ООП – Інкапсуляція та поліморфізм

№ уроку: 3 **Курс:** Python Essential

Засоби навчання: PyCharm

Огляд, ціль та призначення уроку

Після завершення уроку учні матимуть уявлення про такі принципи парадигми об'єктно-орієнтованого програмування, як інкапсуляція та поліморфізм у Python та практично їх використовувати.

Вивчивши матеріал даного заняття, учень зможе:

- Мати уявлення про інкапсуляцію
- Види модифікаторів доступу, яким чином можна з ними працювати
- Розуміти, що таке поліморфізм
- Використовувати дані принципи практично

Зміст уроку

1. Що таке інкапсуляція?
2. Модифікатори доступу
3. Різниця між модифікаторами доступу
4. Реалізація інкапсуляції в Python
5. Сеттери та геттери
6. Що таке поліморфізм?
7. Реалізація поліморфізму у Python

Резюме

Інкапсуляція — це механізм мови, який дозволяє об'єднати всі методи та атрибути всередині одного класу. Ми використовували цей механізм постійно, коли всередині класу створювали багато методів та атрибутів.
Ще одна властивість інкапсуляції: можливість обмеження доступу до методів та змінних. Інкапсуляція робить деякі методи або атрибути доступними тільки всередині об'єкта, але **не доступними поза** об'єктом.
Інкапсуляція в Python працює на рівні угоди між програмістами про те, які атрибути загальнодоступні, а які — внутрішні.
Підкреслення на початку імені атрибуту говорить про те, що змінна або метод не призначений для використання поза методами класу, однак **атрибут доступний за цим іменем**.

```
class Person:
    def _get_secret(self):
        print("It is my big secret!")
>>> me = Person()
>>> me._get_secret()
It is my big secret!
```

Нижнє підкреслення слугує для інших програмістів індикатором: "Цей метод/змінну краще не використовувати - вона потрібна для внутрішнього механізму". При цьому Python ніяк не контролює та не забороняє використовувати.
Якщо перед методом/атрибутом поставити 2 підкреслення, Python відреагує на таке найменування і захистить змінну від зовнішнього використання:

```
class Person:
    def _get_secret(self):
        print("It is my big secret!")

    def __get_the_biggest_secret(self):
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
        print("Quieter! It is my biggets secret!")
>>> me = Person()
>>> me._get_secret()
It is my big secret!
>>> me._get_the_biggest_secret()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Person' object has no attribute '__get_the_biggest_secret'
```

Це захистить ваш код від невмілого програміста. Насправді, Python маскує "приховану" змінну за шаблоном: "**_ім'яКласуІм'яМетоду**". За цим шаблоном Python перейменував наш метод get_the_biggest_secret на _Person get_the_biggest_secret:

```
class Person:
    def _get_secret(self):
        print("It is my big secret!")

    def __get_the_biggest_secret(self):
        print("Quieter! It is my biggets secret!")
>>> me = Person()
>>> me._Person__get_the_biggest_secret()
Quieter! It is my biggets secret!
```

Здивуйте колег на співбесіді цим знанням, але **не використовуйте його у роботі**. Це призводить до великих плутанин.

Сеттер (*установник*) Python – це методи, які використовуються для встановлення значень властивостей. В ООП дуже корисно встановлювати значення приватних атрибутів у класі. Як правило, гетери та сеттери в основному використовуються для забезпечення інкапсуляції даних в ООП.

Геттери (*одержувачі*) у Python – це методи, які використовуються в ООП для доступу до приватних атрибутів класу. Функція setattr() у Python узгоджується з функцією getattr() у Python. Вона змінює значення атрибутів об'єкта.

Що таке поліморфізм?

