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 лабораторної роботи: «Патерни проектування»

Тема проєкта: «Аудіо редактор»

Виконала:
студентка групи ІА-33
Котик Іванна

Перевірив:
асистент кафедри ІСТ
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 проєкта: 5. Аудіо редактор (singleton, adapter, observer, mediator, composite, client server). Аудіо редактор повинен володіти наступним функціоналом: представлення аудіо даних будь-якого формату в WAVE-формі, вибір і подальші операції копіювання / вставки / вирізання / деформації по сегменту аудіозапису, можливість роботи з декількома звуковими доріжками, кодування в найбільш поширених форматах (ogg, flac, mp3).

Короткі теоретичні відомості:

Шаблон «Adapter»

Призначення патерну:

Шаблон "Adapter" (Адаптер) використовується для адаптації інтерфейсу одного об'єкту до іншого . Наприклад, існує декілька бібліотек для роботи з принтерами, проте кожна має різний інтерфейс (хоча однакові можливості і призначення). Має сенс розробити уніфікований інтерфейс (сканування, асинхронне сканування, двостороннє сканування, потокове сканування і тому подібне), і реалізувати відповідні адаптери для приведення бібліотек до уніфікованого інтерфейсу. Це дозволить в програмі звертатися до загального інтерфейсу, а не приводити різні сценарії роботи залежно від способу реалізації бібліотеки. Адаптери також називаються "wrappers" (обгортками). Структура патерну Адаптер на рівні об’єктів

Проблема: Ви реалізовуєте аудіо-плеєр, який може програвати аудіо різних форматів. Ви почали аналізувати і бачите що для програвання аудіо із різних форматів краще використовувати різні компоненти.

Таким чином ви реалізуєте складну логіку роботи з різними компонентами. У вас у коді з'являється багато логіки з перевіркою, якщо формат один, то викликаємо такий метод у такого компонента, якщо інший, то викликаємо декілька методів у іншого компонента, і т.д. Логіка роботи з компонентами стає досить складною та заплутаною.

Коли потрібно добавити підтримку нового аудіо-формату, то потрібно добавити роботу з новим компонентом і при цьому внести зміну в уже існуючу логіку, що може привести до помилок там де все коректно працювало.

Рішення: Для вирішення цих проблем можна використати патерн Адаптер. Визначаємо загальний інтерфейс IPlayer для програвання музики через компоненти. Далі для кожного компонента робимо свій адаптер. Адаптер має посилання на об'єкт компонента та реалізує інтерфейс викликаючи методи з компонента, який він адаптує. Таким чином, адаптер ніби обгортає специфічний 59 компонент і далі весь клієнтський код буде працювати з адаптерами через інтерфейс IPlayer. Класи, які будуть працювати з адаптером через цей інтерфейс не будуть знати інтерфейси кожного специфічного компонента. При такій реалізації, якщо буде потрібно добавити підтримку нового формату, то просто створюється новий адаптер, який обгортає новий компонент, але робота з цим адаптером буде виконуватися через існуючий інтерфейс. При цьому існуючий код не буде зазнавати змін, а значить і не "поламаємо" те що вже працює.

Переваги та недоліки:

+ Відокремлює інтерфейс або код перетворення даних від основної бізнес логіки.

+ Можна добавляти нові адаптери не змінюючи код у класі Client.

- Умовним недоліком можна назвати збільшення кількості класів, але за рахунок використання патерна Адаптер програмний код, як привило, стає легше читати.

Шаблон «Builder» (Будівельник)

Використовується для розділення процесу створення складного об’єкта від його представлення. Дає можливість створювати різні варіанти одного об’єкта, не змінюючи його структуру.

Основні елементи:

- Builder — інтерфейс для створення частин продукту.
- ConcreteBuilder — реалізація побудови конкретного продукту.
- Director — визначає послідовність кроків побудови.
- Product — кінцевий об’єкт.

Застосовується, коли об’єкт має багато кроків побудови або кілька варіантів створення.

Шаблон «Command» (Команда)

Перетворює запит у самостійний об’єкт, що містить усю інформацію про операцію.

