OOP C++

VS

OOP Python

THÔNG TIN SINH VIÊN

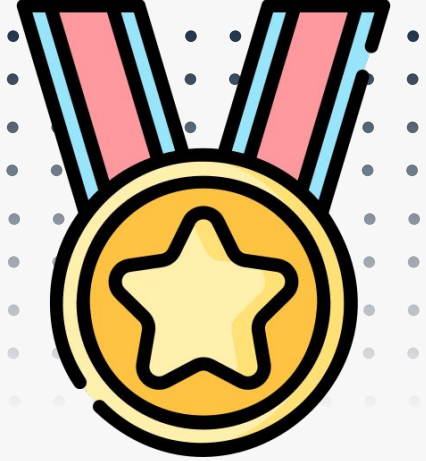
MSSV: 19120662

Họ tên: Đinh Trần Xuân Thi

Lớp: 19CTT4

CÁC CHỦ ĐỀ THUYẾT TRÌNH

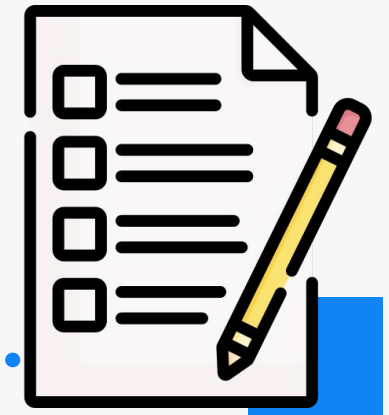
- Lớp, đối tượng và các vấn đề liên quan
- Hàm khởi tạo và hàm hủy
- Thuộc tính tĩnh và phương thức tĩnh
- Kế thừa
- Đa hình



