МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних технологій, автоматики та метрології

кафедра "Електронних обчислювальних машин"

Звіт

з лабораторної роботи №2

дисципліни «Кросплатформні засоби програмування»

на тему: «КЛАСИ ТА ПАКЕТИ»

Варіант 16

Виконав:
ст. гр. КІ-305
Маоиновський В.В

Прийняв:
Олексів М.В.

Львів – 2025

Мета роботи: ознайомитися з процесом розробки класів та пакетів мовою Java.

ЗАВДАННЯ

Написати та налагодити програму на мові Java, що реалізує у вигляді класу предметну область згідно варіанту. Програма має задовольняти наступним вимогам:

• програма має розміщуватися в пакеті Група.Прізвище.Lab2;

16. Аудіоплеєр

• клас має містити мінімум 3 поля, що є об’єктами класів, які описують складові частини предметної області;

• клас має містити кілька конструкторів та мінімум 10 методів;

• для тестування і демонстрації роботи розробленого класу розробити клас-драйвер;

• методи класу мають вести протокол своєї діяльності, що записується у файл;

• розробити механізм коректного завершення роботи з файлом (не надіятися на метод finalize());

• програма має володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.

1. Автоматично згенерувати документацію до розробленої програми.

2. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.

3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.

Виконання завдання:

Клас Main

```
public class AudioPlayerApp {
    public static void main(String[] args) throws Exception {
        try (AudioPlayer player = new AudioPlayer()) {

            // === Додаємо 3 пісні автоматично ===
            player.addTrack(new Track("Neon Samurai", "JJ SHOP", 210));
            player.addTrack(new Track("Midnight Drive", "SynthFox", 185));
            player.addTrack(new Track("Cyber Mirage", "OrionDeMirage", 240));
            out.println(" Added 3 demo tracks to playlist.\n");
```

```
Scanner sc = new Scanner(System.in);
int choice;

do {
   out.println("\n=== AUDIO PLAYER MENU ===");
   out.println("1 - Add track");
   out.println("2 - Play");
   out.println("3 - Pause");
   out.println("4 - Stop");
   out.println("5 - Next track");
   out.println("6 - Previous track");
   out.println("7 - Set volume");
   out.println("8 - Mute / Unmute");
   out.println("9 - Show status");
   out.println("0 - Exit");
   out.println("============================");
   out.print("Enter choice: ");

   while (!sc.hasNextInt()) {
      out.print("Enter number: ");
      sc.next();
   }
   choice = sc.nextInt();

   switch (choice) {
      case 1 -> {
         sc.nextLine(); // очистити буфер
         out.print("Enter track title: ");
         String title = sc.nextLine();
         out.print("Enter artist: ");
         String artist = sc.nextLine();
         out.print("Enter duration (sec): ");
         int dur = sc.nextInt();
         player.addTrack(new Track(title, artist, dur));
         out.println(" Track added.");
      }
```

```
case 2 -> {
    player.play();
    out.println("  Playing current track...");
    out.println(player.getStatus());
}
case 3 -> {
    player.pause();
    out.println("  Paused.");
    out.println(player.getStatus());
}
case 4 -> {
    player.stop();
    out.println("  Stopped.");
    out.println(player.getStatus());
}
case 5 -> {
    player.next();
    out.println(" Next track...");
    out.println(player.getStatus());
}
case 6 -> {
    player.prev();
    out.println("  Previous track...");
    out.println(player.getStatus());
}
case 7 -> {
    out.print("Volume (0-100): ");
    int v = sc.nextInt();
    player.setVolume(v);
    out.println("Volume set to " + v + "%");
    out.println(player.getStatus());
}
case 8 -> {
    out.print("Mute (1) / Unmute (0): ");
    int m = sc.nextInt();
    if (m == 1) {
        player.mute();
```

```
                    out.println(" Muted");
                } else {
                    player.unmute();
                    out.println(" Unmuted");
                }
                out.println(player.getStatus());
            }
            case 9 -> out.println(player.getStatus());
            case 0 -> out.println(" Exiting player...");
            default -> out.println("Unknown command.");
          }
        } while (choice != 0);
    }
  }
}
```

