Міністерство освіти і науки України
Національний університет «Львівська політехніка»

Кафедра ЕОМ

Звіт

до лабораторної роботи № 3

З дисципліни: «Кросплатформенні засоби програмування»

На тему: **«СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ»**

Варіант 16

Виконав: ст.гр. КІ-301
Онисько М.М.
Прийняв: доц.
Майдан М.В.

Львів 2023

Мета: ознайомитися з спадкуванням та інтерфейсами у мові Java.

Завдання:
1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №3, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №3, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab4 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.
4. Дати відповідь на контрольні запитання.

Виконання роботи:

Код програми:

AudioPlayerApp

```java
/**
* lab3 package
*/
package KI301.Onysko.Lab3;

import java.io.*;


/**
* Audioplayer Application class implements main method for AudioPlayer class possibilities demonstration
* @author MYKOLA ONYSKO
* @version 1.0
*/

public class AudioPlayerApp {
	/**
	* @param args
	* @throws FileNotFoundException
	*/
	public static void main(String[] args) throws FileNotFoundException{

		AudioRecorder audioplayer = new AudioRecorder("IPhone", "IPod 4", 99);
		Song song1 = new Song("Запис1");
		audioplayer.startRecording(song1);
		audioplayer.waitingAbout(song1, 130);
		audioplayer.stopRecording(song1);
		audioplayer.saveRecording(song1);
		Song song2 = new Song("Запис2");
		audioplayer.startRecording(song2);
		audioplayer.waitingAbout(song2, 130);
		audioplayer.stopRecording(song2);
		audioplayer.saveRecording(song2);
		audioplayer.displayInfo();
		audioplayer.displaySongs();
		audioplayer.connectHeadset();
```

```
            audioplayer.disconnectHeadset();


        }
}
```