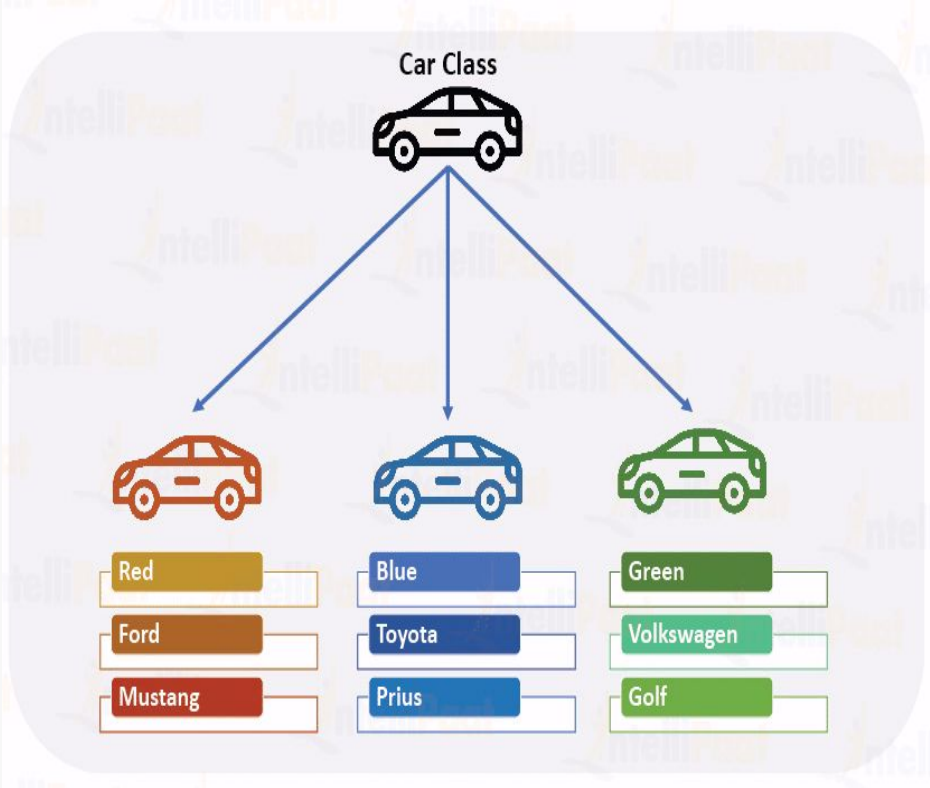
CLASS, OBJECT VÀ CÁC VẤN ĐỀ LIÊN QUAN

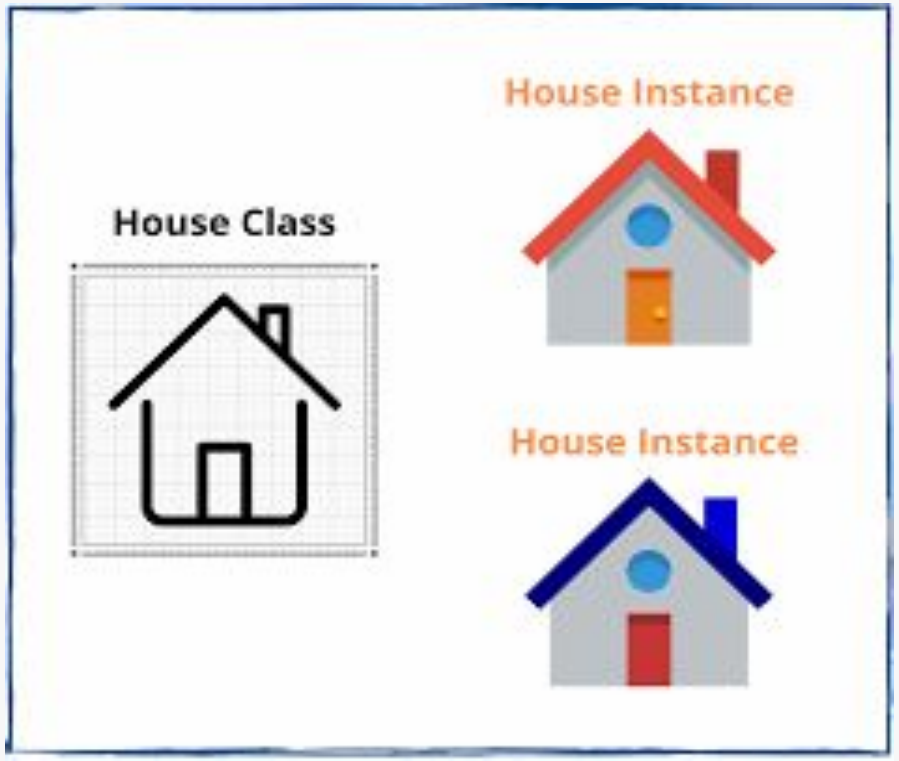
CLASS VÀ OBJECT

- Lớp (class) là một bản thiết kế hoặc nguyên mẫu mà từ đó chúng ta có thể tạo ra các đối tượng
- Đối tượng (object) chính là một thể hiện của một class



CLASS VÀ OBJECT





KHAI BÁO CLASS VÀ OBJECT

C++

```
class Rectangle {
};

Rectangle figure1();
```

Python

```
class Rectangle:
    pass

figure1 = Rectangle()
```

- Trong C++, kết thúc khai báo class bắt buộc phải có dấu ;
- Trong python, từ khóa pass dùng để giữ chỗ, khi có thuộc tính hoặc phương thức trong class thì không cần sử dụng

THÀNH PHẦN CLASS

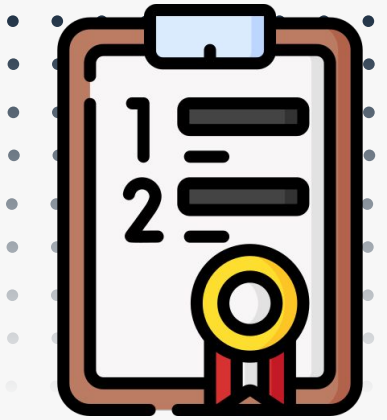
- Một tên lớp duy nhất để phân biệt các lớp khác
- Các thuộc tính (attribute) chứa dữ liệu mô tả đối tượng
- Các phương thức (method) mô tả hành vi đối tượng

ATTRIBUTE TRONG C++

```
class Rectangle {
    // mặc định là private;
    int width;
    int height;
public: // thành phần public
    string color;
};

int main() {
    Rectangle figure1;
    // figure1.width = 1;    Lỗi không thể truy
xuất thành phần private
    figure1.color = "red";
    cout << figure1.color << endl;
}
```

