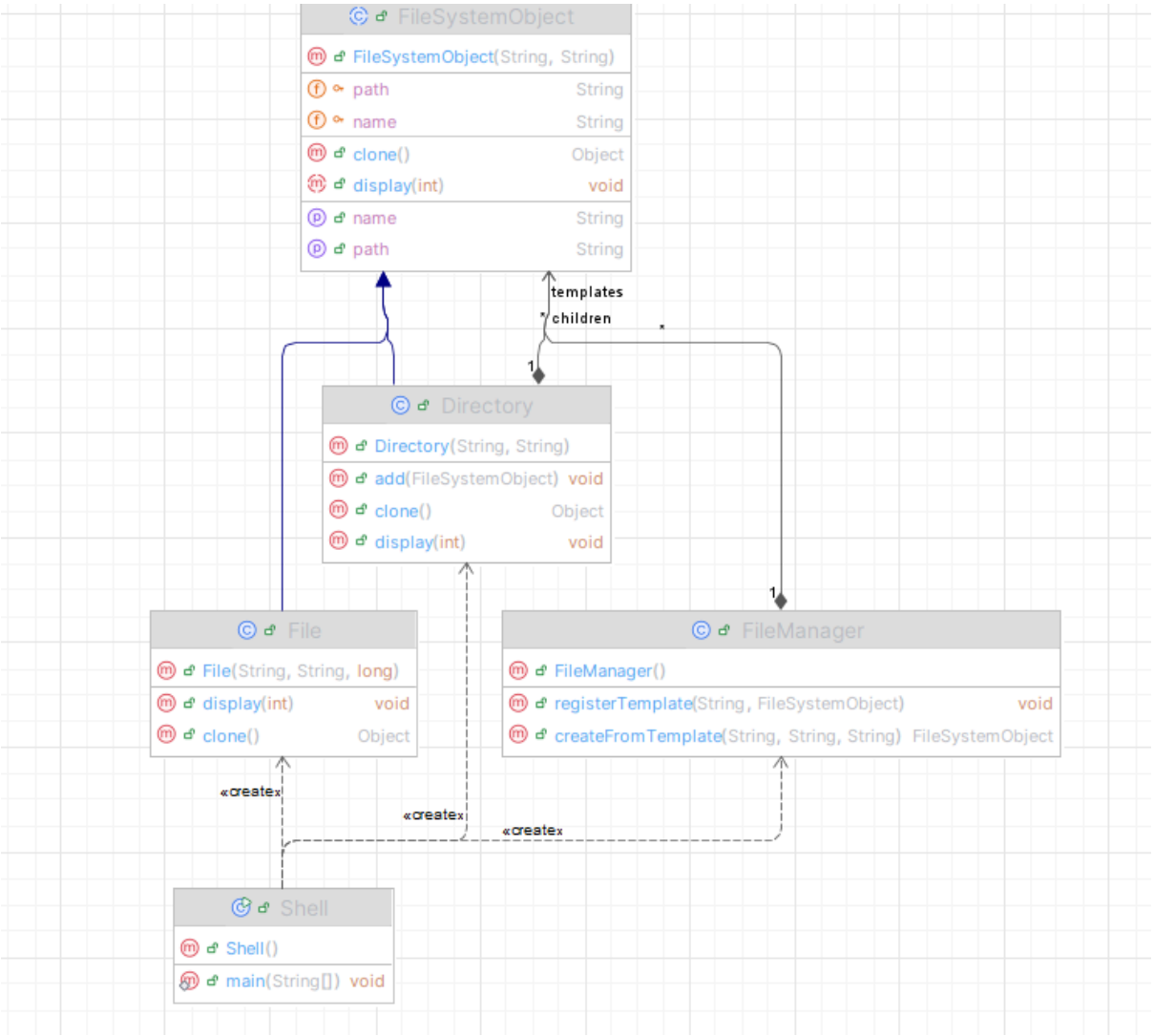


рис 3.1 - діаграма класів

На скріншоті представлена UML-діаграма, яка візуалізує статичну структуру розробленої програмної системи та взаємозв'язки між її ключовими компонентами. Діаграма точно відображає реалізацію патерну «Прототип» у поєднанні з Реєстром Прототипів.

