Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 5

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав
Студент групи ІА-31:
Губар Б. О.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

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

2. **Теоретичні відомості:**

Adapter: Адаптує інтерфейс одного об’єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація інтерфейсів бібліотек принтерів). Спрощує інтеграцію без змін у коді.
Builder: Відокремлює процес створення об’єкта від його представлення, доречний для складних або багатоформних конструкцій (наприклад, відповіді веб-сервера). Забезпечує гнучкий контроль.
Command: Перетворює виклик методу в об’єкт, дозволяючи гнучкі системи команд із відміною, логуванням чи плануванням (наприклад, дії в інтерфейсі). Покращує модульність.
Chain of Responsibility: Передає запити по ланцюжку обробників, поки один не виконає (наприклад, контекстні меню). Зменшує зв’язки та спрощує зміни.
Prototype: Створює об’єкти клонуванням прототипу, корисний для складних об’єктів або динамічних змін (наприклад, редактор рівнів гри). Зменшує ієрархію спадкування.

3. **Хід роботи:**

Тема : CI server (state, command, decorator, mediator, visitor, soa)

1) Ознайомитись з короткими теоретичними відомостями.
2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3) Реалізувати один з розглянутих шаблонів за обраною темою.
4) Реалізувати не менше 3-х класів відповідно до обраної теми.
5) 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.

Обґрунтування застосування патерну

Для системи реалізовано патерн **Command** для організації процесу виконання пайплайну збірки. Оскільки процес CI (Continuous Integration) складається з різних етапів (клонування репозиторію, компіляція, запуск тестів), жорстке кодування цієї послідовності в одному класі зробило б систему негнучкою і складною для розширення. Патерн Command дозволяє інкапсулювати кожну дію

(операцію) в окремий об'єкт із загальним інтерфейсом. Це дає змогу будувати пайплайни динамічно (додавати або прибирати кроки), ставити команди в чергу та легко додавати нові типи завдань (наприклад, деплой або лінтер) без зміни коду виконавця.

