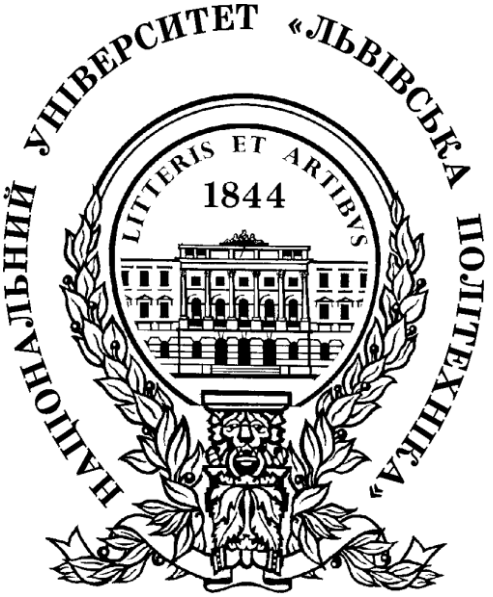
Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



ЗВІТ

до лабораторної роботи №9

З дисципліни: «Кросплатформні засоби програмування»

На тему: «ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ У PYTHON»

Варіант 16

Виконав:

ст. групи КІ-306

Мілян М.О.

Прийняв:

доцент кафедри ЕОМ

Олексів М.В.

Львів – 2024

Мета: оволодіти навиками реалізації парадигм об’єктно-орієнтованого програмування використовуючи засоби мови Python.

Завдання:

1. Написати та налагодити програму на мові Python згідно варіанту. Програма має задовольняти наступним вимогам:
    - класи програми мають розміщуватися в окремих модулях в одному пакеті;
    - точка входу в програму (main) має бути в окремому модулі;
    - мають бути реалізовані базовий і похідний класи предметної області згідно варіанту;
    - програма має містити коментарі.
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.
4. Дати відповідь на контрольні запитання.

Базовий клас згідно варіанту №16: «Аудіоплеєр»
Похідний клас згідно варіанту №16: «Диктофон »

Хід роботи

Код програми:

Main.py

```
from Dictaphone import Dictaphone

# Основна точка входу програми.
if __name__ == "__main__":
    try:
        # Створюється екземпляр класу Dictaphone.
        dictaphone = Dictaphone()

        # Виклик методу для виведення інформації про готовність диктофона до
роботи.
        dictaphone.device_functionality()

        # Початок запису аудіо. Якщо заряд батареї недостатній, запис не почнеться.
        print("Attempting to start recording...")
        dictaphone.start_recording()

        # Зупинка запису.
        print("Attempting to stop recording...")
        dictaphone.stop_recording()

        # Відтворення записаного аудіо. Якщо заряд батареї недостатній, відтворення
не почнеться.
        print("Attempting to play recorded audio...")
        dictaphone.play_recording()

        # Відтворення треку "Recorded Track". Якщо підключено Bluetooth-пристрій,
відтворення піде через нього,
```

```
        # інакше трек відтвориться через вбудований динамік.
        print('Attempting to play the track "Recorded Track"...')
        dictaphone.play_track("Recorded Track")

        # Виведення стану батареї після виконаних дій
        print(f"Current battery capacity: {dictaphone.get_capacity()}mAh")

    except Exception as e:
        # Обробка будь-яких винятків, що можуть виникнути під час виконання
програми.
        print(f"Error: {e}")
```

Dictaphone.py

```
from AudioPlayer import AudioPlayer

from Speaker import Speaker
from BluetoothModule import BluetoothModule
from Battery import Battery

# Клас Dictaphone розширює функціональність AudioPlayer, додаючи можливість запису
аудіо.
class Dictaphone(AudioPlayer):
    def __init__(self, speaker=None, bluetooth_module=None, battery=None):
        # Викликає конструктор батьківського класу AudioPlayer для ініціалізації
динаміка, Bluetooth-модуля та батареї.
        super().__init__(speaker, bluetooth_module, battery)
        self.is_recording = False  # Змінна для відстеження стану запису (пише або
не пише).

    # Метод для початку запису.
    def start_recording(self):
        # Перевіряє рівень заряду батареї, якщо батарея розряджена, запис не
починається.
        if self.get_capacity() <= 0:
            print("Battery is empty. Cannot start recording.")
            return
        # Якщо запис ще не розпочато, починає запис і зменшує заряд батареї на 20
мА·год.
        if not self.is_recording:
            self.is_recording = True
            print("Recording started...")
            self.set_capacity(self.get_capacity() - 20)
        else:
            # Якщо запис вже триває, виводить повідомлення.
            print("Already recording.")

    # Метод для зупинки запису.
    def stop_recording(self):
        # Якщо диктофон записує, зупиняє запис.
        if self.is_recording:
            self.is_recording = False
```

```
            print("Recording stopped.")
        else:
            # Якщо запис не відбувається, виводить повідомлення.
            print("Not recording.")

    # Метод для відтворення записаного аудіо.
    def play_recording(self):
        # Перевіряє рівень заряду батареї, якщо батарея розряджена, відтворення
неможливе.
        if self.get_capacity() <= 0:
            print("Battery is empty. Cannot play recording.")
            return
        # Відтворює запис і зменшує заряд батареї на 30 мА·год.
        print("Playing recorded audio...")
        self.set_capacity(self.get_capacity() - 30)

    # Перевизначений метод для виведення інформації про готовність диктофона до
роботи.
    def device_functionality(self):
        print("Dictaphone is ready for recording and playback.")

    # Метод для формування рядкового представлення об'єкта Dictaphone.
    def __str__(self):
        # Повертає інформацію про стан запису та рівень заряду батареї.
        return f"Dictaphone{{isRecording={self.is_recording},
battery={self.get_capacity()}mAh}}"
```

