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

Лабораторна робота №7

із дисципліни «*Технології розроблення програмного забезпечення*»

Патерни проектування

Аудіоредактор

Виконала
студентка групи ІА–34
Кузьменко В. С.

Перевірив
викладач
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.

Тема роботи:

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

Принципи проєктування

Don't Repeat Yourself (DRY). Принцип DRY спрямований на уникнення дублювання коду. Якщо однаковий фрагмент коду повторюється в кількох місцях, це ускладнює підтримку програми, адже кожна зміна вимагатиме оновлення в усіх місцях. Щоб цього уникнути, необхідно використовувати функції, класи чи інтерфейси для реалізації загального функціоналу, підвищуючи гнучкість і зручність підтримки.

Keep It Simple, Stupid! (KISS). KISS закликає створювати прості, зрозумілі системи. Поділ програми на малі й логічно ізольовані компоненти робить її більш надійною, легкою для розуміння та модифікації. Надмірна складність ускладнює розробку та тестування, тому простота стає ключем до ефективності.

You Only Load It Once (YOLO). Принцип YOLO передбачає одноразову ініціалізацію даних під час запуску програми. Це знижує навантаження на систему, адже повторна обробка або завантаження даних можуть значно вплинути на продуктивність. Ефективне використання цього принципу зменшує час виконання операцій.

Принцип Парето. Цей принцип нагадує, що 80% результатів забезпечують лише 20% зусиль. У контексті програмування варто зосередитися на оптимізації найважливіших частин системи, які приносять найбільшу користь, замість рівномірного розподілу ресурсів на малозначущі задачі.

You Ain’t Gonna Need It (YAGNI). YAGNI рекомендує уникати додавання непотрібної функціональності. Система повинна реалізовувати лише ті можливості, які дійсно потрібні зараз, без передбачення майбутніх сценаріїв, які можуть ніколи не виникнути. Це дозволяє зменшити складність, підвищити продуктивність і скоротити час розробки.

Шаблони проєктування

Mediator (Посередник). Шаблон «Посередник» централізує взаємодію між об'єктами, щоб зменшити їхню залежність одне від одного. Усі комунікації здійснюються через окремий об'єкт-посередник, що спрощує структуру системи, знижує зв'язність компонентів і полегшує внесення змін.

Facade (Фасад). «Фасад» спрощує взаємодію із системою, пропонуючи єдиний інтерфейс для роботи з її складними підсистемами. Він приховує всі деталі реалізації, надаючи користувачам зрозумілий і зручний спосіб доступу до функціоналу системи.

Bridge (Міст). Шаблон «Міст» дозволяє розділити абстракцію та її реалізацію, що робить систему більш гнучкою. Це особливо корисно, коли потрібно незалежно розширювати як абстракцію, так і реалізацію, зменшуючи кількість класів та складність ієрархії.

Template Method (Шаблонний метод). Шаблонний метод визначає загальну структуру алгоритму у базовому класі, залишаючи підкласам можливість реалізовувати окремі його кроки. Такий підхід зберігає основну логіку незмінною, водночас забезпечуючи гнучкість у деталях реалізації.

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Реалізувати функціонал програми через класи та їх взаємодію.
3. Застосувати один із шаблонів проектування.
4. Створити щонайменше три класи за обраною темою.
5. Підготувати звіт із діаграмою класів і прикладами коду.

Шаблон Mediator

Ознайомившись із теоретичними матеріалами, було вирішено реалізувати шаблон *Mediator (Посередник)*. Цей патерн проектування належить до групи поведінкових шаблонів і використовується для організації взаємодії між об’єктами таким чином, щоб вони не залежали безпосередньо один від одного. Замість того, щоб об’єкти напряму викликали методи один одного, усі комунікації здійснюються через спеціальний об’єкт - посередник, який координує їхню взаємодію.

Основна ідея *Mediator* полягає в централізації логіки обміну повідомленнями між компонентами системи, що значно зменшує зв’язність і спрощує підтримку коду. Завдяки цьому шаблону зміни в одному компоненті не вимагають модифікації інших, адже комунікація відбувається виключно через посередника.

Вибір саме цього шаблону обумовлений тим, що розроблюваний застосунок містить велику кількість елементів графічного інтерфейсу, які взаємодіють між собою. Якщо створювати залежності напряму, код швидко стане складним і заплутаним. Застосування Mediator дозволяє уніфікувати управління взаємодіями між елементами, зменшити дублювання коду, підвищити гнучкість і масштабованість системи.

Загальну структуру графічно описано на Рис 1.