AudioPlayer

```
/**
 * Lab3 package
 */
package KI301.Onysko.Lab3;
import java.io.*;
import java.util.ArrayList;
import java.io.FileNotFoundException;
/**
 * Class <code>Audioplayer</code> implements audioplayer
 * @author MYKOLA ONYSKO
 * @version 1.0
 */
public abstract class AudioPlayer{
       PrintWriter fout;
       String brand;
       String model;
       private ArrayList <Playlist> playlists;
       Playlist playlist;
       Song song;
       Battery battery;
       /**
        * Constructor
        * @throws FileNotFoundException
        */
       public AudioPlayer() throws FileNotFoundException{
              String brand = "невідомий";
              String model = "невідома";
              playlists = new ArrayList <Playlist>();
              playlist = new Playlist();
              battery = new Battery();
              song = new Song();
              fout = new PrintWriter(new File("Log.txt"));
       }
       /**
        * Constructor
        * @param <code>brand</code> Brand of audioplayer
        * @param <code>model</code> Model of audioplayer
        * @param <code>charge</code> Battery charge
        * @throws FileNotFoundException
        */
       public AudioPlayer(String brand, String model, int charge) throws
FileNotFoundException{
              this.brand = brand;
              this.model = model;
              playlists = new ArrayList <Playlist>();
              playlist = new Playlist();
              battery = new Battery(charge);
              song = new Song();
              fout = new PrintWriter(new File("Log.txt"));
       }


       /**
        * Method returns a playlist with a list of songs
        * @param <code>playlist</code>
```

```
	 */
	public void addPlaylist(Playlist playlist) {
		this.playlists.add(playlist);
		System.out.println("Додано плейлист " + playlist.getPlaylistName() +
"\n");
		fout.println("Додано плейлист " + playlist.getPlaylistName() + "\n");
		fout.flush();
	}
	/**
	 * Method selects the current playlist
	 * @param <code>playlist</code>
	 */
	public void setPlaylist(Playlist playlist) {
		for(Playlist tmpPlaylist : playlists) {
			if(tmpPlaylist == playlist) {
				this.playlist = playlist;
			}
		}
		System.out.println("Вибрано плейлист " + playlist.getPlaylistName() +
"\n");
		fout.println("Вибрано плейлист " + playlist.getPlaylistName() + "\n");
		fout.flush();
	}
	//є
	/**
	 * Method selects the current music
	 * @param song
	 */
	public void setSong(Song song) {
		for(Song tmpSong : playlist.getSongs()) {
			if(tmpSong == song) {
				this.song = song;
			}
		}
		System.out.println("Вибрано пісню " + song.getSongName() + "\n");
		fout.println("Вибрано пісню " + song.getSongName() + "\n");
		fout.flush();
	}
	/**
	 * Method turns the music on and off
	 */
	public void playback() {
		if(song.playback()) {
			System.out.println("Грає пісня " + song.getSongName() + "\n");
			fout.println("Грає пісня " + song.getSongName() + "\n");
		} else {
			System.out.println("Призупинена пісня " + song.getSongName() +
"\n");
			fout.println("Призупинена пісня " + song.getSongName() + "\n");
		}
		fout.flush();
	}
	/**
	 * Method deletes the current playlist
	 * @param playlist
	 */
	public void removePlaylist(Playlist playlist) {
		int i = this.playlists.indexOf(playlist);
		this.playlists.remove(i);
		System.out.println("Видалено плейлист " + playlist.getPlaylistName() +
"\n");
		fout.println("Видалено плейлист " + playlist.getPlaylistName() + "\n");
		fout.flush();
	}
```

```
	/**
	 * Method switches to the next song in the playlist
	 */
	public void skipToNextSong() {

		int i = playlist.getSongs().indexOf(song);
		if(i >= 0 && i < playlist.getSongs().size() - 1) {
			song = playlist.getSongs().get(++i);
		}else {
			song = playlist.getSongs().get(0);
		}
		System.out.println("Тепер грає пісня: " + song.getSongName());
		fout.println("Тепер грає пісня: " + song.getSongName());
		fout.flush();
	}
	/**
	 * Method switches to the previous song in the playlist
	 */
	public void skipToPreviousSong() {
		int i = playlist.getSongs().indexOf(song);
		if(i > 0) {
			song = playlist.getSongs().get(--i);
		}else if(i==0){
			int lastIndex = playlist.getSongs().size()-1;
			song = playlist.getSongs().get(lastIndex);
		}
		System.out.println("Тепер грає пісня: " + song.getSongName());
		fout.println("Тепер грає пісня: " + song.getSongName());
		fout.flush();
	}
	/**
	 * method displays information about the audio player
	 */
	public void displayInfo( ) {

		System.out.println("Марка аудіопрогравача: " + brand +
					"\nМодель аудіопрогравача: "  + model +
						"\nПлейлисти:\n ");
		fout.println("Марка аудіопрогравача: " + brand +
				"\nМодель аудіопрогравача: "  + model +
					"\nПлейлисти:\n ");
		for(Playlist playlist : playlists) {
			System.out.println(playlist.getPlaylistName());
			fout.println(playlist.getPlaylistName());
		}
		System.out.println("\n");
		fout.println("\n");
		fout.flush();
	}
	/**
	 * Method displays the songs in the selected playlist
	 */
	public void displaySongs() {
		System.out.println("Пісні плейлиста '"+
this.playlist.getPlaylistName()+"':\n");
		fout.println("Пісні плейлиста '"+
this.playlist.getPlaylistName()+"':\n");
		for (Song song : this.playlist.getSongs()){
			System.out.println(song.getSongName());
			fout.println(song.getSongName());
		}
		System.out.println("\n");
		fout.println("\n");
		fout.flush();
```

```
	}
	/**
	 * Method deletes the current music
	 * @param song
	 */
	public void removeSong(Song song) {
		playlist.removeSong(song);
	}
	/**
	 * Method displays the current battery charge
	 */
	public void getCurrentLevel() {
		battery.getCurrentLevel();
		System.out.println("Поточна батарея:" + battery.getCurrentLevel() +
"\n");
		fout.println("Поточна батарея:" + battery.getCurrentLevel() + "\n");
		fout.flush();
	}
	/**