AudioPlayer.py

```
from Speaker import Speaker
from BluetoothModule import BluetoothModule
from Battery import Battery

# Клас AudioPlayer представляє плеєр, що може відтворювати аудіо через динамік або
Bluetooth-пристрій,
# підключатися до Bluetooth і управляти рівнем заряду батареї.
class AudioPlayer:
    def __init__(self, speaker=None, bluetooth_module=None, battery=None):
        # Ініціалізація плеєра. Якщо динамік, модуль Bluetooth або батарея не
передані, використовуються значення за замовчуванням.
        self.speaker = speaker if speaker else Speaker(50)  # Динамік з початковою
гучністю 50
        self.bluetooth_module = bluetooth_module if bluetooth_module else
BluetoothModule()  # Bluetooth-модуль
        self.battery = battery if battery else Battery(3000)  # Батарея з ємністю
3000 мА·год

    # Метод для відтворення аудіо-треку.
    def play_track(self, track):
        # Перевірка, чи батарея не розряджена.
        if self.battery.get_capacity() <= 0:
            print("Battery is empty. Please charge the device.")
```

```
            return

        # Відтворення треку через підключений Bluetooth-пристрій або через
вбудований динамік.
        if self.bluetooth_module.is_connected:
            print(f"Playing track: {track} on Bluetooth device:
{self.bluetooth_module.get_connected_device()}")
        else:
            print(f"Playing track: {track} on built-in speaker.")

        # Зменшення рівня заряду батареї на 50 мА·год після відтворення треку.
        self.battery.drain_battery(50)
        print(f"Battery capacity after playing track:
{self.battery.get_capacity()}mAh")

    # Метод для підключення до Bluetooth-пристрою.
    def connect_bluetooth(self, device):
        # Перевірка, чи достатньо заряду для підключення.
        if self.battery.get_capacity() <= 0:
            print("Battery is empty. Please charge the device.")
            return

        # Підключення до вказаного Bluetooth-пристрою.
        self.bluetooth_module.connect_to_device(device)
        print(f"Connected to Bluetooth device: {device}")

        # Зменшення рівня заряду батареї на 30 мА·год після підключення.
        self.battery.drain_battery(30)
        print(f"Battery capacity after connecting to Bluetooth:
{self.battery.get_capacity()}mAh")

    # Метод для відключення Bluetooth-пристрою.
    def disconnect_bluetooth(self):
        self.bluetooth_module.disconnect()
        print("Disconnected from Bluetooth device.")

    # Метод для збільшення гучності.
    def increase_volume(self):
        # Збільшення гучності на 10 одиниць.
        self.speaker.set_volume(self.speaker.get_volume() + 10)
        print(f"Increased volume to: {self.speaker.get_volume()}")

    # Метод для зменшення гучності.
    def decrease_volume(self):
        # Зменшення гучності на 10 одиниць.
        self.speaker.set_volume(self.speaker.get_volume() - 10)
        print(f"Decreased volume to: {self.speaker.get_volume()}")

    # Метод для заряджання батареї на задану кількість мА·год.
    def charge_battery(self, amount):
        # Збільшення ємності батареї на вказане значення.
        self.battery.set_capacity(self.battery.get_capacity() + amount)
```

```
        print(f"Charged battery by {amount}mAh. New capacity:
{self.battery.get_capacity()}mAh")

    # Метод для отримання поточного рівня заряду батареї.
    def get_capacity(self):
        return self.battery.get_capacity()

    # Метод для встановлення нового рівня заряду батареї.
    def set_capacity(self, capacity):
        self.battery.set_capacity(capacity)

    # Метод для виведення інформації про готовність плеєра до роботи.
    def device_functionality(self):
        print("AudioPlayer is ready for playback.")
```

