Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №4
Технологія розробки програмного забезпечення
«Вступ до паттернів проектування»

Виконала
студентка групи ІА–32:
Ткачук М. С.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

Теоретичні відомості 4
Діаграма класів 7
Патерн Singleton 9
Фрагменти коду 11
Вихідний код 16
Висновок: 16
Контрольні запитання 17

Тема: Вступ до паттернів проектування

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Завдання:

Ознайомитись з короткими теоретичними відомостями.

Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
Реалізувати один з розглянутих шаблонів за обраною темою.

Реалізувати не менше 3-х класів відповідно до обраної теми.

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.

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).

Теоретичні відомості

Шаблон проєктування – це формалізоване, багаторазово використане рішення типових задач проєктування. Кожен патерн має назву, опис проблеми та спосіб її вирішення. Використання патернів допомагає створювати зрозумілі, структуровані та гнучкі системи, які легше підтримувати й розширювати. Патерни також працюють як спільна «мова» між розробниками, дозволяючи швидко пояснювати архітектурні рішення.

Шаблон Singleton

Призначення: забезпечує існування лише одного екземпляра класу та глобальну точку доступу до нього.

Використовується, коли:

- у системі повинен бути один об’єкт певного типу (наприклад, конфігурації);
- необхідно централізувати контроль над операціями.

Переваги: один екземпляр, глобальний доступ.

Недоліки: порушення принципу єдиної відповідальності, складне тестування, ризик поганого дизайну.

Шаблон Iterator

Призначення: забезпечує послідовний обхід елементів колекції без розкриття її внутрішньої структури.

Ідея: логіка перебору виноситься з колекції в окремий клас-ітератор.

Можливості: різні способи обходу (вперед, назад, за особливим правилом).

Переваги:

- уніфікований доступ до колекцій різних типів;
- спрощення класів колекцій.

  Недолік: може бути надмірним для простих структур.

Шаблон Proxy

Призначення: створює об’єкт-замісник, який контролює доступ до реального об’єкта та може додавати додаткову логіку (кешування, контроль, оптимізація).
Приклад: відкладена робота із зовнішнім сервісом, зменшення кількості запитів через проміжний рівень.

Переваги:

- зміна поведінки без змін у клієнтському коді;
- можливість контролю й оптимізації викликів.

  Недоліки:
- потенційне уповільнення;
- ризик некоректної поведінки замісника.

Шаблон State

Призначення: дозволяє змінювати поведінку об’єкта залежно від його внутрішнього стану.
Ідея: кожен стан винесений у окремий клас, а об’єкт-контекст делегує виклики поточному стану.

Переваги:

- логіка різних станів відокремлена;
- легко додавати нові стани;
- код контексту чистіший.

  Недолік: складніша структура, більше класів.

Шаблон Strategy

Призначення: забезпечує можливість підміняти алгоритми під час виконання програми.

Ідея: різні алгоритми оформлюються як окремі стратегії, а об'єкт-контекст лише викликає поточну стратегію.

Приклад: різні способи сортування, різні політики обробки даних.

Переваги:

- алгоритми взаємозамінні;
- менше умовних операторів у коді;
- стратегії можна використовувати повторно.

  Недолік: збільшення кількості класів при простих задачах.