	 * Method displays whether the battery is currently charging
	 */
	public void isCharging() {
		battery.isCharging();
		if(battery.isCharging()==true){
			System.out.println("Батарея заряджається\n");
			fout.println("Батарея заряджається\n");
			fout.flush();
		}else {
			System.out.println("Батарея не заряджається\n");
			fout.println("Батарея не заряджається\n");
			fout.flush();
		}
	}
	/**
	 * Method charges the battery
	 */
	public void chargeBattery() {
		battery.chargeBattery();
		if(battery.isCharging()) {
			if(battery.getCurrentLevel()<100) {
				System.out.println("Батарея почала заряджатись\n");
			  fout.println("Батарея почала заряджатись\n");
			}
			else {
				System.out.println("Батарея не потребує зарядки\n");
			  fout.println("Батарея не потребує зарядки\n");
			}
		}else {
			System.out.println("Батарея перестала заряджатись\n");
		  fout.println("Батарея перестала заряджатись\n");
		}
		fout.flush();
	}
	/**
	 * Method dicharges battery
	 * @param energyConsumed
	 */
	public void dischargeBattery(int energyConsumed) {
		 System.out.println("Заряд батареї зменшився");
		 fout.println("Заряд батареї зменшився");
		 int t = battery.dischargeBattery(energyConsumed);
		 if(t <= 0)
		 {
			  System.out.println("Батарея повністю розряджена\n");
```

```
                    fout.println("Батарея повністю розряджена\n");
                }else {
                        System.out.println("Стан батареї:" + t + "\n");
                       fout.println("Батарея повністю розряджена\n");
                }
        fout.flush();
        }
         // Метод для початку запису аудіо
        abstract void startRecording(Song song);
         // Метод для завершення запису аудіо
        abstract void stopRecording(Song song);

         // Метод для збереження записаного аудіos
        abstract void saveRecording(Song song);

        abstract void waitingAbout(Song song, int time);


        public void dispose()
        {
        fout.close();
        }

}


class Song{

        private String songName;
        private String songLength;
        private boolean isPlayed;
        /*
         * Constructor
         */
        public Song(){
                songName = "невідома";
                songLength = "00:00";
                isPlayed = false;
        }
        /**
         * Constructor
         * @param <code>songName</code> Name of song
         * @param <code>songLength</code> Song length
         */
        public Song(String songName, String songLength){
                this.songName = songName;
                this.songLength = songLength;
                isPlayed = false;
        }

        public Song(String songName){
                this.songName = songName;
                isPlayed = false;
        }
        /**
         * Method returns the name of the song
         * @return songName
         */
        public String getSongName() {
                return songName;
        }
        /*
         * Method returns the length of the song
         * @return songLength
```

```
	 */
	public String getSongLength() {
		return songLength;
	}
	/**
	 * Method to play/pause music
	 * @return true, if the song is not playing, false in other cases
	 */
	public boolean playback() {

		if(isPlayed == false) {
			isPlayed = true;
			return isPlayed;
		}else{
			isPlayed = false;
			return isPlayed;
		}
	}
}
class Playlist{

	private ArrayList <Song> songs;
	private String playlistName;
	/**
	 * Constructor
	 */
	public Playlist(){
		playlistName = "невідома";
		songs = new ArrayList<Song>();
	}
	/**
	 * Constructor
	 * @param <code>playlistName</code> Name of playlist
	 */
	public Playlist(String playlistName){
		this.playlistName = playlistName;
		songs = new ArrayList<Song>();
	}
	/**
	 * Method adds a song
	 * @param song
	 */
	public void addSong(Song song) {
		this.songs.add(song);
	}
	/**
	 * Method returns the name of the playlist
	 * @return playlistName
	 */
	public String getPlaylistName() {
		return playlistName;
	}
	/**
	 * Method sets the current playlist
	 * @param playlistName
	 * @return playlistName
	 */
	public void setPlaylistName(String playlistName) {
		this.playlistName = playlistName;
	}
	/**
	 * Method deletes the song
	 * @param song
	 */
```

```
	public void removeSong(Song song) {
		int i = this.songs.indexOf(song);
		this.songs.remove(i);
	}
	/**
	 * Method returns songs
	 * @return songs
	 */
	public ArrayList <Song> getSongs(){
		return songs;
	}
}
class Battery{

	private int currentLevel;
	private boolean charging;
	private int energyConsumed;
	/**
	 * Constructor
	 */
	public Battery() {
		int currentLevel = 100;
		boolean charging = false;
		int energyConsumed = 0;
	}
	/**
	 * Constructor
	 * @param <code>currentLevel</code>
	 */
	public Battery(int currentLevel) {
		this.currentLevel = currentLevel;
		boolean charging = false;
		int energyConsumed = 0;
	}
	 /**
     * Method to get the current battery level.
     * @return current battery level in percent
     */
	   public int getCurrentLevel() {
	     return currentLevel;
	}
	/**
	 * Method to check if the battery is charging.
	 * @return true, if the battery is charging, false in other cases
	 */
	public boolean isCharging() {
	    return charging;
	}
	/**
	 * Method for charging the battery.
	 * Checks the battery for charging and for reaching the maximum charge level.
	 */
	public boolean chargeBattery() {
	   if(!charging)
	   {
		charging = true;
		return charging;

	   }
	   else {
		 charging = false;
		return charging;
	   }
	}
```

```
    /**
     * A method for discharging the battery of an audioplayer
     * @param energyConsumed
     */
    public int dischargeBattery(int energyConsumed) {
        currentLevel -= energyConsumed;

       if(currentLevel <= 0)
       {
             currentLevel = 0;
       }
       return currentLevel;
    }
}
```