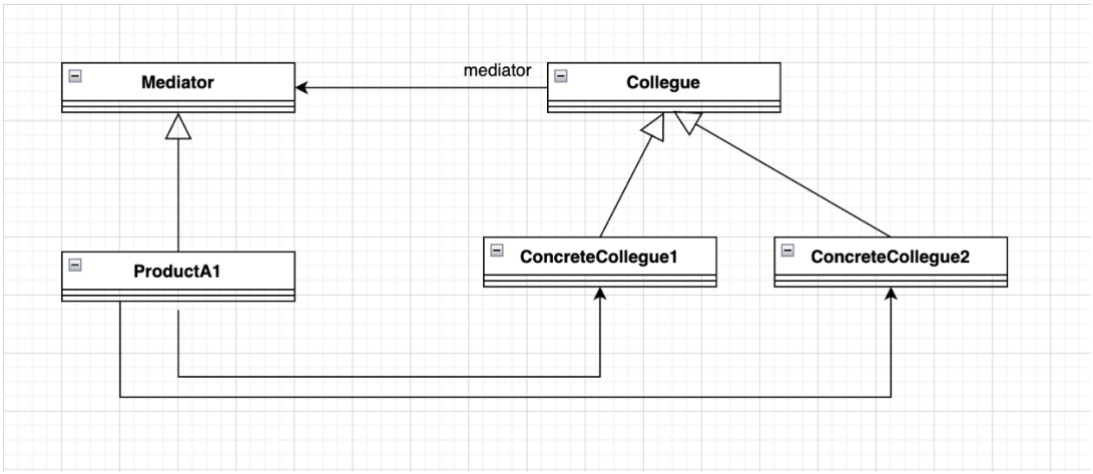


Рис. 1. Загальна структура шаблону Mediator

Компоненти в шаблоні *Mediator* - це незалежні об’єкти, що реалізують окремі частини бізнес-логіки застосунку. Кожен із них виконує власні функції, але не має безпосередніх зв’язків із іншими компонентами. Замість цього вони взаємодіють через посередника, який виступає центральною ланкою комунікації.

Посередник визначає уніфікований інтерфейс для обміну даними між компонентами, а його конкретна реалізація містить логіку, яка координує їхню поведінку. Це означає, що коли в одному з компонентів відбувається певна подія, він не викликає інші об’єкти напряму, а повідомляє посередника. Той, своєю чергою, аналізує ситуацію та вирішує, які інші компоненти мають бути поінформовані або змінені у відповідь.

Такий підхід роз’єднує компоненти між собою, робить систему більш гнучкою та легкою для супроводу. Зміни в одному елементі не потребують втручання в код інших, достатньо лише оновити логіку посередника. Завдяки цьому Mediator забезпечує чітку структуру взаємодії та зменшує складність комунікацій у програмі.

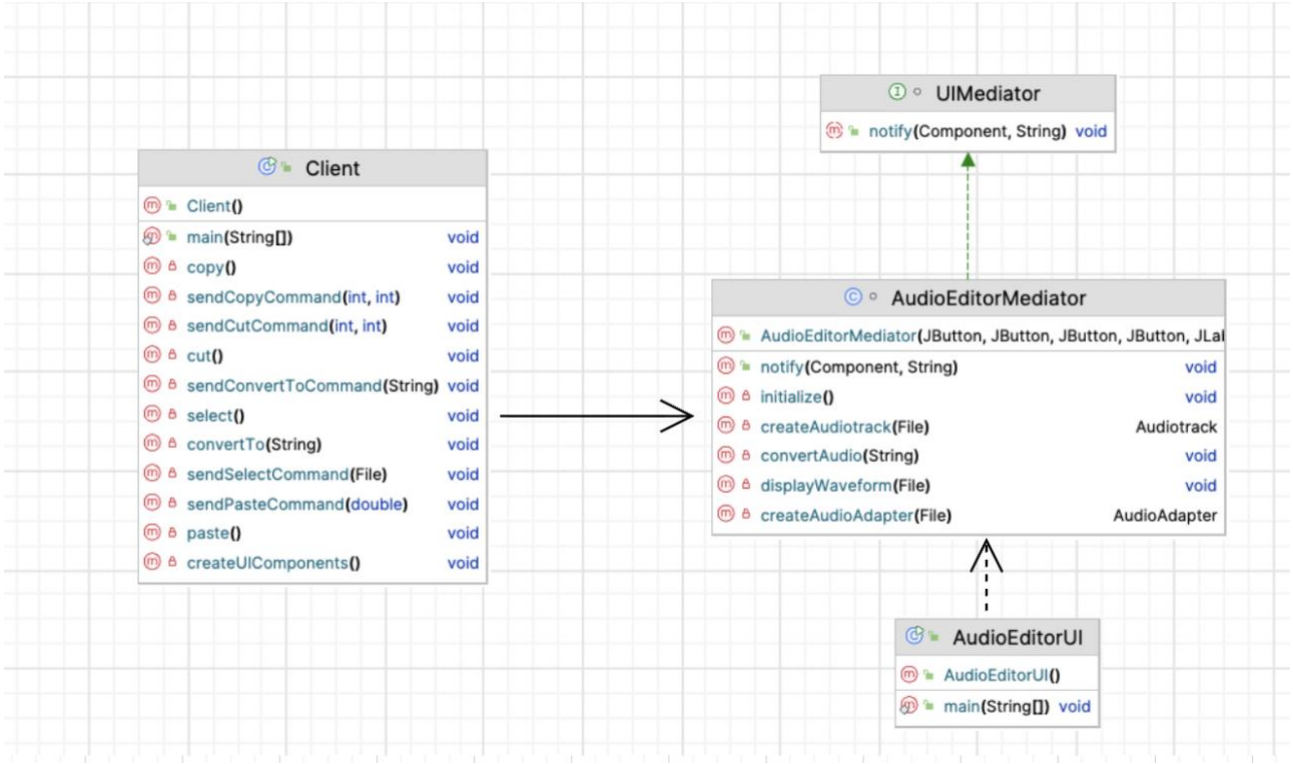


Рис. 2 Структура реалізованого шаблону *Mediator*

Інтерфейс *UIMediator* визначає єдиний метод notify(Component sender, String event), який використовується для посередництва між компонентами інтерфейсу користувача. Його завдання - забезпечити незалежність компонентів один від одного: замість того, щоб напряму викликати методи інших елементів, вони повідомляють про події через медіатор.

