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:
Козир Я. 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 :

21. **Online radio station** (iterator, adapter, factory method, facade, visitor, client-server)

Додаток повинен служити сервером для радіостанції з можливістю мовлення на радіостанцію (64, 92, 128, 196, 224 kb/s) в потоковому режимі; вести облік підключених користувачів і статистику відвідувань і прослуховувань; налаштувати папки з піснями і можливість вести списки програвання або playlists (не відтворювати всі пісні).

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.

Система повинна конвертувати завантажені MP3-файли у формат HLS (сегментований потік). Для цього використовується потужна консольна утиліта FFmpeg (ffmpeg.exe). Вона працює як окремий процес операційної системи і не має нативного інтерфейсу для C# класів. Прямий виклик Process.Start у бізнес-логіці зробив би код залежним від конкретної утиліти та складним для тестування. Створено адаптер FFmpegAdapter, який реалізує внутрішній інтерфейс системи IAudioConverter. Цей клас бере на себе всю складність формування аргументів командного рядка, запуску процесу та обробки результатів, надаючи решті системи простий асинхронний метод ConvertToHlsAsync.

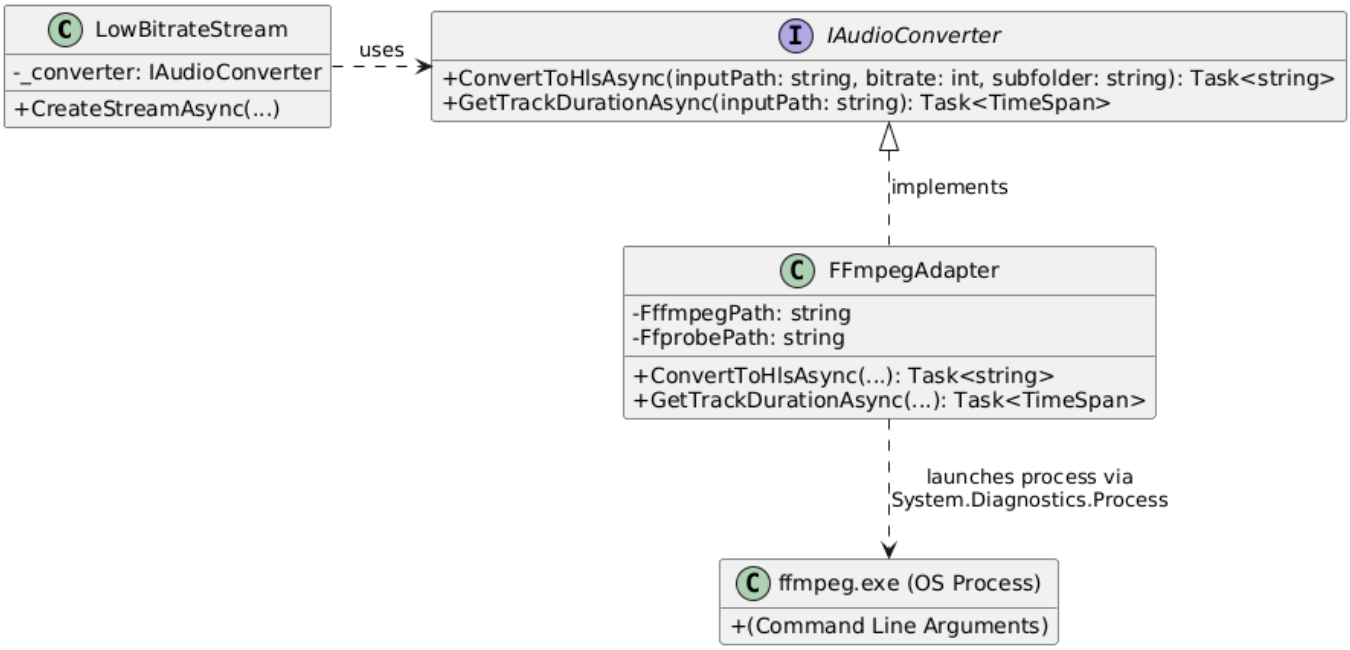


Рисунок 1 - Структура патерну Adapter

- IAudioConverter (Target): Інтерфейс, який очікує система. Описує метод конвертації та отримання тривалості.
- FFmpegAdapter (Adapter): Реалізує цей інтерфейс. Всередині методів формує аргументи для CLI (Command Line Interface) і запускає зовнішній процес.
- ffmpeg.exe (Adaptee): Зовнішня утиліта, яка виконує реальну роботу, але має несумісний інтерфейс (консольний ввід/вивід).
- LowBitrateStream (Client): Клас бізнес-логіки, який викликає метод конвертації, не знаючи, що під капотом працює окремий exe-файл.

```
public async Task<string> ConvertToHlsAsync(string inputPath, int bitrate, string subfolder)
{
    var baseFolder = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "streams", Path.GetFileNameWithoutExtension
    var outputFolder = Path.Combine(baseFolder, subfolder);
    Directory.CreateDirectory(outputFolder);

    var playlistPath = Path.Combine(outputFolder, "index.m3u8");