Speaker.py

```
# Клас Speaker представляє динамік із можливістю керування гучністю.
class Speaker:
    def __init__(self, volume):
        # Ініціалізує динамік із початковою гучністю.
        self.volume = volume

    # Метод для отримання поточного рівня гучності.
    def get_volume(self):
        return self.volume

    # Метод для встановлення нового рівня гучності.
    def set_volume(self, volume):
        self.volume = volume

    # Метод для формування рядкового представлення об'єкта Speaker.
    def __str__(self):
        # Повертає інформацію про поточний рівень гучності динаміка.
        return f"Speaker{{volume={self.volume}}}"
```

BluetoothModule.py

```
# Клас BluetoothModule представляє Bluetooth-модуль, який може підключатися та
відключатися від пристроїв.
class BluetoothModule:
    def __init__(self):
        # Ініціалізує модуль зі станом "не підключено" та порожньою назвою
підключеного пристрою.
        self.is_connected = False
        self.connected_device = "None"

    # Метод для підключення до вказаного Bluetooth-пристрою.
    def connect_to_device(self, device):
        self.is_connected = True  # Встановлює статус підключення як True.
        self.connected_device = device  # Зберігає назву підключеного пристрою.
```

```
    # Метод для відключення від Bluetooth-пристрою.
    def disconnect(self):
        self.is_connected = False  # Встановлює статус підключення як False.
        self.connected_device = "None"  # Очищує назву підключеного пристрою.

    # Метод для перевірки стану підключення.
    def is_connected(self):
        return self.is_connected

    # Метод для отримання назви підключеного пристрою.
    def get_connected_device(self):
        return self.connected_device

    # Метод для формування рядкового представлення об'єкта BluetoothModule.
    def __str__(self):
        # Повертає інформацію про стан підключення та назву підключеного пристрою.
        return f"BluetoothModule{{isConnected={self.is_connected},
connectedDevice='{self.connected_device}'}}"
```

Battery.py

```
# Клас Battery представляє батарею з певною ємністю, що дозволяє керувати її
зарядом.
class Battery:
    def __init__(self, capacity):
        # Ініціалізує батарею з заданою ємністю.
        self.capacity = capacity

    # Метод для отримання поточної ємності батареї.
    def get_capacity(self):
        return self.capacity

    # Метод для встановлення нової ємності батареї.
    def set_capacity(self, capacity):
        self.capacity = capacity

    # Метод для розряду батареї на вказану величину.
    def drain_battery(self, amount):
        self.capacity -= amount  # Зменшує ємність батареї на вказану величину.
        if self.capacity < 0:
            self.capacity = 0  # Якщо ємність менша за 0, встановлює її в 0.

    # Метод для формування рядкового представлення об'єкта Battery.
    def __str__(self):
        # Повертає інформацію про поточну ємність батареї в мА·год.
        return f"Battery{{capacity={self.capacity}mAh}}"
```

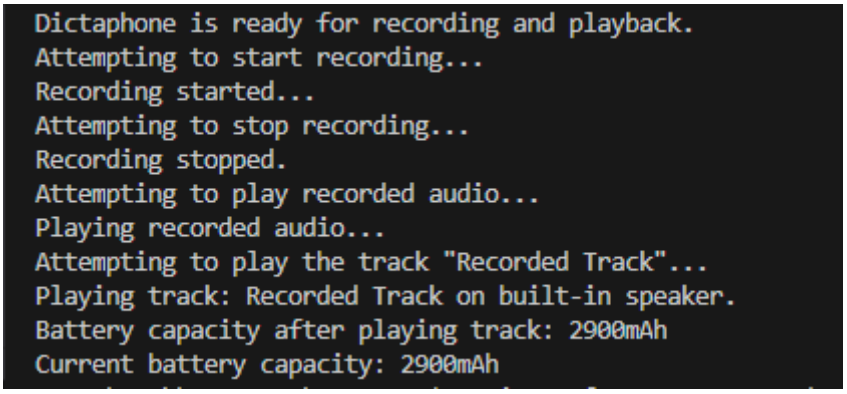


Рис.1 Вивід результату у консоль

Висновок: На лабораторній роботі я оволодів навиками реалізації парадигм об’єктно-орієнтованого програмування використовуючи засоби мови Python.