Клас *AudioEditorMediator* є конкретною реалізацією посередника (Concrete Mediator), що реалізує інтерфейс *UIMediator*. Він отримує посилання на елементи інтерфейсу користувача (JButton, JLabel тощо) через конструктор і керує їхньою взаємодією. Саме в цьому класі визначена логіка, що робити, коли користувач натискає кнопку, обирає файл чи запускає конвертацію.

Основні методи:

- notify(Component, String) - реагує на події від компонентів і викликає відповідні дії.
- initialize() - ініціалізує стан медіатора або компонентів.
- createAudiotrack(File) - створює аудіотрек із вибраного файлу.
- convertAudio(String) - виконує конвертацію у вказаний формат.
- displayWaveform(File) - відображає хвильову форму аудіо.
- createAudioAdapter(File) - створює адаптер для роботи з файлом аудіо.

Компоненти, з якими працює медіатор:

- JButton - наприклад, loadFileButton, convertButton, cutButton, pasteButton тощо;
- JLabel - наприклад, fileLabel, який показує інформацію про поточний файл;
- JPanel - може використовуватись для візуалізації хвильової форми (waveformPanel);
- Logger - для фіксації дій користувача або помилок.

Клас *AudioEditorUI* виконує роль головного класу застосунку, який:

- створює графічний інтерфейс користувача (вікна, кнопки, панелі, мітки);
- ініціалізує об’єкт *AudioEditorMediator*, передаючи йому всі створені компоненти;
- встановлює взаємодію: коли користувач натискає кнопку чи обирає файл - UI повідомляє про це медіатор.

Тобто, *AudioEditorUI* є “точкою входу” до програми, що налаштовує середовище взаємодії між користувачем і медіатором.

Клас *Client* виконує роль керуючого клієнта, який моделює дії користувача та надсилає команди до медіатора.

У ньому визначені методи для виконання типових дій аудіоредактора:

- copy(), cut(), paste() - для редагування фрагментів аудіо;
- convertTo(String format) - для конвертації у заданий формат;
- select(), sendSelectCommand(File) - для вибору файлу;
- createUIComponents() - створює або ініціалізує елементи UI;
- main(String[] args) - точка запуску програми.

Таким чином, *Client* взаємодіє з *AudioEditorMediator*, відправляючи команди, які медіатор координує між компонентами інтерфейсу.

Узагальнення:

- UIMediator - інтерфейс посередника, що визначає спосіб зв’язку між компонентами.
- AudioEditorMediator - конкретний посередник, який керує логікою взаємодії.
- AudioEditorUI - створює інтерфейс і передає посилання на компоненти медіатору.
- Client - ініціює дії користувача та працює з медіатором.

Код системи

Лістинг 1- UIMediator

```
package org.example.swing;

import java.awt.*;

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

Лістинг 2- AudioEditorUI

```
package org.example.swing;

import org.example.database.DatabaseInitializer;
import org.example.logs.Logger;
import org.example.composite.*;

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

public class AudioEditorUI {
    public static void main(String[] args) {
        DatabaseInitializer.initializeDatabase();

        JFrame frame = new JFrame("Audio Editor");
        frame.setSize(600, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(null);

        JButton loadFileButton = new JButton("Load Audio File");
        loadFileButton.setBounds(50, 50, 180, 30);

        JLabel fileLabel = new JLabel("No file selected");
        fileLabel.setBounds(250, 50, 300, 30);

        JButton convertToMp3Button = new JButton("Convert to MP3");
        convertToMp3Button.setBounds(50, 100, 180, 30);

        JButton convertToOggButton = new JButton("Convert to OGG");
        convertToOggButton.setBounds(50, 150, 180, 30);

        JButton convertToFlacButton = new JButton("Convert to FLAC");
        convertToFlacButton.setBounds(50, 200, 180, 30);

        JPanel waveformPanel = new JPanel();
        waveformPanel.setBounds(50, 250, 500, 100);
        waveformPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));

        Logger logger = new Logger();

        UIContainer mainContainer = new UIContainer();

        UIContainer buttonPanel = new UIContainer();
        buttonPanel.add(new UIElement(loadFileButton));
        buttonPanel.add(new UIElement(convertToMp3Button));
        buttonPanel.add(new UIElement(convertToOggButton));
        buttonPanel.add(new UIElement(convertToFlacButton));

        mainContainer.add(buttonPanel);
        mainContainer.add(new UIElement(fileLabel));
```

```
        mainContainer.add(new UIElement(waveformPanel));

        frame.add(loadFileButton);
        frame.add(fileLabel);
        frame.add(convertToMp3Button);
        frame.add(convertToOggButton);
        frame.add(convertToFlacButton);
        frame.add(waveformPanel);

        AudioEditorMediator mediator = new AudioEditorMediator(
                loadFileButton, convertToMp3Button, convertToOggButton,
convertToFlacButton,
                fileLabel, waveformPanel, logger);

        mainContainer.operation();

        frame.setVisible(true);