Хід Роботи

Діаграма класів

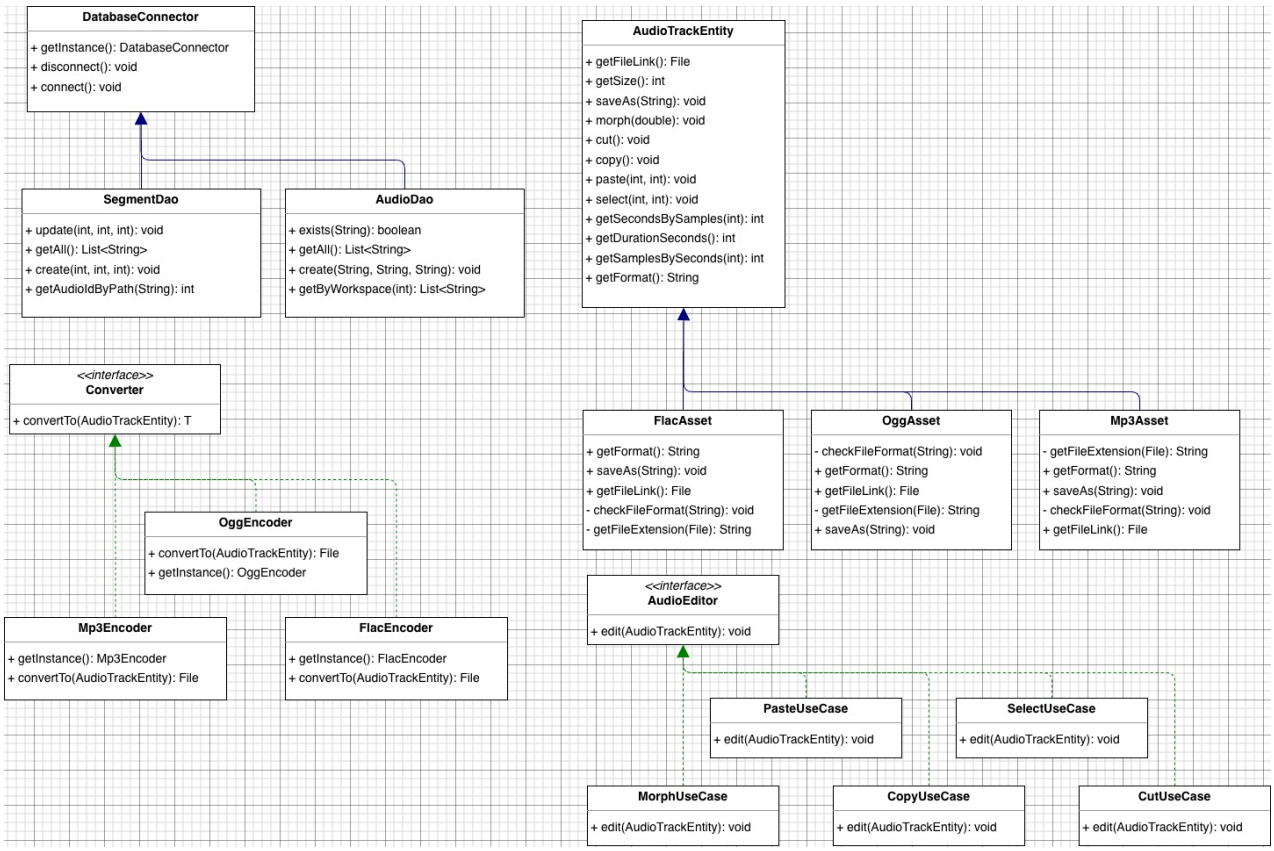


Рисунок 1 – Діаграма класів

На рисунку 1 накведено діаграму класів аудіоредактора, що відображає логічну структуру програмного забезпечення, основні класи, їхні атрибути, методи та взаємозв’язки між ними.

Центральним елементом системи є клас AudioTrackEntity, який описує сутність аудіотреку та містить методи для виконання основних операцій редагування cut(), copy(), paste(), morph(), а також методи для роботи з форматом файлу, отримання семплів і тривалості (getDurationSeconds(), getFormat(), getSampleBySeconds()).

Цей клас є базовим для роботи з різними типами звукових файлів і виступає ядром усієї системи.

Від AudioTrackEntity наслідуються класи Mp3Asset, OggAsset та FlacAsset, що реалізують підтримку різних аудіоформатів. Кожен із цих класів має власні методи для збереження (saveAs()), перевірки формату (checkFileFormat()), а також отримання розширення файлу (getFileExtension()), що дозволяє системі працювати з популярними форматами MP3, OGG та FLAC.

Окремий клас AudioEditor реалізує високорівневу логіку редагування файлів. Він містить метод edit(), який викликає відповідні операції залежно від сценарію редагування: копіювання, вставлення, вирізання або деформації звуку. Для реалізації цих сценаріїв використовуються окремі класи CutUseCase, CopyUseCase, SelectUseCase і MorphUseCase, кожен із яких виконує власну операцію редагування через метод edit(AudioTrackEntity).

Для кодування звукових доріжок у різні формати використовується абстрактний клас Converter<T>, який містить універсальний метод convertTo(AudioTrackEntity). Від нього наслідуються конкретні реалізації Mp3Encoder, OggEncoder та FlacEncoder, які забезпечують перетворення аудіо у відповідний формат.

Компонент DatabaseConnector відповідає за взаємодію з локальною базою даних SQLite. Він реалізує підключення, відключення та отримання екземпляра підключення (getInstance(), connect(), disconnect()). З ним працюють два DAO-класи AudioDao і SegmentDao, які реалізують доступ до даних аудіофайлів і сегментів.
AudioDao забезпечує методи create(), getAll(), exists() і getByWorkspace(), а SegmentDao: операції create(), update() та getAudioIdByPath(). Це дає змогу системі зберігати інформацію про проєкти, сегменти та аудіофайли у структурованому вигляді.

