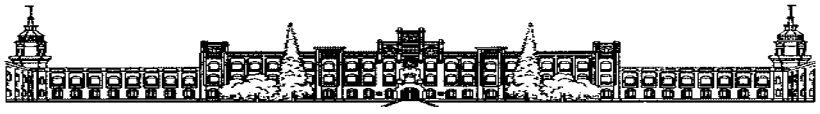


Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №7
Технологія розробки програмного забезпечення
«Патерни проектування»

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

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

Тема: Патерни проектування

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» 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 «Mediator»

Патерн Mediator використовується для організації взаємодії між об’єктами через один центральний об’єкт-посередник. Замість того, щоб компоненти напряму викликали один одного, вони спілкуються тільки з медіатором, який координує їхню поведінку. Це значно зменшує кількість зв’язків у системі та спрощує зміну логіки взаємодії. Медіатор особливо корисний у великих UI-формах, де багато контролів повинні реагувати на зміни один одного. Недолік: з часом медіатор може стати надмірно складним і перетворитися на "God Object".

Шаблон «Facade»

Патерн Facade забезпечує спрощений інтерфейс для складної підсистеми. Він приховує внутрішню структуру системи та пропонує єдиний доступний клас із зрозумілими методами. Клієнтський код працює лише з фасадом, не торкаючись внутрішніх деталей. Це полегшує використання модуля, робить код чистішим і дозволяє змінювати внутрішню реалізацію без впливу на користувачів. Недолік: фасад може зменшувати гнучкість, обмежуючи можливість прямого доступу до низькорівневих елементів.

Шаблон «Bridge»

Патерн Bridge розділяє абстракцію та її реалізацію на дві незалежні ієрархії. Абстракція містить посилання на інтерфейс реалізації і делегує їй роботу. Це дозволяє комбінувати різні абстракції з різними реалізаціями без вибуху кількості класів. Патерн застосовується, коли потрібно підтримувати багато варіантів поведінки та багато варіантів відображення або роботи з даними. Недолік:підвищена складність через створення кількох паралельних ієрархій.

Шаблон «Template Method»

Патерн Template Method визначає етапи алгоритму в базовому класі та дозволяє підкласам перевизначати окремі кроки, не змінюючи сам алгоритм.

Це дає можливість повторно використовувати спільну логіку й адаптувати лише різні частини поведінки. Корисний, коли алгоритм має спільну структуру, а конкретні реалізації деяких кроків відрізняються. Недоліки: залежність підкласів від жорсткої структури алгоритму та ризик ускладнення підтримки при великій кількості перевизначених методів.