Основні елементи діаграми:

1. **Ієрархія прототипів:**
    - В основі архітектури лежить абстрактний клас FileSystemObject, який виступає в ролі **базового прототипу**. Він визначає загальний інтерфейс та стан для всіх об'єктів файлової системи.
    - Класи File та Directory є **конкретними прототипами**. Вони успадковують FileSystemObject, що показано суцільною лінією з трикутною стрілкою (відношення наслідування). Кожен з них надає власну реалізацію методів, зокрема ключового методу clone().
2. **Композиційні зв'язки (Агрегація):**
    - Зв'язок між Directory та FileSystemObject (позначений як children) показаний як агрегація (лінія з порожнім ромбом). Це означає, що екземпляр Directory містить у собі колекцію (0..*) інших об'єктів FileSystemObject, що дозволяє створювати ієрархічні деревоподібні структури.
    - Аналогічний зв'язок існує між FileManager та FileSystemObject (позначений як templates). FileManager агрегує колекцію прототипів, виконуючи роль їхнього сховища або **реєстру**.
3. **Клієнт та залежності:**
    - Клас Shell виступає в ролі **клієнта**. Його зв'язки з іншими класами показані пунктирними лініями (відношення залежності) з підписом «create». Це вказує на те, що Shell не зберігає постійних посилань на ці класи, а лише створює їхні екземпляри в методі main для ініціалізації системи та демонстрації її роботи.

Бачимо, що діаграма повноцінно ілюструє архітектуру реалізованої системи. Вона чітко показує розподіл ролей відповідно до патерну «Прототип»: є і ієрархія об'єктів, здатних до клонування, і централізований менеджер (FileManager) для керування цими прототипами, і клієнт (Shell), який взаємодіє з системою на високому рівні, не вдаючись у деталі створення об'єктів.

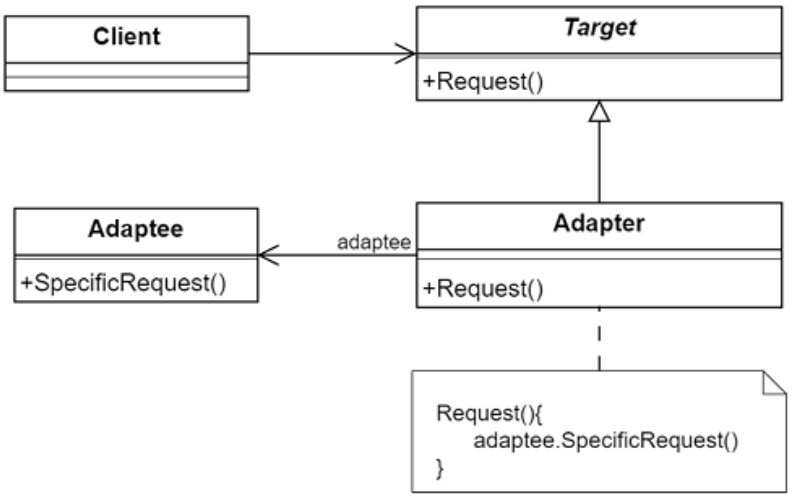
4. Висновки

Висновки: В ході виконання даної лабораторної роботи було успішно вивчено та реалізовано патерн проектування «Прототип» для вирішення задачі гнучкого створення об'єктів у файловому менеджері. Була створена ієрархія класів, здатних до клонування, де для простих об'єктів, як File, застосовувалось поверхневе копіювання, а для складних ієрархічних структур, як Directory, було реалізовано механізм глибокого копіювання, що забезпечило повну незалежність копій. Впровадження класу FileManager у ролі реєстру прототипів дозволило централізувати керування шаблонами та повністю абстрагувати клієнтський код від процесів ініціалізації конкретних класів. Практична реалізація продемонструвала ключові переваги патерну: значне спрощення процесу створення складних об'єктів, підвищення продуктивності за рахунок уникнення викликів конструкторів та гнучкість системи, що дозволяє динамічно додавати нові типи об'єктів без зміни існуючого коду. Таким чином, мета лабораторної роботи була повністю досягнута, а ефективність патерну «Прототип» була підтверджена на практиці.