Діаграма класів демонструє, як система аудіоредактора розділена на логічні шари:

- рівень даних (DAO-класи, DatabaseConnector),
- рівень бізнес-логіки (AudioEditor, UseCase-класи),
- рівень моделей (AudioTrackEntity і похідні класи для форматів),
- рівень перетворень (Converter і Encoder-класи).

Патерн Singleton

Патерн Singleton це підхід, який гарантує, що в програмі існуватиме лише один екземпляр певного класу, а доступ до нього здійснюється через глобальну точку входу. Така модель є доцільною у випадках, коли:

- створення об’єкта є ресурсозатратним;
- об’єкт повинен мати єдиний узгоджений стан у межах усієї системи;
- наявність множинних екземплярів може спричинити логічні конфлікти або надмірне навантаження.

Singleton широко застосовується у програмних системах, де необхідно централізовано керувати ресурсами, наприклад підключенням до бази даних, роботою з файловою системою або службами обробки даних. Класичне визначення й обґрунтування цього патерну наведено в одній із найвідоміших праць у сфері шаблонів проєктування у книзі "Design Patterns" авторства Gamma та ін. [9].

У цьому проєкті патерн Singleton був обраний для реалізації енкодерів аудіофайлів (OGG, FLAC, MP3). Створення кодувальних модулів є ресурсоємним і не потребує множинних екземплярів, тому використання одного глобального об’єкта забезпечує стабільну та передбачувану роботу системи. Такий підхід дозволяє зменшити навантаження на пам’ять, уникнути дублювання внутрішніх структур і забезпечити централізований контроль над

процесом кодування. Схему реалізації патерну Singleton для трьох енкодерів наведено на рисунку 2.3.

Інтерфейс AudioEncoder задає загальний контракт: кожен енкодер повинен приймати аудіосегмент і повертати результат у вигляді згенерованого файлу.

Кожен клас енкодера:

- має приватний конструктор, що забороняє створення нових екземплярів іззовні;
- містить приватне статичне поле sharedReference, у якому зберігається єдиний екземпляр класу;
- надає публічний статичний метод get(), що створює інстанцію під час першого звернення та повертає ту саму копію під час наступних викликів;

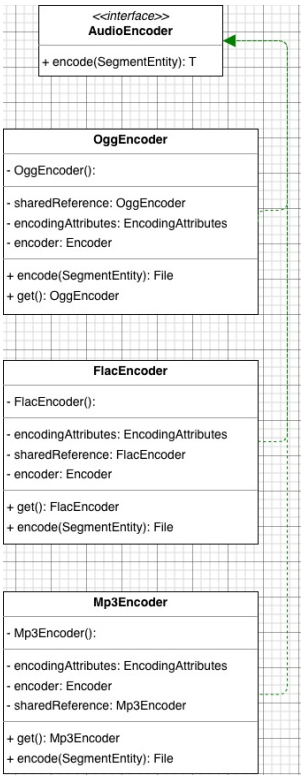


Рисунок 2 – Схема патерну singleton

Фрагменти коду:

AudioEncoder.java

```
package app.soundlab.audioencoder;

public interface AudioEncoder {
    String encode(String resourceName);
}
```

FlacEncoder.java

```
package app.soundlab.audioencoder;

public class FlacEncoder implements AudioEncoder {
    private static volatile FlacEncoder sharedReference;
    private final String formatName = "FLAC";

    private FlacEncoder() {
    }

    public static FlacEncoder get() {
        FlacEncoder current = sharedReference;
        if (current == null) {
            synchronized (FlacEncoder.class) {
                current = sharedReference;
                if (current == null) {
                    current = new FlacEncoder();
                    sharedReference = current;
```

```
            }
        }
    }
    return current;
  }

  @Override
  public String encode(String resourceName) {
    return "[FlacEncoder] Converted '%s' to %s format.".formatted(resourceName,
formatName);
  }
}
```

Mp3Encoder.java

```
package app.soundlab.audioencoder;

public class Mp3Encoder implements AudioEncoder {
  private static volatile Mp3Encoder sharedReference;
  private final String formatName = "MP3";

  private Mp3Encoder() {
  }

  public static Mp3Encoder get() {
    Mp3Encoder current = sharedReference;
    if (current == null) {
      synchronized (Mp3Encoder.class) {
        current = sharedReference;
```

```
            if (current == null) {
                current = new Mp3Encoder();
                sharedReference = current;
            }
        }
    }
    return current;
}

@Override
public String encode(String resourceName) {
    return "[Mp3Encoder] Converted '%s' to %s format.".formatted(resourceName,
formatName);
}
}
```

