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

Технології розроблення програмного забезпечення

Тема проєкту ‘Shell (total commander)’

Виконала: студентка групи ІА-33

Маляревич Анжела Валентинівна

Перевірив:

Мягкий Михайло Юрійович

Київ - 2025

Зміст

Тема: Патерни проектування.

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

Тема роботи:

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

Шаблон «Adapter» (Адаптер)

Призначення: дозволяє узгодити роботу класів з різними інтерфейсами, «обгортаючи» один уніфікованим інтерфейсом.
Ідея: створюється спільний інтерфейс (наприклад, IPlayer), а для кожного конкретного класу — свій адаптер, який реалізує цей інтерфейс.
Переваги:

- додає нові адаптери без зміни існуючого коду;
- робить код зрозумілішим.

Недолік: збільшується кількість класів.

Шаблон «Builder» (Будівельник)

Призначення: відділяє процес створення об’єкта від його представлення.
Ідея: кожен етап побудови об’єкта виділяється в окремий метод будівельника, що дозволяє створювати різні варіанти об’єкта з тих самих кроків.
Переваги:

- гнучкість та незалежність від змін у деталях;

- один код може будувати різні об’єкти.

Недолік: залежність від конкретних реалізацій будівельників.

Шаблон «Command» (Команда)

Призначення: перетворює виклик методу на окремий об’єкт-команду.
Ідея: кожна дія (наприклад, натискання кнопки) реалізується у власному класі-команді, що відокремлює UI від логіки.
Переваги:

- легко додавати нові команди;
- можна скасовувати, повторювати або логувати дії;
- простіше тестувати логіку без UI.

Шаблон «Chain of Responsibility» (Ланцюжок відповідальності)

Призначення: дозволяє послідовно передавати запит по ланцюжку обробників, поки хтось його не обробить.
Ідея: кожен обробник має посилання на «наступного» і вирішує, чи передавати далі.
Переваги:

- клієнт не знає, хто саме обробить запит;
- легко додавати або видаляти обробники.

Недолік: запит може залишитись без обробки.

Шаблон «Prototype» (Прототип)

Призначення: створює нові об’єкти шляхом копіювання існуючого «прототипу».
Ідея: кожен клас має метод clone(), який повертає копію об’єкта. Це спрощує створення схожих екземплярів.
Переваги:

- швидше створення складних об’єктів;

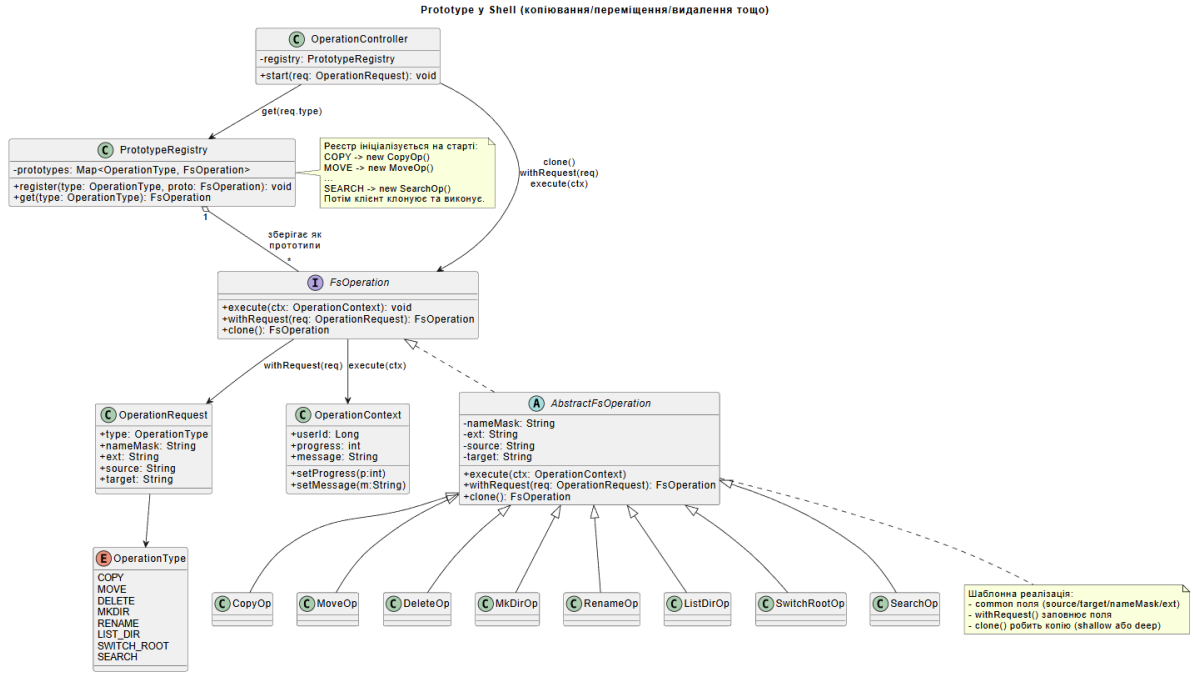
- зменшення ієрархії класів;
- клоновані об’єкти можна змінювати незалежно.

