Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

ЗВІТ

до лабораторної роботи №3

З дисципліни: «Кросплатформні засоби програмування»

На тему: «СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ»

Варіант 16

Виконала:

ст. групи КІ-306

Мілян М.О.

Прийняв:

доцент кафедри ЕОМ

Олексів М.В.

Львів – 2024

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 №2, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №2, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab3 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

Тема згідно варіанту №16 – «Диктофон»

Хід роботи

Код програми:

DictaphoneDriver.java

```
package KI.Milian.Lab3;

import java.io.IOException;

/**
 * Клас-драйвер для тестування класу Dictaphone.
 */
public class DictaphoneDriver {
    public static void main(String[] args) {
        try {
            // Створюємо диктофон
            Dictaphone dictaphone = new Dictaphone();

            // Використання функціональності диктофона
            dictaphone.deviceFunctionality();
            dictaphone.startRecording();
            dictaphone.stopRecording();
            dictaphone.playRecording();
            dictaphone.playTrack("Recorded Track");

            // Закриття ресурсів
            dictaphone.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Dictaphone.java

```
package KI.Milian.Lab3;

import java.io.IOException;
```

```java
/**
 * Інтерфейс для пристроїв, що можуть записувати звук.
 */
interface Recordable {
    /**
     * Метод для початку запису.
     */
    void startRecording();

    /**
     * Метод для зупинки запису.
     */
    void stopRecording();
}

/**
 * Клас, що описує диктофон. Розширює абстрактний клас AudioPlayer та реалізує інтерфейс Recordable.
 */
public class Dictaphone extends AudioPlayer implements Recordable {
    private boolean isRecording;

    public Dictaphone(Speaker speaker, BluetoothModule bluetoothModule, Battery battery) throws IOException {
        super(speaker, bluetoothModule, battery);
        this.isRecording = false;
    }

    public Dictaphone() throws IOException {
        super();
        this.isRecording = false;
    }

    @Override
    public void startRecording() {
        if (getCapacity() <= 0) {
            System.out.println("Battery is empty. Cannot start recording.");
            return;
        }
        if (!isRecording) {
            isRecording = true;
            System.out.println("Recording started...");
            setCapacity(getCapacity()- 20); // Витрачаємо 20 мАг на початок запису
        } else {
            System.out.println("Already recording.");
        }
    }

    @Override
    public void stopRecording() {
        if (isRecording) {
            isRecording = false;
            System.out.println("Recording stopped.");
        } else {
            System.out.println("Not recording.");
        }
    }

    @Override
    public void deviceFunctionality() throws IOException {
        System.out.println("Dictaphone is ready for recording and playback.");
        log("Dictaphone initialized.");
    }

    public void playRecording() throws IOException {
```

```
        if (getCapacity() <= 0) {
            System.out.println("Battery is empty. Cannot play recording.");
            return;
        }
        System.out.println("Playing recorded audio...");
        setCapacity(getCapacity()- 30); // Витрачаємо 30 мАг на відтворення
запису
    }

    @Override
    public String toString() {
        return "Dictaphone{isRecording=" + isRecording + ", battery=" +
getCapacity() + "mAh}";
    }
}
```

AudioPlayer.java

```
package KI.Milian.Lab3;

import java.io.FileWriter;
import java.io.IOException;

/**
 * Клас, що описує аудіоплеєр.
 */
public abstract class AudioPlayer {
    private Speaker speaker;
    private BluetoothModule bluetoothModule;
    private Battery battery;
    private FileWriter logWriter;

    public AudioPlayer(Speaker speaker, BluetoothModule bluetoothModule, Battery
battery) throws IOException {
        this.speaker = speaker;
        this.bluetoothModule = bluetoothModule;
        this.battery = battery;
        this.logWriter = new FileWriter("audio_player_log.txt", true);
    }

    public AudioPlayer() throws IOException {
        this(new Speaker(50), new BluetoothModule(), new Battery(3000));
    }

    /**
     * Відтворює трек і зменшує заряд батареї.
     */
    public void playTrack(String track) throws IOException {
        if (battery.getCapacity() <= 0) {
            log("Battery is empty. Cannot play track.");
            System.out.println("Battery is empty. Please charge the device.");
            return;
        }

        if (bluetoothModule.isConnected()) {
            log("Playing track: " + track + " on device: " +
bluetoothModule.getConnectedDevice());
            System.out.println("Playing track: " + track + " on Bluetooth
device: " + bluetoothModule.getConnectedDevice());
        } else {
            log("Playing track: " + track + " on built-in speaker.");
            System.out.println("Playing track: " + track + " on built-in
speaker.");
        }
        battery.drainBattery(50); // Витрачаємо 50 мАг на кожен трек
```

```
        log("Battery capacity after playing track: " + battery.getCapacity() +
"mAh");
        System.out.println("Battery capacity after playing track: " +
battery.getCapacity() + "mAh");
    }