- Các thuộc tính phải được khai báo trong class
- Tất cả các đối tượng của class đều có thuộc tính đó
- Mặc định thì phạm vi truy cập là private

ATTRIBUTE TRONG PYTHON



```
class Rectangle:
    def __init__(self):
        self.width = 1
        self.height = 1

figure1 = Rectangle
figure2 = Rectangle
figure1.color = 'red' #chỉ figure1 mới có thuộc tính color
figure1.width = 2  # có thể truy cập ngoài lớp
print(figure1.color)
# print(figure2.color) lỗi figure2 không có thuộc tính color
```

- Thuộc tính có thể khai báo trong phương thức hoặc khai báo ngoài lớp
- Thông thường sẽ khai báo thuộc tính trong hàm __init__
- Phạm vi truy cập mặc định là public

METHOD TRONG C++

- Phạm vi truy cập mặc định là private
- Có thể định nghĩa phương thức trong class hoặc ngoài class
- Khi định nghĩa phương thức ngoài class thì phải sử dụng toán tử :: và tên class

CON TRỎ THIS TRONG C++

- Là một từ khóa đề cập đến thể hiện hiện tại của lớp
- Là tham số ẩn với tất cả các phương thức của lớp
- Ở trong các phương thức thì con trỏ **this** đang tham chiếu tới đối tượng được gọi
- Sử dụng con trỏ this rõ ràng khi tham số của phương thức trùng tên với thuộc tính

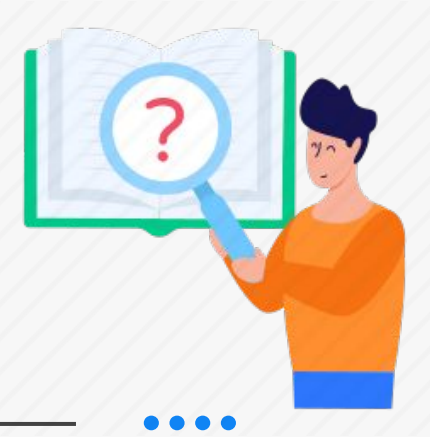
METHOD TRONG C++

```
class Rectangle {
    // mặc định là private;
    int width;
    int height;
public: // thành phần public
    string color;
    void setWidth(int width) {this->width = width; // dùng
this rõ ràng}
    void setHeight(int height) {this->height = height; //
dùng this rõ ràng}
    int getPerimeter();
    int getArea();
};
int Rectangle::getPerimeter() {return (width + height) * 2;}
int Rectangle::getArea() {return width * height;}
```

METHOD TRONG C++

```
int main() {
    Rectangle figure1;
    figure1.color = "red";
    cout << "Figure1 color: " << figure1.color << endl;
    figure1.setWidth(10);
    figure1.setHeight(5);
    cout << "Figure1 area:" << figure1.getArea() << endl;
    cout << "Figure1 p:" << figure1.getPerimeter() << endl;

    return 0;
}
```

```
Figure1 color: red
Figure1 area: 50
Figure1 perimeter: 30
```



METHOD TRONG PYTHON

- Các phương thức bắt buộc phải thêm tham số **self** đầu tiên trong định nghĩa phương thức
- Tham số self được Python cung cấp giá trị
- Tham số **self** tương tự con trỏ **this** trong C++

METHOD TRONG PYTHON

```
class Rectangle:
    def __init__(self):
        self.width = 1
        self.height = 1
    def getPerimeter(self):
        return (self.width + self.height) * 2
    def getArea(self):
        return self.width + self.height
    def setWidth(self, width):
        self.width = width
    def setHeight(self, height):
        self.height = height
```

```
figure1 = Rectangle()
figure1.setWidth(10)
figure1.setHeight(5)
print("Figure1 color: ", figure1.color)
print("Figure1 perimeter: ",
figure1.getPerimeter())
print("Figure1 area: ", figure1.getArea())
```

```
Figure1 color:  red
Figure1 perimeter:  30
Figure1 area:  15
```

TẦM VỰC

Tầm vực	Phạm vi truy cập
private	bên trong lớp, bên ngoài không truy xuất được
protected	bên trong lớp và lớp kế thừa
public	bên trong lớp lẫn bên ngoài lớp đều truy xuất được