Основні елементи:

- Command — інтерфейс з методом execute().
- ConcreteCommand — реалізує команду, викликаючи метод отримувача.
- Receiver — знає, як виконати дію.
- Invoker — ініціює виконання команди.
- Client — створює команду та передає її викликачеві.

Підтримує відкладене виконання, історію команд і можливість відміни дій.

Шаблон «Prototype» (Прототип)

Дозволяє створювати нові об’єкти копіюванням вже існуючих екземплярів.

Основні елементи:

- Prototype — інтерфейс із методом clone().
- ConcretePrototype — реалізує власне копіювання.
- Client — створює нові об’єкти через клонування прототипів.

Застосовується, коли створення об’єкта є складним або ресурсоємним.

Шаблон «Chain of Responsibility» (Ланцюжок відповідальності)

Дозволяє передавати запит уздовж ланцюга об’єктів, доки один із них не обробить його.

Основні елементи:

- Handler — базовий клас, який має посилання на наступний обробник.
- ConcreteHandler — перевіряє, чи може обробити запит, інакше передає далі.
- Client — створює та з’єднує ланцюг обробників.

Приклади використання: системи логування, обробка подій, механізми доступу.

Хід роботи:

1. Діаграма класів.

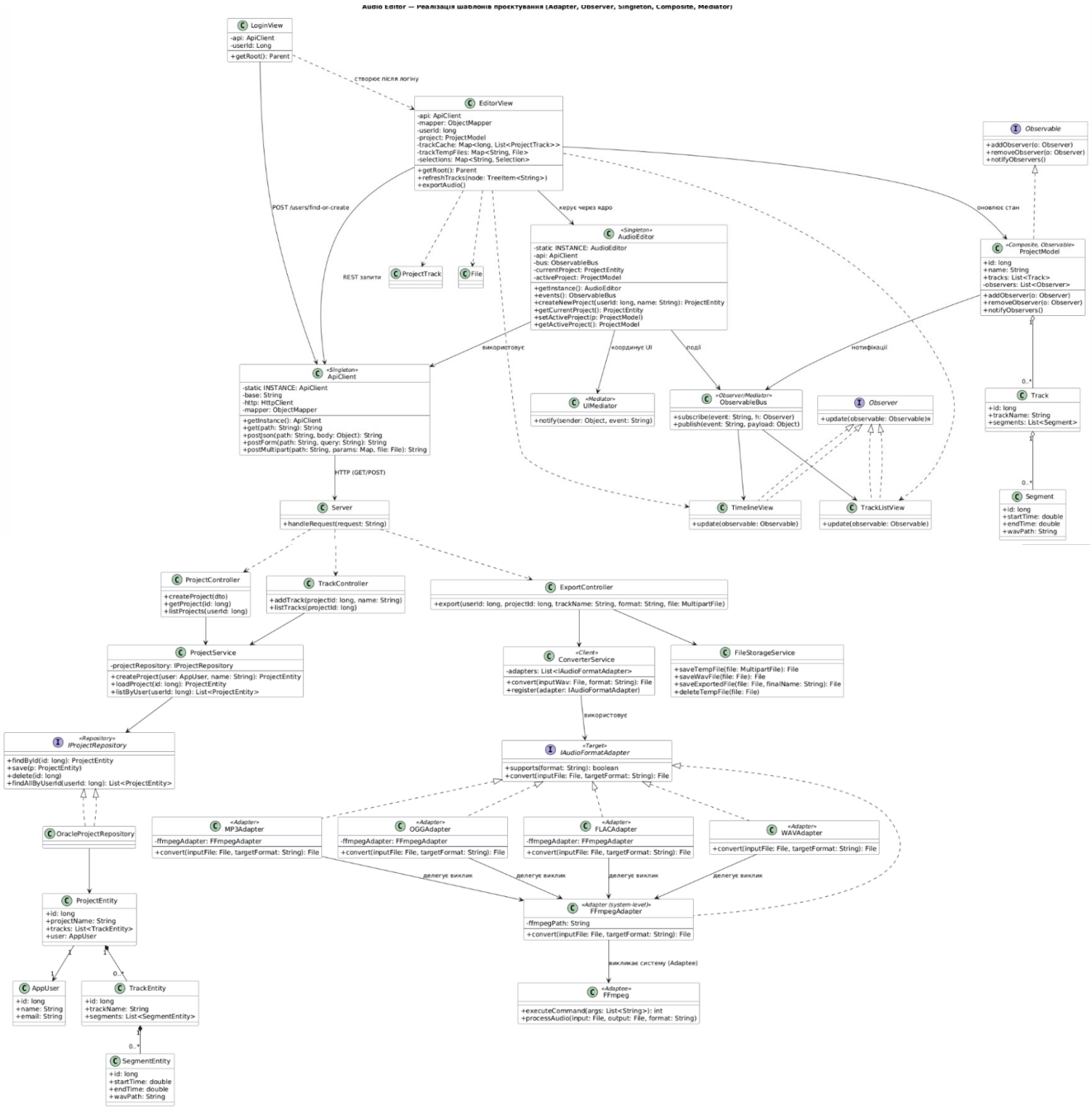


Рисунок 1.1 – Діаграма класів

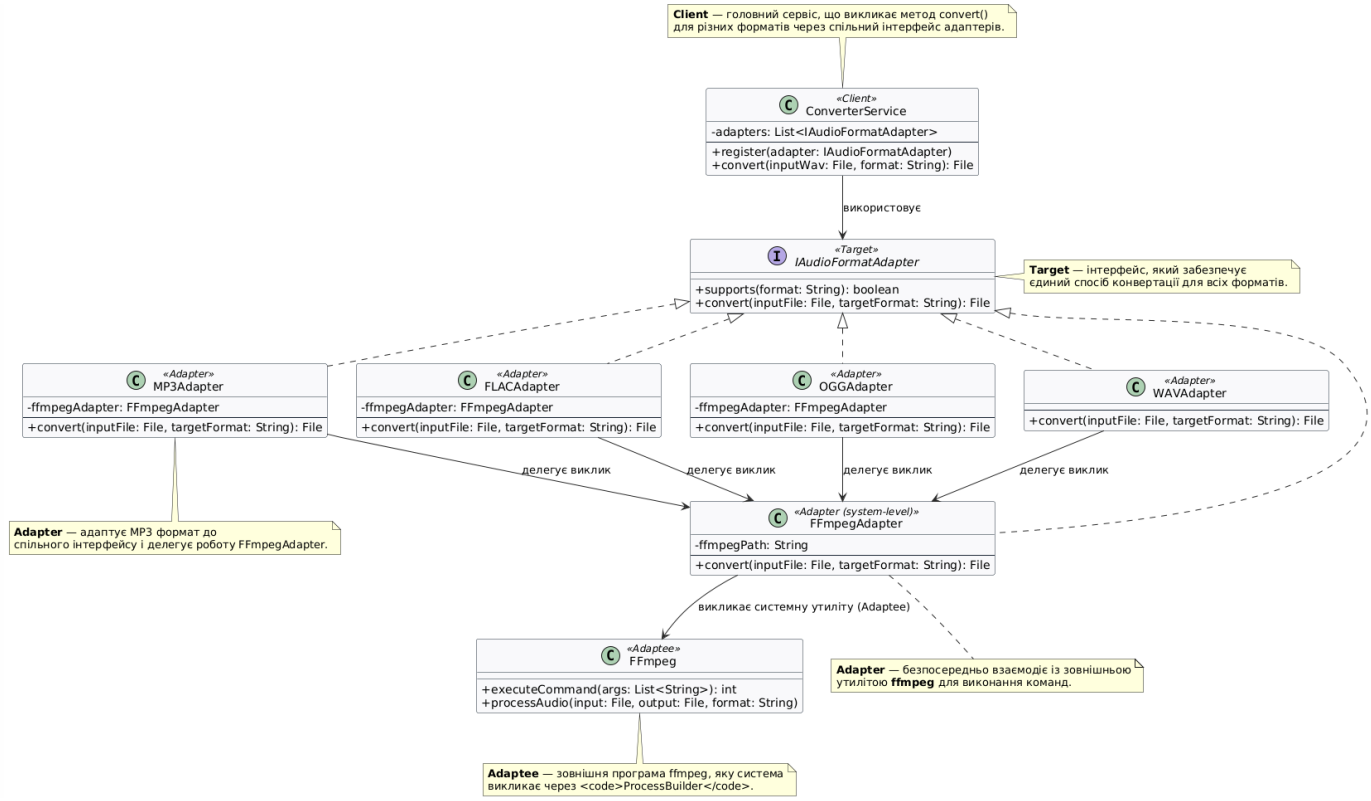


Рисунок 1.2 – Представлення патенту «Адаптер» на діаграмі класів

2.1. Реалізація шаблону «Адаптер» у системі для Audio Editor.

