Lập trình hướng đối tượng

Th.S. Nguyễn Thành An
Trường ĐH Khoa học Tự nhiên,
Đại học Quốc gia - TP. HCM
ntan@selab.hcmus.edu.vn

Nội dung

- Lớp đối tượng
- Thuộc tính và phương thức
- Phương thức khởi tạo
- Getter và Setter
- Thuộc tính và phương thức tĩnh
- Overriding
- Nạp chồng hàm (overloading)
- Kế thừa
- Đa hình

Lớp đối tượng

- Các “thực thể” hay “đối tượng” trong cuộc sống được biểu diễn, lưu trữ và sử dụng trong chương trình máy tính
  - VD: học sinh, tài khoản, hoá đơn, nút ấn, thanh cuộn, …
- Mỗi “đối tượng” bao gồm hai thành phần cơ bản: thông tin và hành vi
  - VD:
    - Học sinh: tên, lớp, ĐTB, … - đọc, ghi, tính ĐTB, …
    - Nút ấn: vị trí, màu sắc, trạng thái, … - kích hoạt,...
    - Tài khoản: username, password, … - chứng thực, ...

Lớp đối tượng

- Các “đối tượng” này được biểu diễn trong chương trình máy tính là các class
  - Thông tin → thuộc tính (attribute)
  - Hành vi → phương thức (method)
- Hiện tại, lập trình hướng đối tượng (OOP) đang là xu thế của cả thế giới.
  - Lập trình hướng thủ tục (Procedure) →
  - Lập trình hướng đối tượng (OOP) →
  - Lập trình hàm (Functional Programming) (maybe) → …
- Rất nhiều ngôn ngữ hỗ trợ OOP: **Python**, C++, Java, C#, ...

Lớp đối tượng

```
class Student:
```

tên class

```
def __init__(self, name, gpa):
    self.name, self.gpa = name, gpa

def greet(self):
    print('Hi, I am', self.name)
```

thuộc tính

phương thức

```
std = Student('An', 5.)    # khởi tạo instance
std.greet()                # thực hiện method
```

Lớp đối tượng

- Biến được tạo ra từ một class gọi là instance.
- Vòng đời của instance
  - Khởi tạo
  - Hoạt động
    - Thay đổi thông tin
    - Thực hiện hành vi
  - Huỷ: giải phóng bộ nhớ
- Python: lập trình viên thường kiểm soát: Khởi tạo và Hoạt động

Thuộc tính và phương thức

- Thuộc tính: lưu trữ thông tin, dữ liệu → biến/hằng
  - Kiểu dữ liệu cơ bản
  - Cấu trúc dữ liệu
  - Instance của class khác
  - Có thể khai báo trong constructor
- Phương thức: hàm
  - có hoặc không return giá trị
  - method thông thường bắt buộc có argv[0] = **self**
  - có thể áp dụng kỹ thuật tham số với giá trị mặc định
- Truy cập thuộc tính và phương thức thông qua self (giống this của C++/Java...)

Lớp đối tượng

```
class Student:
```

tên class

```
def __init__(self, name, gpa):
    self.name, self.gpa = name, gpa

def greet(self):
    print('Hi, I am', self.name)
```

thuộc tính

phương thức

```
std = Student('An', 5.)    # khởi tạo instance
std.greet()                # thực hiện method
```

Thuộc tính và phương thức

- Thuộc tính và phương thức có 3 tầm vực truy xuất (accessibility):
  - public name, greet()
  - protected _name, _greet()
  - private __name, __greet()
- Đây là quy ước, còn thức tế vẫn có thể “cưỡng chế” truy cập

```
  def __greet(self):
    print('Hi, I am', self._name)

std = Student('An', 5.)   # khởi tạo instance
std._Student__greet()     # thực hiện method
```

```
Hi, I am An
```

Thuộc tính và phương thức

- Một số phương thức mặc định phải “override” để tuỳ chỉnh
  - __init__ → constructor
  - __str__ → ép kiểu thành str
  - __ge__ → so sánh với >=
  - __le__ → so sáng với <=
  - ....

```
'__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__'
```

