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 25

Виконав:

ст. групи КІ-306

Тимків Н.В.

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 №25: «Кондиціонер»
Похідний клас згідно варіанту №25: «Пристрій кліматконтролю»
GitHub Repository:

https://github.com/NazarTymkiiv/CPPT_Tymkiv_NV_KI-36_2.git

Хід роботи

Код програми:

Main.py

```
from ClimateControlDevice import ClimateControlDevice
from Conditioner import Conditioner

if __name__ == "__main__":
    """Головна функція, що демонструє використання ClimateControlDevice."""
    try:
        device = ClimateControlDevice()

        device.turn_on()
        device.change_temperature(22)
        device.switch_mode("Cool")
        device.increase_fan_speed()
        device.enable_energy_saving_mode()
        device.auto_adjust_temperature(30)
        device.perform_specific_maintenance()
        device.get_status()
        device.disable_energy_saving_mode()
        device.turn_off()

    except Exception as e:
        raise RuntimeError(f"Сталася помилка: {str(e)}")
```

ClimateControlDevice.py

```
from Conditioner import Conditioner

class ClimateControlDevice(Conditioner):
    """Клас ClimateControlDevice представляє пристрій для кліматконтролю.
    Він розширює функціональність базового кондиціонера та підтримує
    режими енергозбереження та автоматичне регулювання температури."""

    def __init__(self):
        """Конструктор для створення пристрою кліматконтролю."""
        super().__init__()
        self.energy_saving_mode = False

    def enable_energy_saving_mode(self):
        """Вмикає режим енергозбереження."""
        self.energy_saving_mode = True
        print("Режим енергозбереження увімкнено.")

    def disable_energy_saving_mode(self):
        """Вимикає режим енергозбереження."""
        self.energy_saving_mode = False
        print("Режим енергозбереження вимкнено.")

    def is_energy_saving_mode_enabled(self):
        """Перевіряє, чи увімкнено режим енергозбереження.

        Повертає:
            bool: True, якщо режим енергозбереження увімкнено, інакше False
        """
        return self.energy_saving_mode

    def perform_specific_maintenance(self):
        """Виконує специфічне обслуговування пристрою кліматконтролю."""
        print("Виконується специфічне обслуговування пристрою кліматконтролю.")

    def auto_adjust_temperature(self, outdoor_temperature):
        """Автоматично регулює температуру на основі зовнішніх умов.

        Аргументи:
            outdoor_temperature (float): зовнішня температура
        """
        if self.is_on:
            if self.energy_saving_mode:
                target_temperature = outdoor_temperature + 2  # Менша різниця для
економії енергії
            else:
                target_temperature = outdoor_temperature - 2  # Більша різниця для
комфорту
            self.change_temperature(target_temperature)
```

```
            print(f"Автоматично встановлено температуру на {target_temperature}
градусів.")
        else:
            print("Неможливо автоматично регулювати температуру. Пристрій
вимкнено.")
```