Шаблон «Adapter» використовується для узгодження інтерфейсів об’єктів, які не сумісні між собою. Його суть полягає в тому, щоб створити спеціальний клас-адаптер, який «обгортає» сторонній або існуючий компонент і забезпечує взаємодію з ним через уніфікований інтерфейс. Такий підхід дозволяє підключати різні бібліотеки або зовнішні системи без зміни основного коду програми.

Проблема:

Під час розробки програмного забезпечення Audio Editor виникла потреба експортувати аудіо у різних форматах — MP3, OGG, FLAC, WAV.

Початкова реалізація включала велику кількість умовних перевірок (if-else) для кожного формату, що робило код громіздким і складним для розширення.

Додавання нового формату вимагало змін у вже існуючій логіці, що могло призвести до помилок та зниження стабільності програми.

Рішення:

Для усунення цих недоліків було застосовано шаблон Adapter.

Було створено уніфікований інтерфейс IAudioFormatAdapter, який визначає спільні методи для всіх форматів.

Кожен конкретний адаптер (MP3Adapter, OGGAdapter, FLACAdapter, WAVAdapter) реалізує цей інтерфейс і делегує виконання реальної конвертації класу FFmpegAdapter.

FFmpegAdapter, у свою чергу, взаємодіє із зовнішньою утилітою ffmpeg (яка є Adaptee) через системні виклики (ProcessBuilder).

Клієнтський код (ConverterService) звертається лише до спільного інтерфейсу і не залежить від конкретної реалізації адаптера.

2.2. Код реалізації шаблона «Адаптер».

