LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG PYTHON

MỤC LỤC

CHƯƠNG 1. KHÁI NIỆM LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lập trình hướng đối tượng (Object-Oriented Programming – OOP) là một phương pháp lập trình sử dụng các đối tượng (Object) để xây dựng hệ thống phần mềm.

Khái niệm về OOP trong Python tập trung vào việc tạo code sử dụng lại. Khái niệm này còn được gọi là DRY (Don't Repeat Yourself).

Các nguyên lý của lập trình hướng đối tượng:

- *Tính kế thừa (Inheritance)*: cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa.
- *Tính đóng gói (Encapsulation)*: là quy tắc yêu cầu trạng thái bên trong của một đối tượng được bảo vệ và tránh truy cập được từ code bên ngoài (tức là code bên ngoài không thể trực tiếp nhìn thấy và thay đổi trạng thái của đối tượng đó).
- *Tính đa hình (Polymorphism)*: là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.
- *Tính trừu tượng (Abstract)*: trong ngôn ngữ Python, tính đa hình không được rõ nét. Python không cho phép khai báo lớp trừu tượng mà chỉ tạo ra một lớp Abstract Base Class (ABC) để người dùng thừa kế nếu muốn thực thi tính trừu tượng.

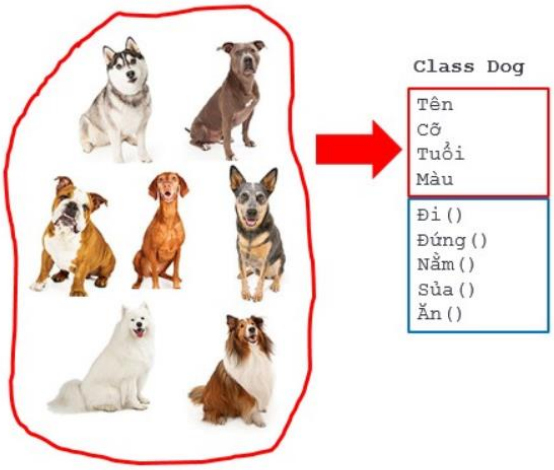
CHƯƠNG 2. LỚP & ĐỐI TƯỢNG

2.1 Lớp (Class)

Một lớp trong lập trình hướng đối tượng đại diện cho tập hợp các đối tượng có cùng các đặc điểm, hành vi, phương thức hoạt động.

Trong đó:

- đặc điểm của đối tượng được thể hiện ra bên ngoài là các thuộc tính (property) của lớp;
- các hành vi hay phương thức hoạt động của đối tượng gọi là phương thức của lớp;
- các thuộc tính và phương thức được gọi chung là thành viên của lớp.



2.1.1 Khai báo class

Khai báo một lớp bắt đầu bằng từ khóa **class** theo sau là tên của lớp và dấu hai chấm (:).

Dưới đây là cấu trúc tổng quát khi khai báo một lớp:

```
class className:

    # khai báo thuộc tính của lớp

    # khai báo thuộc tính của đối tượng
```

```
    # khai báo phương thức
```

Giải thích:

- Lớp được khai báo bởi từ khóa class, theo sau là tên của lớp và dấu hai chấm

Ví dụ về khai báo class:

```
class Dog:
    # Thuộc tính của lớp
    DogCount = 0
    def __init__(self, name, size, age, color):#hàm khởi tạo đối tượng
        self.name = name # Thuộc tính của đối tượng
        self.size = size # Thuộc tính của đối tượng
        self.age = age # Thuộc tính của đối tượng
        self.color = color # Thuộc tính của đối tượng

    def Go(self):
        print("I'm going...")

    def Stay(self, place):
        print("I'm staying at {}".format(place))

    def Lie(self, place):
        print("I'm lying at {}".format(place))

    def Bark(self):
        print("Whoop...")

    def Eat(self, food):
        print("I'm eating {}".format(food))

# khở i tạo đố i tượng
bull = Dog("Bull", "large", 2, "Yellow");
bull.Stay("garden")
bull.Bark()
```

Sau khi biên dịch đoạn code trên và chạy thử, kết quả thu được:

```
I'm staying at garden

Whoop...
```

2.1.2 Phương thức

Phương thức là tập hợp các lệnh Python dùng để thực hiện một cách trọn vẹn một nhiệm vụ nào đó.

