



## **Unit 8**

Classes and Objects; Inheritance

Hướng đối tượng

# Hướng đối tượng là gì?

- Là kỹ thuật bao bọc dữ liệu, nhằm tạo ra một kiểu dữ liệu mới.
- Các khái niệm:
  - Trừu tượng (**Abstraction**)
  - Đa hình (**Polymorphism**)
  - Đóng gói (**Encapsulation**)
  - Kế thừa (**Inheritance**)

# OOP, định nghĩa Class

- Python được xây dựng như một ngôn ngữ lập trình hướng thủ tục (procedural language).
  - OOP có tồn tại, hoạt động tốt nhưng không chặt chẽ
- Khai báo class như sau

```
class name:  
    statements
```

# Fields (thuộc tính)

**name = value**

– Ví dụ:

```
# main
p1 = Diem()
p1.x = 2
p1.y = -5
```

**diem.py**

```
1 class Diem:
2     x = 0
3     y = 0
```

- Thuộc tính là biến nằm trong lớp, mô tả đặc tính của đối tượng.
- Được khai báo trực tiếp trong lớp hay trong hàm tạo (constructors)
- Python không có tính chất đóng gói (encapsulation) hay private fields

# Sử dụng Class

import **class**

- client programs must import the classes they use

## point\_main.py

```
1  from Diem import *
2
3  # main
4  p1 = Diem()
5  p1.x = 7
6  p1.y = -3
7  ...
8
9  # Python objects are dynamic (can add fields any time!)
10 p1.name = "Tyler Durden"
```

# Object Methods

```
def name(self, parameter, ..., parameter) :  
    statements
```

- **self** *must* be the first parameter to any object method
  - represents the "implicit parameter" (`this` in Java)
- *must* access the object's fields through the `self` reference

```
class Diem:  
    def dich_chuyen(self, dx, dy) :  
        self.x += dx  
        self.y += dy  
    ...
```

# Exercise Answer

## diem.py

```
1  from math import *
2
3  class Diem:
4      x = 0
5      y = 0
6
7      def dat_vi_tri(self, x, y):
8          self.x = x
9          self.y = y
10
11     def khoang_cach_tu_tam_o(self):
12         return sqrt(self.x * self.x + self.y * self.y)
13
14     def khoang_cach(self, other):
15         dx = self.x - other.x
16         dy = self.y - other.y
17         return sqrt(dx * dx + dy * dy)
```

# Calling Methods

- Có 2 cách”

1) **object.method (parameters)**

hoặc

2) **Class.method (object, parameters)**

- Ví dụ:

```
p = Diem(3, -4)
```

```
p.dich_chuyen(1, 5)
```

```
Diem.dich_chuyen(p, 1, 5)
```



# Constructors (hàm tạo)

```
def __init__(self, parameter, ..., parameter) :  
    statements
```

- a constructor is a special method with the name `__init__`
- Ví dụ:

```
class Diem:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
    ...
```

- How would we make it possible to construct a `Diem()` with no parameters to get (0, 0)?

# toString and `__str__`

```
def __str__(self):  
    return string
```

- equivalent to Java's `toString` (converts object to a string)
- invoked automatically when `str` or `print` is called

Exercise: Write a `__str__` method for `diem` objects that returns strings like `"(3, -14)"`

```
def __str__(self):  
    return "(" + str(self.x) + ", " + str(self.y) + ")"
```

# Complete Point Class

diem.py

```
1  from math import *
2
3  class Diem:
4      def __init__(self, x, y):
5          self.x = x
6          self.y = y
7
8      def khoang_cach_tu_tam_o(self):
9          return sqrt(self.x * self.x + self.y * self.y)
10
11     def khoang_cach(self, other):
12         dx = self.x - other.x
13         dy = self.y - other.y
14         return sqrt(dx * dx + dy * dy)
15
16     def dich_chuyen(self, dx, dy):
17         self.x += dx
18         self.y += dy
19
20     def __str__(self):
21         return "(" + str(self.x) + ", " + str(self.y) + ")"
```

# Operator Overloading

- **operator overloading:** You can define functions so that Python's built-in operators can be used with your class.
  - See also: <http://docs.python.org/ref/customization.html>

Operator	Class Method
-	<code>__sub__(self, other)</code>
+	<code>__add__(self, other)</code>
*	<code>__mul__(self, other)</code>
/	<code>__truediv__(self, other)</code>

## Unary Operators

-	<code>__neg__(self)</code>
+	<code>__pos__(self)</code>

Operator	Class Method
<code>==</code>	<code>__eq__(self, other)</code>
<code>!=</code>	<code>__ne__(self, other)</code>
<code>&lt;</code>	<code>__lt__(self, other)</code>
<code>&gt;</code>	<code>__gt__(self, other)</code>
<code>&lt;=</code>	<code>__le__(self, other)</code>
<code>&gt;=</code>	<code>__ge__(self, other)</code>

# Bài tập

- **Viết class PhanSo** để biểu diễn phân số như  $1/2$  hay  $-3/8$ .
- Viết hàm rút gọn phân số; ví dụ, rút gọn  $4/12$  thành  $1/3$  hay  $6/-9$  thành  $-2/3$ .
  - Hint: Sử dụng hàm tìm ước chung lớn nhất (greatest common divisor)  $\rightarrow$  `math.gcd(a,b)`
- Định nghĩa hàm **cong** và **nhan** 2 phân số
- Định nghĩa các toán tử  $+$ ,  $*$ ,  $==$ ,  $<$

# Generating Exceptions

```
raise ExceptionType ("message")
```

- useful when the client uses your object improperly
- **types:** `ArithmeticError`, `AssertionError`, `IndexError`, `NameError`, `SyntaxError`, `TypeError`, `ValueError`

- Example:

```
class BankAccount:  
    ...  
    def deposit(self, amount):  
        if amount < 0:  
            raise ValueError("negative amount")  
        ...
```

# Inheritance (thừa kế)

```
class name (superclass) :  
    statements
```

– Ví dụ:

```
class Diem3D (Diem) :      # Diem3D extends Diem  
    z = 0  
    ...
```

- Python also supports *multiple inheritance*

```
class name (superclass, ..., superclass) :  
    statements
```

*(if > 1 superclass has the same field/method, conflicts are resolved in left-to-right order)*

# Calling Superclass Methods

- methods: **class.method (object, parameters)**
- constructors: **class.\_\_init\_\_(parameters)**

```
class Diem3D(Diem):  
    z = 0  
    def __init__(self, x, y, z):  
        Diem.__init__(self, x, y)  
        self.z = z  
  
    def dich_chuyen(self, dx, dy, dz):  
        Diem.dich_chuyen(self, dx, dy)  
        self.z += dz
```