Недоліки:

- складно реалізувати глибоке копіювання;
- надмірне використання ускладнює код.

Хід роботи

Діаграма класів



Діаграма показує, як кожна операція успадковується від абстрактного прототипу та може бути клонована для повторного використання

FsOperation.java

```java
package ua.kpi.ia33.shellweb.patterns.prototype.fs;

import ua.kpi.ia33.shellweb.patterns.state.op.OperationRequest;
import ua.kpi.ia33.shellweb.patterns.state.op.OperationContext;



public interface FsOperation extends Cloneable {  18 usages  9 implementations
    FsOperation withRequest(OperationRequest req); // заповнюємо source/target/nameMask/ext  2 usages  1 implementation
    void execute(OperationContext ctx) throws Exception; // реальна дія  9 implementations
    FsOperation clone();  1 implementation
}
```

AbstractFsOperation.java

AbstractFsOperation.java × | factorymethod\FsOperation.java | DefaultFsOperationFactory.java | dt

```java
package ua.kpi.ia33.shellweb.patterns.prototype.fs;

import ua.kpi.ia33.shellweb.patterns.state.op.OperationRequest;
import ua.kpi.ia33.shellweb.patterns.state.op.OperationContext;
import ua.kpi.ia33.shellweb.domain.User;

import java.nio.file.Path;

public abstract class AbstractFsOperation implements FsOperation {  8 usages  8 inheritors
    protected Path source, target;
    protected String nameMask, ext;  2 usages
    protected User user;

    @Override  2 usages
    public FsOperation withRequest(OperationRequest r) {
        this.source   = r.getSource();
        this.target   = r.getTarget();
        this.nameMask = r.getNameMask();
        this.ext      = r.getExt();
        this.user     = r.getUser();
        return this;
    }

    @Override
    public FsOperation clone() {
        try { return (FsOperation) super.clone(); }
        catch (CloneNotSupportedException e) { throw new AssertionError(e); }
    }

    @Override  8 implementations
    public abstract void execute(OperationContext ctx) throws Exception;
}
```

інтерфейс FsOperation, який визначає методи clone(), execute() та withRequest(). абстрактний клас AbstractFsOperation, що реалізує механізм клонування (через super.clone()) та заповнення параметрів операції.

На фото видно ключову частину патерну Prototype це клонування об’єкта замість його створення через new