Хід Роботи

Діаграма класів

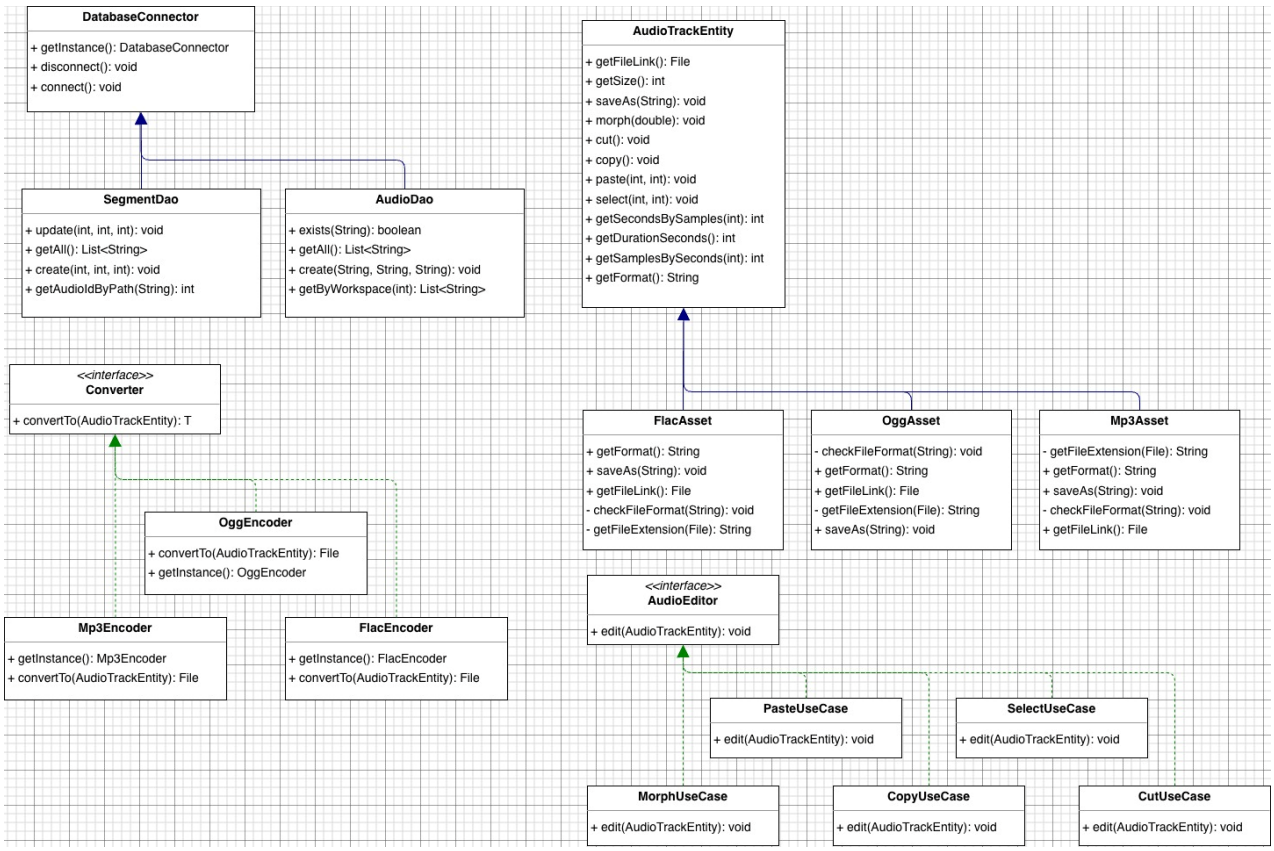


Рисунок 1 – Діаграма класів

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 Mediator

На діаграмі показано реалізацію шаблону Mediator. UiCoordinator є центральним посередником, який отримує події від інтерфейсу та визначає, яку дію потрібно виконати. MainWindow не взаємодіє з іншими компонентами напряму, а надсилає всі події через notify. Інтерфейс UiBus задає єдиний спосіб комунікації компонентів з посередником. Такий підхід зменшує кількість зв’язків між елементами інтерфейсу, робить структуру зрозумілішою і дозволяє легко розширювати функціонал без змін у вже існуючих класах.

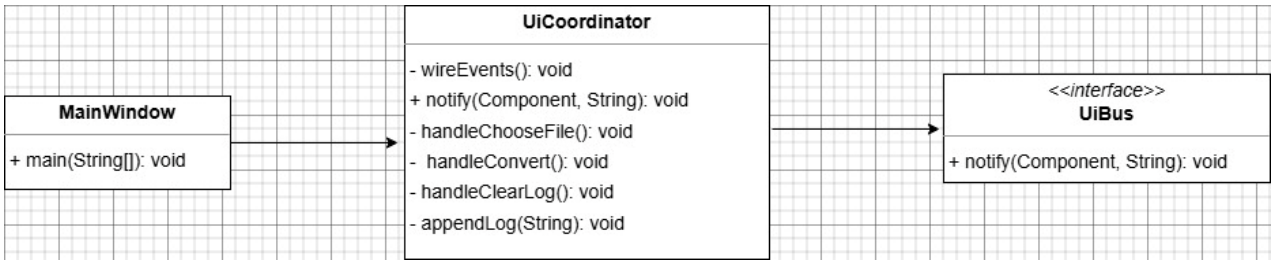


Рисунок 2 – Схема патерну Mediator

Фрагменти коду:

