

ПРОГРАММА

ПРОФЕССИОНАЛЬНОЙ ПЕРЕПОДГОТОВКИ

**“Python-разработка для БПЛА: паттерны проектирования, API-интерфейсы и библиотеки для оптимизации решений”**

**Задание 5.**

**Работа с Singleton, Adapter, Decorator**

Преподаватель курса

**Максимов Егор Васильевич**

Выполнил

**Зубков Евгений Юрьевич**

**№ потока БПЛА\_256-1, группа 1**

**Москва 2024 г.**

**Задание 5. Работа с Singleton, Adapter, Decorator**

**Инструкция:**

Шаг 1. Создать класс MySingleton. Добавить в него статическое поле instance и метод get\_instance для получения единственного экземпляра класса.

Шаг 2. В методе **init** класса MySingleton проверить, существует ли уже экземпляр в instance. Если нет - создать новый и сохранить в instance.

Шаг 3.Вызвать метод get\_instance дважды в коде и убедиться, что он возвращает один и тот же экземпляр класса.

Шаг 4. Создать класс MyClass с методом test. Создать класс MyClassAdapter, в нем вызвать метод test из экземпляра MyClass.

Шаг 5. Создать функцию my\_function. Написать декоратор my\_decorator, который будет выводить имя функции перед ее вызовом.

Шаг 6. Применить декоратор к функции my\_function и убедиться, что при вызове функции выводится ее имя.

**Решение:**

class MySingleton:  
 \_instance = None # Статическое поле для хранения единственного экземпляра класса  
  
 # данный метод вызывается ДО создания экземпляра класса  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 if cls.\_instance is None:  
 cls.\_instance = super(MySingleton, cls).\_\_new\_\_(cls)  
 return cls.\_instance  
  
 # данный метод вызывается ПОСЛЕ создания экземпляра класса (инициализация)  
 def \_\_init\_\_(self):  
 if MySingleton.\_instance is None:  
 MySingleton()  
  
  
 @staticmethod  
 def get\_instance():  
 MySingleton()  
  
  
# Вызов метода get\_instance дважды  
singleton1 = MySingleton.get\_instance()  
singleton2 = MySingleton.get\_instance()  
  
# Проверка, что оба вызова возвращают один и тот же экземпляр  
print(f"Идентификаторы объектов ID: {id(singleton1), id(singleton2)}, Одинаковы: {singleton1 is singleton2}")  
  
  
  
# Создаем класс  
class MyClass:  
 def test(self):  
 print("Метод test из MyClass вызван")  
  
  
# Создаем класс - адаптер (оборачиваем MyClass)  
class MyClassAdapter:  
 def \_\_init\_\_(self, adapter):  
 self.adapter = adapter # Сохраняем экземпляр адаптируемого класса  
  
 def call\_test(self):  
 self.adapter.test() # Вызываем метод test из экземпляра MyClass  
  
  
my\_class = MyClass()  
adapter = MyClassAdapter(my\_class)  
adapter.call\_test()  
  
  
def my\_decorator(func):  
 def first\_func(\*args, \*\*kwargs):  
 print(f"Вызов функции: {func.\_\_name\_\_}")  
 return func(\*args, \*\*kwargs)  
 return first\_func  
  
  
@my\_decorator  
def my\_function():  
 print("Выполнение my\_function")  
  
  
my\_function()