МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

Кафедра

Інформаційних технологій

Факультет

Механіки, енергетики та ІТ

Практична робота №1

На тему: “Твірні шаблони проєктування”

Виконала студентка групи: IТ-31

Розвєзєва Христина Тарасівна

Перевірив: Татомир Андрій

Мета: освоїти роботу з твірними шаблонами проєктування та навчитися їх використовувати.

Твірні шаблони проєктування забезпечують механізми створення об'єктів, що збільшують гнучкість та ефективність програми. Ці шаблони приховують процес створення об'єктів, дозволяючи керувати створенням екземплярів на більш високому рівні. Головною метою є інкапсуляція логіки створення об'єктів, щоб зробити код менш залежним від конкретних класів і підвищити масштабованість та розширюваність програмного забезпечення.

До основних твірних шаблонів належать:

1. Factory Method (Фабричний метод)
2. Abstract Factory (Абстрактна фабрика)
3. Builder (Будівельник)
4. Prototype (Прототип)
5. Singleton (Одинак)

**Singleton** (Одинак)

**Опис:** Шаблон Singleton гарантує, що клас матиме лише один екземпляр, і надає глобальну точку доступу до цього екземпляра. Це корисно в ситуаціях, коли потрібно мати один і той самий об'єкт у всіх частинах програми, наприклад, для керування підключеннями до бази даних, логування або конфігурацій.

**Властивості:**

* Забезпечує наявність лише одного екземпляра класу.
* Контролює доступ до цього екземпляра.
* Уникнення створення додаткових екземплярів (використовується той самий екземпляр протягом усього часу виконання програми).

**Приклад:**

class CarFactorySingleton:

\_instance = None

def \_\_new\_\_(cls):

if cls.\_instance is None:

cls.\_instance = super(CarFactorySingleton, cls).\_\_new\_\_(cls)

return cls.\_instance

def \_\_init\_\_(self):

self.car\_brand = "Tesla" ( Ми задали, що за замовчуванням фабрика вироблятиме автомобілі Tesla)

def set\_car\_brand(self, brand):

self.car\_brand = brand

def get\_car\_brand(self):

return self.car\_brand

(Використовую Singleton)

factory1 = CarFactorySingleton()

factory2 = CarFactorySingleton()

(Перевіряю, що це один і той самий об'єкт)

print(factory1 is factory2) # True, бо обидва посилаються на той самий екземпляр

(Виводжу марку автомобіля, яку виробляє фабрика)

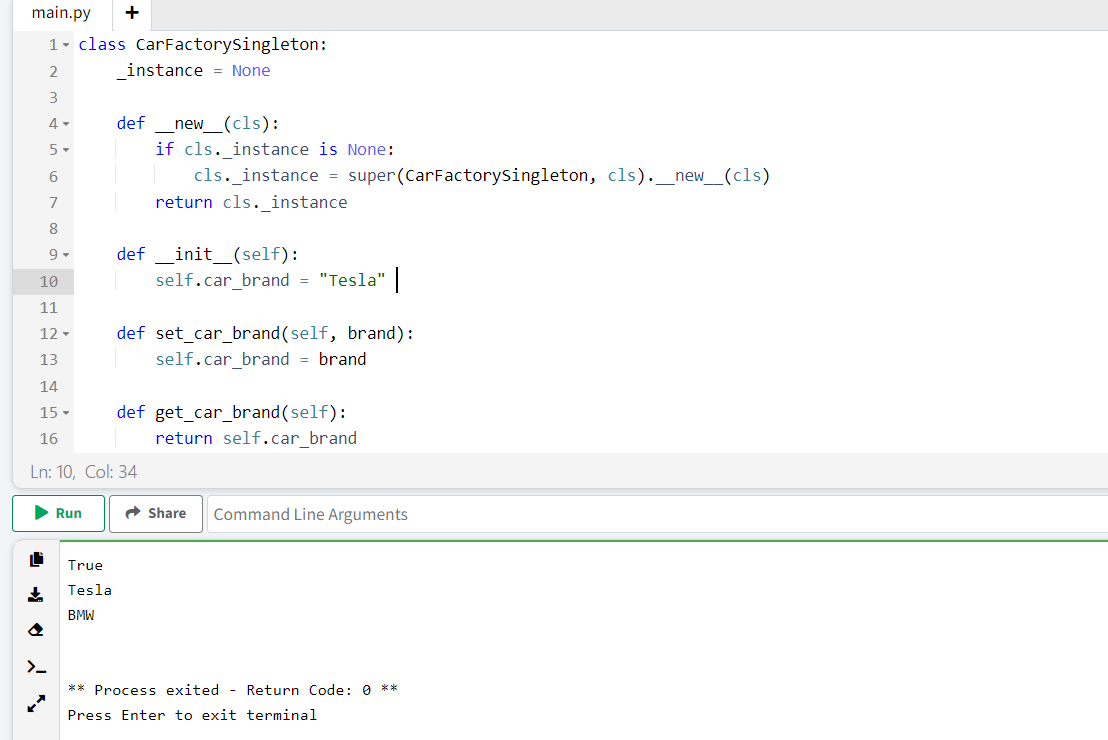
print(factory1.get\_car\_brand()) # Tesla

(Змінюю марку автомобіля через один екземпляр фабрики)

factory1.set\_car\_brand("BMW")

(Перевіряю, що зміни відбилися в обох екземплярах)

print(factory2.get\_car\_brand())



- \_instance: CarFactorySingleton

CarFactorySingleton

+ \_\_new\_\_(): CarFactorySingleton|

+ \_\_init\_\_()

+ set\_car\_brand(brand: str)

+ get\_car\_brand(): str

UML- діаграма

Вище зображено UML-діаграму, яка показує, що клас має приватне поле

\_instance для зберігання єдиного екземпляра. А публічні методи керують створенням екземпляру та роботою з маркою автомобіля.

Містить такі методи:

Містить методи:

* \_\_new\_\_(): перевіряє, чи вже існує екземпляр, і повертає його, якщо є, або створює новий.
* \_\_init\_\_(): ініціалізує об'єкт (встановлює початкову марку автомобіля).
* set\_car\_brand(brand: str): змінює марку автомобіля.
* get\_car\_brand(): повертає поточну марку автомобіля.

Висновок: Шаблон **Singleton** гарантує, що клас має лише один екземпляр, і забезпечує глобальну точку доступу до цього екземпляра. Це корисно для керування ресурсами, які мають бути спільними для всієї програми. В UML-діаграмі Singleton показується як клас із приватним статичним полем для зберігання єдиного екземпляра і методами, які контролюють його створення та доступ.