ООП – Инкапсуляция и полиморфизм

№ урока: 3 **Курс:** Python Essential

Средства обучения: PyCharm

Обзор, цель и назначение урока

После завершения урока обучающиеся будут иметь представление о таких парадигмах объектно-ориентированного программирования, как инкапсуляция и полиморфизм в Python.

Изучив материал данного занятия, учащийся сможет:

- Иметь представление об инкапсуляции
- Понимать, что такое полиморфизм
-

Содержание урока

1. Что такое инкапсуляция?
2. Реализация инкапсуляции в Python
3. Что такое полиморфизм?
4. Пример в Python

Резюме

Инкапсуляция — это механизм языка, который позволяет объединить все методы и атрибуты внутри одного класса. Мы использовали этот механизм постоянно, когда внутри класса создавали много методов и атрибутов.
Еще одно свойство инкапсуляции: возможность ограничения доступа к методам и переменным. Инкапсуляция делает некоторые методы или атрибуты доступными только внутри самого объекта, но **не доступными вне** объекта.
Инкапсуляция в Python работает на уровне соглашения между программистами о том, какие атрибуты являются общедоступными, а какие — внутренними.
Подчеркивание в начале имени атрибута говорит о том, что переменная или метод не предназначен для использования вне методов класса, однако **атрибут доступен по этому имени.**

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

Нижнее подчеркивание служит для других программистов индикатором: "Этот метод\переменную лучше не использовать — она нужна для внутреннего механизма". При этом Python никак не контролирует и не запрещает использовать.
Если перед методом\атрибутов поставить 2 подчеркивания, Python отреагирует на такое наименование и защитит переменную от внешнего использования:

```
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._get_secret()
```


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

```
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'
```

Это защитит ваш код от неумелого программиста. На самом деле, Python маскирует "скрытую" переменную за шаблоном: "**_имяКлассаИмяМетода**". За этим шаблоном, Python переименовал 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!
```

Удивите коллег на собеседовании этим знанием, но **не используйте его в работе**. Это приводит к большим путаницам.

Что такое полиморфизм?

Термин полиморфизм пришел в программирование с биологии. Полиморфизм в биологии — способность организмов существовать в состояниях с различной внутренней структурой или в разных внешних формах во время своего жизненного цикла. Звучит сложно, давайте разбираться на более жизненном примере.

Парадигма ООП впервые была реализована в языке SmallTalk. Создатель языка был биологом и принципы ООП были сформулированы исходя из его наблюдений за живыми организмами. Поэтому все принципы ООП пришли в программирование из биологии.

У вас есть смартфон. У смартфона есть интерфейс с помощью которого вы с ним взаимодействуете и даете ему команды. На языке Python это выглядело бы так:

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

В один день ваш смартфон сломался. Вы покупаете новый от той же компании, но другой модели. Теперь вам необходимо заново учиться вызывать все эти методы? Скорее всего нет, так как интерфейс смартфона не поменялся: у него так же есть метод позвонить, поставить будильник и прочее. При этом его внутренние характеристики и алгоритмы могли измениться или добавиться новый функционал.

Из этого делаем вывод, что хоть и class (**модель**) смартфона поменялись, его методы (**интерфейс**) остались такими же и вы сохранили способность работать с ним. Это и есть полиморфизм: объект поменялся, а интерфейс остался тот же.

Реализация в Python

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

    def call_to(self, contact_name):
        pass

    def set_alarm_for(self, time):
        pass

    def open_sms_from(self, contact_name):
        pass

    def lock_screen(self):
        pass

class Nokia(BasePhone):
```


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

```
    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]

    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):
```


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

```
        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 выполнения кода:

```
================ 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?']

=============== 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!
```


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

Функция test_my_phone не знает, какой смартфон ей передают. Она лишь знает методы, которые должен поддерживать смартфон. А мы создали два смартфона с одинаковым интерфейсом, но разными внутренними алгоритмами. Это и есть **полиморфизм**.
Вы могли заметить класс BasePhone. Он создан, чтобы определить все обязательные методы\атрибуты, которые должны поддерживать все остальные смартфоны.
Изучение чужого кода - хороший способ понять механизмы работы языка. Не стесняйтесь задавать вопросы тренеру по коду выше.

Закрепление материала

- Что такое инкапсуляция?
- Что такое полиморфизм?
- Что указывает на то, что атрибут является внутренним?
- В каких ситуациях применяется полиморфизм?
- Что такое «Утиная типизация»?

Дополнительное задание

Задание
Опишите два класса Base и его наследника Child с методами method(), который выводит на консоль фразы соответственно "Hello from Base" и "Hello from Child"

Самостоятельная деятельность учащегося

Задание 1
Создайте класс, описывающий автомобиль. Какие атрибуты и методы должны быть полностью инкапсулированы? Доступ к таким атрибутам и изменение данных реализуйте через специальные методы (get, set).

Задание 2
Создайте 2 класса – языка, например, английский и испанский. У обоих классов должен быть метод greeting(). Оба создают разные приветствия. Создайте два соответствующих объекта из двух классов выше и вызовите действия этих двух объектов в одной функции (функция **hello_friend**).

Задание 3
Используя ссылки в конце данного урока, ознакомьтесь с таким средством инкапсуляции как свойства. Ознакомьтесь с декоратором property в Python. Создайте класс, описывающий температуру и позволяющий задавать и получать температуру по шкале Цельсия и Фаренгейта, причём данные могут быть заданы в одной шкале, а получены в другой.

Рекомендуемые ресурсы

Статьи в Википедии о ключевых понятиях, рассмотренных на этом уроке
https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование
https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование_на_Python
https://ru.wikipedia.org/wiki/Инкапсуляция_(программирование)
https://ru.wikipedia.org/wiki/Полиморфизм_(информатика)
https://ru.wikipedia.org/wiki/Свойство_(программирование)
http://www.programiz.com/python-programming/property
https://docs.python.org/3/library/functions.html#property

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