TẦM VỰC TRONG C++

Trong C++, người ta sử dụng từ khóa public, private, protected để chỉ tầm vực của một khối

```
class Rectangle {
private:
    int width;
    int height;
    int getPerimeter(){return (width + height) * 2;}
public: // thành phần public
    string color;
    int getArea(){return width * height;}
};
```

```
int main() {
    Rectangle figure1;
    figure1.width = 2;    //Sai
    figure1.height = 3;    //Sai
    figure1.getPerimeter(); //Sai
    figure1.color = "red"; //Đúng
    figure1.getArea() //Đúng
    cout << figure1.color << endl; //Đúng
}
```

TẦM VỰC TRONG PYTHON

- public: để tên thuộc tính và phương thức như bình thường

```
class Rectangle:
    def __init__(self):
        self.width = 1
        self.height = 1
    def getPerimeter(self):
        return (self.width + self.height) * 2
    def getArea(self):
        return self.width + self.height
```

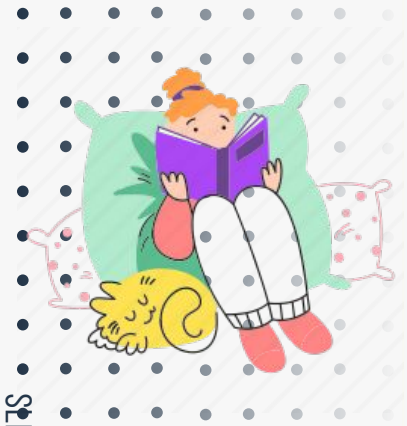
TẦM VỰC TRONG PYTHON

- private: thêm 2 dấu gạch dưới "__" vào tên phương thức và thuộc tính, tuy nhiên chỉ là giả private, chúng ta có thể truy cập qua cú pháp: **object._className__variable**

```
class Rectangle:
    def __init__(self):
        self.width = 1
        self.__height = 1

figure1 = Rectangle()
figure1.__height = 5 #Sai
figure1._Rectangle__height = 5
#Đúng
```

TẦM VỰC TRONG PYTHON

- protected: thêm 1 dấu gạch dưới "_" vào tên thuộc tính và phương thức. Tuy nhiên đó chỉ là quy ước, thực tế chúng ta vẫn truy cập bình thường



```
class Rectangle:
    def __init__(self):
        self.width = 1
        self._height = 1

figure1 = Rectangle()
figure1._height = 5 #Đúng

print(figure1._height)
```

```
Output: 5
```

TÍNH ĐÓNG GÓI

- Gom các dữ liệu và phương thức có liên quan với nhau vào trong một class
- Các thuộc tính nên có tầm vực là private để ẩn dữ liệu
- Các phương thức nên có tầm vực là public

GETTER VÀ SETTER

- Dùng getter để lấy được giá trị của các thuộc tính
- Dùng setter để gán giá trị cho thuộc tính với những ràng buộc cho dữ liệu đúng

CLASS C++ SỬA LẠI

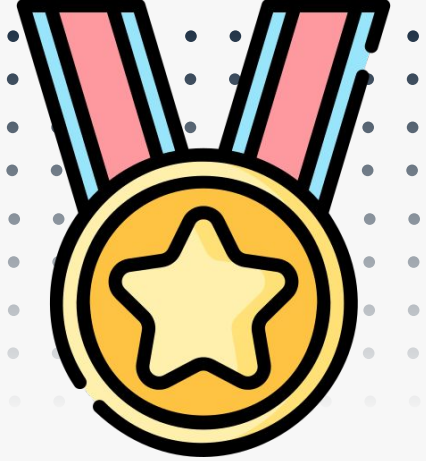
```
class Rectangle {
private:
    int width;
    int height;
    string color;
public:
    int getWidth();
    void setWidth(int width);
    int getHeight();
    void setHeight(int height);
    string getColor();
    void setColor(string color);
    int getPerimeter();
    int getArea();
};
```

```
int Rectangle::getWidth() {return width;}

void Rectangle::setWidth(int width) {this->width = width;}

int Rectangle::getHeight() {return height;}

void Rectangle::setWidth(int width) {this->height = height;}

string Rectangle::getColor() {return color;}

void Rectangle::setColor(string color) {this->color = color;}

int Rectangle::getPerimeter() {return (width + height) * 2;}

int Rectangle::getArea() {return width * height;}
```