    var segmentPath = Path.Combine(outputFolder, "seg%03d.ts");

    var args = $"-i \"{inputPath}\" " +
               $"-c:a aac -b:a {bitrate}k " +
               $"-f hls -hls_time 10 -hls_list_size 0 " +
               $"-hls_segment_filename \"{segmentPath}\" " +
               $"\"{playlistPath}\"";

    var psi = new ProcessStartInfo
    {
        FileName = FfmpegPath,
        Arguments = args,
        UseShellExecute = false,
        RedirectStandardOutput = true,
        RedirectStandardError = true,
        CreateNoWindow = true
    };

    var outputBuilder = new StringBuilder();
    var errorBuilder = new StringBuilder();

    using var process = new Process { StartInfo = psi };

    process.OutputDataReceived += (sender, e) => { if (e.Data != null) outputBuilder.AppendLine(e.Data); };
    process.ErrorDataReceived += (sender, e) => { if (e.Data != null) errorBuilder.AppendLine(e.Data); };

    process.Start();
    process.BeginOutputReadLine();
```

Рисунок 2 – Реалізація Адаптера FfmpegAdapter.cs

```
using System.Threading.Tasks;

namespace OnlineRadioStation.Domain
{
    13 references
    public interface IAudioConverter
    {
        4 references
        Task<string> ConvertToHlsAsync(string inputPath, int bitrate, string subfolder);
        3 references
        Task<TimeSpan> GetTrackDurationAsync(string inputPath);


    }
}
```

Рисунок 3 – Цільовий інтерфейс IAudioConverter.cs

```
namespace OnlineRadioStation.Domain
{
    2 references
    public class LowBitrateStream : IAudioStream
    {
        2 references
        private readonly IAudioConverter _converter;
        2 references
        private const int Bitrate = 64;

        1 reference
        public LowBitrateStream(IAudioConverter converter)
        {
            _converter = converter;
        }

        1 reference
        public int GetBitrate() => Bitrate;

        2 references
        public async Task<string> CreateStreamAsync(string inputAudioPath, string subfolder)
        {
            return await _converter.ConvertToHlsAsync(inputAudioPath, Bitrate, subfolder);
        }
    }
}
```

Рисунок 4 – LowBitrateStream.cs

Рисунок 5 – Результат роботи адаптера: фізично створені файли HLS-потоку (сегменти) у файловій системі сервера