AudioRecorder

```
package KI301.Onysko.Lab3;
import java.io.*;
import java.util.ArrayList;


/**
 * Interface for an audio recorder.
 */
interface Recorder {

        /**
     * Method to connect the headset. Subclasses should implement this method.
     */
      void connectHeadset();
       /**
     * Method to disconnect the headset. Subclasses should implement this method.
     */
      void disconnectHeadset();

}

/**
 * Class implementing the Recorder interface and extending the AudioPlayer class.
 * This class represents an AudioRecorder with the ability to record and save audio.
 * @author MYKOLA ONYSKO
 * @version 1.0
 */
public class AudioRecorder extends AudioPlayer implements Recorder {

        private String fileName;
        private int time;
        Playlist playlistRec;
        boolean check = false;
        boolean connectedHeadset = false;

          /**
           * Constructor for an AudioRecorder without specified brand and model.
           * Initializes default values and creates the "Recordered" playlist.
           * @throws FileNotFoundException If the log file cannot be created.
           */
          public AudioRecorder() throws FileNotFoundException {
              super();
              // Створення плейлиста "Recordered"
              playlistRec = new Playlist("Recordered");
          }
```

```java
    /**
     * Constructor for an AudioRecorder with specified brand, model, and
initial battery charge.
     * Initializes values, sets the battery charge, and creates the
"Recordered" playlist.
     * @param brand Brand of the audioplayer
     * @param model Model of the audioplayer
     * @param charge Initial battery charge
     * @throws FileNotFoundException If the log file cannot be created.
     */
    public AudioRecorder(String brand, String model, int charge) throws
FileNotFoundException {
        super(brand, model, charge);
        // Створення плейлиста "Recordered"
        playlistRec = new Playlist("Recordered");
    }
    /**
     * Formats time in minutes and seconds.
     * @param time The time in seconds.
     * @return Formatted time in the "mm:ss" format.
     */
    private static String formatTime(int time) {
        int minutes = time / 60;
        int seconds = time % 60;
        return String.format("%d:%02d", minutes, seconds);
    }

    @Override
    public void startRecording(Song song) {
        System.out.println("Початок запису аудіо");
        check = true;
    }

    @Override
    public void stopRecording(Song song) {
        System.out.println("Зупинка запису аудіо");
    }

    @Override
    public void saveRecording(Song song) {
       check = false;
        playlistRec.addSong(song);
        System.out.println("Пісню '" + song.getSongName() + "' додано до папки
'Recordered'.");
    }

    @Override
    public void waitingAbout(Song song, int time) {
       if(check == false) {
             System.out.println("НЕ МОЖНА");
       }
       else{
             this.time = time;
              String formattedTime = formatTime(time); // Перетворюємо хвилини
в секунди та форматуємо
              System.out.println("Пісня '" + song.getSongName() + "'
записується " + formattedTime);
            }

    }

    @Override
    public void displaySongs() {
```

```
                    System.out.println("Пісні плейлиста '"+
this.playlistRec.getPlaylistName()+"':\n");
                    for (Song song : this.playlistRec.getSongs()){
                          System.out.println(song.getSongName());
                    }
                    System.out.println("\n");

             }
             /**
              * {@inheritDoc}
              */
               @Override
               public void connectHeadset() {
                    if(connectedHeadset == false) {
                          connectedHeadset = true;
                          System.out.println("Навушники підключено");
                    }else {
                          System.out.println("Навушники вже підключено!");
                    }
                    // TODO Auto-generated method stub

               }
             /**
              * {@inheritDoc}
              */
               @Override
               public void disconnectHeadset() {
                    if(connectedHeadset == true) {
                          connectedHeadset = false;
                          System.out.println("Навушники відключено");
                    }else {
                          System.out.println("Навушники вже відключено!");
                    }

               }

 }
```