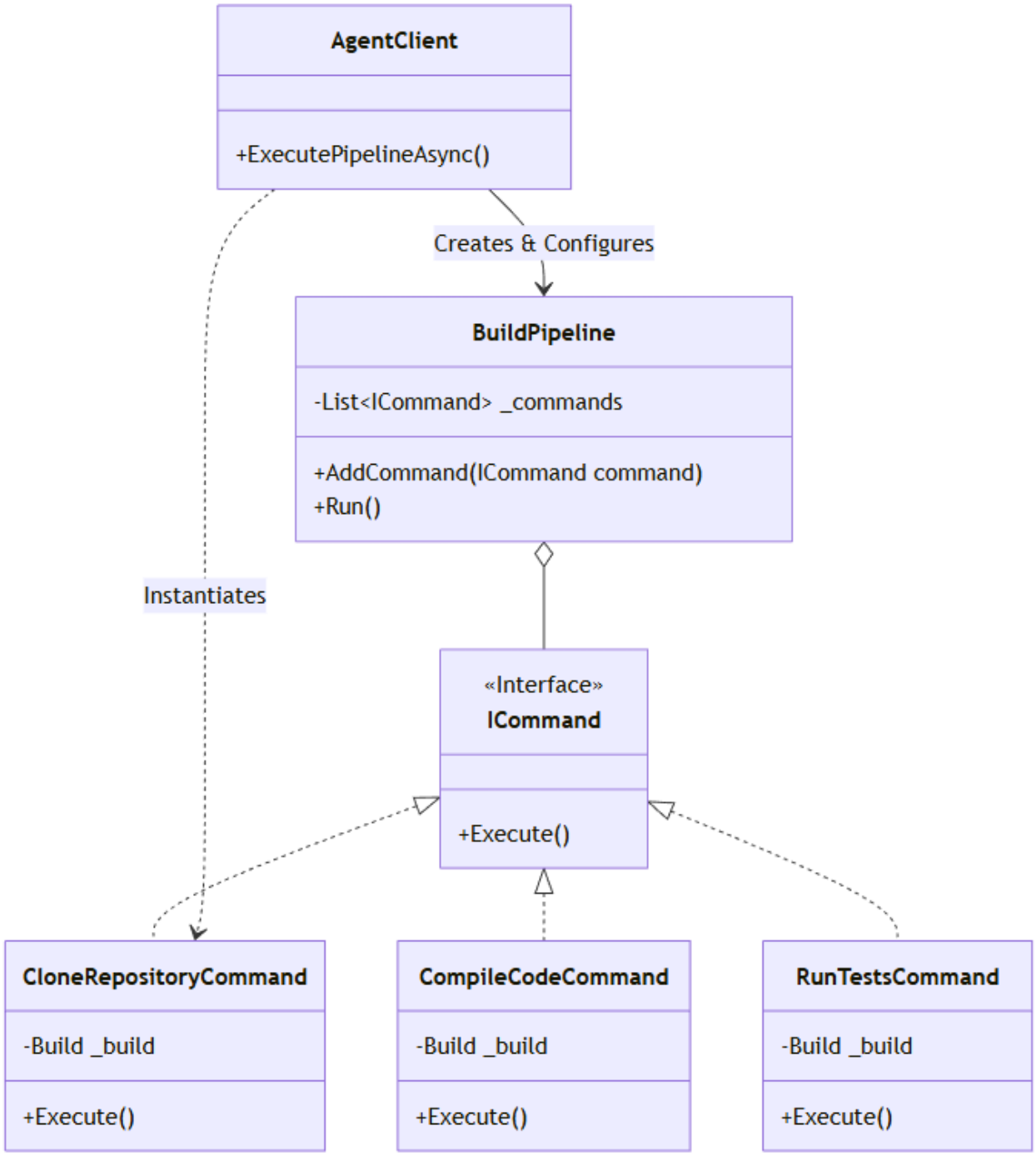


Рисунок 1 - Структура патерну Command

ICommand (Command) – це загальний інтерфейс для всіх команд. Він оголошує єдиний метод Execute(), який запускає виконання операції.

Concrete Commands (CloneRepositoryCommand, CompileCodeCommand, RunTestsCommand) – конкретні реалізації команд. Кожен клас містить власну логіку:

- CloneRepositoryCommand відповідає за взаємодію з git.
- CompileCodeCommand відповідає за виклик dotnet build.
- Всі вони зберігають посилання на контекст (Build), необхідний для виконання (наприклад, URL репозиторію).

BuildPipeline (Invoker) – ініціатор виконання. Він зберігає список команд (_commands) і по черзі викликає у них метод Execute(). Цей клас не знає деталей реалізації команд, він лише керує порядком їх запуску.

AgentClient (Client) – клієнтський код (у файлі Program.cs агента). Він створює об'єкт пайплайну, налаштовує його (додає необхідні команди в потрібному порядку) і запускає виконання.

```
namespace CiServer.Core.Commands;

0 references
public interface ICommand
{
    0 references
    void Execute();
}
```

Рисунок 2 – Контракт методу Execute

```
public void AddCommand(ICommand command)
{
    _commands.Add(command);
}

0 references
public void Run()
{
    Console.WriteLine(">>> Pipeline started.");
    foreach (var command in _commands)
    {
        try
        {
            command.Execute();
        }
        catch (Exception ex)
        {
            Console.WriteLine($"!!! Pipeline failed on command {command.GetType().Name}: {ex.Message}");
            throw;
        }
    }
    Console.WriteLine(">>> Pipeline finished successfully.");
}
```

Рисунок 3 – Виклик command.Execute() -- суть патерну

```
public void Execute()
{
    Console.WriteLine($"[COMPILER] Building project in {_workingDir}...");

    if (!Directory.Exists(_workingDir))
    {
        throw new Exception("Source code not found. Did git clone fail?");
    }

    RunProcess("dotnet", "build", _workingDir);

    Console.WriteLine("[COMPILER] Build successful.");
}
```

Рисунок 4 – Фрагмент CompileCodeCommand

```
var pipeline = new BuildPipeline();
var gitCmd = new CloneRepositoryCommand(build);
var buildCmd = new CompileCodeCommand(build);
var testCmd = new RunTestsCommand(build);

var decoratedGit = new HttpReportDecorator(gitCmd, client, build.BuildId);
var decoratedBuild = new HttpReportDecorator(buildCmd, client, build.BuildId);
var decoratedTest = new HttpReportDecorator(testCmd, client, build.BuildId);

pipeline.AddCommand(decoratedGit);
pipeline.AddCommand(decoratedBuild);
pipeline.AddCommand(decoratedTest);

bool success = true;
try
{
    pipeline.Run();
}
catch
{
    success = false;
}
```