CLASS PYTHON SỬA LẠI

```python
class Rectangle:
    def __init__(self):
        self.__width = 1
        self.__height = 1
        self.__color = ''
    def getPerimeter(self):
        return (self.__width + self.__height) * 2
    def getArea(self):
        return self.__width * self.__height
```

```python
    def getWidth(self):
        return self.__width
    def setWidth(self, width):
        self.__width = width
    def getHeight(self):
        return self.__height
    def setHeight(self, height):
        self.__height = height
    def getColor(self):
        return self.color
    def setColor(self, color):
        self.__color = color
```



CONSTRUCTOR VÀ DESTRUCTOR

CONSTRUCTOR

- Khi vừa được khai báo thì đối tượng cần có thông tin khởi tạo ban đầu
- Có thể sử dụng hàm để khởi tạo. Tuy nhiên, người dùng có thể quên gọi hàm, có thể sinh ra lỗi

⇨ Giải pháp: **constructor**

CONSTRUCTOR

- Constructor (hàm khởi tạo, hàm dựng) được gọi ngay khi đối tượng được tạo ra
- Chỉ được gọi khi đối tượng được tạo ra
- Dùng để khởi tạo giá trị ban đầu cho các thuộc tính
- Không có kiểu trả về

CONSTRUCTOR

C++

- Tên hàm trùng với tên lớp
- Một class có thể có nhiều constructor
- Có 3 loại: default, parameter và copy

Python

- Tên hàm là __init__
- Một class chỉ có một constructor
- Có 2 loại: default, parameter

PARAMETER CONSTRUCTOR TRONG C++

Có một hoặc nhiều tham số

```
class Rectangle {
private:
    int width;
    int height;
    string color;
public:
    Rectangle(int);
    Rectangle(int, int, string);
    void print();
};
```

```
Rectangle::Rectangle(int width) {
    this->width = width;
    height = 1;
    color = "transparent";
}
Rectangle::Rectangle(int width, int height, string color) {
    this->width = width;
    this->height = height;
    this->color = color;
}
```

PARAMETER CONSTRUCTOR TRONG C++

Sử dụng parameter constructor

```
int main() {
    Rectangle figure1(5);
    Rectangle figure2(5, 10, "blue");
    figure1.print();
    figure2.print();
    return 0;
}
```

DEFAULT CONSTRUCTOR TRONG C++

Là constructor không có tham số nào

```
class Rectangle {
private:
    int width;
    int height;
    string color;
public:
    Rectangle();
    void print();
};
```

```
Rectangle::Rectangle() {
    width = 1;
    height = 1;
    color = "transparent";
}
```

```
int main() {
    Rectangle figure1;
    figure1.print();
}
```

```
Width: 1
Height: 1
```

DEFAULT CONSTRUCTOR TRONG C++

Compiler sẽ cung cấp default constructor nếu chưa có bất kì constructor nào

```
class Rectangle {
private:
    int width;
    int height;
    string color;
public:
    void print();
};
```

```
int main() {
    Rectangle figure1;
    figure1.print();
}
```

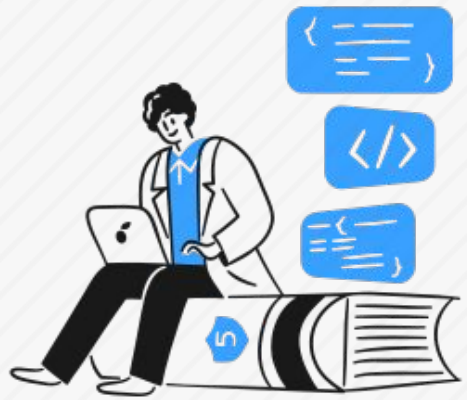
```
Output:
Width: 1990356781
Height: 4201136
```