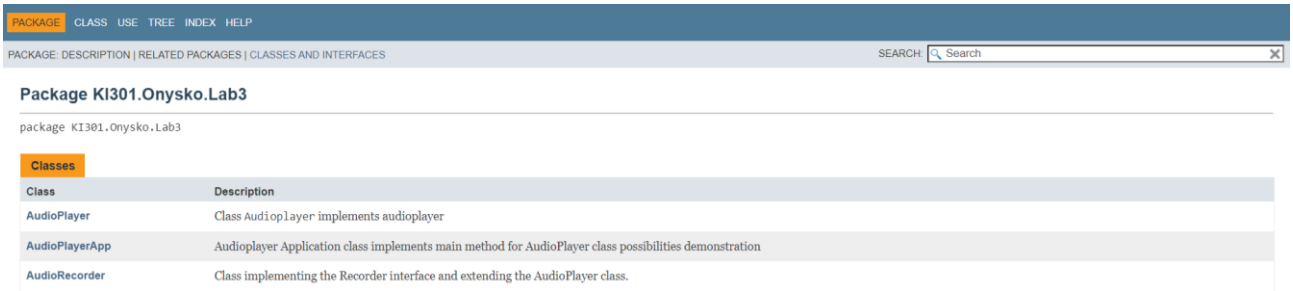
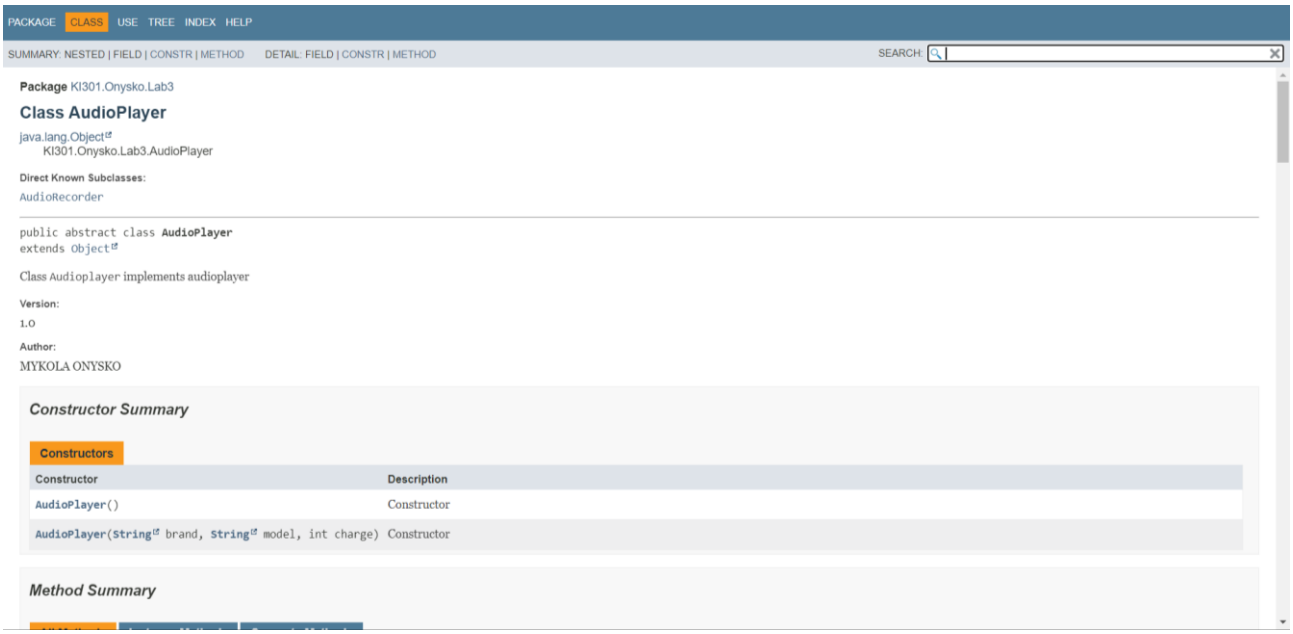