MainWindow.java

```
package app.soundlab.ui;


import javax.swing.*;

import java.awt.*;


public class MainWindow {

    public static void main(String[] args) {

        SwingUtilities.invokeLater(MainWindow::launch);

    }


    private static void launch() {

        JFrame frame = new JFrame("Mediator demo");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setSize(480, 360);


        JButton chooseFileButton = new JButton("Choose audio...");

        JButton convertButton = new JButton("Convert to MP3");

        JButton clearLogButton = new JButton("Clear log");

        convertButton.setEnabled(false);


        JLabel selectedFileLabel = new JLabel("No file selected");
```

```
        selectedFileLabel.setBorder(BorderFactory.createTitledBorder("Selected
file"));

        JTextArea eventLog = new JTextArea();
        eventLog.setEditable(false);
        eventLog.setLineWrap(true);
        JScrollPane logScroll = new JScrollPane(eventLog);
        logScroll.setBorder(BorderFactory.createTitledBorder("Mediator events"));

        JProgressBar progressBar = new JProgressBar();
        progressBar.setStringPainted(true);

        new UiCoordinator(
                chooseFileButton,
                convertButton,
                clearLogButton,
                selectedFileLabel,
                eventLog,
                progressBar
        );

        JPanel controlPanel = new JPanel(new GridLayout(1, 3, 8, 8));
        controlPanel.add(chooseFileButton);
```

```
        controlPanel.add(convertButton);
        controlPanel.add(clearLogButton);

        JPanel content = new JPanel(new BorderLayout(8, 8));
        content.setBorder(BorderFactory.createEmptyBorder(8, 8, 8, 8));
        content.add(controlPanel, BorderLayout.NORTH);
        content.add(selectedFileLabel, BorderLayout.CENTER);
        content.add(progressBar, BorderLayout.SOUTH);

        frame.setLayout(new BorderLayout(8, 8));
        frame.add(content, BorderLayout.NORTH);
        frame.add(logScroll, BorderLayout.CENTER);

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

UiBus.java

```
package app.soundlab.ui;

import java.awt.*;
```

```
interface UiBus {
    void notify(Component sender, String event);
}
```

UiCoordinator.java

```
package app.soundlab.ui;

import javax.swing.*;
import java.awt.*;
import java.io.File;

class UiCoordinator implements UiBus {
    private final JButton chooseFileButton;
    private final JButton convertButton;
    private final JButton clearLogButton;
    private final JLabel selectedFileLabel;
    private final JTextArea eventLog;
    private final JProgressBar progressBar;

    private File selectedFile;

    UiCoordinator(JButton chooseFileButton,
                  JButton convertButton,
                  JButton clearLogButton,
                  JLabel selectedFileLabel,
                  JTextArea eventLog,
                  JProgressBar progressBar) {
```

```
        this.chooseFileButton = chooseFileButton;
        this.convertButton = convertButton;
        this.clearLogButton = clearLogButton;
        this.selectedFileLabel = selectedFileLabel;
        this.eventLog = eventLog;
        this.progressBar = progressBar;
        wireEvents();
    }

    private void wireEvents() {
        chooseFileButton.addActionListener(e -> notify(chooseFileButton,
"chooseFile"));
        convertButton.addActionListener(e -> notify(convertButton, "convert"));
        clearLogButton.addActionListener(e -> notify(clearLogButton, "clearLog"));
    }

    @Override
    public void notify(Component sender, String event) {
        switch (event) {
            case "chooseFile" -> handleChooseFile();
            case "convert" -> handleConvert();
            case "clearLog" -> handleClearLog();
            default -> throw new IllegalArgumentException("Unknown event: " +
event);
        }
    }

    private void handleChooseFile() {
        JFileChooser fileChooser = new JFileChooser();
```

```
    int result = fileChooser.showOpenDialog(null);
    if (result == JFileChooser.APPROVE_OPTION) {
        selectedFile = fileChooser.getSelectedFile();
        selectedFileLabel.setText(selectedFile.getName());
        appendLog("Selected file: " + selectedFile.getAbsolutePath());
        convertButton.setEnabled(true);
    } else {
        appendLog("File selection cancelled.");
    }
}

private void handleConvert() {
    if (selectedFile == null) {
        JOptionPane.showMessageDialog(null, "Please choose a file first.");
        return;
    }
    appendLog("Preparing fake conversion for " + selectedFile.getName());
    convertButton.setEnabled(false);
    progressBar.setIndeterminate(true);
    progressBar.setString("Converting...");
    progressBar.setStringPainted(true);