OperationPrototypes.java

SearchOp.java

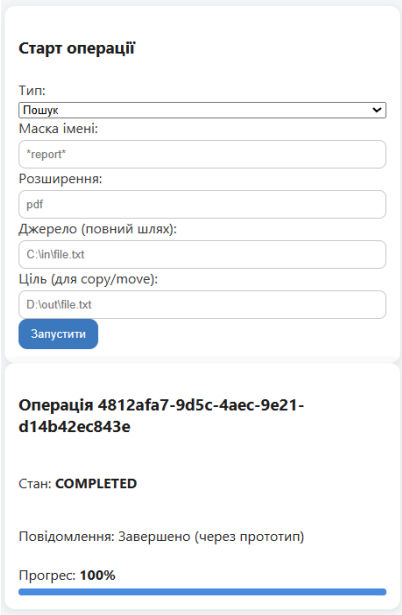
OperationPrototypes.java | SearchOp.java | OperationContext.java | op\OperationType.java

```java
package ua.kpi.ia33.shellweb.patterns.prototype.fs;

import ua.kpi.ia33.shellweb.patterns.state.op.OperationContext;
import ua.kpi.ia33.shellweb.service.SearchService;
import ua.kpi.ia33.shellweb.domain.SearchQuery;

public class SearchOp extends AbstractFsOperation {  1 usage
    private final SearchService search;  2 usages
    public SearchOp(SearchService search) { this.search = search; }  1 usage

    @Override
    public void execute(OperationContext ctx) throws Exception {
        if (user == null) throw new IllegalStateException("User is required for SEARCH");
        SearchQuery q = search.createQuery(user, nameMask, ext);
        ctx.setProgress(100);
        ctx.setMessage("Пошук завершено: запит #" + q.getId());
    }
```

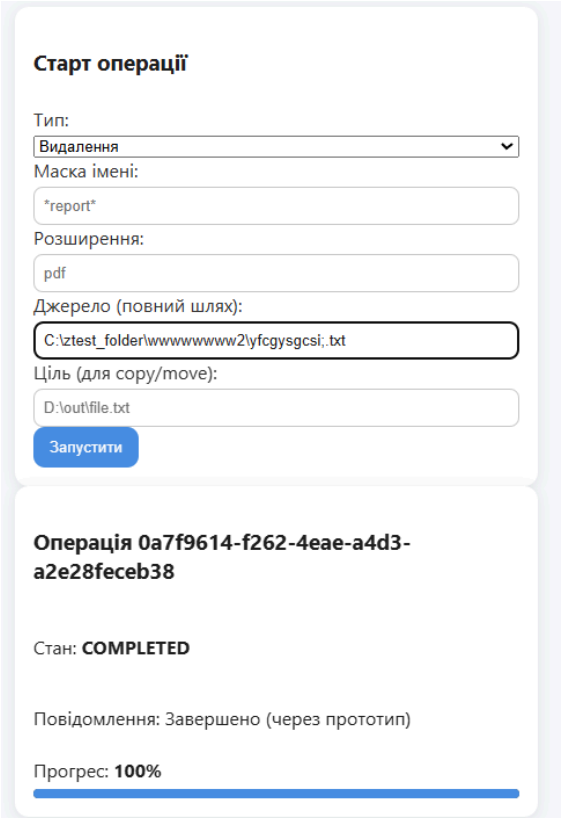
Клас OperationPrototypes, що містить карту прототипів. Саме він зберігає базові екземпляри операцій і повертає їх для клонування. Клас SearchOp, який є конкретним прототипом і реалізує логіку пошуку файлів.

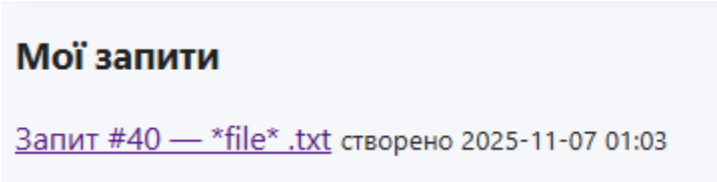
Ці фото демонструють, як налаштовано Prototype у твоїй системі: операція береться з реєстру → клонуються → виконуються

Форма виконання операції (пошук)