Рис.1 Документація Package KI301.Onysko.Lab3



PACKAGE CLASS USE TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH:

Package KI301.Onysko.Lab3

Class AudioPlayer

java.lang.Object
KI301.Onysko.Lab3.AudioPlayer

Direct Known Subclasses:
AudioRecorder

public abstract class AudioPlayer
extends Object

Class Audioplayer implements audioplayer

Version:
1.0

Author:
MYKOLA ONYSKO

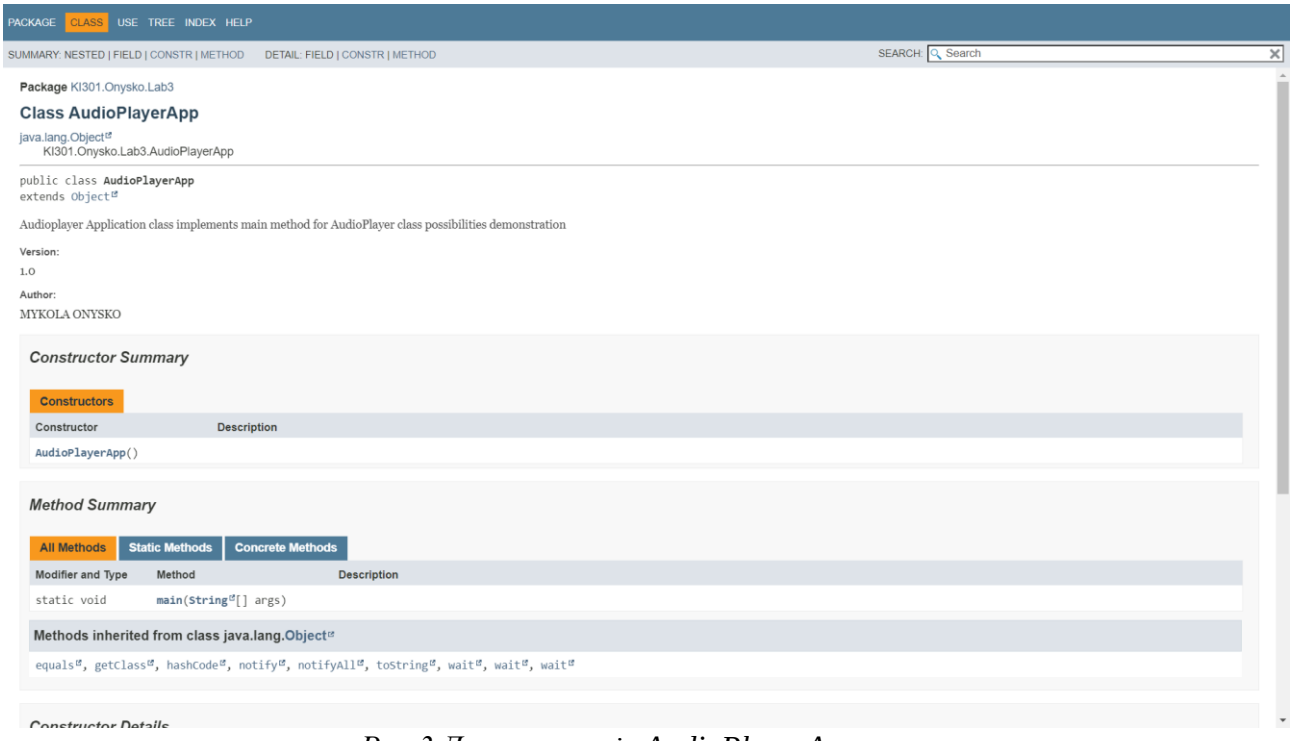
Constructor Summary

Constructors

Constructor	Description
AudioPlayer()	Constructor
AudioPlayer(String brand, String model, int charge)	Constructor

Method Summary

Рис.2 Документація AudioPlayer



PACKAGE CLASS USE TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Package KI301.Onysko.Lab3

Class AudioPlayerApp

java.lang.Object
KI301.Onysko.Lab3.AudioPlayerApp

public class AudioPlayerApp
extends Object

Audioplayer Application class implements main method for AudioPlayer class possibilities demonstration

Version:
1.0

Author:
MYKOLA ONYSKO

Constructor Summary

Constructors

Constructor	Description
AudioPlayerApp()	

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static void	main(String[] args)	

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Рис.3 Документація AudioPlayerApp