    // Keep demo self-contained: simulate background work
    new SwingWorker<Void, Void>() {
        @Override
        protected Void doInBackground() throws InterruptedException {
            Thread.sleep(1500); // fake work
            return null;
        }
```

```
            @Override
            protected void done() {
                progressBar.setIndeterminate(false);
                progressBar.setValue(100);
                progressBar.setString("Done");
                appendLog("Finished conversion for " + selectedFile.getName());
                convertButton.setEnabled(true);
            }
        }.execute();
    }

    private void handleClearLog() {
        eventLog.setText("");
        progressBar.setValue(0);
        progressBar.setString("");
        appendLog("Log cleared.");
    }

    private void appendLog(String message) {
        eventLog.append(message + System.lineSeparator());
    }
}
```

Вихідний код:
https://github.com/mandarinchik21/AudioEditor/tree/main/lab7

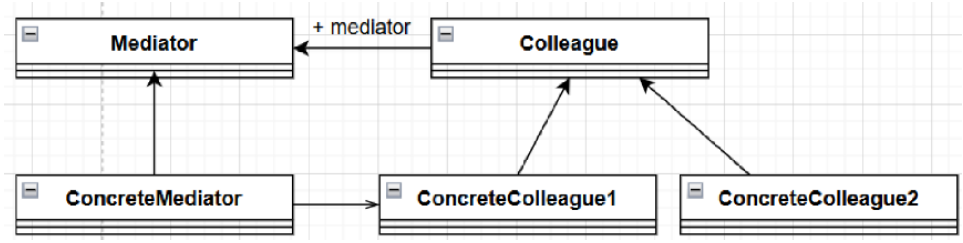
Висновок: У ході виконання лабораторної роботи було розглянуто та проаналізовано патерни «Mediator», «Facade», «Bridge» і «Template Method», а також досліджено їхнє застосування при проєктуванні аудіоредактора. Реалізація обраного шаблону показала, що використання цих підходів дозволяє значно зменшити зв’язність між компонентами, спростити роботу зі складними підсистемами, розділити абстракції та реалізації, а також структурувати алгоритми обробки даних. Завдяки впровадженню патернів система стала більш зрозумілою, гнучкою та масштабованою, що спрощує подальший розвиток функціоналу та підтримку коду. Отримані знання підтверджують практичну цінність шаблонів проєктування у побудові сучасного програмного забезпечення.

Контрольні запитання

1. Яке призначення шаблону «Посередник»?

Патерн «Посередник» (Mediator) організовує взаємодію між об’єктами через центральний об’єкт-медіатор, зменшуючи кількість прямих зв’язків між компонентами та спрощуючи координацію складних взаємодій.

2. Нарисуйте структуру шаблону «Посередник».



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

- Mediator інтерфейс або базовий клас посередника.
- ConcreteMediator реалізує логіку взаємодії між компонентами.
- Colleague базовий клас елементів, що взаємодіють через медіатор.
- ConcreteColleague компоненти, які виконують дії, але звертаються до медіатора.

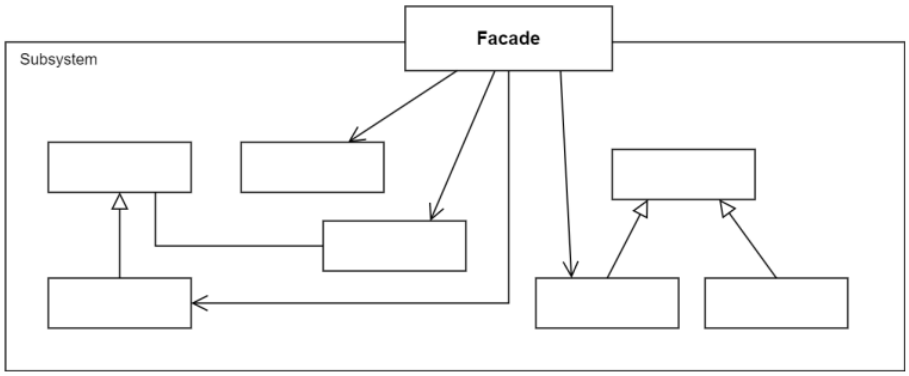
Взаємодія: колеги не знають один про одного та надсилають запити медіатору, який координує їхню поведінку.