DEFAULT CONSTRUCTOR TRONG C++

Nhưng nếu đã có một số constructor, thì phải tự định nghĩa default constructor

```
class Rectangle {
private:
    int width;
    int height;
    string color;
public:
    Rectangle(int);
    void print();
};
```

```
int main() {
    Rectangle figure1; // Lỗi
    figure1.print();   // Lỗi
    return 0;
}
```



COPY CONSTRUCTOR TRONG C++

Tạo đối tượng có thuộc tính giống một đối tượng có sẵn

```
class Rectangle {
private:
    int width;
    int height;
    string color;
public:
    Rectangle(int, int, string);
    Rectangle(const Rectangle&);
    void print();
};
```

```
Rectangle::Rectangle(const  Rectangle& srcFigure) {
    width = srcFigure.width;
    height = srcFigure.height;
    color = srcFigure.color;
}
```

```
int main() {
    Rectangle figure1(5, 10, "blue");
    Rectangle figure2(figure1);
    figure2.print();
    return 0;
}
```

COPY CONSTRUCTOR TRONG C++

Nếu không định nghĩa copy constructor, compiler sẽ tự động cung cấp

```
class Rectangle {
private:
    int width;
    int height;
    string color;
public:
    Rectangle(int, int, string);
    void print();
};
```

```
int main() {
    Rectangle figure1(5, 10, "blue");
    Rectangle figure2(figure1);
    figure2.print();
    return 0;
}
```

COPY CONSTRUCTOR TRONG C++

Default copy constructor theo dạng shallow copy (member-wise copy), cho nên:

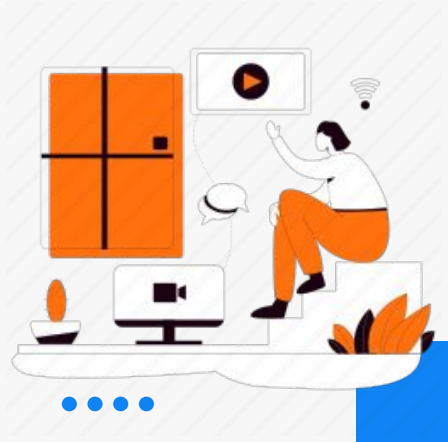
- Kiểu dữ liệu bình thường: không cần viết
- Kiểu dữ liệu con trỏ: phải viết rõ copy constructor

COPY CONSTRUCTOR TRONG C++

```
Array::Array(int length) {
    this->length = length;
    if (length == 0) {arr = nullptr;}
    else {
        arr = new int[length];
        for (int i = 0; i < length; i++) {arr[i] = 0;}}}
```

```
Array::Array(const Array& src) {
    length = src.length;
    if (length == 0) {arr = nullptr;}
    else {
        arr = new int[length];
        for (int i = 0; i < length; i++) {arr[i] = src.arr[i];}
    }
}
```

```
class Array {
private:
    int length;
    int* arr;
public:
    Array(int);
    Array(const Array&);
};
```



DEFAULT CONSTRUCTOR TRONG PYTHON

Khá giống với default constructor trong C++, ngoại trừ có thêm tham số self và tên hàm là __init__

```
class Rectangle:
    def __init__(self):
        self.__width = 1
        self.__height = 1
        self.__color = 'transparent'
    def print(self):
        print('Width: ', self.__width)
        print('Height: ', self.__height)
        print('Color: ', self.__color)
```

```
figure1 = Rectangle()
figure1.print()
```

```
Width:  1
Height:  1
Color:  transparent
```

PARAMETER CONSTRUCTOR TRONG PYTHON

Cũng khá giống với parameter constructor trong C++, ngoại trừ có thêm tham số self và tên hàm là __init__

```
class Rectangle:
    def __init__(self, width, height, color):
        self.__width = width
        self.__height = height
        self.__color = color
    def print(self):
        print('Width: ', self.__width)
        print('Height: ', self.__width)
        print('Color: ', self.__color)
```

```
figure1 = Rectangle(5, 10, 'red')
figure1.print()
```

```
Width:  5
Height:  10
Color:  red
```