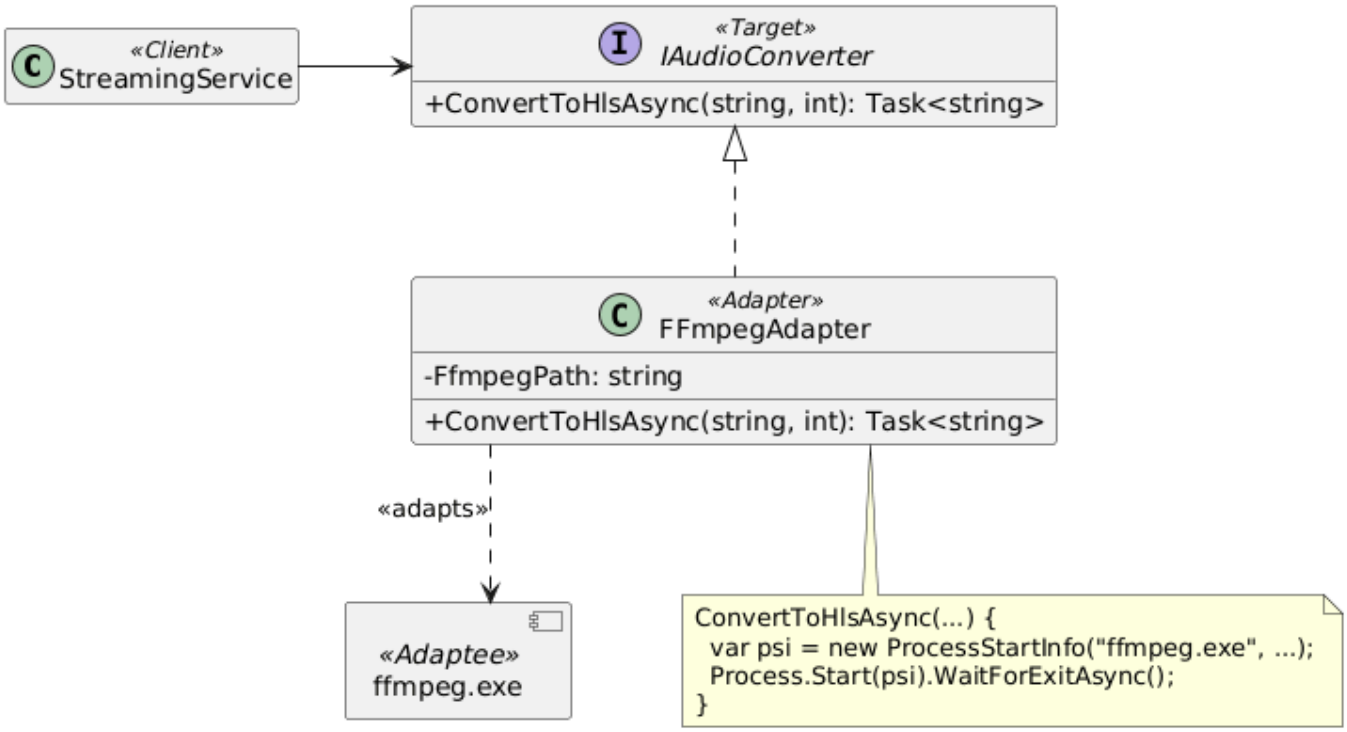
4. Висновок

У ході виконання лабораторної роботи було розглянуто базові шаблони проєктування, зокрема Adapter, Builder, Command та інші. Отримані знання дали можливість зрозуміти роль патернів у побудові гнучких програмних систем. На прикладі веб-застосунку «Online Radio Station» було реалізовано шаблон Adapter для інтеграції зовнішньої утиліти ffmpeg.exe. Було створено цільовий інтерфейс IAudioConverter, а клас FFmpegAdapter виступив "перехідником", що приховує складну логіку роботи з процесами операційної системи

(System.Diagnostics.Process). Це дало змогу відокремити бізнес-логіку класів потоків (LowBitrateStream, HighBitrateStream) від низькорівневої реалізації конвертації. Такий підхід робить систему гнучкою: якщо в майбутньому знадобиться замінити FFmpeg на іншу бібліотеку (наприклад, Xuggle), достатньо буде написати новий адаптер, не змінюючи код самої системи.

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».