Cú pháp định nghĩa phương thức:

```
def MethodName(self,<các tham số khác nếu có>):

        # các lệnh của phương thức
```

Giải thích:

- Tham số đầu tiên của phương thức luôn luôn là `self`, ngoài ra có thể bao gồm các tham số khác (xem phương thức `Eat`, `Lie`, `Stay`).
- `self` là tham số đặc biệt, nó ám chỉ đối tượng mà chúng ta đang làm việc. Khi gọi phương thức, không bao giờ truyền giá trị cho tham số này.

Trong ví dụ trên: `Go`, `Stay`, `Lie`, `Bark`, `Eat` là phương thức.

Cú pháp để gọi các phương thức của đối tượng:

```
Tên_đối_tượng.Tên_phương_phức(<tham số>)
```

(Xem dòng bôi vàng ở ví dụ trên để hiểu hơn về cách gọi phương thức của đối tượng)

2.1.2.1 Phương thức khởi tạo (Constructor)

Đây là một phương thức đặc biệt của lớp. Nó được thực thi khi khởi tạo đối tượng (xem mục 2.2 để biết cách khởi tạo một đối tượng).

Đặc điểm của phương thức khởi tạo:

- Luôn có tên `__init__`
- Nội dung của phương thức là để khởi tạo, đồng thời cũng là để xác định các thuộc tính của đối tượng.

Cú pháp khai báo phương thức khởi tạo:

```
def __init__(self, <các tham số khác nếu có>):

    # body of the constructor
```

Đoạn code dưới đây là phương thức khởi tạo cho lớp `Dog`:

```
def __init__(self, name, size, age, color):
        self.name = name # Thuộc tính của đối tượng
        self.size = size # Thuộc tính của đối tượng
        self.age = age # Thuộc tính của đối tượng
        self.color = color # Thuộc tính của đối tượng
```

Trong ví vụ này:

- Ngoài tham số mặc định `self`, còn có tham số khác là `name`, `size`, `age`, `color`
- `name`, `size`, `age`, `color` là các thuộc tính của đối tượng thuộc lớp `Dog`.

2.1.2.2 Phương thức hủy (destructor)

Là một phương thức đặc biệt của lớp, dùng để hủy một đối tượng của lớp đó. Trong Python, phương thức hủy không thật sự cần thiết, bởi Python có cơ chế tự động quản lý bộ nhớ (tự động hủy đối tượng khi không sử dụng nữa). Tuy nhiên, người dùng vẫn có thể tự khai báo phương thức hủy cho lớp nếu muốn.

Cú pháp khai báo phương thức hủy:

```
def __del__(self):

  # body of destructor
```

Ví dụ phương thức hủy cho lớp `Dog`:

```
class Dog:
    # Class attributes
    DogCount = 0
    def __init__(self, name, size, age, color):
        self.name = name # object attributes
        self.size = size # object attributes
        self.age = age # object attributes
        self.color = color # object attributes

    def __del__(self):
        print("A dog object is being deleted.")

obj = Dog("bull", "large", 2, "yellow")
del obj
```

Kết quả sau khi chạy code:

```
A dog object is being deleted.
```

2.1.3 Thành viên tĩnh của lớp

Thuộc tính hoặc phương thức trở thành dạng tĩnh là thuộc tính hoặc phương thức của lớp chứ không phải của đối tượng. Nó được dùng chung cho mọi đối tượng.

Thành viên tĩnh được sử dụng để lưu trữ và chia sẻ giá trị dùng chung giữa tất cả đối tượng được tạo từ lớp.