Контрольні запитання

1. Яке призначення шаблону «Адаптер»?
Шаблон «Адаптер» (Adapter) призначений для перетворення інтерфейсу одного класу в інтерфейс, очікуваний клієнтом. Він дозволяє класам працювати разом, навіть якщо їхні інтерфейси несумісні, виступаючи в ролі посередника між ними.

2. Нарисуйте структуру шаблону «Адаптер».



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

- **Target:** Інтерфейс, який очікує та використовує клієнт (Client).
- **Client:** Клієнтський клас, який взаємодіє з об'єктами через інтерфейс Target.
- **Adaptee:** Клас з несумісним інтерфейсом, який потрібно адаптувати.
- **Adapter:** Клас, що реалізує інтерфейс Target і містить посилання на об'єкт Adaptee. Коли клієнт викликає метод Adapter, той, у свою чергу, перетворює цей виклик у відповідний виклик методу об'єкта Adaptee.

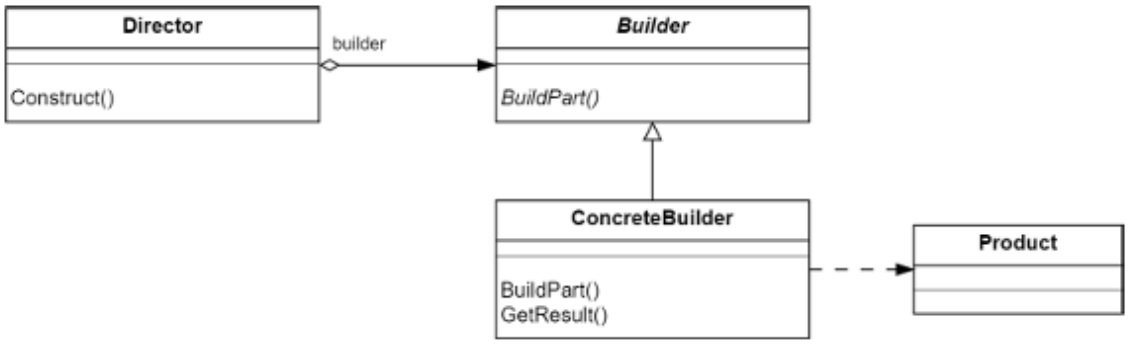
4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

- **Адаптер класів** використовує **наслідування**. Клас Adapter успадковує одночасно і клас Adaptee, і інтерфейс Target. Цей підхід можливий у мовах, що підтримують множинне наслідування.
- **Адаптер об'єктів** використовує **композицію**. Клас Adapter реалізує інтерфейс Target і містить у собі екземпляр класу Adaptee. Цей підхід є більш гнучким, оскільки дозволяє адаптувати цілу ієрархію класів Adaptee.

5. Яке призначення шаблону «Будівельник»?
Шаблон «Будівельник» (Builder) призначений для того, щоб відокремити процес конструювання складного об'єкта від його представлення. Це

дозволяє використовувати один і той же процес конструювання для створення різних варіацій об'єкта.

6. Нарисуйте структуру шаблону «Будівельник».