Термін поліморфізм прийшов у програмування з біології. Поліморфізм у біології — здатність організмів існувати у станах із різною внутрішньою структурою чи різних зовнішніх формах під час свого життєвого циклу. Звучить складно, давайте розбиратися на більш життєвому прикладі. Парадигма ООП вперше була реалізована у мові SmallTalk. Автор мови був біологом і принципи ООП були сформульовані виходячи з його спостережень за живими організмами. Тому всі принципи ООП прийшли в програмування з біології.
Ви маєте смартфон. У смартфона є інтерфейс за допомогою якого ви з ним взаємодієте та даєте йому команди. Мовою Python це виглядало б так:

```
my_phone.call_to("David")
my_phone.set_alarm_for("6:00")
my_phone.open_sms_from("Kate")
my_phone.lock_screen()
```

Одного дня ваш смартфон зламався. Ви купуєте новий від тієї ж компанії, але іншої моделі. Тепер вам необхідно знову вчитися викликати всі ці методи? Швидше за все — ні, тому що інтерфейс смартфона не змінився: у нього також є спосіб зателефонувати, поставити будильник та інше. При цьому його внутрішні характеристики та алгоритми могли змінитися або міг бути доданий новий функціонал.
З цього робимо висновок, що хоч і class (**модель**) смартфона змінилися, його методи (**інтерфейс**) залишилися такими ж, і ви зберегли здатність працювати з ним. Це і є поліморфізм: об'єкт змінився, а інтерфейс залишився той самий.

Реализація в Python

```
class BasePhone:
    def __init__(self, owner_name):
        self.owner_name = owner_name

    def call_to(self, contact_name):
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
        pass

    def set_alarm_for(self, time):
        pass

    def open_sms_from(self, contact_name):
        pass

    def lock_screen(self):
        pass

class Nokia(BasePhone):
    DATA = {
        "contacts": {
            "David": {
                "phone": "+380991231231",
                "SMS": [
                    "Hello, guy!"
                ]
            },
            "Kate": {
                "phone": "+380991765231",
                "SMS": [
                    "Can we meet next morning?"
                ]
            }
        },
        "alarms": [],
        "status": "unlocked",
    }

    def call_to(self, contact_name):
        if not self.DATA['status'] == "lock":
            contact_phone = self.DATA['contacts'][contact_name]["phone"]
            print(f"Calling to {contact_name} with phone number {contact_phone}")

    def set_alarm_for(self, time):
        if not self.DATA['status'] == "lock":
            self.DATA["alarms"].append(time)
            print(f"Alarms was set for {time}. All alarms: {self.DATA['alarms']}")

    def  open_sms_from(self,  contact_name):
        if not self.DATA['status'] == "lock":
            SMS = self.DATA['contacts'][contact_name]["SMS"]
            print(f"All SMS from {contact_name}: {SMS}")

    def lock_screen(self):
        self.DATA["status"] = "lock"

class Samsung(BasePhone):
    DATA = {
        "contacts": {
            "David": {
                "phone": "+380991231231",
                "SMS": [
                    "Hello, guy!"
                ]
            },
            "Kate": {
                "phone": "+380991765231",
                "SMS": [
                    "Can we meet next morning?"
                ]
            }
        },
        "alarms": [],
        "status": "unlocked",
    }

    def _get_contact(self, contact_name):
        return self.DATA["contacts"][contact_name]
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
    def _get_contact_sms(self, contact_name):
        contact = self._get_contact(contact_name)
        return contact["SMS"]

    def _get_contact_phone(self, contact_name):
        contact = self._get_contact(contact_name)
        return contact["phone"]

    def _get_alarms(self):
        return self.DATA["alarms"]

    def _set_status(self, lock):
        self.DATA["status"] = "lock"

    def _set_alarm_for(self, time):
        self.DATA["alarms"].append(time)

    def _is_phone_unlock(self):
        phone_status = self.DATA["status"]

        if phone_status == "unlocked":
            return True
        else:
            print("WARNING: You can't get access to locked phone!")
            return False

    def lock_screen(self):
        self._set_status("lock")

    def call_to(self, contact_name):
        if self._is_phone_unlock():
            phone_number  =  self._get_contact_phone(contact_name)
            print(f"Calling to {contact_name} with phone number {phone_number}")

    def set_alarm_for(self, time):
        if self._is_phone_unlock():
            self._set_alarm_for(time)
            all_alarms = self._get_alarms()
            print(f"Alarms was set for {time}. All alarms: {all_alarms}")

    def open_sms_from(self, contact_name):
        if self._is_phone_unlock():
            contact_sms = self._get_contact_sms(contact_name)
            print(f"All SMS from {contact_name}: {contact_sms}")

def test_my_phone(phone):
    phone.call_to("David")
    phone.set_alarm_for("6:00")
    phone.open_sms_from("Kate")
    phone.lock_screen()
    phone.open_sms_from("Kate")


nokia_phone = Nokia(owner_name="Oleg")     # same to Nokia("Oleg")
samsung_phone = Samsung(owner_name="Oleg") # same to Samsung("Oleg")

print(" Test Nokia phone ".center(50, "="))
test_my_phone(nokia_phone)

print()

print(" Test Samsung phone ".center(50, "="))
test_my_phone(samsung_phone)
```