    }
}
```

Лістинг 3- AudioEditorMediator

```
package org.example.swing;
import org.example.converter.AudioFlacConverter;
import org.example.converter.AudioMp3Converter;
import org.example.converter.AudioOggConverter;
import org.example.audiotrack.*;
import org.example.logs.Logger;

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

class AudioEditorMediator implements UIMediator {
    private JButton loadFileButton;
    private JButton convertToMp3Button;
    private JButton convertToOggButton;
    private JButton convertToFlacButton;
    private JLabel fileLabel;
    private JPanel waveformPanel;
    private Logger logger;
    private File selectedFile;

    public AudioEditorMediator(JButton loadFileButton, JButton
convertToMp3Button, JButton convertToOggButton, JButton convertToFlacButton,
                               JLabel fileLabel, JPanel waveformPanel, Logger
logger) {
        this.loadFileButton = loadFileButton;
        this.convertToMp3Button = convertToMp3Button;
        this.convertToOggButton = convertToOggButton;
        this.convertToFlacButton = convertToFlacButton;
        this.fileLabel = fileLabel;
        this.waveformPanel = waveformPanel;
        this.logger = logger;

        initialize();
    }

    private void initialize() {
        loadFileButton.addActionListener(e -> notify(loadFileButton,
"loadFile"));
        convertToMp3Button.addActionListener(e -> notify(convertToMp3Button,
"convertToMp3"));
```

```
        convertToOggButton.addActionListener(e -> notify(convertToOggButton,
"convertToOgg"));
        convertToFlacButton.addActionListener(e -> notify(convertToFlacButton,
"convertToFlac"));
    }

    @Override
    public void notify(Component sender, String event) {
        switch (event) {
            case "loadFile":
                JFileChooser fileChooser = new JFileChooser();
                int returnValue = fileChooser.showOpenDialog(null);
                if (returnValue == JFileChooser.APPROVE_OPTION) {
                    selectedFile = fileChooser.getSelectedFile();
                    fileLabel.setText("Selected: " + selectedFile.getName());
                    displayWaveform(selectedFile);
                    logger.fileOpen();
                }
                break;

            case "convertToMp3":
                convertAudio("mp3");
                break;

            case "convertToOgg":
                convertAudio("ogg");
                break;

            case "convertToFlac":
                convertAudio("flac");
                break;

            default:
                throw new IllegalArgumentException("Unknown event: " + event);
        }
    }

    private void convertAudio(String format) {
        if (selectedFile == null) {
            JOptionPane.showMessageDialog(null, "Please load a file first.");
            return;
        }

        AudioAdapter adapter = createAudioAdapter(selectedFile);
        File convertedFile;
        switch (format) {
            case "mp3":
                convertedFile = AudioMp3Converter.getInstance().convertTo(new
Mp3(adapter.adaptFile().getAbsolutePath()));
                break;
            case "ogg":
                convertedFile = AudioOggConverter.getInstance().convertTo(new
Ogg(adapter.adaptFile().getAbsolutePath()));
                break;
            case "flac":
                convertedFile = AudioFlacConverter.getInstance().convertTo(new
Flac(adapter.adaptFile().getAbsolutePath()));
                break;
            default:
                throw new IllegalArgumentException("Unsupported format: " +
format);
        }
        JOptionPane.showMessageDialog(null, "File converted to " +
format.toUpperCase() + " successfully!");
    }
```

```
    private void displayWaveform(File audioFile) {
        waveformPanel.removeAll();
        waveformPanel.setLayout(new BorderLayout());
        waveformPanel.add(new AudioWaveformPanel(audioFile),
BorderLayout.CENTER);
        waveformPanel.revalidate();
        waveformPanel.repaint();
    }

    private AudioAdapter createAudioAdapter(File file) {
        Audiotrack audiotrack = createAudiotrack(file);
        return new AudioAdapter(audiotrack);
    }

    private Audiotrack createAudiotrack(File file) {
        String fileName = file.getName().toLowerCase();
        if (fileName.endsWith(".mp3")) {
            return new Mp3(file.getAbsolutePath());
        } else if (fileName.endsWith(".ogg")) {
            return new Ogg(file.getAbsolutePath());
        } else if (fileName.endsWith(".flac")) {
            return new Flac(file.getAbsolutePath());
        } else {
            throw new IllegalArgumentException("Unsupported file format: " +
fileName);
        }
    }
}
```

Лістинг 4- Client

```
package org.example.swing;

import org.example.slider.RangeSlider;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class Client extends JDialog {
    {
        Socket socket = null;
        try {
            socket = new Socket("localhost", 55555);

            objectOutputStream = new
ObjectOutputStream(socket.getOutputStream());
            objectInputStream = new ObjectInputStream(socket.getInputStream());