Операція видалення





Результати пошуку

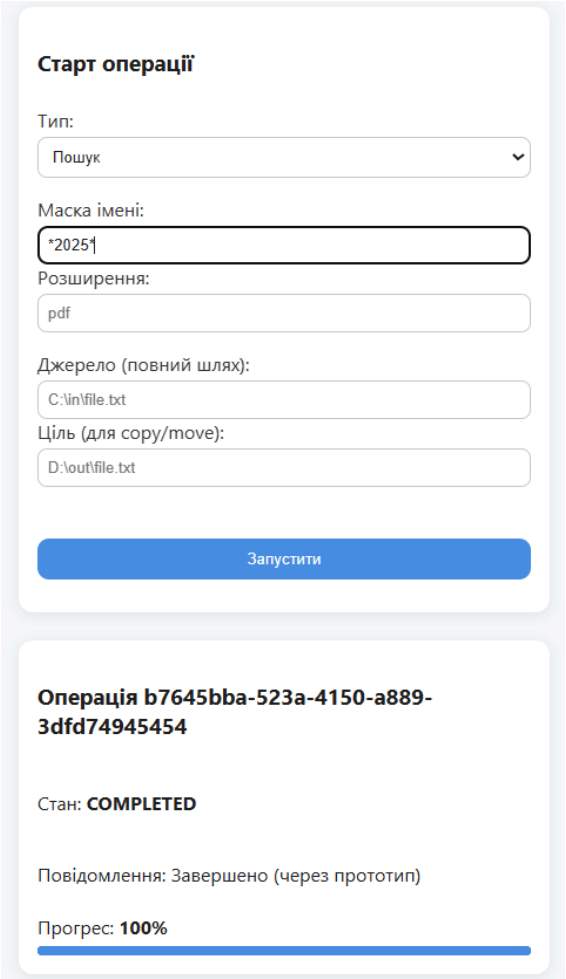
Поточний стан: **unknown**

Запит #40: ***file*** .txt

Тип	Ім'я	Шлях
file	dx11filelist.txt	D:\SteamLibrary\steamapps\common\The Witcher 3\bin\config\r4game\user_config_matrix\pc\dx11filelist.txt
file	dx12filelist.txt	D:\SteamLibrary\steamapps\common\The Witcher 3\bin\config\r4game\user_config_matrix\pc\dx12filelist.txt
file	test_file.txt	D:\xampp\perl\vendor\lib\auto\share\module\File-ShareDir\test_file.txt

← Повернутись

Пошук за маскою “*2025*”



