Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

Звіт

З лабораторної роботи №9

Варіант – 10

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Основи об’єктно-орієнтованого програмування у Python»

Виконав: ст. гр. КІ-306

Миценко О. С.

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

Львів 2023

Мета роботи: 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.

Варіант завдання:

Базовий клас: Будинок

Похідний клас: Офісний центр

Код програми:

House.py

```
from datetime import datetime

class House:
    def __init__(self, address="No information", number_of_floors=0, has_garden=False):
        self.address = address
        self.number_of_floors = number_of_floors
        self.has_garden = has_garden

    def display_details(self):
        print(f"Адреса будинку: {self.address}")
        print(f"Кількість поверхів: {self.number_of_floors}")
        print(f"Наявність саду: {'Так' if self.has_garden else 'Ні'}")

    def set_number_of_floors(self, number_of_floors):
        self.number_of_floors = number_of_floors
       # self.log_message(f"Встановлено кількість поверхів: {number_of_floors}")

    def set_address(self, address):
        self.address = address
      #  self.log_message(f"Оновлено адресу: {address}")

    def set_has_garden(self, has_garden):
        self.has_garden = has_garden
      #  self.log_message("Оновлено інформацію про сад.")

    def add_floor(self):
        self.number_of_floors += 1
       # self.log_message("Додано поверх.")

    def remove_floor(self):
```

```
        if self.number_of_floors > 0:
            self.number_of_floors -= 1
            #self.log_message("Видалено поверх.")

         # self.log_message("Не можна видалити поверх. Кількість поверхів вже
мінімальна.")

    def get_number_of_floors(self):
      #  self.log_message("Дана інформація про кількість поверхів.")
        return self.number_of_floors

    def get_address(self):
        #self.log_message("Дана інформація про адресу.")
        return self.address

    def has_garden_info(self):
       # self.log_message("Дана інформація про наявність саду.")
        return self.has_garden



# Приклад використання класу
if __name__ == "__main__":
    house = House("Some address1", 2, True)

    house.display_details()
    house.set_number_of_floors(3)
    house.set_address("New address")
    house.add_floor()
    house.remove_floor()

    print(f"Кількість поверхів: {house.get_number_of_floors()}")
    print(f"Адреса будинку: {house.get_address()}")
    print(f"Наявність саду: {house.has_garden_info()}")

    del house  # закриття файлу перед знищенням об'єкту
```

office_center.py

```
from house import House
from SomeClass import SomeClass
class OfficeCenter(House, SomeClass):
    def __init__(self, address, number_of_floors, office_space,
                 has_meeting_room, number_of_desks, has_projector, has_whiteboard):
        super().__init__(address, number_of_floors)
        # Ініціалізуємо конструктори від унаслідованих класів

        self.office_space = office_space
        self.has_meeting_room = has_meeting_room
        self.number_of_desks = number_of_desks
        self.has_projector = has_projector
        self.has_whiteboard = has_whiteboard

    def display_class_name(self):
        print(f"New class name: {self.__class__.__name__}")

    def get_office_space(self):
        return self.office_space

    def set_office_space(self, office_space):
        self.office_space = office_space
```

```
    def allocate_office_space(self, square_meters):
        self.office_space += square_meters

    def equip_meeting_room(self, projector, whiteboard):
        self.has_meeting_room = True
        self.has_projector = projector
        self.has_whiteboard = whiteboard

    def add_desks(self, desks_to_add):
        self.number_of_desks += desks_to_add

    def remove_desks(self, desks_to_remove):
        if desks_to_remove > self.number_of_desks:
            print("Cannot remove more desks than available.")
        else:
            self.number_of_desks -= desks_to_remove

    def add_projector(self, has_projector):
        self.has_projector = has_projector
        # Additional logic related to adding a projector

    def add_whiteboard(self, has_whiteboard):
        self.has_whiteboard = has_whiteboard
        # Additional logic related to adding a whiteboard

    def display_details(self):
        print("Адреса офісного центру:", self.address)
        print("Кількість поверхів:", self.number_of_floors)
        print("Офісний простір (в м²):", self.office_space)
        print("Наявність зала для зустрічей:", "Так" if self.has_meeting_room else "Hi")
        print("Кількість робочих місць:", self.number_of_desks)
        print("Наявність проектора:", "Так" if self.has_projector else "Hi")
        print("Наявність дошки:", "Так" if self.has_whiteboard else "Hi")

    def __str__(self):
        return f"OfficeCenter: {self.address}, {self.number_of_floors} floors"

# Приклад використання класу
if __name__ == "__main__":
    office = OfficeCenter("Some address", 3, 500, True, 50, True, True)
    office.display_details()

    office.set_office_space(600)
    office.allocate_office_space(100)
    office.equip_meeting_room(True, False)
    office.add_desks(10)
    office.remove_desks(5)
    office.add_projector(True)
    office.add_whiteboard(False)

    office.display_details()

    office.display_class_name()
```