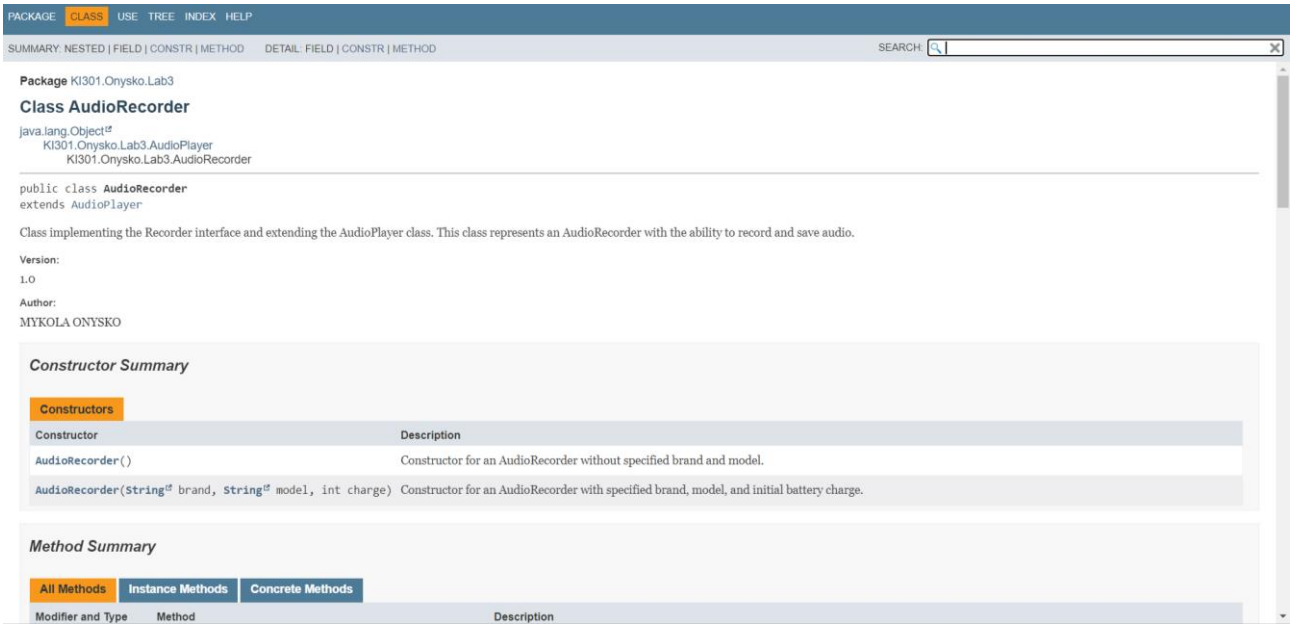
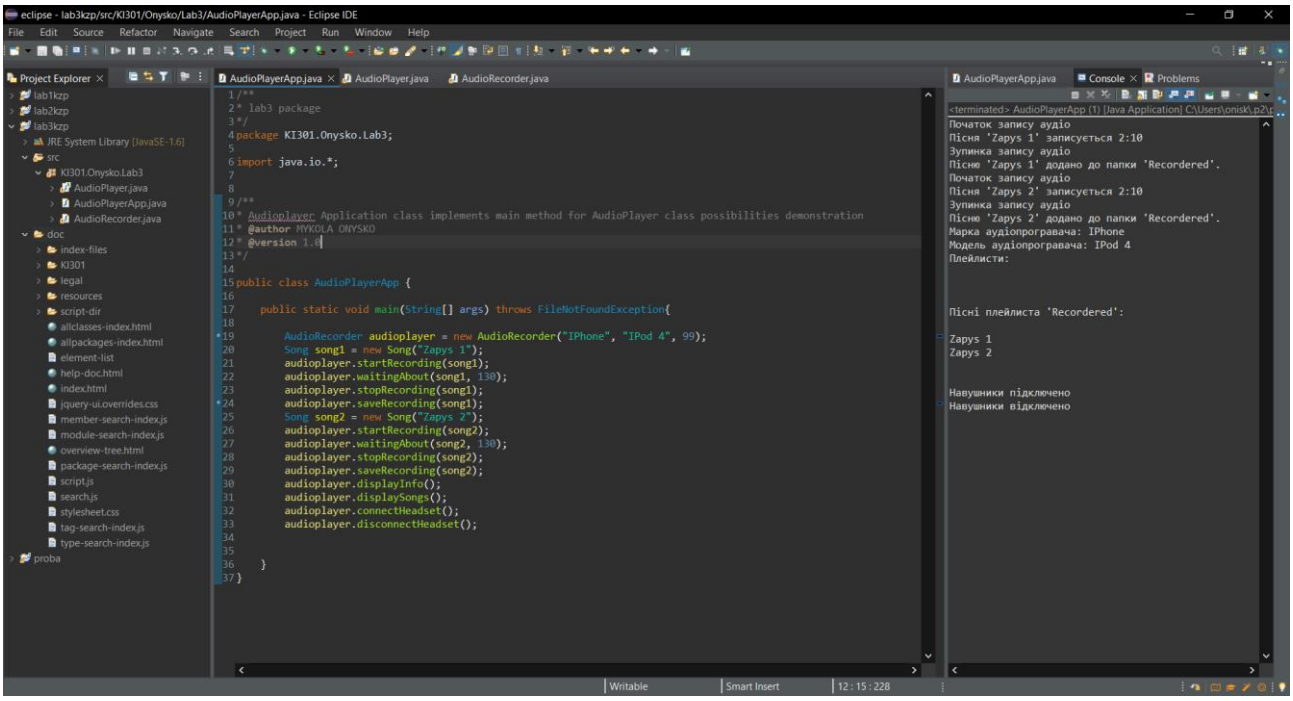