Клас ConverterService:

```
@Service
public class ConverterService {
    private final IAudioFormatAdapter adapter;
    private final FileStorageService storage;

    public ConverterService(
            @Qualifier("FFmpegAdapter") IAudioFormatAdapter adapter,
```

```
            FileStorageService storage
        ) {
            this.adapter = adapter;
            this.storage = storage;
        }

        public File export(File wavFile, String targetFormat, String exportBaseName) throws Exception
{
            File converted = adapter.convert(wavFile, targetFormat);
            String finalName = exportBaseName + "." + targetFormat.toLowerCase();
            return storage.saveExportedFile(converted, finalName);
        }
    }
```

Інтерфейс IAudioFormatAdapter:

```
    public interface IAudioFormatAdapter {
        File convert(File inputFile, String targetFormat) throws Exception;
    }
```

Клас MP3Adapter:

```
    public class MP3Adapter implements IAudioFormatAdapter {

        private final FFmpegAdapter ffmpegAdapter;

        public MP3Adapter(FFmpegAdapter ffmpegAdapter) {
            this.ffmpegAdapter = ffmpegAdapter;
        }

        @Override
```

```
    public File convert(File inputFile, String targetFormat) throws Exception {
        System.out.println("Використовується MP3Adapter → FFmpeg");
        return ffmpegAdapter.convert(inputFile, "mp3");
    }
}
```