OggEncoder.java

```
package app.soundlab.audioencoder;

public class OggEncoder implements AudioEncoder {
    private static volatile OggEncoder sharedReference;
    private final String formatName = "OGG";

    private OggEncoder() {
    }

    public static OggEncoder get() {
        OggEncoder current = sharedReference;
```

```
        if (current == null) {
            synchronized (OggEncoder.class) {
                current = sharedReference;
                if (current == null) {
                    current = new OggEncoder();
                    sharedReference = current;
                }
            }
        }
        return current;
    }

    @Override
    public String encode(String resourceName) {
        return "[OggEncoder] Converted '%s' to %s format.".formatted(resourceName,
formatName);
    }
}
```

App.java

```
package app.soundlab;

import app.soundlab.audioencoder.AudioEncoder;
import app.soundlab.audioencoder.FlacEncoder;
import app.soundlab.audioencoder.Mp3Encoder;
import app.soundlab.audioencoder.OggEncoder;

public final class App {
```

```
    public static void main(String[] args) {
        System.out.println("Singleton encoder demo:");

        demonstrate("lofi-beat.wav", Mp3Encoder.get());
        demonstrate("podcast-episode.wav", FlacEncoder.get());
        demonstrate("game-soundtrack.wav", OggEncoder.get());

        AudioEncoder mp3First = Mp3Encoder.get();
        AudioEncoder mp3Second = Mp3Encoder.get();

        System.out.println();
        System.out.println("Are both MP3 encoder references identical?: " + (mp3First == mp3Second));
        System.out.println("(Same object, because only one encoder instance exists in the JVM.)");
    }

    private static void demonstrate(String resourceName, AudioEncoder encoder) {
        System.out.println();
        System.out.printf("Requesting %s for '%s'%n", encoder.getClass().getSimpleName(), resourceName);
        System.out.println("Result: " + encoder.encode(resourceName));
    }
}
```

Вихідний код:
https://github.com/mandarinchik21/AudioEditor/tree/main/lab4

Висновок:

У ході виконання лабораторної роботи було опрацьовано основні шаблони проєктування та їх роль у створенні гнучких, масштабованих і підтримуваних програмних систем. На прикладі аудіоредактора було розглянуто практичне застосування шаблонів, зокрема реалізовано патерн Singleton, що забезпечує контрольований доступ до єдиного екземпляра енкодерів аудіофайлів. Це дозволило оптимізувати використання ресурсів та централізувати процес кодування. Побудована діаграма класів показала логічну структуру системи та взаємодію між моделями, бізнес-логікою, DAO-рівнем і компонентами кодування. Реалізація кількох класів і демонстрація їхньої взаємодії підтвердили важливість правильного вибору патернів при створенні складних модулів, таких як редактор аудіо. Робота дала змогу краще зрозуміти призначення та переваги шаблонів Singleton, Iterator, Proxy, State і Strategy, а також навички практичного застосування патернів у програмній архітектурі.

Контрольні запитання

1. Що таке шаблон проєктування?

Це типове, багаторазове рішення поширеної задачі проєктування, оформлене як готова схема або підхід

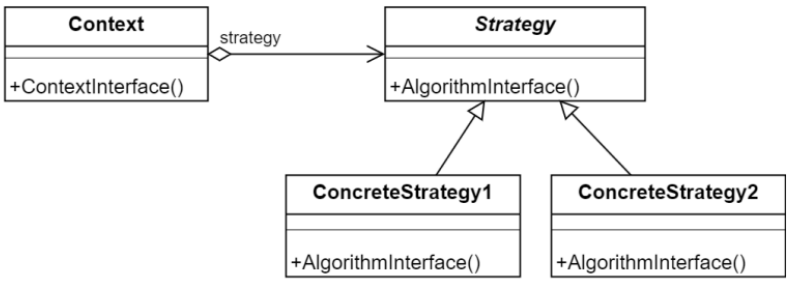
2. Навіщо використовувати шаблони проєктування?

Щоб спростити архітектуру, підвищити гнучкість системи, зменшити дублювання коду та полегшити підтримку.

3. Яке призначення шаблону «Стратегія»?

Дозволяє замінювати алгоритм роботи об’єкта іншим алгоритмом під час виконання програми.

4. Нарисуйте структуру шаблону «Стратегія».