2.1.3.1 Thuộc tính tĩnh

Trong Python, thuộc tính tĩnh cần được khởi tạo giá trị ngay khi khai báo.

Ví dụ sau đây minh họa cho việc sử dụng thuộc tính tĩnh.

```
class Dog:
    # Thuộc tính của lớp - thuộc tính tĩnh DogCount
    DogCount = 0
    def __init__(self, name, size, age, color):
        self.name = name # object attributes
        self.size = size # object attributes
        self.age = age # object attributes
        self.color = color # object attributes
        Dog.DogCount = Dog.DogCount + 1
    def __del__(self):
        print("A dog object is being deleted.")
        Dog.DogCount = Dog.DogCount - 1

obj1 = Dog("bull", "large", 2, "yellow")
obj2 = Dog ("Poodle", "small", 1, "white")
print ("Number of dogs: {}".format(Dog.DogCount))
```

Trong ví dụ này, `DogCount` là thuộc tính tĩnh.

Kết quả thu được khi chạy code:

```
Number of dogs: 2
```

Chú ý:

Truy cập thuộc tính tĩnh của lớp thì phải sử dụng tên lớp chứ không sử dụng tên đối tượng (xem đoạn highlight màu vàng trong ví dụ trên).

2.1.3.2 Phương thức của lớp và phương thức tĩnh

Trong Python, phương thức của lớp và phương thức tĩnh là phương thức thuộc lớp chứ không phải thuộc đối tượng.

Ví dụ:

```
class Dog:
    # Class attributes
    DogCount = 0
    def __init__(self, name, size, age, color):
        self.name = name # object attributes
        self.size = size # object attributes
        self.age = age # object attributes
        self.color = color # object attributes
        Dog.DogCount = Dog.DogCount + 1

    @classmethod
    def CreateDog(cls, name, size, age, color):
        Dog.DogCount = Dog.DogCount + 1
        return cls(name, size, age, color)

    @staticmethod
    def Report():
        print("Number of dogs: {}".format(Dog.DogCount))

obj = Dog.CreateDog("bull", "large", 2, "yellow")
Dog.Report()
```

Từ ví dụ trên ta thấy, `CreateDog` là phương thức của lớp và `Report` là phương thức tĩnh. Phương thức của lớp thì sẽ có từ khóa `@classmethod` đứng trước, phương thức tĩnh có từ khóa `@staticmethod` đứng trước.

Sự khác biệt giữa hai loại phương thức này:

Phương thức của lớp (`@classmethod`)	**Phương thức tĩnh (`@staticmethod`)**
Có tham số đầu tiên là `cls`	**Không có** tham số `cls`
Có thể truy xuất và chỉnh sửa trạng thái của lớp.	Không thể chỉnh sửa hay truy xuất trạng thái của lớp.
Dùng để tạo ra các đối tượng của lớp (tương tự như phương thức khởi tạo)	Dùng để xử lý các công việc mang tính chất tổng quát

2.2 Đối tượng

Là một đối tượng trong thế giới thực, có thuộc tính và hành vi. Trong Python, đối tượng là một thể hiện của lớp.

Cú pháp để khởi tạo đối tượng:

```
Tên_biến = Tên_lớp(<các tham số cần thiết>)
```

Ví dụ về khởi tạo đối tượng:

```
obj1 = Dog ("bull", "large", 2, "yellow")
obj2 = Dog ("Poodle", "small", 1, "white")
```

Ví dụ trên khởi tạo 2 đối tượng `obj1` và `obj2` thuộc lớp `Dog`.

CHƯƠNG 3. CÁC TÍNH CHẤT CỦA HƯỚNG ĐỐI TƯỢNG

3.1 Tính đóng gói (Encapsulation)

Là khả năng che dấu thông tin của đối tượng với môi trường bên ngoài. Tính chất này giúp đảm bảo tính toàn vẹn và bảo mật của đối tượng.