        } catch (IOException e) {
            System.out.println("Server is not running.");
            System.exit(1);
        }
    }

    private JPanel contentPane;
    private JButton button;
```

```
    private JPanel wave;
    private WavePanel wavePanel;
    private JButton copyButton;
    private JButton convertToButton;
    private JButton pasteButton;
    private JButton cutButton;
    private RangeSlider rangeSlider;
    private ObjectOutputStream objectOutputStream;
    private ObjectInputStream objectInputStream;

    public Client() {
        setContentPane(contentPane);
        setModal(true);
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                select();
            }
        });
        copyButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                copy();
            }
        });
        cutButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                cut();
            }
        });
        pasteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                paste();
            }
        });
    }



    private void select() {
        JFileChooser fileChooser = new JFileChooser();
        int state = fileChooser.showOpenDialog(null);
        if (state == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            fileChooser.setVisible(false);
            try {
                sendSelectCommand(selectedFile);
                int[] data = (int[]) objectInputStream.readObject();
                wavePanel.setData(data);
                SwingUtilities.updateComponentTreeUI(contentPane);

                rangeSlider.setMinimum(0);
                rangeSlider.setMaximum(data.length);

                rangeSlider.setValue(((int) (rangeSlider.getMaximum() * 0.5)));
                rangeSlider.setUpperValue(((int) (rangeSlider.getMaximum() *
0.75)));
                rangeSlider.setVisible(true);
            } catch (IOException | ClassNotFoundException e) {
                System.out.println(e.getMessage());
            }
        }
    }
```

```
    private void copy() {
        try {
            sendCopyCommand(rangeSlider.getValue(),
rangeSlider.getUpperValue());

            int[] data = (int[]) objectInputStream.readObject();
            wavePanel.setData(data);
            SwingUtilities.updateComponentTreeUI(contentPane);

            rangeSlider.setMinimum(0);
            rangeSlider.setMaximum(data.length);

            rangeSlider.setVisible(true);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }

    private void cut() {
        try {
            sendCutCommand(rangeSlider.getValue(), rangeSlider.getUpperValue());

            int[] data = (int[]) objectInputStream.readObject();
            wavePanel.setData(data);
            SwingUtilities.updateComponentTreeUI(contentPane);

            rangeSlider.setMinimum(0);
            rangeSlider.setMaximum(data.length);

            rangeSlider.setValue(((int) (rangeSlider.getMaximum() * 0.5)));
            rangeSlider.setUpperValue(((int) (rangeSlider.getMaximum() *
0.75)));
        } catch (IOException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    private void paste() {
        try {
            double x = wave.getMousePosition().getX() / wave.getWidth();
            System.out.println(x);
            sendPasteCommand(x);

            int[] data = (int[]) objectInputStream.readObject();
            wavePanel.setData(data);
            SwingUtilities.updateComponentTreeUI(contentPane);

            rangeSlider.setMinimum(0);
            rangeSlider.setMaximum(data.length);

            rangeSlider.setValue(((int) (rangeSlider.getMaximum() * 0.5)));
            rangeSlider.setUpperValue(((int) (rangeSlider.getMaximum() *
0.75)));
        } catch (Exception e) {
            System.out.println("Мишка знаходиться у неправильному місці. Будь
ласка, оберіть місце на звуковій доріжці.");
        }
    }

    private void convertTo(String format) {
        try {
            sendConvertToCommand(format);
            JOptionPane.showMessageDialog(null, "Файл було успішно форматовано у
" + format);
```

```
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }


    private void sendCopyCommand(int l, int u) throws IOException {
        objectOutputStream.writeObject("copy");
        objectOutputStream.writeObject(l);
        objectOutputStream.writeObject(u);
    }

    private void sendCutCommand(int l, int u) throws IOException {
        objectOutputStream.writeObject("cut");
        objectOutputStream.writeObject(l);
        objectOutputStream.writeObject(u);
    }

    private void sendSelectCommand(File file) throws IOException {
        objectOutputStream.writeObject("select");
        objectOutputStream.writeObject(file);
    }

    private void sendPasteCommand(double x) throws IOException {
        objectOutputStream.writeObject("paste");
        objectOutputStream.writeObject(x);
    }

    private void sendConvertToCommand(String format) throws IOException {
        objectOutputStream.writeObject("convertTo" + format);
    }

    private void createUIComponents() {
        wave = new JPanel();
        wavePanel = new WavePanel();
        rangeSlider = new RangeSlider();
        rangeSlider.setVisible(false);

        convertToButton = new JButton();

        JPopupMenu popupMenu = new JPopupMenu();

        JMenuItem mp3 = new JMenuItem("mp3");
        JMenuItem ogg = new JMenuItem("ogg");
        JMenuItem flac = new JMenuItem("flac");

        mp3.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                convertTo("Mp3");
            }
        });

        ogg.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                convertTo("Ogg");
            }
        });

        flac.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                convertTo("Flac");
```

```
            }
        });

        popupMenu.add(mp3);
        popupMenu.add(ogg);
        popupMenu.add(flac);

        convertToButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {;
                popupMenu.show(convertToButton, 0, convertToButton.getHeight());
            }
        });


    }

    public static void main(String[] args) {
        Client dialog = new Client();
        dialog.pack();
        dialog.setVisible(true);
        System.exit(0);
    }