CONSTRUCTOR TRONG PYTHON

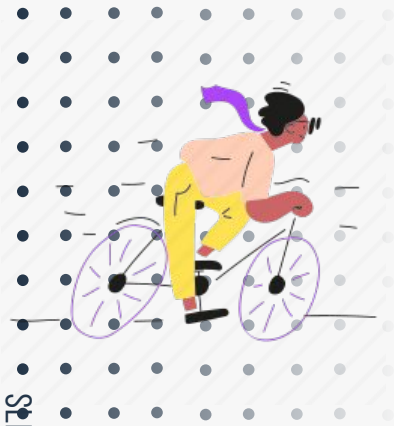
Python chỉ cho phép có 1 constructor, nếu cố tính viết nhiều constructor thì hàm sau sẽ ghi đè lên hàm trước

```
class Rectangle:
    def __init__(self):
        self.__width = 1
        self.__height = 1
        self.__color = 'transparent'
    def __init__(self, width, height, color):
        self.__width = width
        self.__height = height
        self.__color = color
```

```
figure1 =Rectangle() #Lỗi
figure2 = Rectangle(5, 10, 'red') #Đúng
```

DESTRUCTOR

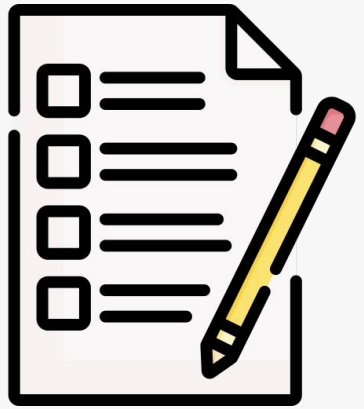
- Có lúc đối tượng sử dụng tài nguyên và sau khi kết thúc cần giải phóng tài nguyên đó
- Có thể tạo hàm để giải phóng tài nguyên đó. Tuy nhiên người dùng có thể quên gọi hàm đó



Giải pháp: **destructor**

DESTRUCTOR

- Destructor (hàm hủy) được gọi tự động khi đối tượng bị hủy
- Dùng để giải phóng tài nguyên đã sử dụng
- Không có kiểu trả về
- Không có tham số



DESTRUCTOR

C++

- Tên hàm trùng với tên lớp và có thêm dấu ~ ở trước
- Một class chỉ có một destructor
- Cần thiết khi trong class có dữ liệu là con trỏ

Python

- Tên hàm là __del__
- Một class chỉ có một destructor
- Không cần thiết, Python sẽ tự giải phóng bộ nhớ

DESTRUCTOR TRONG C++

Cần thiết khi thuộc tính có kiểu dữ liệu con trỏ, cần phải xóa bộ nhớ đã cấp phát

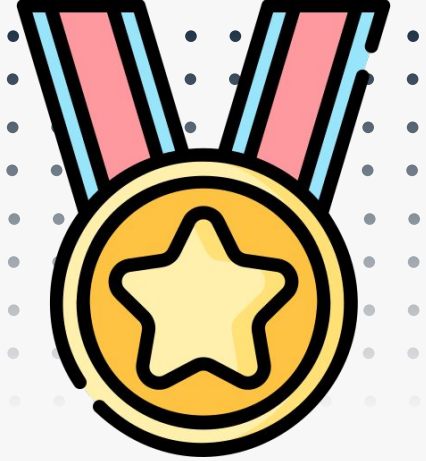
```
class Array {
private:
    int length;
    int* arr;
public:
    Array(int);
    Array(const Array&);
    ~Array();
};
```

```
Array::~Array() {
    if (length > 0) {
        delete[] arr;
        arr = nullptr;
    }
}
```

DESTRUCTOR TRONG PYTHON

Không cần thiết, giải phóng bộ nhớ có Python lo

```
class Rectangle:
    def __init__(self, width, height, color):
        self.__width = width
        self.__height = height
        self.__color = color

    def __del__(self):
        print("Destructor")
```



STATIC ATTRIBUTE VÀ STATIC METHOD

STATIC ATTRIBUTE

- Thuộc tính tĩnh (Static attribute) là thuộc tính chung của lớp
- Không của riêng bất kỳ đối tượng nào