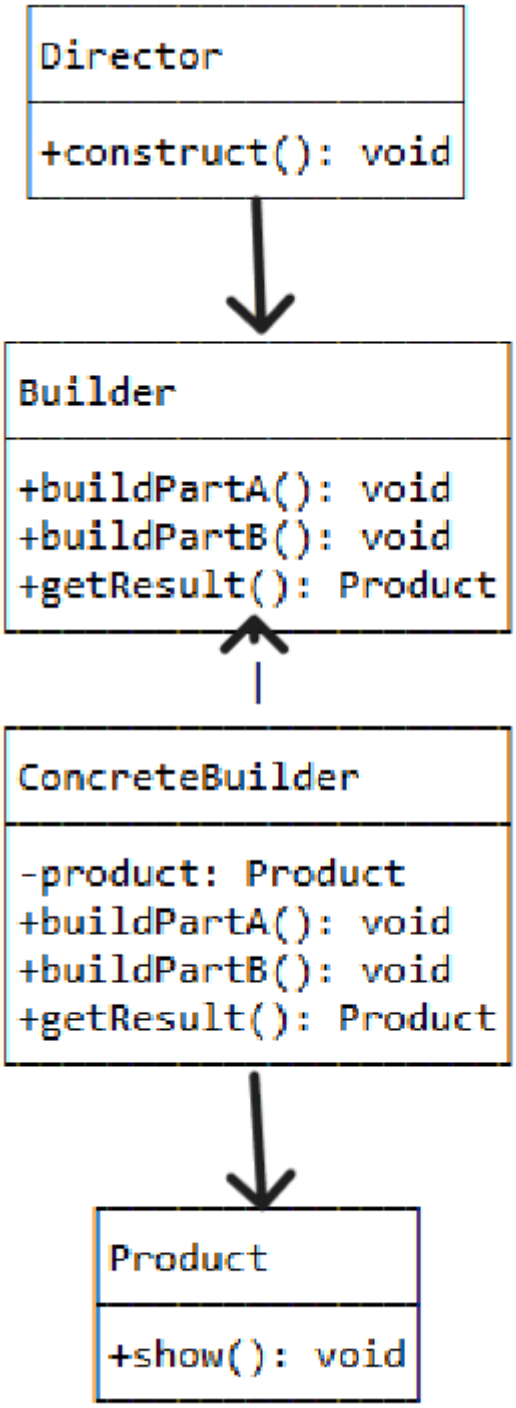


3) Які класи входять в шаблон «Адаптер», та яка між ними взаємодія?

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.

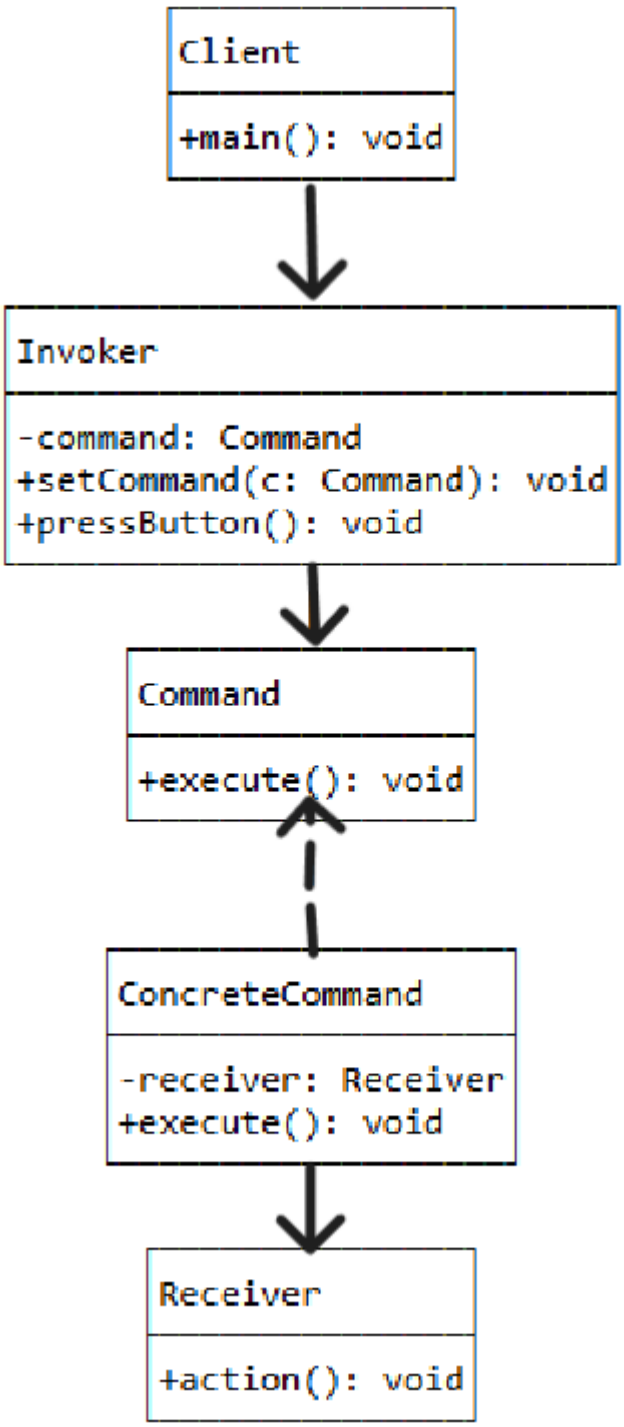
6) Нарисуйте структуру шаблону «Будівельник».