}
```

Висновок

Під час виконання даної лабораторної роботи було реалізовано шаблон *Mediator (Посередник)*, який виявився особливо корисним для проєктування аудіоредактора з великою кількістю взаємодіючих елементів інтерфейсу. Завдяки використанню цього патерну вдалося централізувати обмін повідомленнями між компонентами, уникнувши надмірних залежностей між ними. Це значно спростило структуру коду, полегшило його розширення та подальшу підтримку. Уся логіка взаємодії зосереджена в одному класі-посереднику, що зробило систему більш гнучкою, масштабованою та зрозумілою. Mediator допоміг реалізувати принципи низької зв’язності та єдиної відповідальності, завдяки чому навіть додавання нових елементів не потребує змін у решті компонентів.

Під час виконання роботи мене також було ознайомлено з ключовими принципами проєктування та шаблонами, що сприяють ефективній розробці програмного забезпечення. Принцип *Don't Repeat Yourself (DRY)* дозволяє уникати дублювання коду, спрощуючи його підтримку та зменшуючи

ймовірність помилок. *Keep It Simple, Stupid! (KISS)* забезпечує зрозумілість, надійність і легкість модифікації системи завдяки спрощенню її компонентів. *You Only Load It Once (YOLO)* оптимізує продуктивність, запобігаючи повторному завантаженню даних і економлячи ресурси. *Принцип Парето* показує, що 80% результату можна досягти завдяки 20% зусиль, що допомагає концентруватися на найважливіших аспектах розробки. *You Ain't Gonna Need It (YAGNI)* дозволяє уникати зайвої складності, реалізуючи лише необхідну функціональність.

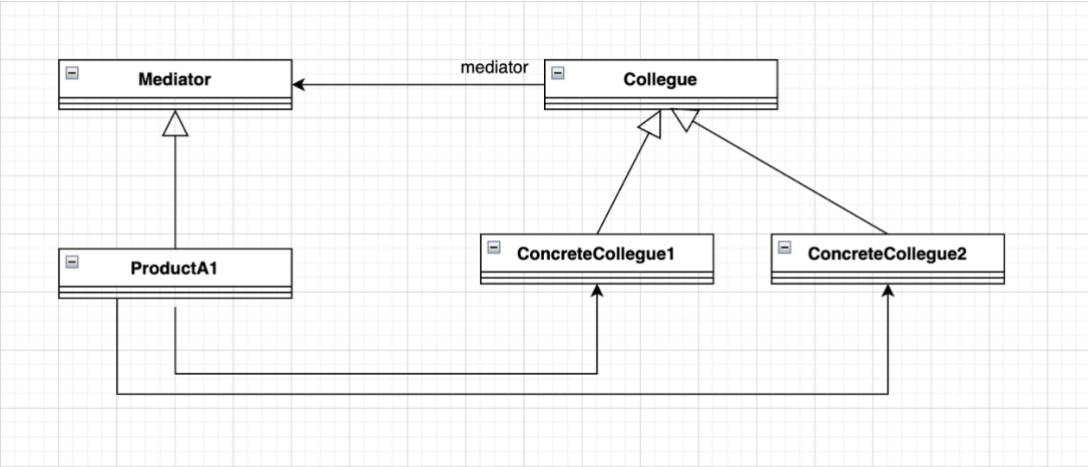
Шаблони проєктування також відіграють важливу роль у створенні гнучких та масштабованих систем. *Mediator* централізує взаємодію між об'єктами, зменшуючи їхню взаємозалежність і полегшуючи підтримку. *Facade* приховує внутрішню складність підсистем, надаючи єдиний зручний інтерфейс для роботи з ними. *Bridge* дозволяє окремо розвивати абстракцію та її реалізацію, що підвищує гнучкість системи. *Template Method* стандартизує структуру алгоритмів, залишаючи підкласам можливість налаштовувати окремі кроки.