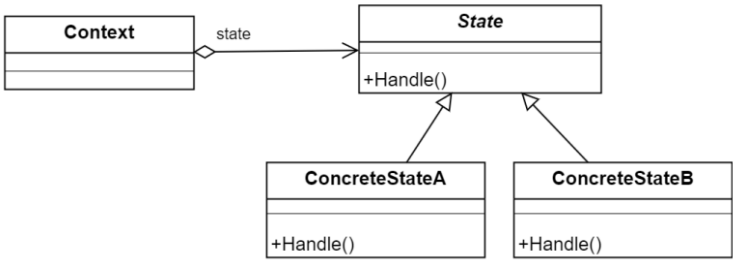
5. Які класи входять у шаблон «Стратегія», та взаємодія?

- Strategy інтерфейс алгоритму.
- ConcreteStrategy конкретні реалізації алгоритмів.
- Context зберігає вибрану стратегію та викликає її.
  Контекст делегує роботу обраній стратегії.

6. Яке призначення шаблону «Стан»?

Змінювати поведінку об’єкта залежно від його внутрішнього стану, винісши логіку станів у окремі класи.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять у шаблон «Стан», та взаємодія?

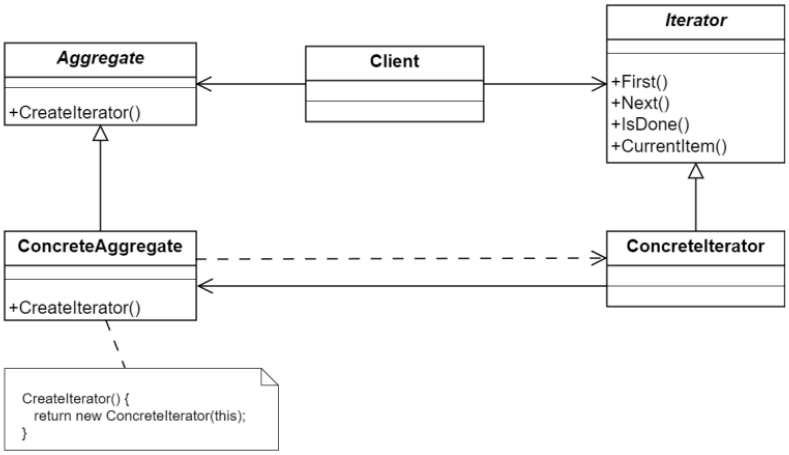
- State інтерфейс поведінки.
- ConcreteState окремі стани з власною логікою.
- Context зберігає поточний стан і делегує йому роботу.

  Зміна стану - зміна поведінки контексту.

9. Яке призначення шаблону «Ітератор»?

Забезпечити послідовний обхід елементів колекції без розкриття її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



11. Які класи входять у шаблон «Ітератор», та взаємодія?

- Iterator – інтерфейс проходження (First, Next, Current).
- ConcreteIterator – реалізує обхід конкретної колекції.
- Aggregate – інтерфейс колекції.
- ConcreteAggregate – реальна колекція, створює ітератор.
  Ітератор обходить елементи, не змінюючи колекцію.

12. В чому полягає ідея шаблону «Одинак»?

Забезпечити існування тільки одного екземпляра класу та глобальну точку доступу до нього.

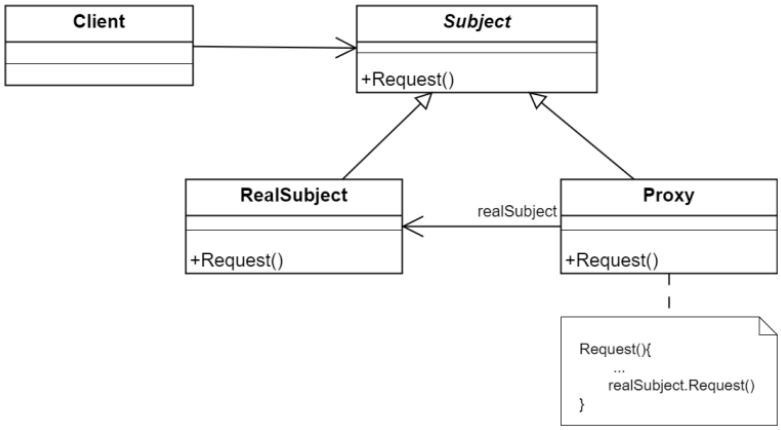
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Бо він створює глобальний стан, ускладнює тестування, порушує принцип єдиної відповідальності та може маскувати погану архітектуру.

14. Яке призначення шаблону «Проксі»?

Контролювати доступ до реального об’єкта, додаючи проміжну логіку (кешування, відкладені операції, перевірки).

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять у шаблон «Проксі», та взаємодія?

- Subject – спільний інтерфейс.
- RealSubject – реальний об’єкт.
- Proxy – замісник, контролює/перехоплює виклики.

  Клієнт працює з Proxy, а той вже керує доступом до RealSubject.