    /**
     * Підключаємося до Bluetooth-пристрою і зменшуємо заряд батареї.
     */
    public void connectBluetooth(String device) throws IOException {
        if (battery.getCapacity() <= 0) {
            log("Battery is empty. Cannot connect to Bluetooth.");
            System.out.println("Battery is empty. Please charge the device.");
            return;
        }

        bluetoothModule.connectToDevice(device);
        log("Connected to Bluetooth device: " + device);
        System.out.println("Connected to Bluetooth device: " + device);
        battery.drainBattery(30); // Витрачаємо 30 мАг на підключення до
Bluetooth
        log("Battery capacity after connecting to Bluetooth: " +
battery.getCapacity() + "mAh");
        System.out.println("Battery capacity after connecting to Bluetooth: " +
battery.getCapacity() + "mAh");
    }

    public void disconnectBluetooth() throws IOException {
        bluetoothModule.disconnect();
        log("Disconnected from Bluetooth device.");
        System.out.println("Disconnected from Bluetooth device.");
    }

    public void increaseVolume() throws IOException {
        speaker.setVolume(speaker.getVolume() + 10);
        log("Increased volume to: " + speaker.getVolume());
        System.out.println("Increased volume to: " + speaker.getVolume());
    }

    public void decreaseVolume() throws IOException {
        speaker.setVolume(speaker.getVolume() - 10);
        log("Decreased volume to: " + speaker.getVolume());
        System.out.println("Decreased volume to: " + speaker.getVolume());
    }

    public void chargeBattery(int amount) throws IOException {
        battery.setCapacity(battery.getCapacity() + amount);
        log("Charged battery by " + amount + "mAh. New capacity: " +
battery.getCapacity() + "mAh");
        System.out.println("Charged battery by " + amount + "mAh. New capacity:
" + battery.getCapacity() + "mAh");
    }

    protected void log(String message) throws IOException {
        logWriter.write(message + "\n");
    }

    public void close() throws IOException {
        if (logWriter != null) {
            logWriter.close();
        }
    }

    public int getCapacity() {
        return battery.getCapacity();
    }
```

```
    public void setCapacity(int capacity) {
        battery.setCapacity(capacity);
    }

    /**
     * Абстрактний метод для підкласів, що мають реалізовувати специфічні
функції.
     */
    public abstract void deviceFunctionality() throws IOException;


    @Override
    public String toString() {
        return "AudioPlayer{speaker=" + speaker + ", bluetoothModule=" +
bluetoothModule + ", battery=" + battery + "}";
    }
}
```

Battery.java

```
package KI.Milian.Lab3;

/**
 * Клас, що описує акумулятор для аудіоплеєра.
 */
public class Battery {
    private int capacity;  // Ємність батареї в мАг (мілліампер-години)

    public Battery(int capacity) {
        this.capacity = capacity;
    }

    public int getCapacity() {
        return capacity;
    }

    public void setCapacity(int capacity) {
        this.capacity = capacity;
    }

    /**
     * Зменшує заряд батареї на вказану кількість мАг.
     * Якщо заряд стає меншим за 0, встановлюється на 0.
     */
    public void drainBattery(int amount) {
        capacity -= amount;
        if (capacity < 0) {
            capacity = 0;
        }
    }

    @Override
    public String toString() {
        return "Battery{capacity=" + capacity + "mAh}";
    }
}
```

Speaker.java

```
package KI.Milian.Lab3;

/**
 * Клас, що описує динамік аудіоплеєра.
 */
public class Speaker {
    private int volume; // Гучність динаміка
```

```
    public Speaker(int volume) {
        this.volume = volume;
    }

    public int getVolume() {
        return volume;
    }

    public void setVolume(int volume) {
        this.volume = volume;
    }

    @Override
    public String toString() {
        return "Speaker{volume=" + volume + "}";
    }
}
```

BluetoothModule.java

```
package KI.Milian.Lab3;

/**
 * Клас, що описує Bluetooth модуль для аудіоплеєра.
 */
public class BluetoothModule {
    private boolean isConnected; // Стан підключення
    private String connectedDevice; // Підключений пристрій

    public BluetoothModule() {
        this.isConnected = false;
        this.connectedDevice = "None";
    }

    public void connectToDevice(String device) {
        this.isConnected = true;
        this.connectedDevice = device;
    }

    public void disconnect() {
        this.isConnected = false;
        this.connectedDevice = "None";
    }

    public boolean isConnected() {
        return isConnected;
    }

    public String getConnectedDevice() {
        return connectedDevice;
    }

    @Override
    public String toString() {
        return "BluetoothModule{isConnected=" + isConnected + ",
connectedDevice='" + connectedDevice + "'}";
    }
}
```

```
Dictaphone is ready for recording and playback.
Recording started...
Recording stopped.
Playing recorded audio...
Playing track: Recorded Track on built-in speaker.
Battery capacity after playing track: 2900mAh