Результати пошуку

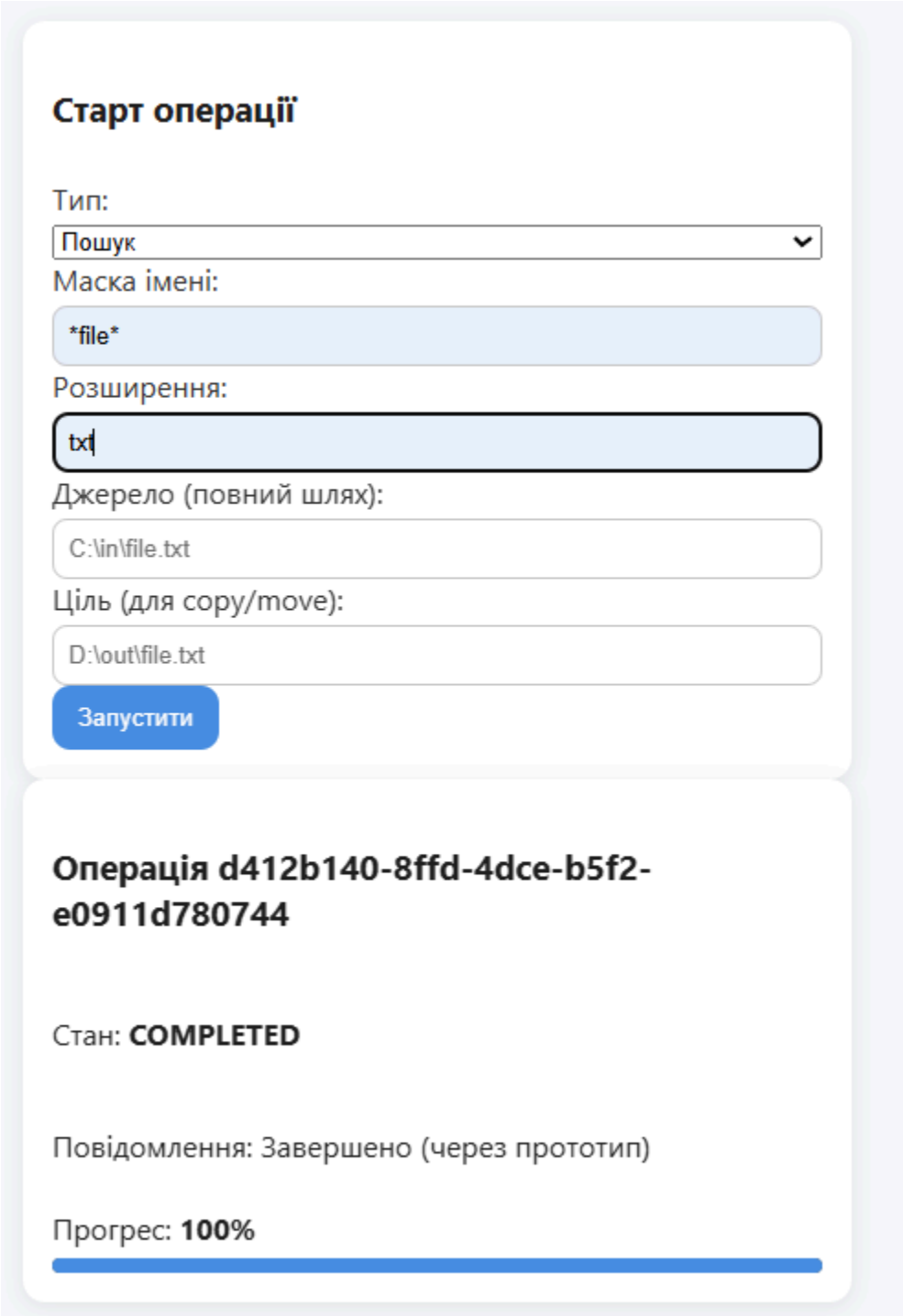
Поточний стан: **unknown**

Запит #41: ***2025*** .

Тип	Ім'я	Шлях
file	config_2025-07-30.json	D:\SteamLibrary\steamapps\common\wallpaper_engine\config_backups\config_2025-07-30.json
file	config_2025-08-26.json	D:\SteamLibrary\steamapps\common\wallpaper_engine\config_backups\config_2025-08-26.json
file	config_2025-08-30.json	D:\SteamLibrary\steamapps\common\wallpaper_engine\config_backups\config_2025-08-30.json
file	config_2025-09-11.json	D:\SteamLibrary\steamapps\common\wallpaper_engine\config_backups\config_2025-09-11.json
file	config_2025-09-15.json	D:\SteamLibrary\steamapps\common\wallpaper_engine\config_backups\config_2025-09-15.json
file	config_2025-09-20.json	D:\SteamLibrary\steamapps\common\wallpaper_engine\config_backups\config_2025-09-20.json
file	config_2025-09-23.json	D:\SteamLibrary\steamapps\common\wallpaper_engine\config_backups\config_2025-09-23.json
file	config_2025-10-09.json	D:\SteamLibrary\steamapps\common\wallpaper_engine\config_backups\config_2025-10-09.json
file	Л.р.№2_des-20251001T125301Z-1-001.zip	D:\бісики\Л.р.№2_des-20251001T125301Z-1-001.zip

← Повернутись

Пошук file


Старт операції
Тип:
Пошук
Маска імені:
file
Розширення:
txt
Джерело (повний шлях):
C:\in\file.txt
Ціль (для copy/move):
D:\out\file.txt
Запустити
Операція d412b140-8ffd-4dce-b5f2-e0911d780744
Стан: COMPLETED
Повідомлення: Завершено (через прототип)
Прогрес: 100%