Клас FFmpegAdapter:

```
@Component
public class FFmpegAdapter implements IAudioFormatAdapter {

    @Value("${ffmpeg.path:ffmpeg}")
    private String ffmpegPath;

    @Override
    public File convert(File inputFile, String targetFormat) throws Exception {
        String baseName = inputFile.getName().replaceAll("\\.[^.]+$", "");
        File out = File.createTempFile(baseName + "-", "." + targetFormat);

        List<String> cmd = new ArrayList<>();
        cmd.add(ffmpegPath);
        cmd.add("-y");
        cmd.add("-i");
        cmd.add(inputFile.getAbsolutePath());
        if ("mp3".equalsIgnoreCase(targetFormat)) {
            cmd.add("-codec:a"); cmd.add("libmp3lame");
            cmd.add("-qscale:a"); cmd.add("2");
        }
        if ("ogg".equalsIgnoreCase(targetFormat)) {
```

```
            cmd.add("-codec:a"); cmd.add("libvorbis");
            cmd.add("-qscale:a"); cmd.add("5");
        }
        cmd.add(out.getAbsolutePath());

        ProcessBuilder pb = new ProcessBuilder(cmd);
        pb.redirectErrorStream(true);
        Process p = pb.start();
        try (BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream(),
StandardCharsets.UTF_8))) {
            while (br.readLine() != null) {  }
        }
        int code = p.waitFor();
        if (code != 0) {
            throw new IOException("ffmpeg exited with " + code);
        }
        return out;
    }
}
```

2.3. Пояснення коду реалізації шаблону «Адаптер».

У застосунку Audio Editor шаблон Adapter використано для узгодження інтерфейсу зовнішньої утиліти ffmpeg з внутрішнім сервісом конвертації аудіо. Метою є приховати складність взаємодії з командною утилітою та забезпечити роботу програми через єдиний інтерфейс.

Інтерфейс IAudioFormatAdapter визначає спільний метод convert(), який реалізують усі адаптери.

Клас FFmpegAdapter виступає як адаптер і реалізує цей інтерфейс, перетворюючи виклики Java-коду у команди для зовнішньої програми ffmpeg.

Саме тут відбувається делегування: адаптер не виконує конвертацію самостійно, а передає цю задачу утиліті ffmpeg через об’єкт ProcessBuilder.

Таким чином, FFmpegAdapter адаптує зовнішній інтерфейс ffmpeg до внутрішнього стандарту системи.

Клас ConverterService виконує роль Client — він взаємодіє не з конкретним адаптером, а з інтерфейсом IAudioFormatAdapter.

Через інжекцію залежності (@Qualifier("FFmpegAdapter")) він отримує реалізацію адаптера і викликає adapter.convert(), не знаючи про те, як саме виконується конвертація.

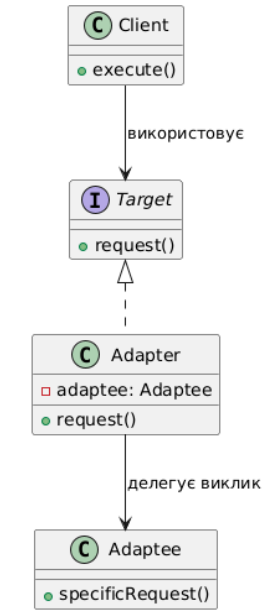
Отриманий результат передається у FileStorageService для збереження.

Відповіді на контрольні запитання:

1. *Яке призначення шаблону «Адаптер»?*

Шаблон Adapter використовується для адаптації інтерфейсу одного класу до іншого, коли їхні інтерфейси несумісні. Він дозволяє об’єктам із різними інтерфейсами працювати разом, обгортаючи один об’єкт у спеціальний адаптер, що реалізує потрібний інтерфейс.

2. *Нарисуйте структуру шаблону «Адаптер».*



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

Client (Клієнт) – виконує запити через стандартний інтерфейс, не знаючи про конкретну реалізацію.

Target (Цільовий інтерфейс) – визначає інтерфейс, з яким очікує працювати клієнт.

Adapter (Адаптер) – реалізує інтерфейс Target і перетворює виклики клієнта у формат, зрозумілий адаптованому класу.

Adaptee (Адаптований клас) – має несумісний інтерфейс, який потрібно адаптувати для клієнта.