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

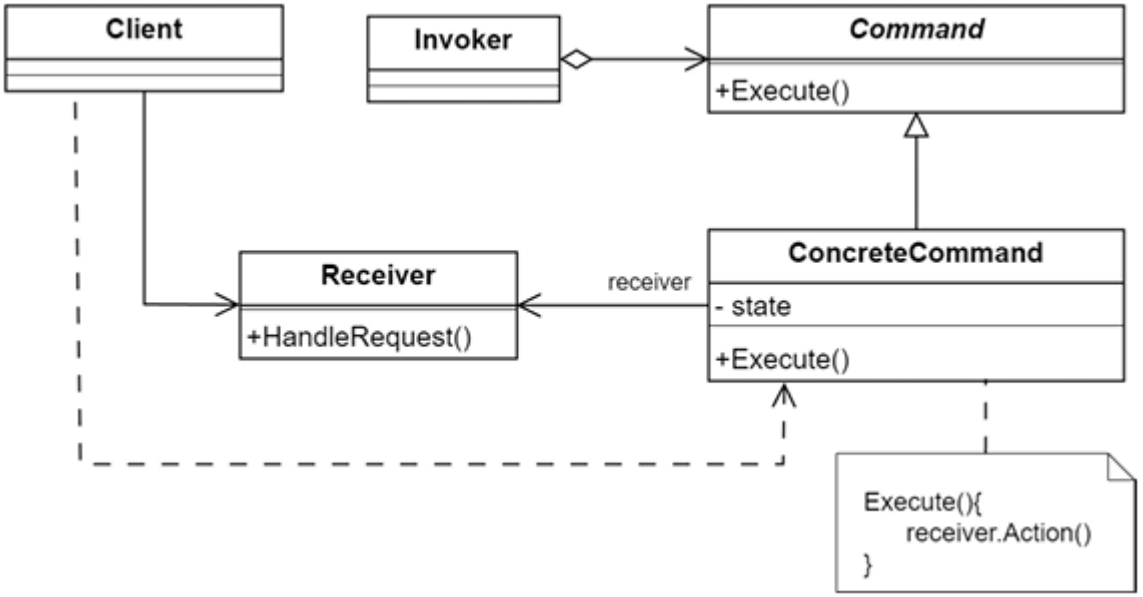
- **Product:** Складний об'єкт, який створюється.
- **Builder:** Абстрактний інтерфейс, що визначає кроки для створення частин об'єкта Product.
- **ConcreteBuilder:** Конкретна реалізація інтерфейсу Builder, яка конструює та збирає частини об'єкта.
- **Director:** Клас, який керує процесом конструювання, викликаючи методи Builder у правильній послідовності. Director не знає про конкретну реалізацію Builder чи Product.

8. У яких випадках варто застосовувати шаблон «Будівельник»"?
Шаблон варто застосовувати, коли:

- Процес створення об'єкта є складним, багатоетапним і не повинен бути прив'язаний до класів, що його складають.
- Необхідно створювати різні представлення одного й того ж об'єкта.
- Потрібно уникнути "телескопічного конструктора" (конструктора з великою кількістю необов'язкових параметрів).

9. Яке призначення шаблону «Команда»?
Шаблон «Команда» (Command) призначений для інкапсуляції запиту (виклику операції) у вигляді об'єкта. Це дозволяє параметризувати клієнтські об'єкти різними запитами, ставити запити в чергу, логувати їх, а також підтримувати операції скасування (undo).

10. Нарисуйте структуру шаблону «Команда».



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

- **Command:** Інтерфейс, що оголошує метод для виконання операції (зазвичай execute()).
- **ConcreteCommand:** Клас, що реалізує інтерфейс Command. Він містить посилання на об'єкт Receiver і викликає його методи для виконання запиту.
- **Receiver:** Об'єкт, який безпосередньо виконує операцію (містить бізнес-логіку).
- **Invoker:** Об'єкт, який ініціює виконання команди. Він не знає нічого про Receiver, а лише викликає метод execute() у об'єкта Command.
- **Client:** Створює об'єкт ConcreteCommand і встановлює для нього Receiver.