Висновок

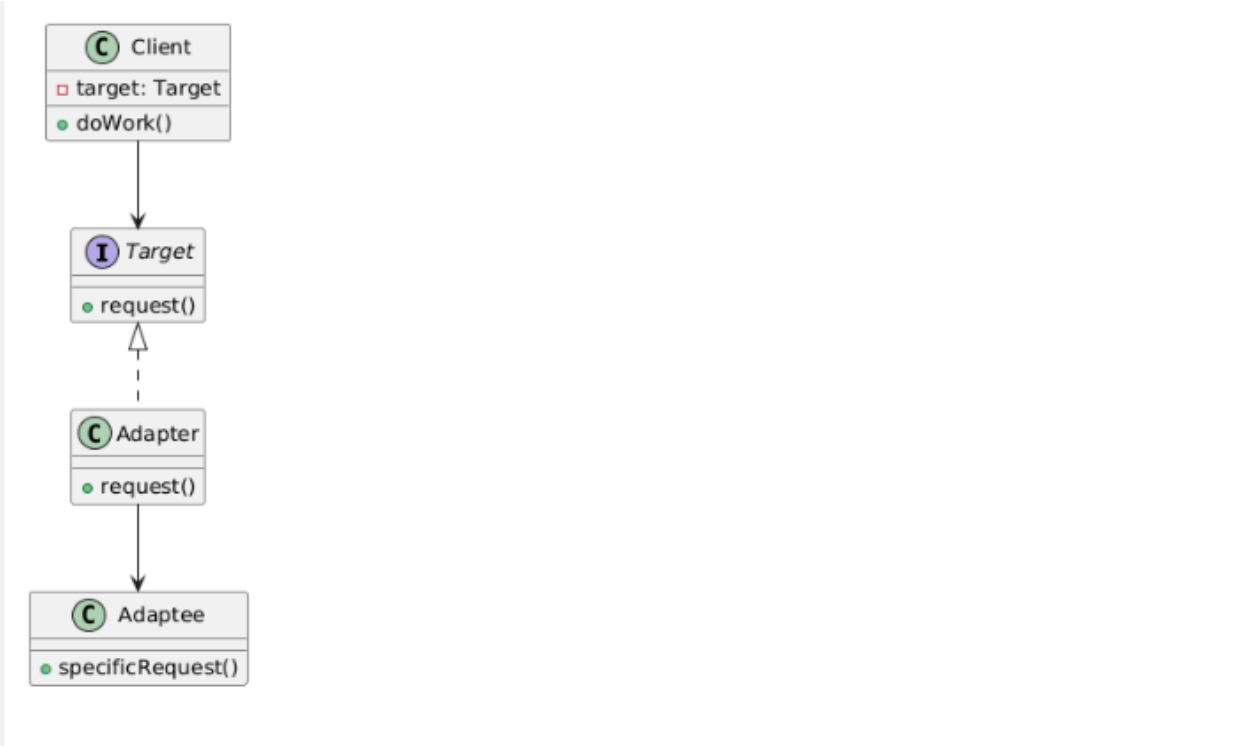
У цій лабораторній роботі я реалізувала патерн Prototype для створення та повторного використання файлових операцій у проєкті Shell. Я винесла спільну логіку в базовий прототип, створила набір конкретних операцій і побудувала реєстр прототипів, з якого контролер отримує потрібний шаблон і клонує його з новими параметрами. Це дозволило уникнути дублювання коду та зробило систему гнучкою

Питання до лабораторної роботи

1. Яке призначення шаблону «Адаптер»?

«Адаптер» дозволяє змусити працювати разом два класи з несумісними інтерфейсами. Він обгортає існуючий клас і перетворює його інтерфейс у той, який очікує клієнт

2. Нарисуйте структуру шаблону «Адаптер».



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

- Client – працює тільки з інтерфейсом Target.
- Target – інтерфейс, який очікує клієнт.
- Adaptee – існуючий клас із “незручним” інтерфейсом.
- Adapter – реалізує Target та всередині викликає методи Adaptee.

Клієнт звертається до Adapter через Target, а адаптер усередині делегує виклики до Adaptee

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

Класів?

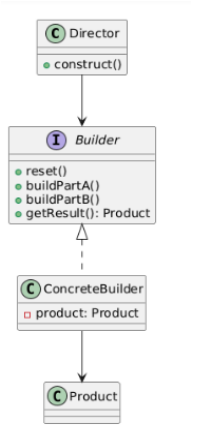
Об’єктний адаптер – використовує композицію: адаптер містить посилання на об’єкт Adaptee. Гнучкіший, можна підміняти реалізацію в рантаймі.

Класовий адаптер – використовує наслідування від Adaptee (і реалізує Target). Підходить там, де є множинне наслідування; менш гнучкий, жорстко прив’язаний до конкретного класу

5. Яке призначення шаблону «Будівельник»?

