Лабораторна робота №4

Звіт

З дисципліни “Об'єктно-орієнтоване програмування ”

на тему: “Метакласи та метапрограмування ”.

Студента 3 курсу: Групи МІТ-31 Демиденко Андрій

**Київ - 2024р.**

**Комплексний звіт**

**Мета:** розглянути концепцію метапрограмування у Пайтон. Ознайомитися

з поняттям метакласу у Пайтон, навчитися створювати та використовувати класи

з наслідуванням метакласів.

**Завдання для самостійного опрацювання варіант**

**Варіант №6**

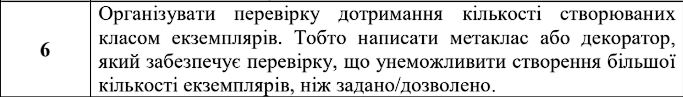


Рис. 4.1 – завдання для варіанту

**Завдання №1**

Метою є створення механізму, що обмежує кількість екземплярів певного класу. Це може бути корисно в ситуаціях, де потрібен контроль над кількістю одночасно існуючих об'єктів, наприклад, у випадках керування підключеннями до бази даних, пулів ресурсів або ліцензійних обмежень.

## Основні компоненти

### 1. Метаклас InstanceLimitMeta (4.1):

Метаклас — це клас для класів, тобто об'єкти, які керують створенням інших класів. У Python метакласи дозволяють змінювати поведінку класів під час їх створення **(\_\_new\_\_)** та при виклику для створення екземпляра **(\_\_call\_\_).** У нашому випадку метаклас реалізує логіку підрахунку створених об'єктів та перевіряє, чи не перевищено встановлений ліміт.

* **\_max\_instances** — класова змінна, що визначає максимальну кількість дозволених об'єктів (у нашому випадку — 3).
* **\_\_new\_\_** — викликається під час створення класу. Тут до словника атрибутів класу додається змінна \_instance\_count, яка зберігатиме поточну кількість створених об'єктів цього класу.
* **\_\_call\_\_** — викликається щоразу, коли створюється новий екземпляр класу. У цьому методі перевіряється, чи не досягнуто ліміту. Якщо ліміт перевищено — викидається виняток, і створення екземпляра блокується.

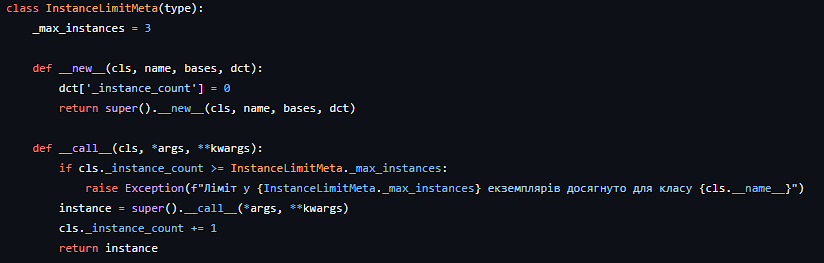


Рис. 4.1 – Метаклас **InstanceLimitMeta.**

### 2. Клас MyClass (4.2):

Цей клас використовує метаклас InstanceLimitMeta за допомогою параметра metaclass. Завдяки цьому під час кожного створення екземпляра буде задіяна логіка перевірки **з \_\_call\_\_** метакласу. У конструкторі **\_\_init\_\_** реалізовано просте виведення повідомлення для демонстрації моменту створення екземпляра.

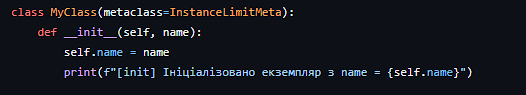
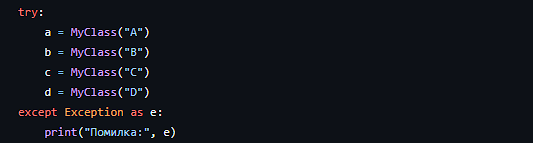


Рис. 4.2 – Клас **MyClass.**

## Демонстрація роботи (4.3):

У блоці **try-except** створюються чотири об'єкти класу **MyClass**. Перші три створюються успішно, але при спробі створити четвертий (**d = MyClass("D")**) викидається виняток, оскільки досягнуто встановленого ліміту. У блоці **except** цей виняток перехоплюється, і виводиться зрозуміле повідомлення.



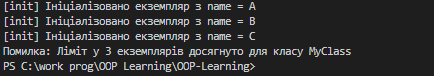


Рис. 4.3 – Демонстрація роботи.

## Особливості реалізації

* **Використання метакласу дозволяє централізовано контролювати поведінку класу без зміни його структури**. Це особливо зручно для створення обмежень, інтерфейсів або автоматичних перевірок.
* **Обмеження встановлюється на рівні класу, а не екземпляра**, що робить цю логіку незалежною від середовища створення об'єктів.
* **Код є гнучким** — змінити ліміт або повторно використати метаклас для іншого класу можна без потреби переписувати логіку.

**Висновок:** Цей приклад демонструє ефективність використання метакласів для реалізації обмежень та модифікації логіки створення об’єктів. На відміну від стандартного підходу, де подібні перевірки реалізуються вручну в конструкторі або через зовнішні менеджери, метаклас дозволяє інкапсулювати контроль на більш високому рівні — рівні класу. Це полегшує повторне використання логіки, покращує читабельність коду та демонструє глибше розуміння об'єктної моделі Python.

[Git Reposytory](https://github.com/AndrewDem0/OOP-Learning)