Інкапсуляція та абстракція

GENIUS space

Інкапсуляція: обмеження доступу до атрибутів та методів

Інкапсуляція - це один із принципів Об'єктно-Орієнтованого Програмування (ООП), який передбачає обмеження доступу до атрибутів та методів об'єкта. Головна ідея інкапсуляції полягає в тому, щоб приховати деякі дані та деталі реалізації від зовнішнього світу і надати об'єкту контроль над доступом до цих даних.

Основні концепції інкапсуляції:

Захищеність (Protected): Атрибути та методи, які є захищеними, можуть бути доступні тільки з класів-спадкоємців (підкласів). Вони також зазвичай позначаються префіксом “_”.

Приватність (Private): Атрибути та методи, які є приватними, не можуть бути доступні ззовні класу. Вони зазвичай позначаються префіксом “__".

Публічність (Public): Атрибути та методи, які є публічними, можуть бути доступні з будь-якого місця програми.

Інкапсуляція допомагає забезпечити цілісність об'єкта, а також зменшує вплив змін в реалізації класу на інші частини програми. Вона сприяє абстракції та прихованню деталей реалізації, що спрощує використання класів у більш складних програмах.

У цьому прикладі **_name** і **_age** - це захищені атрибути, і до них можна отримати доступ та змінити їх значення за допомогою публічних методів **get_name()** і **set_name()**. Прямий доступ до захищених атрибутів небажаний, оскільки це порушує інкапсуляцію.

```python
class Student:
    def __init__(self, name, age):
        self._name = name  # Захищений атрибут
        self._age = age    # Захищений атрибут

    # Публічний метод для отримання імені
    def get_name(self):
        return self._name

    # Публічний метод для зміни імені
    def set_name(self, name):
        self._name = name

# Створення об'єкта класу Student
student = Student("Василь", 20)

# Отримання імені за допомогою публічного методу
print(student.get_name())  # Василь

# Зміна імені за допомогою публічного методу
student.set_name("Петро")
print(student.get_name())  # Петро

# Прямий доступ до захищених атрибутів (не рекомендується)
student._name = "Іван"
print(student.get_name())  # Іван
```

Абстракція:
визначення загального інтерфейсу класу

Абстракція - це один із ключових принципів Об'єктно-Орієнтованого Програмування (ООП), який передбачає визначення загального інтерфейсу для класу, приховуючи деталі його реалізації. Абстракція дозволяє спрощувати розробку та розуміння коду шляхом виділення ключових аспектів та операцій, які важливі для користувача класу, і приховуючи складні деталі внутрішньої реалізації.

Основні концепції абстракції:

- **Загальний інтерфейс (**Common Interface**)**: Абстракція визначає загальний інтерфейс, який представляє собою набір методів та властивостей, доступних користувачам класу. Цей інтерфейс визначає, як клас може використовуватися, але не розкриває деталей його реалізації.
- **Приховання деталей (**Hiding Details**)**: Абстракція дозволяє приховати деталі реалізації класу від користувачів. Це означає, що користувачі можуть використовувати клас, не знаючи, як саме він працює всередині.
- **Відокремлення від реалізації (**Separation from Implementation**)**: Абстракція дозволяє виділити інтерфейс класу від його конкретної реалізації. Це дозволяє змінювати реалізацію класу без впливу на користувачів цього класу.

GENIUS space

```python
from abc import ABC, abstractmethod

# Створення абстрактного класу з загальним інтерфейсом
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

# Створення класу-спадкоємця, який реалізує абстрактний клас Shape
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

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

    def perimeter(self):
        return 2 * 3.14 * self.radius
```

```python
# Створення класу-спадкоємця, який реалізує абстрактний клас Shape
class Square(Shape):
    def __init__(self, side_length):
        self.side_length = side_length

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

    def perimeter(self):
        return 4 * self.side_length

# Функція, яка використовує абстрактний клас Shape
def print_shape_info(shape):
    print(f"Area: {shape.area()}")
    print(f"Perimeter: {shape.perimeter()}")

# Створення об'єктів і виклик функції
circle = Circle(5)
square = Square(4)

print("Circle:")
print_shape_info(circle)

print("Square:")
print_shape_info(square)
```

У цьому прикладі клас **Shape** є абстрактним класом, який визначає загальний інтерфейс для всіх геометричних фігур. Класи **Circle** і **Square** спадкуються від **Shape** і реалізують його абстрактні методи **area()** і **perimeter()**. Функція **print_shape_info()** може приймати будь-який об'єкт, який є нащадком **Shape**, і виводити інформацію про нього, не знаючи конкретної реалізації.

Абстрактні класи та методи

Абстрактні класи:
Абстрактний клас - це клас, який не може бути інстанційований безпосередньо,
але він може служити як базовий клас для інших класів. Абстрактні класи мають хоча б один **абстрактний метод**, тобто метод без конкретної реалізації. Вони визначають інтерфейс для класів-підкласів.

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Гав-гав!"

class Cat(Animal):
    def speak(self):
        return "Мяу-мяу!"

# Не можна створити екземпляр абстрактного класу Animal:
# animal = Animal()  # Видасть помилку
```

У цьому прикладі **Animal** - це абстрактний клас, який має абстрактний метод **speak()**.
Класи **Dog** і **Cat** є його підкласами і обов'язково реалізують метод **speak**.

GENIUS space

Абстрактні методи:

Абстрактний метод - це метод, який визначений в абстрактному класі, але не має конкретної реалізації в цьому класі. Він обов'язково має бути реалізований у всіх підкласах абстрактного класу.

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

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

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

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width
```

У цьому прикладі **Shape** - це абстрактний клас з абстрактним методом **area()**. Класи **Circle** і **Rectangle** є підкласами і реалізують цей метод згідно своїх потреб.

Абстрактні класи та методи допомагають створювати більш структурований і розширюваний код, оскільки вони вимагають від підкласів дотримуватися певного інтерфейсу та надають загальний шаблон для створення об'єктів.