Відповіді на теоретичні питання

1. Яке призначення шаблону «Посередник» (Mediator)?

Шаблон «*Посередник*» використовується для організації взаємодії між об'єктами без необхідності, щоб вони напряму знали один про одного. Він зменшує зв'язність системи, концентруючи логіку взаємодії в одному об'єкті - медіаторі.

2. Нарисуйте структуру шаблону «Посередник».



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

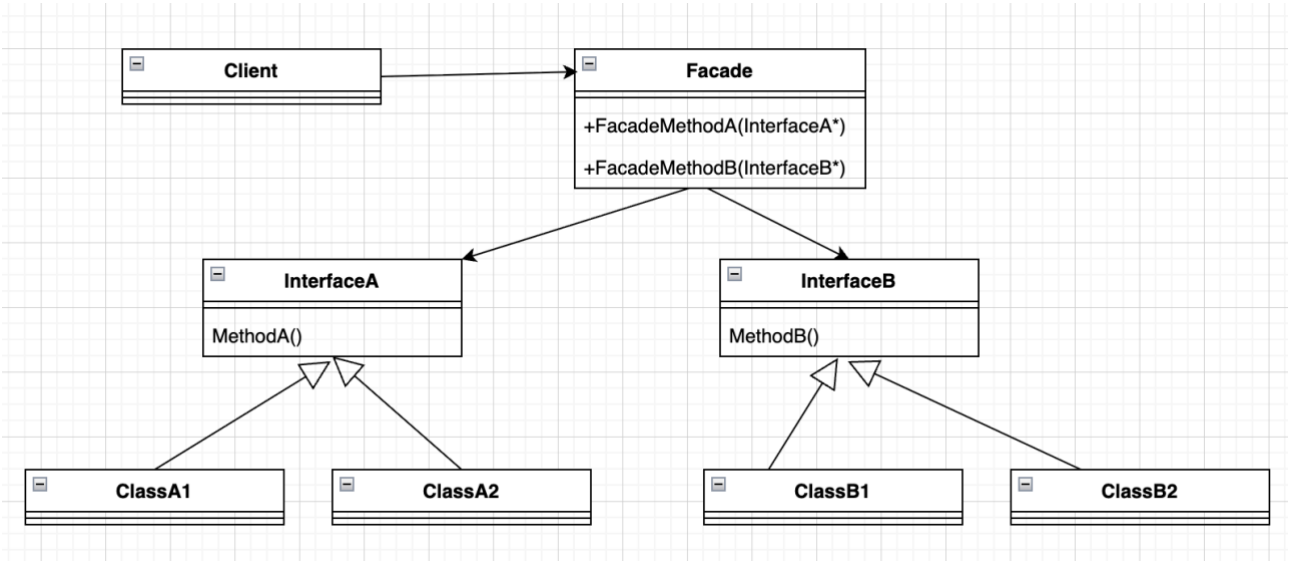
- Mediator - інтерфейс, що визначає методи взаємодії між компонентами.
- ConcreteMediator - конкретний посередник, який координує дії компонентів.
- Component (Colleague) - базовий клас для компонентів, які взаємодіють через медіатор.
- ConcreteComponentA/B - окремі компоненти, які повідомляють медіатор про події.

Взаємодія: компоненти не спілкуються напряму, а передають події медіатору через метод notify(). Медіатор вирішує, як інші компоненти повинні відреагувати.

4. Яке призначення шаблону «Фасад» (Facade)?

Шаблон «*Фасад*» надає спрощений уніфікований інтерфейс до складної системи класів. Він приховує внутрішню реалізацію підсистеми і дозволяє клієнту працювати лише з одним “фасадним” класом.

5. Нарисуйте структуру шаблону «Фасад».



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

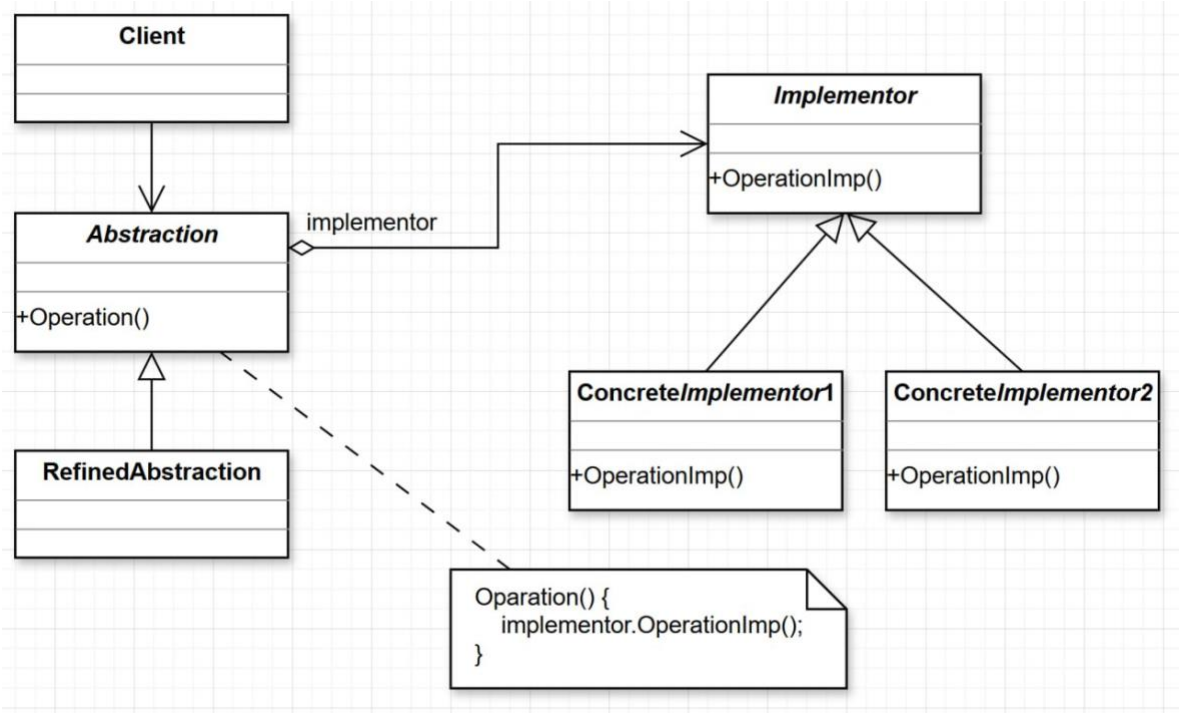
- Facade - основний клас, який інкапсулює складну логіку кількох підсистем.
- SubsystemA, SubsystemB, ... - класи, що реалізують справжню функціональність.
- Client - викликає методи фасаду, не знаючи про деталі підсистем.

Взаємодія: клієнт викликає метод фасаду, а той уже звертається до кількох підсистем, узгоджуючи їхню роботу.

7. Яке призначення шаблону «Міст» (Bridge)?