Клас Client викликає методи через інтерфейс Target. Об’єкт Adapter перехоплює цей виклик і делегує його класу Adaptee, який безпосередньо виконує потрібну дію. Таким чином, адаптер узгоджує інтерфейси, дозволяючи об’єктам, що не сумісні між собою, працювати разом.

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

класів?

Адаптер на рівні об’єктів — реалізує адаптацію за допомогою композиції:

адаптер містить посилання на об’єкт іншого класу (adaptee) і делегує йому виклики методів.

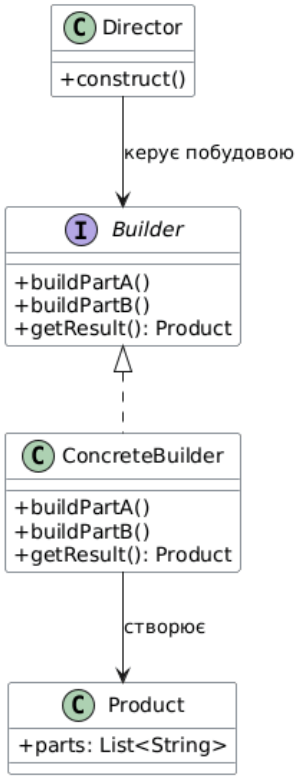
Адаптер на рівні класів — реалізує адаптацію через успадкування:

адаптер наслідує як інтерфейс, який потрібно забезпечити (Target), так і клас, який адаптується (Adaptee).

5. *Яке призначення шаблону «Будівельник»?*

Шаблон «Builder» (Будівельник) використовується для відділення процесу створення об'єкту від його представлення.

6. *Нарисуйте структуру шаблону «Будівельник».*



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

Builder — інтерфейс, який визначає основні кроки створення об’єкта.

ConcreteBuilder — реалізує конкретні кроки побудови продукту.

Director — визначає порядок виклику методів Builder.

Product — кінцевий об’єкт, який створюється.

Director використовує об’єкт типу Builder, викликаючи його методи у певній послідовності. ConcreteBuilder реалізує ці методи й створює Product, який потім повертається клієнту.

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

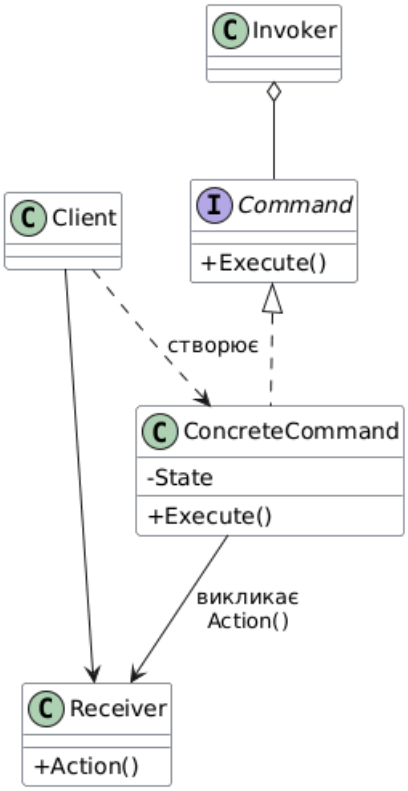
Слід використовувати шаблон Будівельник коли:

- алгоритм створення складного об'єкта не повинен залежати від того, з яких частин складається об'єкт та як вони стикуються поміж собою;
- процес конструювання повинен забезпечити різні подання об'єкта, що конструюється.

9. Яке призначення шаблону «Команда»?

Шаблон «command» (команда) перетворить звичайний виклик методу в клас. Таким чином дії в системі стають повноправними об'єктами.

10.Нарисуйте структуру шаблону «Команда».



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

Command — оголошує інтерфейс для виконання операції;

ConcreteCommand — визначає зв'язок між об'єктом-отримувачем Receiver та дією, реалізує операцію Execute шляхом виклику відповідних операцій об'єкта Receiver;

Client — створює об'єкт класу ConcreteCommand та встановлює його отримувача;

Invoker — звертається до команди щоб та виконала запит;

Receiver — має у своєму розпорядженні усю інформацію про способи виконання операцій, необхідних для задоволення запиту. У ролі отримувача може виступати будь-який клас.