4. Яке призначення шаблону «Фасад»?

Фасад (Facade) спрощує доступ до складної системи, надаючи один уніфікований інтерфейс замість великої кількості елементів підсистеми. Він

приховує внутрішню реалізацію та зменшує залежність клієнтів від складності системи.

5. Нарисуйте структуру шаблону «Фасад».



6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

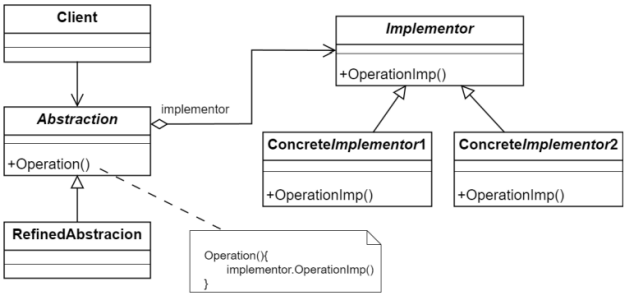
- Facade єдиний спрощений інтерфейс.
- Subsystem classes класи підсистеми з реальною логікою.
- Client працює тільки через Facade.

Взаємодія: клієнт викликає методи фасаду, а фасад всередині звертається до потрібних класів підсистеми.

7. Яке призначення шаблону «Міст»?

Патерн «Міст» (Bridge) розділяє абстракцію та її реалізацію на дві незалежні ієрархії, дозволяючи змінювати їх окремо та уникнути вибуху підкласів при комбінуванні абстракцій та реалізацій.

8. Нарисуйте структуру шаблону «Міст».



9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

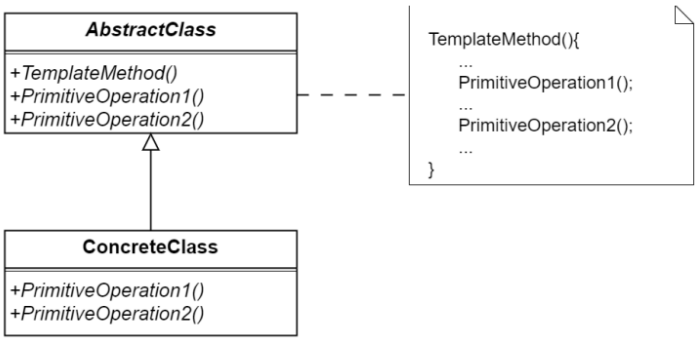
- Abstraction інтерфейс високого рівня.
- RefinedAbstraction розширена абстракція.
- Implementor інтерфейс реалізації.
- ConcreteImplementor конкретна реалізація.

Взаємодія: абстракція містить посилання на Implementor та делегує йому реальну роботу. Ієрархія абстракцій і реалізацій існують незалежно.

10. Яке призначення шаблону «Шаблонний метод»?

Патерн «Шаблонний метод» визначає загальну структуру алгоритму в базовому класі та дозволяє підкласам перевизначати окремі кроки алгоритму без зміни його основної послідовності.

11. Нарисуйте структуру шаблону «Шаблонний метод».



12. Які класи входять в шаблон «Шаблонний метод», та яка між ними взаємодія?

- AbstractClass містить шаблон алгоритму (templateMethod) та визначає абстрактні кроки.
- ConcreteClass реалізує конкретні версії абстрактних кроків.

Взаємодія: клієнт викликає templateMethod(), який виконує загальний алгоритм, а частини алгоритму виконуються через перевизначені методи підкласів.

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

- Template Method описує шаблон алгоритму та дозволяє змінювати окремі кроки.
- Factory Method описує спосіб створення об’єктів, дозволяючи підкласам визначати конкретний продукт.

Коротко:

- Template Method → про алгоритм
- Factory Method → про створення об’єктів

14. Яку функціональність додає шаблон «Міст»?

Патерн «Міст» додає можливість незалежно розвивати абстракцію та реалізацію, комбінувати різні реалізації з різними абстракціями, а також уникати дублювання коду та вибуху кількості підкласів при масштабуванні системи.