Рисунок 5 – Блок для конфігурації процесу збірки

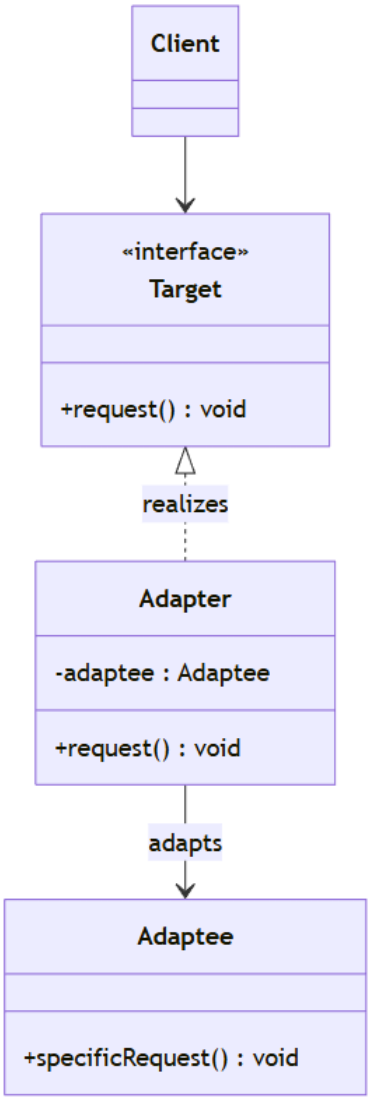
4. Висновок

У ході виконання лабораторної роботи було розглянуто поведінкові шаблони проєктування. На прикладі системи «CI Server» було реалізовано шаблон Command для побудови процесу виконання завдань збірки. Логіка окремих кроків

(клонування, компіляція, тестування) була винесена в окремі класи, що реалізують інтерфейс ICommand. Це дозволило відокремити об'єкт, що ініціює дію (BuildPipeline), від об'єктів, що знають, як її виконати. Завдяки цьому архітектура Агента стала гнучкою: ми можемо динамічно формувати склад пайплайну, змінювати порядок виконання операцій та додавати нові типи команд (наприклад, аналіз коду) без зміни логіки самого виконавця. Це підтверджує ефективність патерну Command для систем, що потребують черг виконання або параметризації об'єктів діями.

5. **Контрольні питання:**

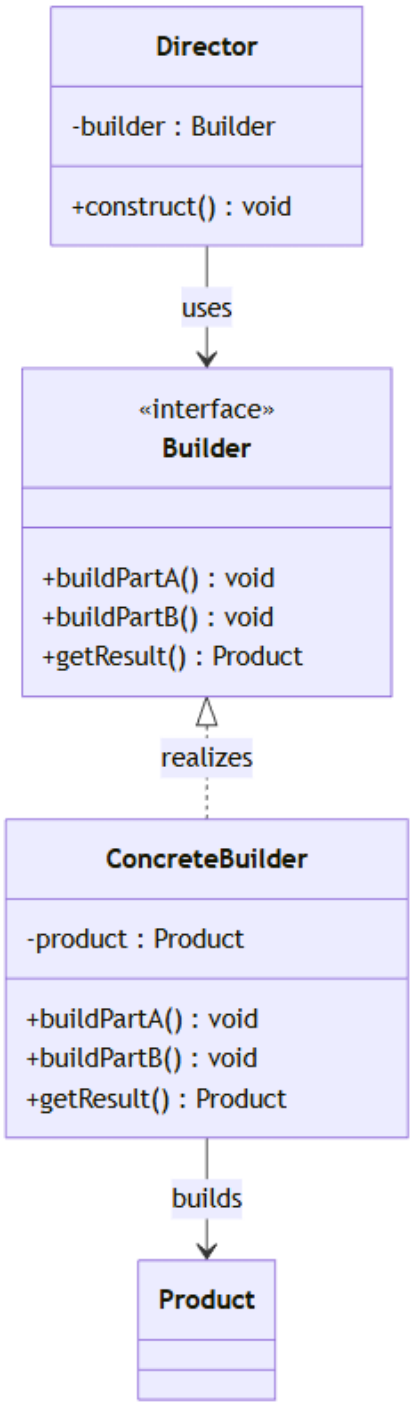
1) Яке призначення шаблону «Адаптер»? Адаптує інтерфейс одного об’єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація бібліотек принтерів).
2) Нарисуйте структуру шаблону «Адаптер».



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

Client, Target, Adapter, Adaptee. Client працює з Target через адаптований інтерфейс, Adapter перенаправляє виклики до Adaptee.