Клієнт створює об'єкт ConcreteCommand та встановлює для нього отримувача. Викликач Invoker зберігає об'єкт ConcreteCommand. Викликач надсилає запит,

викликаючи операцію команди Execute. Якщо підтримується скасування виконаних дій, то ConcreteCommand перед викликом Execute зберігає інформацію про стан, достатню для виконання скасування. Об'єкт ConcreteCommand викликає операції отримувача для виконання запиту.

12.Розкажіть як працює шаблон «Команда».

Шаблон «Команда» перетворює виклик методу на окремий об’єкт, завдяки чому дії в системі стають незалежними та керованими елементами.

Об’єкт команди не виконує дію самостійно, а перенаправляє запит об’єкту-отримувачу (Receiver). Такий підхід дозволяє створювати гнучку систему команд, де можна:

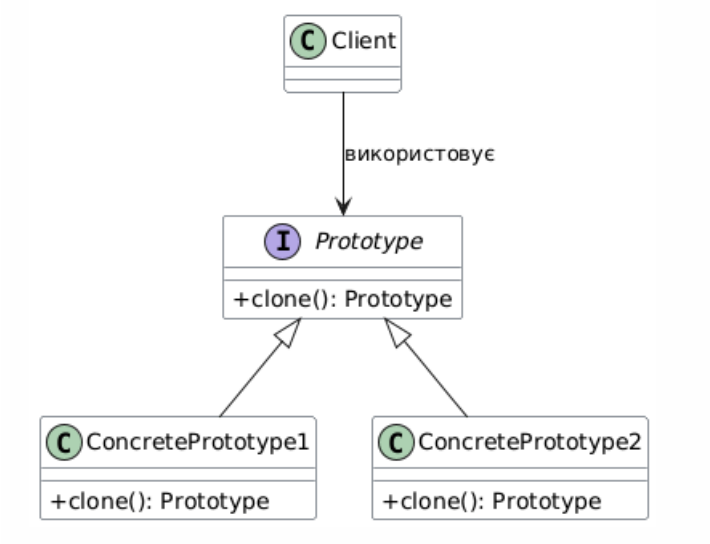
- додавати нові команди без зміни існуючого коду;
- організовувати черги виконання команд;
- реалізовувати відміну, повторення або логування дій.

Наприклад, команда «вставити символ» може зберігати параметри дії (символ, позицію) і, при потребі, виконати відміну шляхом виклику зворотної дії «видалити символ».

13.Яке призначення шаблону «Прототип»?

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.

14.Нарисуйте структуру шаблону «Прототип».



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

Prototype — визначає інтерфейс для клонування самого себе;

ConcretePrototype — реалізує операцію клонування самого себе;

Client — створює новий об'єкт, звертаючись до прототипу із запитом клонувати себе.

Клієнт звертається до прототипу, щоб той створив свого клона.

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

Приклади використання:

Обробка подій у графічних інтерфейсах — наприклад, коли натискання клавіші передається від кнопки до вікна, а потім до системного рівня, поки не буде оброблено.

Системи підтримки користувачів — запит проходить через рівні операторів (1-й, 2-й, 3-й рівень), доки хтось не знайде рішення.

Логування подій — повідомлення проходить через кілька логерів (консоль, файл, віддалений сервер), кожен з яких може обробити або передати далі.

Системи доступу та перевірки прав користувача — якщо перший об’єкт не має права прийняти рішення, запит передається наступному.

Висновки:

У рамках лабораторної роботи було реалізовано шаблон проєктування Adapter на прикладі програмного застосунку Audio Editor. Під час роботи шаблон використано для узгодження інтерфейсу зовнішньої утиліти FFmpeg із внутрішніми сервісами системи через інтерфейс IAudioFormatAdapter. Це дало змогу забезпечити конвертацію аудіофайлів у різні формати (MP3, OGG, FLAC, WAV) без зміни основного коду та спростити розширення функціоналу.

Також було вивчено та проаналізовано структурні й поведінкові шаблони проєктування — Adapter, Builder, Command, Prototype та Chain of Responsibility.