Клас AudioPlayer

```
package KI305.Malynovskyi.Lab2;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.util.Objects;

/**
 * Class <code>AudioPlayer</code> models a simple audio player device.
 * <p>
 * It contains several components (engine, playlist, speaker, storage)
 * and writes a protocol of operations to a log file.
 * </p>
 * @author Malynovskyi
 * @version 1.0
 */
public class AudioPlayer implements AutoCloseable {
```

```
    private final PlaybackEngine engine;
    private final Playlist playlist;
    private final Speaker speaker;
    private final Storage storage;
    private final PrintWriter fout;

    /**
     * Default constructor: creates empty playlist and default components.
     * @throws FileNotFoundException if log file cannot be created
     */
    public AudioPlayer() throws FileNotFoundException {
        this(new Playlist(), new Speaker(), new Storage("Internal", 1024));
    }

    /**
     * Constructor with components.
     * @param pl playlist
     * @param spk speaker
     * @param st storage
     * @throws FileNotFoundException if log file cannot be created
     */
    public AudioPlayer(Playlist pl, Speaker spk, Storage st) throws
FileNotFoundException {
        this.engine = new PlaybackEngine();
        this.playlist = Objects.requireNonNull(pl);
        this.speaker = Objects.requireNonNull(spk);
        this.storage = Objects.requireNonNull(st);
        this.fout = new PrintWriter(new File("Log.txt"));
        log("AudioPlayer created");
    }

    /** Starts playback from current index. */
    public void play() {
        if (playlist.isEmpty()) {
            log("play() -> playlist is empty");
            return;
        }
```

```
    Track t = playlist.current();
    engine.play(t);
    log("play() -> " + t.title());
}

/** Pauses playback. */
public void pause() {
    engine.pause();
    log("pause()");
}

/** Stops playback. */
public void stop() {
    engine.stop();
    log("stop()");
}

/** Skips to the next track and starts playing it. */
public void next() {
    if (playlist.isEmpty()) {
        log("next() -> playlist is empty");
        return;
    }
    Track t = playlist.next();
    engine.play(t);
    log("next() -> " + t.title());
}

/** Returns to the previous track and starts playing it. */
public void prev() {
    if (playlist.isEmpty()) {
        log("prev() -> playlist is empty");
        return;
    }
    Track t = playlist.prev();
    engine.play(t);
    log("prev() -> " + t.title());
```

```
}

/**
 * Seeks to a position (in seconds) on the current track.
 * @param seconds position in seconds
 */
public void seekTo(int seconds) {
    engine.seekTo(seconds);
    log("seekTo(" + seconds + "s)");
}

/**
 * Adds a track to the playlist.
 * @param t track to add
 */
public void addTrack(Track t) {
    playlist.add(t);
    log("addTrack(" + t + ") size=" + playlist.size());
}

/**
 * Removes a track by index.
 * @param index index of track
 */
public void removeTrack(int index) {
    Track removed = playlist.remove(index);
    log("removeTrack(index=" + index + ") -> " + removed);
}

/**
 * Sets volume [0..100].
 * @param value volume
 */
public void setVolume(int value) {
    speaker.setVolume(value);
    log("setVolume(" + value + ")");
}
```

```
    /** Mutes the speaker. */
    public void mute() {
        speaker.mute();
        log("mute()");
    }

    /** Unmutes the speaker. */
    public void unmute() {
        speaker.unmute();
        log("unmute()");
    }

    /** @return textual playback status */
    public String getStatus() {
        String s = "Status{state=" + engine.getState() + ", volume=" +
speaker.getVolume()
                + ", muted=" + speaker.isMuted() + ", track=" + (playlist.isEmpty() ?
"-" : playlist.current().title())
                + "}";
        log("getStatus() -> " + s);
        return s;
    }

    /** Writes a line into the log with timestamp and flushes. */
    private void log(String message) {
        fout.print(LocalDateTime.now() + " :: " + message + "\n");
        fout.flush();
    }

    /** Releases resources (log file). */
    public void dispose() {
        log("dispose()");
        fout.close();
    }

    /** AutoCloseable support for try-with-resources. */
```

```
    @Override
    public void close() {
        dispose();
    }
}
```

Клас PlaybackEngine

```
package KI305.Malynovskyi.Lab2;

/**
 * Class <code>PlaybackEngine</code> simulates playback state machine.
 */
class PlaybackEngine {

   enum State { STOPPED, PLAYING, PAUSED }

   private State state = State.STOPPED;
   private Track current;
   private int positionSec = 0;

   public void play(Track t) {
      current = t;
      positionSec = 0;
      state = State.PLAYING;
   }

   public void pause() {
      if (state == State.PLAYING) state = State.PAUSED;
   }

   public void stop() {
      state = State.STOPPED;
      positionSec = 0;
   }

   public void seekTo(int seconds) {
      if (seconds < 0) seconds = 0;
```

```
        positionSec = seconds;
    }

    public State getState() {
        return state;
    }

    public Track getCurrent() { return current; }

    public int getPositionSec() { return positionSec; }
}
```

Клас Battery

```
package KI305.Malynovskyi.Lab2;

import java.util.ArrayList;
import java.util.List;

/**
 * Class <code>Playlist</code> stores tracks and a cursor.
 */
class Playlist {
    private final List<Track> tracks = new ArrayList<>();
    private int index = 0;

    public void add(Track t) {
        tracks.add(t);
        if (tracks.size() == 1) index = 0;
    }

    public Track remove(int i) throws IndexOutOfBoundsException {
        Track r = tracks.remove(i);
        if (index >= tracks.size()) index = Math.max(0, tracks.size() - 1);
        return r;
    }

    public boolean isEmpty() { return tracks.isEmpty(); }
```

```
    public int size() { return tracks.size(); }

    public Track current() { return tracks.get(index); }

    public Track next() {
        if (tracks.isEmpty()) return null;
        index = (index + 1) % tracks.size();
        return tracks.get(index);
    }

    public Track prev() {
        if (tracks.isEmpty()) return null;
        index = (index - 1 + tracks.size()) % tracks.size();
        return tracks.get(index);
    }
}
```