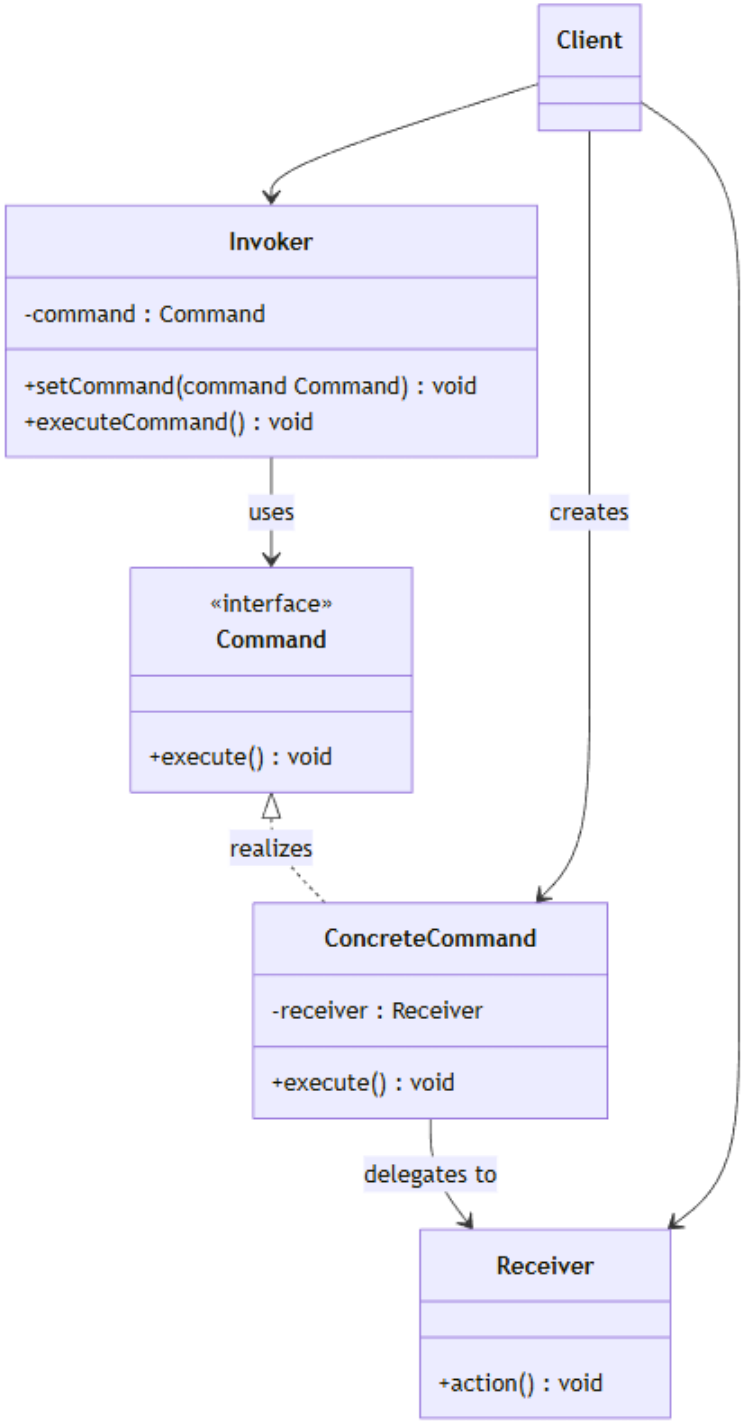
4) Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів? Об’єктний адаптер використовує композицію, класовим — успадкування.
5) Яке призначення шаблону «Будівельник»? Відокремлює процес створення об’єкта від його представлення, придатний для складних або багатоформних конструкцій.
6) Нарисуйте структуру шаблону «Будівельник».



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

Director, Builder, ConcreteBuilder, Product. Director керує будівництвом, Builder визначає інтерфейс, ConcreteBuilder реалізує його, Product — результат.

8) У яких випадках варто застосовувати шаблон «Будівельник»? При складному процесі створення або необхідності різних форм об’єкта.
9) Яке призначення шаблону «Команда»? Перетворює виклик методу в об’єкт для гнучкої системи команд із відміною, логуванням чи плануванням.
10) Нарисуйте структуру шаблону «Команда».