Шаблон «*Міст*» розділяє абстракцію та реалізацію, дозволяючи змінювати їх незалежно одна від одної. Він усуває жорстке зв’язування між ними, роблячи систему більш гнучкою.

8. Нарисуйте структуру шаблону «Міст».



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

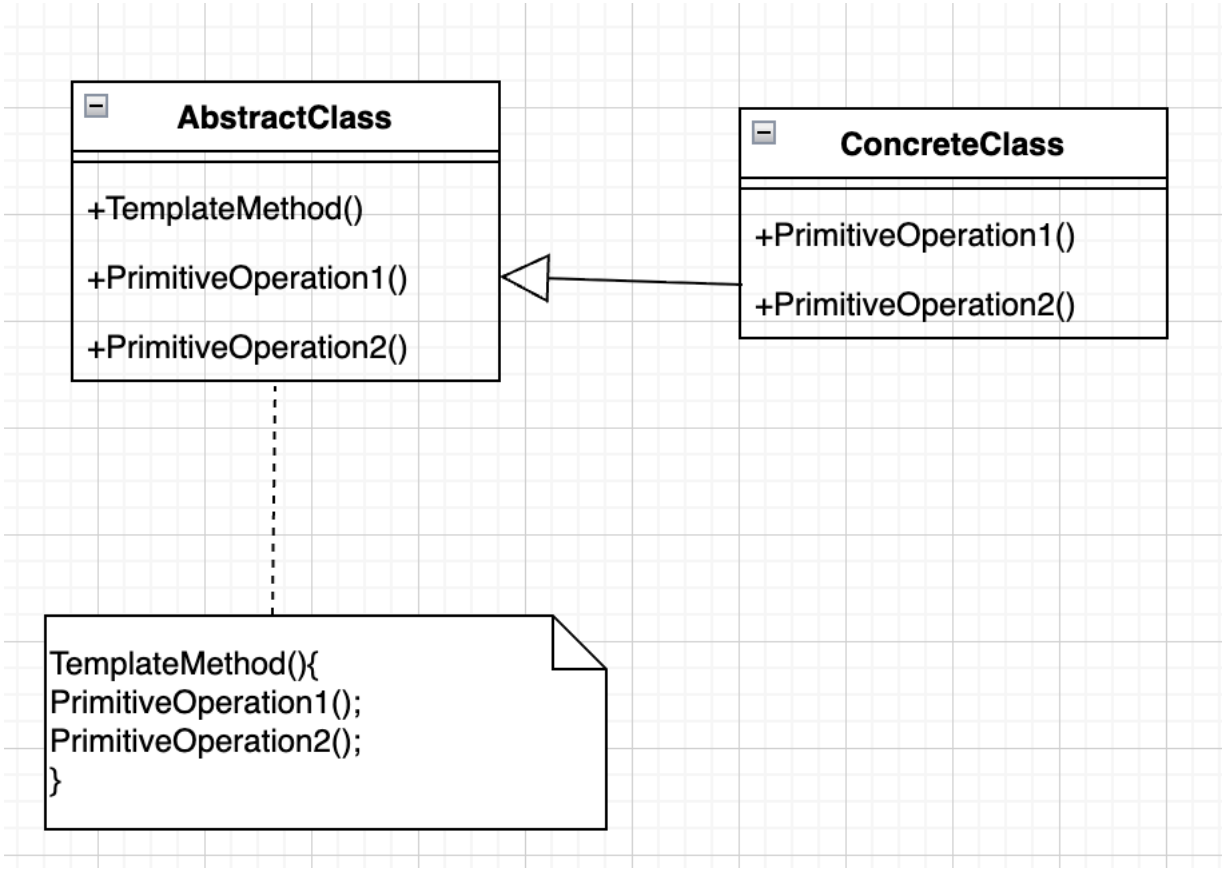
- Abstraction - абстрактний клас, що містить посилання на об’єкт реалізації (Implementor).
- RefinedAbstraction - конкретна реалізація абстракції.
- Implementor - інтерфейс реалізації.
- ConcreteImplementorA/B - конкретні реалізації інтерфейсу.

Взаємодія: Abstraction викликає методи Implementor, делегуючи реальну роботу конкретній реалізації. Таким чином, можна змінювати або розширювати абстракцію і реалізацію окремо.

10. Яке призначення шаблону «Шаблонний метод» (Template Method)?

Шаблон «*Шаблонний метод*» визначає загальну структуру алгоритму, дозволяючи підкласам змінювати окремі кроки, не змінюючи сам алгоритм.

11. Нарисуйте структуру шаблону «Шаблонний метод».



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

- AbstractClass - визначає загальний алгоритм у методі templateMethod() і описує абстрактні кроки.
- ConcreteClass - реалізує конкретні кроки алгоритму.

Взаємодія: клієнт викликає templateMethod(), який у свою чергу викликає визначені в підкласі операції.

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

- *Шаблонний метод* - визначає послідовність дій алгоритму, дозволяючи змінювати окремі кроки.
- *Фабричний метод* - відповідає за створення об’єктів, дозволяючи підкласам вирішувати, який саме клас буде створено.

Тобто *Template Method* це про алгоритм, а *Factory Method* - про створення об’єктів

14. Яку функціональність додає шаблон «Міст»?

Шаблон «*Міст*» додає можливість незалежно змінювати абстракцію і реалізацію, не змінюючи код іншої частини. Він спрощує розширення системи, зменшує дублювання коду та дозволяє легко поєднувати різні реалізації з різними абстракціями.