Conditioner.py

```
from Compressor import Compressor
from Filter import Filter
from Thermostat import Thermostat

class Conditioner:
    """Клас для моделювання роботи кондиціонера.
    Він включає в себе такі компоненти, як компресор, фільтр, термостат,
    і дозволяє керувати станом кондиціонера."""

    def __init__(self, compressor=None, filter_obj=None, thermostat=None):
        """Конструктор за замовчуванням. Створює новий об'єкт кондиціонера з
початковими налаштуваннями.

        Аргументи:
            compressor (Compressor, необов'язковий): об'єкт компресора
            filter_obj (Filter, необов'язковий): об'єкт фільтра
            thermostat (Thermostat, необов'язковий): об'єкт термостата
        """
        self.compressor = compressor if compressor else Compressor()
        self.filter = filter_obj if filter_obj else Filter()
        self.thermostat = thermostat if thermostat else Thermostat()
        self.is_on = False
        self.mode = "Cool"
        self.fan_speed = 1

    def perform_specific_maintenance(self):
        """Абстрактний метод для виконання специфічного обслуговування.
        Повинен бути реалізований у підкласах."""
        raise NotImplementedError("Цей метод повинен бути реалізований у
підкласах")

    def turn_on(self):
        """Вмикає кондиціонер. Запускає компресор та змінює стан на "увімкнено"."""
        if not self.is_on:
            self.compressor.start()
            self.is_on = True
            print("Кондиціонер увімкнено.")
        else:
            print("Кондиціонер вже увімкнено.")

    def turn_off(self):
        """Вимикає кондиціонер. Зупиняє компресор та змінює стан на "вимкнено"."""
        if self.is_on:
            self.compressor.stop()
            self.is_on = False
```

```
            print("Кондиціонер вимкнено.")
        else:
            print("Кондиціонер вже вимкнено.")

    def change_temperature(self, temperature):
        """Змінює температуру кондиціонера. Змінює налаштування термостата, якщо
кондиціонер увімкнено.

        Аргументи:
            temperature (float): нова температура
        """
        if self.is_on:
            self.thermostat.set_temperature(temperature)
            print(f"Температуру встановлено на {temperature} градусів.")
        else:
            print("Неможливо змінити температуру. Кондиціонер вимкнено.")

    def switch_mode(self, new_mode):
        """Перемикає режим роботи кондиціонера.

        Аргументи:
            new_mode (str): новий режим роботи (наприклад, "Cool", "Heat")
        """
        if self.is_on:
            self.mode = new_mode
            print(f"Режим змінено на {new_mode}.")
        else:
            print("Неможливо змінити режим. Кондиціонер вимкнено.")

    def increase_fan_speed(self):
        """Збільшує швидкість вентилятора на 1, якщо вона ще не досягла
максимуму."""
        if self.is_on:
            if self.fan_speed < 3:
                self.fan_speed += 1
                print(f"Швидкість вентилятора збільшено до {self.fan_speed}.")
            else:
                print("Досягнуто максимальну швидкість вентилятора.")
        else:
            print("Неможливо змінити швидкість вентилятора. Кондиціонер вимкнено.")

    def decrease_fan_speed(self):
        """Зменшує швидкість вентилятора на 1, якщо вона ще не досягла мінімуму."""
        if self.is_on:
            if self.fan_speed > 1:
                self.fan_speed -= 1
                print(f"Швидкість вентилятора зменшено до {self.fan_speed}.")
            else:
                print("Досягнуто мінімальну швидкість вентилятора.")
        else:
            print("Неможливо змінити швидкість вентилятора. Кондиціонер вимкнено.")

    def clean_filter(self):
```

```
        """Очищує фільтр кондиціонера."""
        self.filter.clean_filter()
        print("Фільтр очищено.")

    def check_filter_status(self):
        """Перевіряє стан фільтра (чистий або забруднений)."""
        is_clean = self.filter.is_clean()
        print(f"Стан фільтра: {'чистий' if is_clean else 'забруднений'}")

    def perform_maintenance(self):
        """Виконує технічне обслуговування кондиціонера, включаючи очищення фільтра
та перезапуск компресора."""
        self.clean_filter()
        self.compressor.stop()
        self.compressor.start()
        print("Виконано технічне обслуговування кондиціонера.")

    def get_status(self):
        """Виводить поточний статус кондиціонера, включаючи стан, режим,
температуру та швидкість вентилятора."""
        print(f"Статус кондиціонера: {'увімкнено' if self.is_on else 'вимкнено'}, "
              f"Режим: {self.mode}, Температура:
{self.thermostat.get_temperature()}, "
              f"Швидкість вентилятора: {self.fan_speed}")
```