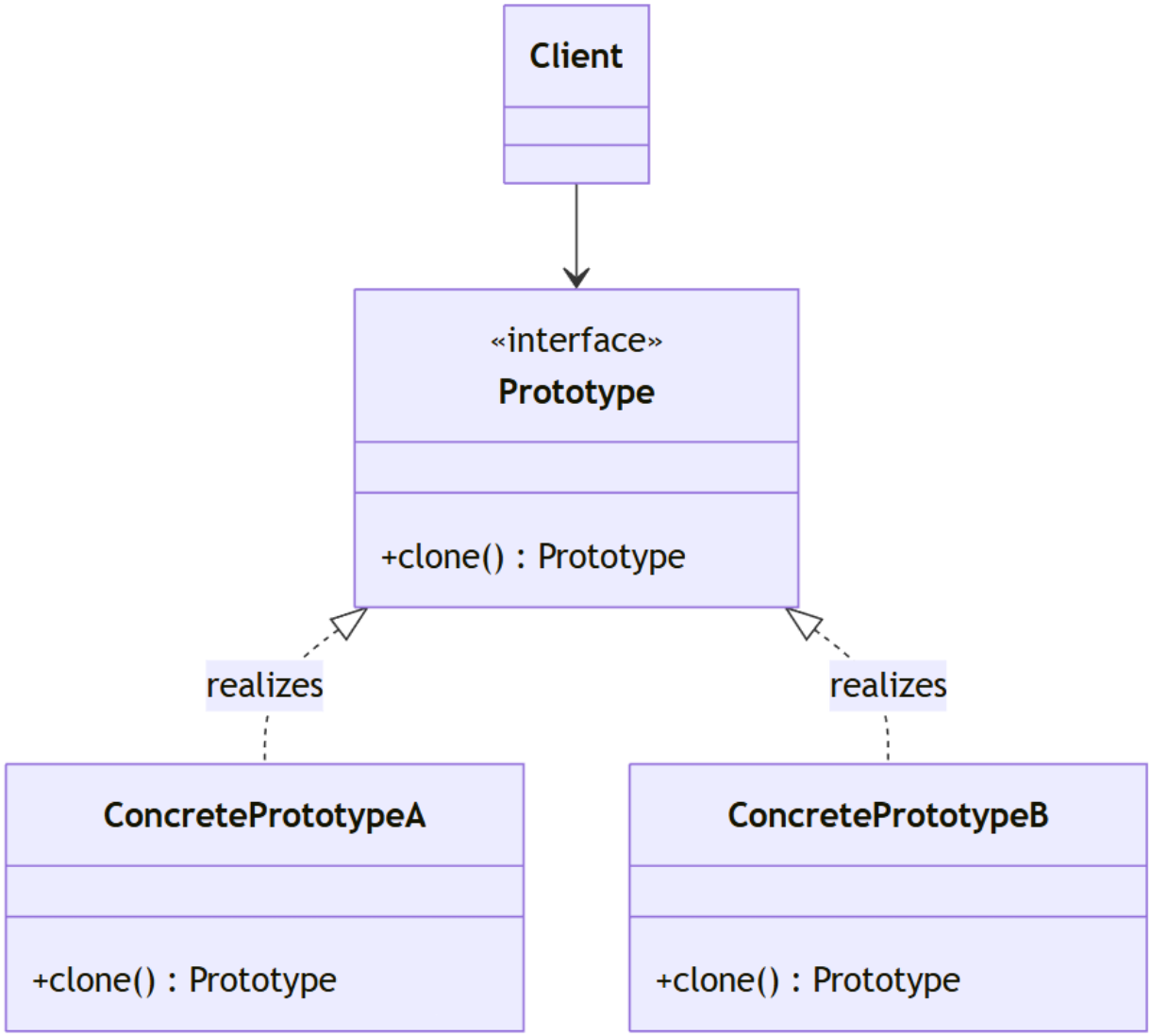
11) Які класи входять в шаблон «Команда», та яка між ними взаємодія?

Client, Invoker, Command, ConcreteCommand, Receiver. Client створює команду, Invoker виконує її, Receiver реалізує дію.

12) Розкажіть як працює шаблон «Команда». Команда інкапсулює запит як об’єкт, який передається Invoker до Receiver для виконання, підтримуючи додаткові функції (скасування, логування).

13) Яке призначення шаблону «Прототип»? Створює об’єкти клонуванням прототипу, зменшуючи ієрархію спадкування.

14) Нарисуйте структуру шаблону «Прототип».



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

Client, Prototype. Client клонувати об’єкт через метод Prototype.

16) Які можна привести приклади використання шаблону «Ланцюжок відповідальності»? Формування контекстного меню в UI, обробка запитів у ієрархії документів.