Клас Speaker

```
package KI305.Malynovskyi.Lab2;

/**
 * Class <code>Speaker</code> controls volume and mute state.
 */
class Speaker {
    private int volume = 50;
    private boolean muted = false;

    public void setVolume(int v) {
        if (v < 0) v = 0;
        if (v > 100) v = 100;
        this.volume = v;
    }
```

```
    public void mute() { muted = true; }
    public void unmute() { muted = false; }

    public int getVolume() { return volume; }

    public boolean isMuted() { return muted; }
}
```

Клас Storage

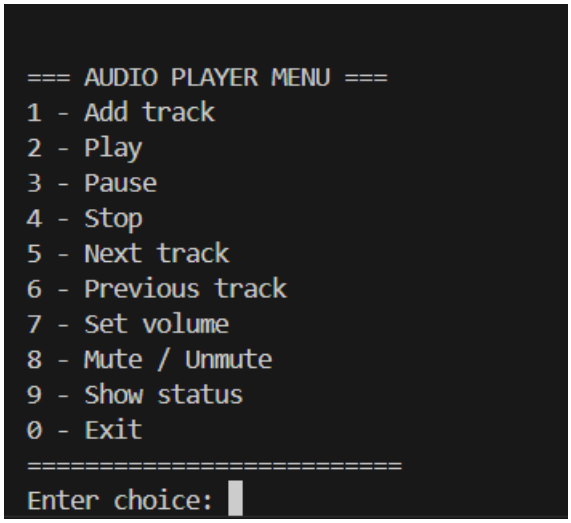
```
package KI305.Malynovskyi.Lab2;
/**
 * Class <code>Storage</code> represents a storage device where audio files live.
 */
class Storage {
    private final String name;
    private final int capacityMb;
    private int usedMb = 0;

    public Storage(String name, int capacityMb) {
        this.name = name;
        this.capacityMb = capacityMb;
    }
    public boolean allocate(int sizeMb) {
        if (usedMb + sizeMb > capacityMb) return false;
        usedMb += sizeMb;
        return true;
    }
    public void free(int sizeMb) {
        usedMb = Math.max(0, usedMb - sizeMb);
    }
    public String getInfo() {
        return name + " [" + usedMb + "/" + capacityMb + " MB]";
    }
}
```

Клас Track

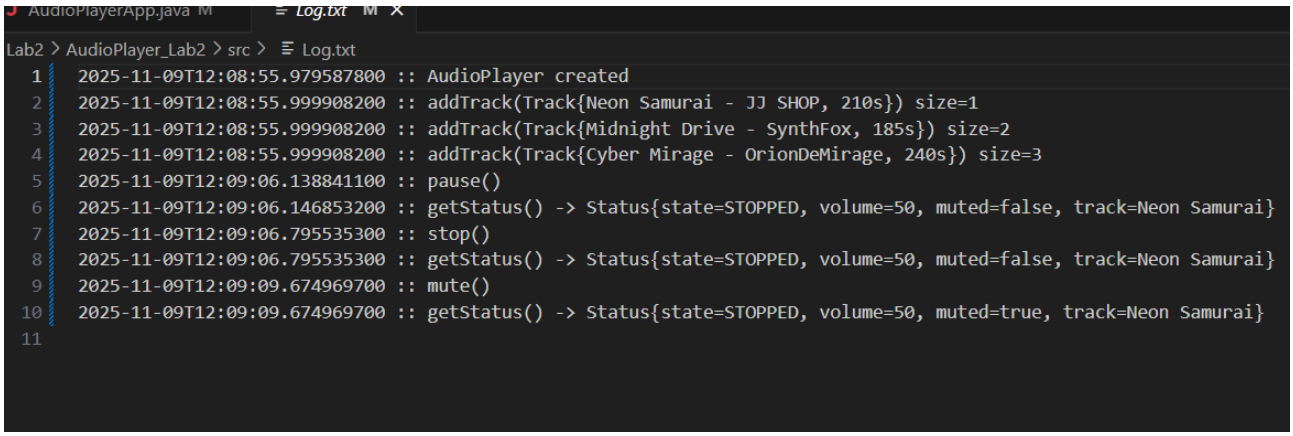
```
package KI305.Malynovskyi.Lab2;

/**
```

```
 * Immutable value object for a track.
 */
public record Track(String title, String artist, int durationSec) {
    @Override
    public String toString() {
        return "Track{" + title + " - " + artist + ", " + durationSec + "s}";
    }
}
```

Результат виконання програми:



Txt-файл з записаною інформацією:



Висновок: На лабораторній роботі я ознайомився з процесом розробки класів та пакетів мовою Java.

https://github.com/Ma1ik-111/KZP/tree/main