Рис.4 Документація AudioRecorder




```java
/**
* lab3 package
*/
package KI301.Onysko.Lab3;

import java.io.*;


/**
* Audioplayer Application class implements main method for AudioPlayer class possibilities demonstration
* @author MYKOLA ONYSKO
* @version 1.0
*/
public class AudioPlayerApp {

    public static void main(String[] args) throws FileNotFoundException{

        AudioRecorder audioplayer = new AudioRecorder("IPhone", "IPod 4", 99);
        Song song1 = new Song("Zapys 1");
        audioplayer.startRecording(song1);
        audioplayer.waitingAbout(song1, 130);
        audioplayer.stopRecording(song1);
        audioplayer.saveRecording(song1);
        Song song2 = new Song("Zapys 2");
        audioplayer.startRecording(song2);
        audioplayer.waitingAbout(song2, 130);
        audioplayer.stopRecording(song2);
        audioplayer.saveRecording(song2);
        audioplayer.displayInfo();
        audioplayer.displaySongs();
        audioplayer.connectHeadset();
        audioplayer.disconnectHeadset();


    }
}
```

Рис.5 Код та вивід програми

Висновок:

У ході виконання завдання була розроблена програма на мові Java, яка розширює абстрактний клас, реалізований у лабораторній роботі №3, для конкретної предметної області. Створено новий підклас, який забезпечує свій коректний функціонал та реалізує принаймні один інтерфейс.

Завдання було успішно виконано, і програма демонструє використання концепцій спадкування та інтерфейсів у мові Java. Крім того, забезпечено можливість генерації документації для подальшого зручного використання та розвитку програмного продукту

Контрольні питання:

1. Синтаксис реалізації спадкування.

Відповідь: Спадкування в мові Java використовується за допомогою ключового слова `extends` для класів і `implements` для інтерфейсів. Наприклад, `class Subclass extends Superclass` або `class MyClass implements MyInterface`.

2. Що таке суперклас та підклас?
Відповідь: Суперклас - це клас, від якого спадковується інший клас, відомий як підклас. Суперклас має загальні атрибути та методи, які можуть бути успадковані й використані підкласом.

3. Як звернутися до членів суперкласу з підкласу?
Відповідь: Для звернення до членів суперкласу використовується ключове слово `super`. Наприклад, `super.method()` викликає метод суперкласу.

4. Коли використовується статичне зв’язування при виклику методу?
Відповідь: Статичне зв'язування відбувається під час компіляції, коли виклик методу визначається на підставі типу посилання.

5. Як відбувається динамічне зв’язування при виклику методу?
Відповідь: Динамічне зв'язування відбувається під час виконання програми, коли виклик методу визначається на підставі типу об'єкта, на який вказує посилання.

6. Що таке абстрактний клас та як його реалізувати?
Відповідь: Абстрактний клас - це клас, який має хоча б один абстрактний метод. Його можна створити за допомогою ключового слова `abstract`. Наприклад, `abstract class MyAbstractClass { abstract void myAbstractMethod(); }`.

7. Для чого використовується ключове слово instanceof?
Відповідь: Ключове слово `instanceof` використовується для перевірки, чи об'єкт є екземпляром певного класу чи реалізує певний інтерфейс.

8. Як перевірити чи клас є підкласом іншого класу?
Відповідь: Це можна зробити за допомогою ключового слова `instanceof`. Наприклад, `if (obj instanceof Superclass) { // код }`.

9. Що таке інтерфейс?
Відповідь: Інтерфейс визначає набір методів, які повинні бути реалізовані класами, що його імплементують. Його можна створити за допомогою ключового слова `interface`.

10. Як оголосити та застосувати інтерфейс?
Відповідь: Інтерфейс оголошується за допомогою ключового слова `interface`, наприклад, `interface MyInterface { void myMethod(); }`. Для його застосування використовується ключове слово `implements` у класі, який реалізує цей інтерфейс. Наприклад, `class MyClass implements MyInterface { // код }`.