«Будівельник» відділяє процес побудови складного об’єкта від його представлення. Дозволяє по кроках конструювати об’єкт, не прив’язуючись до конкретного класу результату

6. Нарисуйте структуру шаблону «Будівельник».



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

- Director – знає, в якій послідовності викликати кроки будівництва.
- Builder – інтерфейс з методами типу buildPartA(), buildPartB().
- ConcreteBuilder – конкретна реалізація, збирає Product.
- Product – кінцевий об’єкт.

Клієнт передає Builder у Director, директор викликає кроки побудови, а клієнт потім забирає результат через getResult()

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

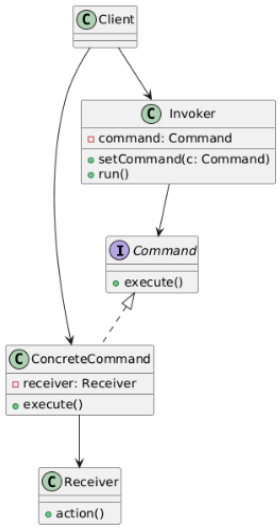
Коли об’єкт:

- має багато полів / частин і різні варіанти конфігурації;
- потрібно конструювати по кроках (можливі різні послідовності);
- важливо відділити “як будувати” від “що саме будуємо” (різні продукти, один сценарій будівництва).

9. Яке призначення шаблону «Команда»?

«Команда» інкапсулює запит у вигляді об’єкта. Це дозволяє ставити команди в чергу, логувати, відміняти/повторювати дії, не знаючи деталей отримувача.

10. Нарисуйте структуру шаблону «Команда».



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

- Command – інтерфейс з методом execute().
- ConcreteCommand – зберігає посилання на Receiver і реалізує execute(), викликаючи receiver.action().
- Receiver – реальний виконавець логіки.
- Invoker – “відправник” команди (кнопка, меню), викликає command.execute().
- Client – створює конкретні команди, налаштовує Invoker.

Клієнт створює ConcreteCommand, передає в Invoker. Коли користувач діє (натискає кнопку), invoker викликає execute(), а команда звертається до receiver.

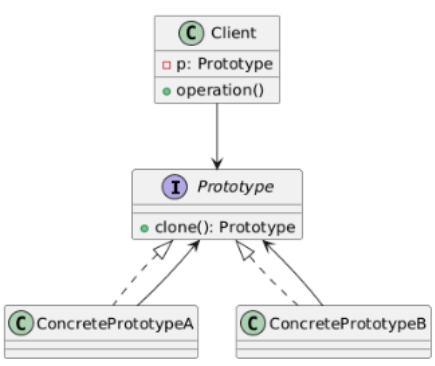
12. Розкажіть як працює шаблон «Команда».

Клієнт обгортає дію в об’єкт-команду, передає цю команду об’єкту-викликачеві (invoker). Коли потрібно виконати дію, invoker не знає деталей – він просто викликає command.execute(). Всередині команда звертається до свого receiver й виконує потрібну операцію. Команди можна зберігати в списку, виконувати пізніше, відкотити або повторити

13. Яке призначення шаблону «Прототип»?

«Прототип» дозволяє створювати нові об’єкти шляхом клонування існуючих. Замість того щоб створювати об’єкт через new і налаштовувати його, ми беремо готовий прототип і копіюємо його методом clone().

14. Нарисуйте структуру шаблону «Прототип».



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

- Prototype – інтерфейс з методом clone().
- ConcretePrototypeA/B – конкретні реалізації, які вміють клонувати самі себе.
- Client – зберігає посилання на прототип і створює нові об’єкти, викликаючи prototype.clone().

Клієнт не знає деталей створення об’єкта, він просто просить у прототипу копію. Всі налаштування “переїжджають” у новий екземпляр.

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

Шаблон «Ланцюжок відповідальності» доречно використовувати там, де запит має пройти через послідовність перевірок або обробників, і кожен обробник сам вирішує обробити запит чи передати далі. Класичні приклади: обробка HTTP-запитів у фільтрах, система логування, перевірки доступу, каскадна обробка подій у UI або послідовні бізнес-перевірки (наприклад, погодження заявки різними рівнями керівництва)