Thuộc tính và phương thức

```
class Student:
 def __init__(self, name, gpa):
   self._name = name
   self._gpa = gpa

 def __le__(self, std):
   return self._gpa <= std._gpa

a = Student('An', 5.)
b = Student('Binh', 6.)
c = a <= b
print(c)
```

Phương thức khởi tạo

- Constructor: __init__
- Có thể khai báo các thuộc tính trong constructor
- Có thể có hoặc không tham số (ngoài self)
- Có thể dùng tham số với giá trị mặc định

```
def __init__(self, name, gpa=0):
  self._name = name
  self._gpa = gpa
```

- Được gọi thực thi khi khởi tạo instance của class

```
a = Student('An', 5.)
b = Student('Binh', 6.)
```

Getter và Setter

- Getter là nhóm các phương thức cho phép lấy giá trị của thuộc tính
- Setter là nhóm các phương thức cập nhật giá trị cho thuộc tính
- Cặp nhóm này giúp kiểm soát truy xuất thuộc tính và đảm bảo nguyên tắc encapsulation của OOP.

```
def get_name(self):
  return self._name

def get_gpa(self):
  return self._gpa
```

```
def set_name(self, name):
  if len(name) < 1 or len(name) > 30:
    raise 'Invalid length'
  self._name = name

def set_gpa(self, gpa):
  if gpa < 0 or gpa > 10:
    raise 'Invalid gpa'
  self._gpa = gpa
```

Thuộc tính và Phương thức tĩnh

- Static attribute và static method:
  - truy cập trực tiếp thông qua tên class, không cần khởi tạo instance
  - được duy trì từ lúc nạp class

```
class Student:
  count = 0

  def __init__(self, name, gpa=0):
    self._name = name
    self._gpa = gpa
    Student.count += 1

print(Student.count)
a = Student('A')
print(Student.count)
b = Student('B')
print(Student.count)
```

```
class Student:

  def __init__(self, name, gpa=0):
    self._name = name
    self._gpa = gpa

  @staticmethod
  def print_classname():
    print('Student')

Student.print_classname()
```

không có self

Nạp chồng hàm

- Overloading:
  - một class có nhiều phương thức cùng tên
  - phân biệt nhau dựa trên function signature (tên hàm, danh sách tham số)
  - tạo ra sự linh hoạt và tiện dụng

Python không hỗ trợ nạp chồng hàm như các ngôn ngữ khác

- Sử dụng hàm với tham số mặc định để xử lý các trường hợp này.

Kế thừa

- Một class A kế thừa class B
  - A → Child class
  - B → Super class / Base Class
- A “thừa kế” thuộc tính và phương thức của B

```
class Person():
    def __init__(self, name, age):
        self._name = name
        self._age = age

    def greet(self):
        print('hello')
```

```
class Student(Person):
    def __init__(self, name, age, address):
        super().__init__(name, age)
        self._address = address
```

```
a = Student('An', 26, 'HCMC')
print(a._name, a._age, a._address)
a.greet()
```

```
➜  Downloads python3 demo.py
An 26 HCMC
hello
```

Kế thừa

- Truy cập đến super class thông qua super()
- Có thể thêm thuộc tính/phương thức cho child class

```
class Student(Person):
    def __init__(self, name, age, address):
        super().__init__(name, age)
        self._address = address

    def salut(self):
        super().greet()
        print('salut')
```

```
a = Student('An', 26, 'HCMC')
print(a._name, a._age, a._address)
a.greet()
```

```
➜  Downloads python3 demo.py
An 26 HCMC
hello
salut
```

Overriding

- Class A kế thừa class B, trong B đã có phương thức f
- Class A muốn thay đổi f thành phương thức mới theo yêu cầu thì cài đặt lại phương thức f như một phương thức thông thường.

```
class B():
    def doSth(self):
        print('hahaha')
```

```
class A(B):
    def doSth(self, speech):
        print(speech)
```

```
a = A()
a.doSth('salut')
```