Tính đóng gói được thể hiện thông qua phạm vi sử dụng:

- *Private*: không thể truy cập thành viên của lớp khi đứng ngoài lớp. Trong Python, thành viên dạng private sẽ có tiền tố là 2 dấu gạch dưới trước tên thành viên (__).
- *Protected*: chỉ truy cập được thành viên của lớp khi đứng ở trong lớp đó hoặc lớp con của nó. Trong Python, thành viên dạng protected sẽ có tiền tố là dấu gạch dưới trước tên (_).
- *Public*: mặc định (khai báo như tên thông thường) thì thành viên ở chế độ public, tức là có thể truy xuất vào các thành viên khi đứng ở bất kì đâu (trong lớp, ngoài lớp…)

Ví dụ:

```
class Dog:
    # Class attributes
    __DogCount = 0
    def __init__(self, name, size, age, color):
        self._name = name # protected attributes
        self._size = size # protected attributes
        self.__age = age # private attributes
        self.__color = color # private attributes
        Dog.DogCount = Dog.DogCount + 1

obj = Dog("bull", "large", 2, "yellow")
print ("Number of dogs: {}".format(Dog.DogCount))
```

Kết quả thu được sau khi chạy đoạn code:

```
Exception has occurred: AttributeError
type object 'Dog' has no attribute 'DogCount'
```

Giải thích:

- `_name`, `_size`: là thuộc tính dạng protected của đối tượng
- `__age`, `__color`: là thuộc tính dạng private của đối tượng

- `__DogCount`: là thuộc tính của lớp, dạng private
- Vì `__DogCount` ở chế độ private nên khi đứng ở ngoài lớp (dòng bôi vàng) không thể nhìn thấy (truy xuất) giá trị của nó. Do vậy, trình biên dịch đưa ra thông báo như kết quả ở trên.

3.2 Tính thừa kế (Inheritance)

Thừa kế là việc tái sử dụng lại một số thuộc tính, phương thức của lớp đã có sẵn. Lớp mới là lớp con (lớp dẫn xuất), lớp đã có sẵn là lớp cha (lớp cơ sở). Lớp con thừa kế các thuộc tính hoặc phương thức được định nghĩa ở lớp cha.

3.2.1 Cú pháp khai báo thừa kế

```
class BaseClassName:
    # khai báo lớp

class ChildClassName(BaseClassName):
    def __init__(self, <các tham số khác nếu có>):
        BaseClassName.__init__(<tham số>) # gọi hàm khởi tạo của lớp cha
       # khai báo lớp
```

Giải thích:

- BaseClass (lớp cơ sở/lớp cha): khai báo như cách khai báo một lớp thông thường
- ChildClass (lớp dẫn xuất/lớp con): để khai báo thừa kế từ lớp cơ sở. Thừa kế từ lớp nào thì tên lớp đó để trong dấu ngoặc tròn.
- Phương thức khởi tạo của lớp con sẽ gọi lại phương thức khởi tạo của lớp cha.

Mục đích của thừa kế:

- *Tái sử dụng code.* Khi tạo một lớp mới, không cần thiết phải viết lại toàn bộ thuộc tính hoặc phương thức mà sử dụng thuộc tính hoặc phương thức của lớp đã có sẵn.

Ví dụ về thừa kế:

```
class Animal:
    def __init__(self, name):
        self._name = name

    def Display(self):
        print("I'm {}".format(self._name))

class Dog(Animal):
    def __init__(self, name, size, age, color):
        super().__init__(name)
```

```
        self.size = size
        self.age  = age
        self.color = color
    def Go(self, place):
        print("I'm going to {}".format(place))

obj1 = Dog("bull", "large", 2, "yellow")

obj1.Display()
obj1.Go("garden")
```

Kết quả sau khi chạy code:

```
I'm bull
I'm going to garden
```

Giải thích ví dụ:

- Lớp `Dog` thừa kế các thuộc tính `Name` và phương thức `Display` từ lớp cơ sở `Animal`.
- Khi khai báo đối tượng `obj1` của lớp `Dog`, có thể gọi các phương thức của riêng lớp `Dog` là `Go` và phương thức thừa kế từ lớp cơ sở (`Display`)

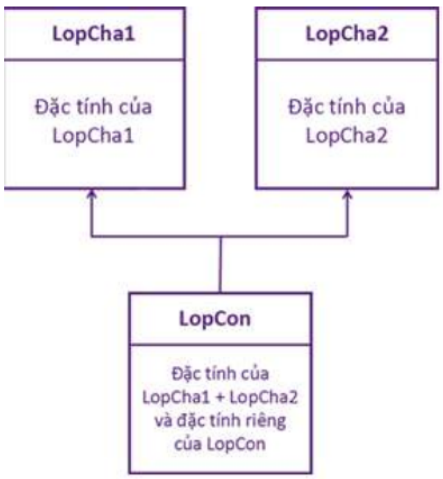
3.2.2 Các loại thừa kế

3.2.2.1 Đơn thừa kế

Lớp con chỉ thừa kế từ một lớp cha (xem ví dụ ở mục 3.2.1).

3.2.2.2 Đa thừa kế

Lớp con thừa kế từ nhiều lớp cha.



Cú pháp khai báo:

```
class LopCha1:
     # khai báo lớp cha 1

class LopCha2:
     # khai báo lớp cha 2

class LopCon(LopCha1, LopCha2):
     # khai báo lớp con
```

Ví dụ:

```
class Zebra:
    def __init__(self):
        print ("Zebra")

    def Display(self):
        print("I'm a zebra")

class Donkey:
    def __init__(self):
        print ("Donkey")
    def Display(self):
        print("I'm a donkey")

class Zonkey(Zebra, Donkey):
    def __init__(self):
        Zebra.__init__(self)
        Donkey.__init__(self)
        print("Zonkey")

obj = Zonkey()
obj.Display() # có vấn đề khi chạy phương thức này!
```

Kết quả sau khi chạy code:

```
Zebra
Donkey
Zonkey
I'm a zebra
```

Trong ví dụ trên, lớp `Zonkey` thừa kế từ 2 lớp cha là `Zebra` và `Donkey`.

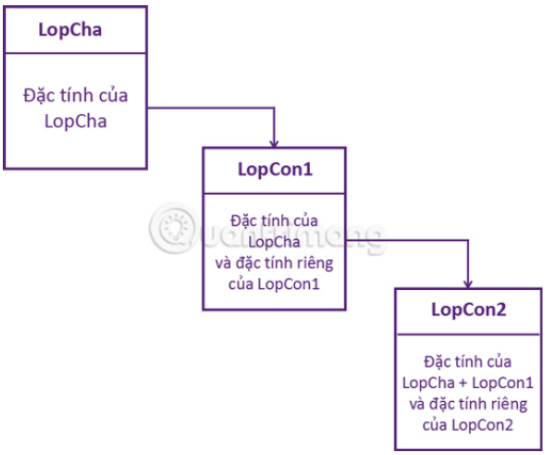
Tuy nhiên, có một vấn đề phát sinh khi thực hiện đa thừa kế: khi các lớp cha có phương thức cùng tên (trong ví dụ trên `Zebra` và `Donkey` đều có phương thức `Display`) thì trình biên dịch không rõ người dung định gọi phương thức nào!???? Như trong ví dụ trên, Python sẽ gọi phương thức `Display` của lớp `Zebra`, lớp xuất hiện trước trong danh sách thừa kế.

Tóm lại:

Trong Python, các lớp cha có thể có các thuộc tính hoặc các phương thức giống nhau. Lớp con sẽ ưu tiên thừa kế thuộc tính, phương thức của lớp đứng đầu tiên trong danh sách thừa kế.

3.2.3 Thừa kế đa cấp

Đây là hiện tượng, một lớp (cháu) thừa kế từ một lớp con. Trong trường hợp này, các thành viên của lớp cha và lớp con sẽ được lớp cháu thừa kế.