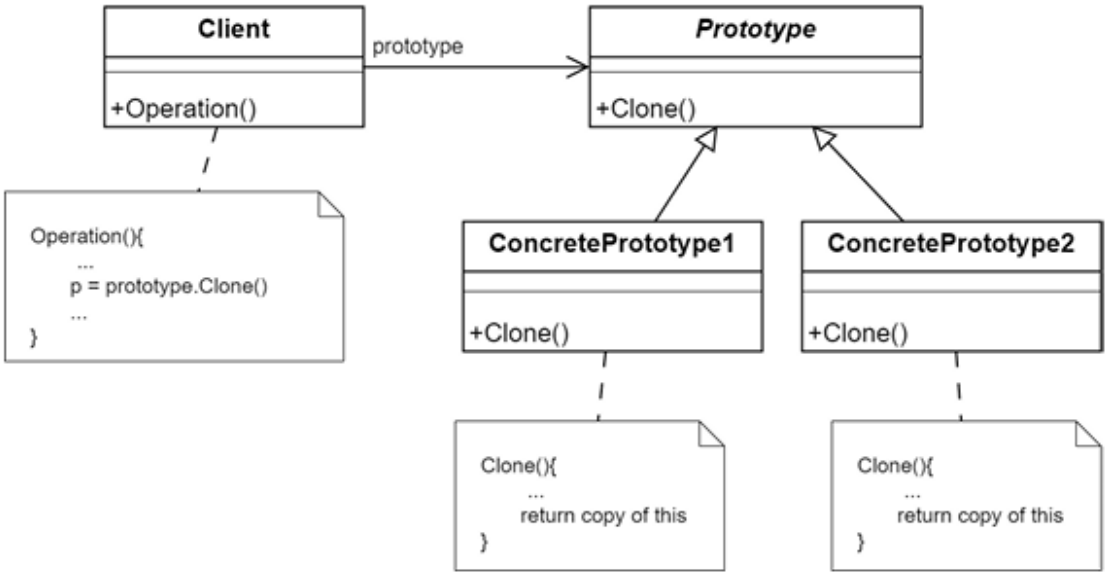
12. Розкажіть як працює шаблон «Команда».

Замість того, щоб Invoker безпосередньо викликав метод Receiver, клієнт створює об'єкт Command, який містить у собі посилання на Receiver та інформацію про те, яку дію потрібно виконати. Цей об'єкт Command передається Invoker. Коли настає час виконати дію, Invoker просто викликає єдиний метод execute() у команди. Команда, у свою чергу, делегує виклик відповідному методу Receiver.

13. Яке призначення шаблону «Прототип»?

Шаблон «Прототип» (Prototype) призначений для створення нових об'єктів шляхом копіювання (клонування) існуючого об'єкта, який називається прототипом. Це дозволяє уникнути прямої залежності від класів об'єктів, що створюються, і може бути ефективнішим за звичайне створення екземпляра.

14. Нарисуйте структуру шаблону «Прототип».



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

- **Prototype:** Абстрактний клас або інтерфейс, що оголошує метод клонування.
- **ConcretePrototype:** Конкретний клас, що реалізує метод клонування. Він відповідає за створення копії самого себе.
- **Client:** Клієнтський клас, який створює новий об'єкт, викликаючи метод клонування у існуючого об'єкта-прототипа.

16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?

- **Системи обробки GUI-подій:** Коли користувач клікає на кнопку, подія може бути оброблена спочатку самою кнопкою, потім її батьківською панеллю, потім вікном, і так далі по ієрархії, доки не знайдеться обробник.

- **Middleware у веб-фреймворках:** Вхідний HTTP-запит послідовно проходить через ланцюжок проміжних обробників (middleware) для аутентифікації, логування, кешування тощо.
- **Системи логування:** Повідомлення може передаватися по ланцюжку логерів (консольний, файловий, мережевий), кожен з яких вирішує, чи обробляти це повідомлення залежно від його рівня важливості.
- **Системи затвердження документів:** Заявка на відпустку або звіт про витрати послідовно передається на затвердження від одного менеджера до іншого по ланцюжку ієрархії.