Результат виконання коду:

```
================ Test Nokia phone ================
Calling to David with phone number +380991231231
Alarms was set for 6:00. All alarms: ['6:00']
All SMS from Kate: ['Can we meet next morning?']
```


CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
=============== Test Samsung phone ===============
Calling to David with phone number +380991231231
Alarms was set for 6:00. All alarms: ['6:00']
All SMS from Kate: ['Can we meet next morning?']
WARNING: You can't get access to locked phone!
```

Функція test_my_phone не знає, який смартфон їй передають. Вона лише знає методи, які має підтримувати смартфон. А ми створили два смартфони з однаковим інтерфейсом, але різними внутрішніми алгоритмами. Це і є **поліморфізм**.
Ви могли побачити клас BasePhone. Він створений, щоб визначити всі обов'язкові методи/атрибути, які повинні підтримувати всі інші смартфони.
Вивчення чужого коду – гарний спосіб зрозуміти механізми роботи мови. Не соромтеся ставити запитання тренеру щодо коду вище.

Закріплення матеріалу

- Що таке інкапсуляція?
- Які модифікатори доступу до атрибутів існують?
- Яка різниця між модифікаторами доступу?
- Що вказує на те, що атрибут є внутрішнім?
- Чи можемо ми використовувати захищений атрибут в середині модуля?
- Яким чином можна дістатись до приватного атрибуту?
- Що таке поліморфізм?
- У яких ситуаціях застосовується поліморфізм?
- Що таке «Качина типізація»?

Додаткові завдання

Завдання 1
Опишіть два класи Base та його спадкоємця Child з методами method(), який виводить на консоль фрази відповідно "Hello from Base" та "Hello from Child".

Завдання 2
Ознайомившись з кодом файлу example_7.py, створіть додаткові класи-нащадки Cone та Paraboloid від класу Shape. Перевизначте їх методи. Створіть екземпляри відповідних класів за скористайтеся всіма методами. В результаті повинно з’явитися зображення. Перегляньте їх.

Самостійна діяльність учня

Завдання 1
Створіть клас, який описує автомобіль. Які атрибути та методи мають бути повністю інкапсульовані? Доступ до таких атрибутів та зміну даних реалізуйте через спеціальні методи (get, set).

Завдання 2
Створіть 2 класи мови, наприклад, англійська та іспанська. В обох класів має бути метод greeting(). Обидва створюють різні привітання. Створіть два відповідні об'єкти з двох класів вище та викличте дії цих двох об'єктів в одній функції (функція **hello_friend**).

Завдання 3
Використовуючи посилання наприкінці цього уроку, ознайомтеся з таким засобом інкапсуляції, як властивості. Ознайомтеся з декоратором property у Python. Створіть клас, що описує температуру і дозволяє задавати та отримувати температуру за шкалою Цельсія та Фаренгейта, причому дані можуть бути задані в одній шкалі, а отримані в іншій.

Рекомендовані ресурси

Статті у Вікіпедії про ключові поняття, розглянуті на цьому уроці

CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

https://uk.wikipedia.org/wiki/Об'єктно-орієнтоване_програмування
https://uk.wikipedia.org/wiki/Інкапсуляція_(програмування)
https://uk.wikipedia.org/wiki/Поліморфізм_(програмування)
https://uk.wikipedia.org/wiki/Властивість_(програмування)
http://www.programiz.com/python-programming/property
https://docs.python.org/3/library/functions.html#property

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com