Наслідування та поліморфізм

Принцип наслідування та його застосування.

Він дозволяє створювати нові класи на основі існуючих (батьківських) класів, спадкуючи їхні атрибути та методи. Це спрощує кодування та сприяє перевикористанню коду. Розглянемо цей принцип на прикладах.

Приклад 1: Батьківський та підкласи

Припустимо, у нас є клас **Vehicle**, який представляє транспортний засіб. У цьому класі є атрибути та методи, які властиві всім транспортним засобам:

```python
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def start_engine(self):
        print("Starting the engine.")
```

Тепер ми можемо створити підкласи, наприклад, **Car** та **Motorcycle**, які успадковують властивості і методи батьківського класу **Vehicle**:

```python
class Car(Vehicle):
    def __init__(self, make, model, year, num_doors):
        super().__init__(make, model, year)
        self.num_doors = num_doors

    def drive(self):
        print("Driving the car.")

class Motorcycle(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)

    def ride(self):
        print("Riding the motorcycle.")
```

За допомогою успадкування ми не повторюємо код для властивостей **make**, **model** та **year**, а знову використовуємо їх у підкласах. Кожен підклас також може мати свої власні атрибути та методи (наприклад, **num_doors** у **Car**).

GENIUS space

Приклад 2: Поліморфізм через успадкування
Поліморфізм - це здатність об'єктів різних класів виконувати однакові методи.
Успадковані методи можуть бути перевизначені у підкласах.

```python
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

class Square(Shape):
    def __init__(self, side_length):
        self.side_length = side_length

    def area(self):
        return self.side_length ** 2

circle = Circle(5)
square = Square(4)

print(circle.area())  # Виведе: 78.5
print(square.area())  # Виведе: 16
```

Обидва класи **Circle** та **Square** успадковують клас **Shape** та перевизначають метод **area()**. Це дозволяє викликати метод **area()** для об'єктів обох класів, і вони повертають правильні значення площі, незважаючи на різний спосіб обчислення площі круга і квадрата.

Поліморфізм:
загальний інтерфейс для різних класів

Поліморфізм - це один із ключових принципів Об'єктно-Орієнтованого Програмування, який дозволяє об'єднати різні класи в єдиний інтерфейс або використовувати об'єкти різних класів через спільний інтерфейс. В інших словах, поліморфізм дозволяє використовувати об'єкти різних класів так, ніби вони є об'єктами одного і того ж базового класу.

```python
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"

# Функція, яка використовує поліморфізм
def animal_sound(animal):
    return animal.speak()

dog = Dog()
cat = Cat()

print(animal_sound(dog))  # Викликаємо метод speak для собаки
print(animal_sound(cat))  # Викликаємо метод speak для кота
```

GENIUS space

Основні ідеї поліморфізму включають:

- **Інтерфейси та абстрактні класи:** Ви можете створити спільний інтерфейс або абстрактний клас, які містять специфікації методів, які повинні бути реалізовані в підкласах. Різні класи можуть реалізовувати ці методи по-різному.
- **Перевизначення методів:** Підкласи можуть перевизначати (override) методи, успадковані від базового класу, для зміни їх поведінки. При цьому, хоча інтерфейс методу залишається тим самим, різні підкласи можуть мати свої власні реалізації.
- **Використання базового класу:** Код може оперувати об'єктами базового класу, не знаючи конкретних деталей підкласу. Це дозволяє створювати загальні функції та методи, які можна використовувати для обробки об'єктів різних класів.