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».



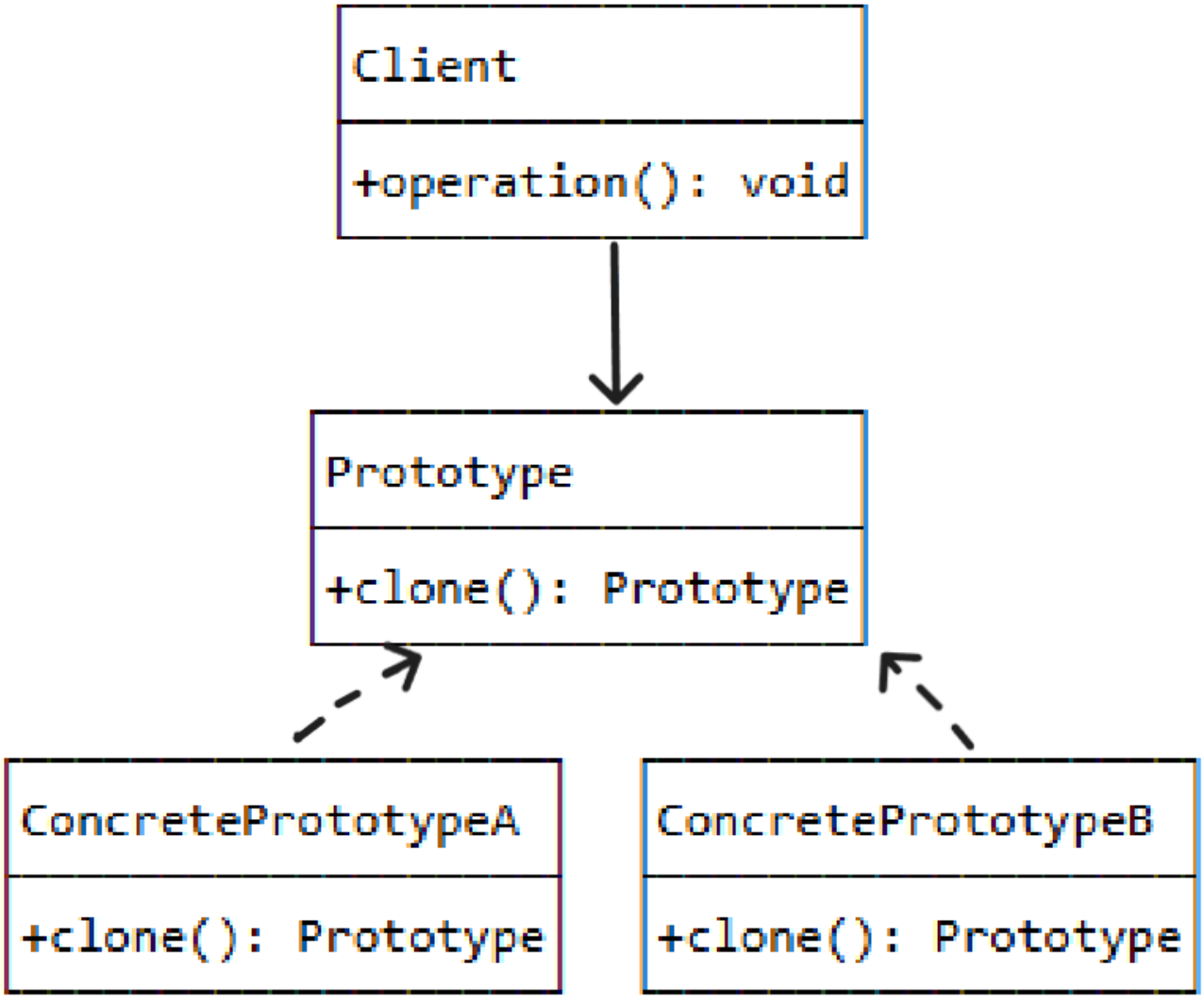
11) Які класи входять в шаблон «Команда», та яка між ними взаємодія?

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.