Process finished with exit code 0
```

Рис.1 Вивід логу у консоль

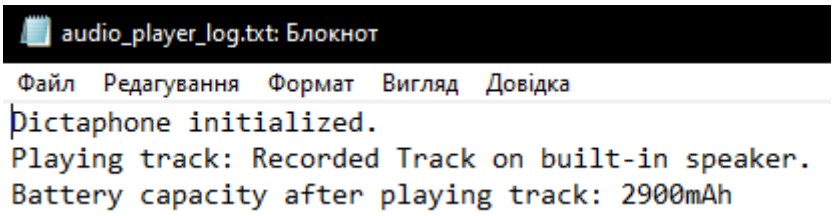


Рис.2 Вивід логу у текстовий файл

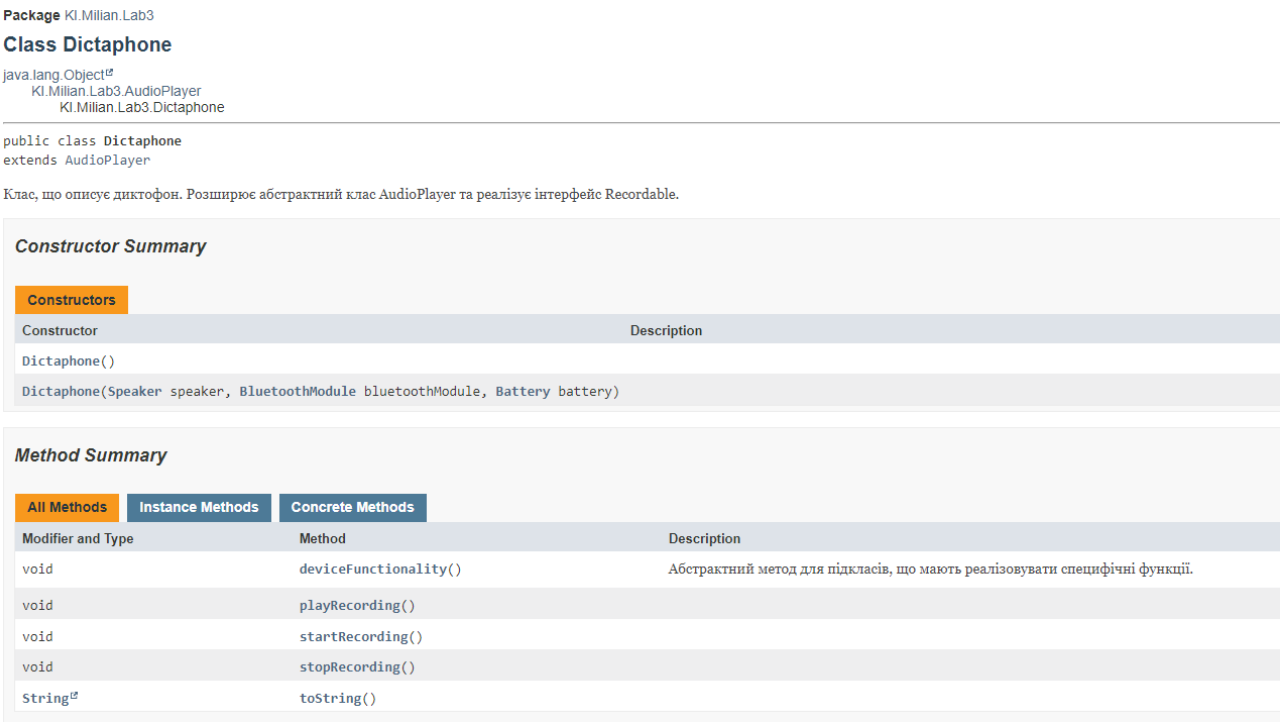
Package KI.Milian.Lab3

```
package KI.Milian.Lab3
```

Classes

Class	Description
AudioPlayer	Клас, що описує аудіоплеєр.
Battery	Клас, що описує акумулятор для аудіоплеєра.
BluetoothModule	Клас, що описує Bluetooth модуль для аудіоплеєра.
Dictaphone	Клас, що описує диктофон.
DictaphoneDriver	Клас-драйвер для тестування класу Dictaphone.
Speaker	Клас, що описує динамік аудіоплеєра.

Рис.3.1 Фрагмент згенерованої документації



Package KI.Milian.Lab3

Class Dictaphone

java.lang.Object
  KI.Milian.Lab3.AudioPlayer
    KI.Milian.Lab3.Dictaphone

```
public class Dictaphone
extends AudioPlayer
```

Клас, що описує диктофон. Розширює абстрактний клас AudioPlayer та реалізує інтерфейс Recordable.

Constructor Summary

Constructors

Constructor	Description
`Dictaphone()`	
`Dictaphone(Speaker speaker, BluetoothModule bluetoothModule, Battery battery)`	

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	deviceFunctionality()	Абстрактний метод для підкласів, що мають реалізовувати специфічні функції.
void	playRecording()	
void	startRecording()	
void	stopRecording()	
String	toString()	

Рис.3.2 Фрагмент згенерованої документації

Висновок: На лабораторній роботі я ознайомився зі спадкуванням та інтерфейсами у мові Java.