Compressor.py

```
class Compressor:
    """Клас, що моделює роботу компресора кондиціонера.
    Компресор може бути увімкнений або вимкнений."""

    def __init__(self):
        """Конструктор за замовчуванням. Створює компресор у вимкненому стані."""
        self.running = False

    def start(self):
        """Увімкнення компресора. Змінює стан компресора на "увімкнено"."""
        self.running = True

    def stop(self):
        """Вимкнення компресора. Змінює стан компресора на "вимкнено"."""
        self.running = False

    def is_running(self):
        """Перевіряє, чи працює компресор.

        Повертає:
            bool: True, якщо компресор увімкнено, False, якщо вимкнено.
        """
        return self.running
```

Filter.py

```
class Filter:
    """Клас Filter моделює фільтр кондиціонера.
    Фільтр може бути чистим або забрудненим."""

    def __init__(self, clean=True):
        """Конструктор, який дозволяє встановити стан фільтра (чистий або
забруднений).

        Аргументи:
            clean (bool): стан фільтра: True для чистого фільтра, False для
забрудненого
        """
        self.clean = clean

    def is_clean(self):
        """Перевіряє, чи є фільтр чистим.

        Повертає:
            bool: True, якщо фільтр чистий, False, якщо він забруднений
        """
        return self.clean

    def clean_filter(self):
        """Очищує фільтр, встановлюючи його стан як чистий."""
        self.clean = True

    def soil(self):
        """Забруднює фільтр, встановлюючи його стан як забруднений."""
        self.clean = False
```

Thermostat.py

```
class Thermostat:
    """Клас Thermostat представляє термостат, який зберігає та управляє
температурою."""

    def __init__(self, temperature=0):
        """Створює новий термостат із заданою температурою.

        Аргументи:
            temperature (float): температура термостата (за замовчуванням: 0)
        """
        self.temperature = temperature

    def get_temperature(self):
        """Повертає поточну температуру термостата.

        Повертає:
```

```
            float: температура термостата
        """
        return self.temperature

    def set_temperature(self, temperature):
        """Встановлює нову температуру термостата.

        Аргументи:
            temperature (float): нова температура термостата
        """
        self.temperature = temperature
```

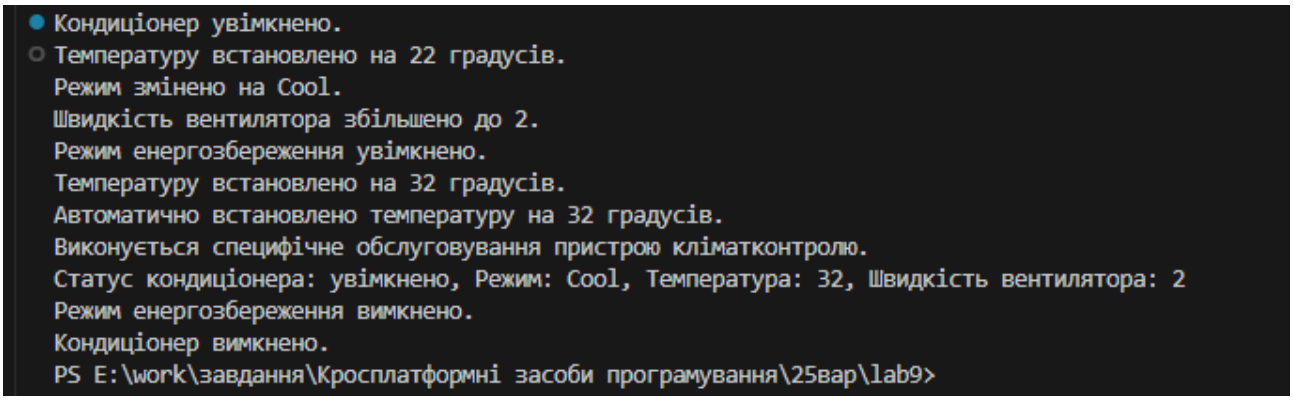

Рис.1 Вивід результату у консоль

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