main.py

```
from house import House
from office_center import OfficeCenter

# Код для тестування класів House і OfficeCenter
if __name__ == "__main__":
    house = House("Lazarenka", 2, True)
```

```
    house.display_details()
    house.set_number_of_floors(3)
    house.set_address("New address")
    house.add_floor()
    house.remove_floor()

    print(f"Кількість поверхів: {house.get_number_of_floors()}")
    print(f"Адреса будинку: {house.get_address()}")
    print(f"Наявність саду: {house.has_garden_info()}")

    del house  # закриття файлу перед знищенням об'єкту

    office = OfficeCenter("Central", 3, 500, True, 50, True, True)
    #office.display_details()

    office.set_office_space(600)
    office.allocate_office_space(100)
    office.equip_meeting_room(True, False)
    office.add_desks(10)
    office.remove_desks(5)
    office.add_projector(True)
    office.add_whiteboard(False)

    #office.display_details()
```

Результати роботи програми:

```
"C:\Users\Oleksandr Mytsenko\Documents\CPPT\CPPTLabs\CPPT_Mytsenko_OS_KI-306_1\Lab9\venv\Scripts\python.exe" "C
Адреса будинку: Lazarenka
Кількість поверхів: 2
Наявність саду: Так
Кількість поверхів: 3
Адреса будинку: New address
Наявність саду: True

Process finished with exit code 0
```

Відповіді на контрольні запитання

1. Що таке модулі?

- Модулі в Python - це файли, які містять Python-код. Вони використовуються для організації коду у логічні групи, і можуть містити функції, класи, змінні та інші об'єкти.

2. Як імпортувати модуль?

- import модуль

3. Як оголосити клас?

- class МійКлас: # Тіло класу

4. Що може міститися у класі?

- атрибути (змінні), методи (функції), конструктори, спеціальні методи (наприклад, __init__, __str__), властивості та інше.

5. Як називається конструктор класу?

- Конструктор класу має ім'я __init__. Він викликається при створенні нового об'єкта класу і використовується для ініціалізації атрибутів об'єкта.
6. Як здійснити спадкування?
- class ПідКлас(БазовийКлас): # Тіло підкласу
7. Які види спадкування існують?
- одиночне спадкування (коли підклас успадковує лише один базовий клас) та множинне спадкування (коли підклас успадковує більше одного базового класу).
8. Які небезпеки є при множинному спадкуванні, як їх уникнути?
- Небезпеки при множинному спадкуванні включають в себе можливі конфлікти імен методів або атрибутів між базовими класами, що може призвести до непередбачуваної поведінки. Для уникнення цих проблем можна використовувати аліаси, викликати методи базових класів безпосередньо або використовувати композицію замість спадкування.
9. Що таке класи-домішки?
- це класи, які містять певний функціонал і можуть бути використані для розширення функціональності інших класів. Вони не призначені для створення об'єктів, але можуть бути включені у інші класи за допомогою спадкування, щоб надати їм певну функціональність.
10.Яка роль функції super() при спадкуванні?
- для виклику методів базового класу з підкласу. Вона допомагає уникнути явного вказівання імен базових класів та робить код більш гнучким при зміні структури спадкування. Наприклад, super().__init__() викликає конструктор базового класу.

Висновок

У ході виконання даної лабораторної роботи, я здобув важливі навички об'єктноорієнтованого програмування мовою Python. Ознайомився з ключовими аспектами цієї парадигми, включаючи створення та використання класів, роботу з об'єктами, та використання спадкування та поліморфізму для покращення ефективності програм.