Đa kế thừa

- Một class kế thừa từ nhiều class khác
- Thừa hưởng thuộc tính và phương thức
- Cẩn thận vấn đề hình thoi trong đa kế thừa

```
class ClassA():
    def __init__(self, height):
        self._height = height

    def doSthA(self, factor):
        print(self._height * factor)
```

```
class ClassB():
    def __init__(self, width):
        self._width = width

    def doSthB(self, factor):
        print(self._width / factor)
```

```
class ClassC(ClassA, ClassB):
    def __init__(self, height, width, size):
        ClassA.__init__(self, height)
        ClassB.__init__(self, width)
        self._size = size

    def doSthC(self):
        print(self._height * self._width)
```

```
c = ClassC(3, 4, 5)
print(c._height, c._width, c._size)
c.doSthA(1)
c.doSthB(1)
c.doSthC()
```

```
➜  Downloads python3 demo.py
3 4 5
3
4.0
12
```

Interface

- Tạo ra các class chỉ có phương thức “rỗng”
- Kế thừa các class “interface”

```python
class ButtonInterface():
    def click(self, num):
        pass
```

```python
class ScreenInterface():
    def touch(self):
        pass
```

```python
class ComponentA(ButtonInterface, ScreenInterface):
    def __init__(self):
        self._name = 'ABC'
        pass

    def click(self, num):
        print('click %d times'%(num))

    def touch(self):
        print('touched')
```

```python
c = ComponentA()
c.click(3)
c.touch()
print(isinstance(c, ButtonInterface))
print(isinstance(c, ScreenInterface))
```

```
→  Downloads python3 demo.py
click 3 times
touched
True
True
```

Đa hình

- Class A kế thừa lớp B, x là instance của A nhưng “ép kiểu” thành B.
- A override phương thức f của B.
- Hỏi x.f() sẽ gọi thực hiện A.f() hay B.f()

```
class A():
    def doSth(self):
        print('hahaha')

class B(A):
    def __init__(self):
        self._name = 'B'

    def doSth(self):
        print('hihihi')
```

```
x = B()
x.__class__ = A
x.doSth()
print(x._name)
print(dir(x))
```

```
→  Downloads python3 demo.py
hahaha
B
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__'
, '__eq__', '__format__', '__ge__', '__getattribute__', '__gt
__', '__hash__', '__init__', '__init_subclass__', '__le__', '
__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__
reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str
__', '__subclasshook__', '__weakref__', '_name', 'doSth']
```

Bài tập 1

- Tạo class Vehicle với thuộc tính max_speed và distance
- Class Vehicle có phương thức run, nhận vào khoảng cách và thời gian chạy. Nếu vận tốc cần chạy vượt quá max_speed thì dừng lại ở vị trí xa nhất có thể theo max_speed.
- Tạo class Bus kế thừa Vehicle, có thêm thuộc tính name, capacity, num_passengers.
- Class Bus thay đổi phương thức run, trong đó thêm vào 2 thông tin: số người lên và số người xuống để cập nhật num_passenger.
- Tiến hành cho xe bus chạy qua 5 trạm, in ra số người có trên xe mỗi lần dừng.

Bài tập 2

- Tạo class Faculty có thuộc tính là list_students và phương thức notify.
- Tạo class Student với name, year.
- Faculty có phương thức register, unregister để thêm sinh viên vào danh sách thông báo của khoa hoặc huỷ đăng ký với tham số là Student.
- Tạo “interface” OnNotifyListener với phương thức rỗng onReceive() với một tham số là message
- Class Student “implement” OnNotifyListener và in ra thông tin message
- Tạo chương trình vận hành minh hoạ với khoa FIT và 3 students.

Bài tập 3

- Tạo ra class Creep với một thuộc tính tĩnh là num_instances, một hằng MAX_NUM_INSTANCE
- Người dùng chỉ gọi phương thức createInstance() để tạo ra creep với một name đưa vào.
- Khi số lượng creep vượt quá mức tối đa, thì không tạo thêm, trả về None.
- Creep có phương thức onDead() được gọi khi bị tiêu diệt để giảm num_instances.
- Tạo chương trình minh hoạ với khoảng 10 creeps.