Cú pháp khai báo:

```
class LopCha:
     # khai báo lớp cha

class LopCon(LopCha):
     # khai báo lớp con

class LopChau(LopCon1):
     # khai báo lớp cháu
```

3.2.4 Thứ tự truy xuất phương thức trong thừa kế

Trong kịch bản đa thừa kế, bất kỳ thuộc tính cần được truy xuất nào, đầu tiên sẽ được tìm kiếm trong lớp hiện tại. Nếu không tìm thấy, tìm kiếm tiếp tục vào lớp cha đầu tiên và từ trái qua phải.

Vậy thứ tự truy xuất sẽ là [LopCon, LopCha1, LopCha2, object].

3.3 Tính đa hình (Polymorphism)

Là khả năng một đối tượng có thể thực hiện một tác vụ theo nhiều cách khác nhau. Tài liệu này chủ yếu tập trung vào tính đa hình trong thừa kế.

Trong Python, đa hình cho phép tạo ra phương thức trùng tên với phương thức ở lớp cha.

Kĩ thuật này gọi là Method overriding (nạp chồng phương thức).

Ví dụ:

```
class Bird:
    def flight(self):
        print("Many birds can fly")

class Sparrow(Bird):
    def flight(self):
        print("Sparrows can fly")

class Ostrich(Bird):
    def flight(self):
        print ("Ostriches cannot fly")

obj1 = Bird();
obj2 = Sparrow();
obj3 = Ostrich();
obj1.flight();
obj2.flight();
obj3.flight();
```

Kết quả sau khi chạy code:

```
Many birds can fly
Sparrows can fly
Ostriches cannot fly
```

Trong ví dụ trên, lớp `Sparrow` và `Ostrich` thừa kế từ lớp `Bird`. Về mặt nguyên tắc, hai lớp này hoàn toàn được phép thừa kế phương thức `flight` của lớp `Bird` nhưng nó lại tự định nghĩa phương thức `flight` theo cách riêng của nó. Và như vậy, mặc dù cùng gọi đến phương thức `flight` nhưng với các đối tượng khác nhau thì lại có nội dung khác nhau.

3.4 Tính trừu tượng (Abstract)

Lớp trừu tượng (abstract class) trong Python được coi như là bản thiết kế hay bộ khung để các class khác tuân theo.

Một lớp có một hoặc nhiều phương thức trừu tượng gọi là lớp trừu tượng.

Phương thức trừu tượng là phương thức chỉ có khai báo (chỉ đưa ra tên và tham số) mà không có định nghĩa (không có nội dung).

Mặc định, Python không cho phép khai báo lớp trừu tượng, nó chỉ có một module chứa lớp cơ sở Abstract Base Class (ABC). Muốn triển khai trừu tượng thì phải thừa kế lớp này.

Ví dụ:

```
from abc import ABC, abstractmethod

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

class Dog(Animal):
    def Say(self):
        print("Whoooopp whooppp...")

class Cat (Animal):
    def Say(self):
        print("Meooo meooo....")
    def Jump(self):
        print("I'm jumping up high....")

obj_dog = Dog()
obj_cat = Cat()

obj_dog.Say()
obj_cat.Say()
```

Trong ví dụ trên:

- Lớp `Animal` muốn trở thành lớp trừu tượng thì nó phải thừa kế lớp `ABC`, phương thức `Say` của nó có thẻ `@abstractmethod` phía trên và không có nội dung gì.
- Lớp `Dog` và `Cat` là lớp con của lớp trừu tượng `Animal`, do vậy 2 lớp này phải định nghĩa phương thức `Say` theo cách riêng của nó.
- Ngoài định nghĩa phương thức `Say` thừa kế từ lớp cha, lớp con vẫn có thể có những phương thức riêng của nó. Ví dụ như phương thức `Jump` của lớp `Cat`.