STATIC ATTRIBUTE

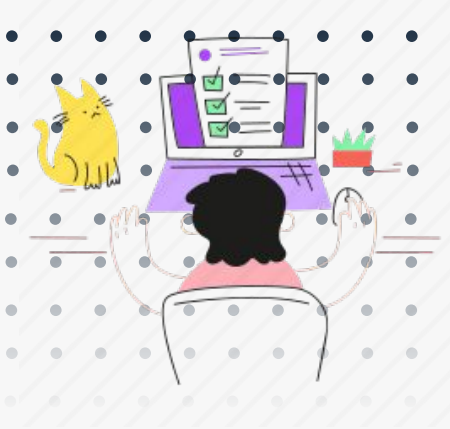
C++

- Thêm **static** ở đằng trước khi khai báo thuộc tính và gán giá trị trước hàm main
- Sử dụng: ClassName::attr (không được dùng obj.attr)

Python

- Định nghĩa trong lớp trong lớp nhưng không phải trong phương thức
- Sử dụng: ClassName.attr hoặc obj.attr

STATIC ATTRIBUTE



C++

- Nếu cố tình sử dụng obj.attr thì khi obj.attr bị thay đổi, ClassName::attr và objs_other.attr cũng sẽ bị thay đổi

Python

- Khi obj.attr bị thay đổi, chỉ có duy nhất obj.attr bị thay đổi. Lúc này, obj.attr sẽ độc lập với ClassName.attr

STATIC ATTRIBUTE

Trong C++

```cpp
int main() {
    Obj obj1;
    cout << Obj::objNum << endl;
    Obj obj2;
    cout << Obj::objNum << endl;
    Obj obj3;
    cout << Obj::objNum << endl;

    obj3.objNum = 10; // Cố tình sử dụng;
    cout << "Class Obj: " << Obj::objNum << endl;
    cout << "obj3: " << obj3.objNum << endl;
    return 0;
}
```

```cpp
class Obj {
public:
    static int objNum;
    Obj() {
        ++objNum;
    }
};

int Obj::objNum = 0;
```

```
1
2
3
Class Obj: 10
obj3: 10
```

STATIC ATTRIBUTE

Trong Python

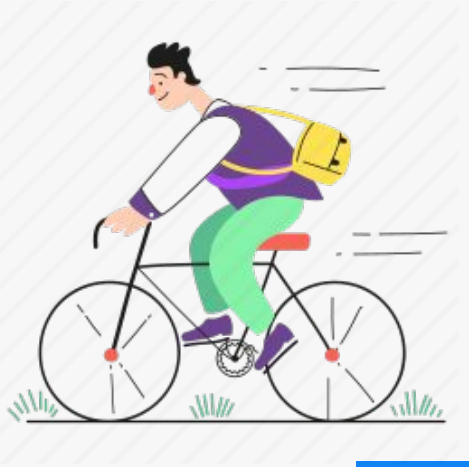
```
class Obj:
    objNum = 0 #static
attribute
    def __init__(self):
        Obj.objNum += 1
```

```
1
2
3
Class Obj: 3
obj3: 10
Class Obj: 7
obj3: 10
```

```
obj1 = Obj()
print(Obj.objNum)
obj2 = Obj()
print(Obj.objNum)
obj3 = Obj()
print(Obj.objNum)

obj3.objNum = 10
print("Class Obj:", Obj.objNum)
print("obj3:", obj3.objNum)

Obj.objNum = 7
print("Class Obj:", Obj.objNum)
print("obj3:", obj3.objNum)
```

STATIC METHOD

Trong C++, class method và static method là một nhưng trong Python thì phân ra làm hai loại phương thức khác nhau



STATIC METHOD TRONG C++

- Khai báo: thêm từ khóa **static** ở đằng trước
- Sử dụng: ClassName::method(args) (**KHÔNG** được dùng obj.method(args)
- Chỉ truy cập được vào các thành phần tĩnh khác của lớp (thuộc tính tĩnh và phương thức tĩnh)

STATIC METHOD TRONG C++

```
class Obj {
public:
    static int objNum;
    int attr;
    Obj() {
        ++objNum;
    }

    static void print() {
        cout << "objNum: " << Obj::objNum << endl;
//Đúng
        // cout << "attr" << attr << endl; //Sai
    }
};
```

CLASS METHOD TRONG PYTHON

- Là phương thức gắn với class chứ không phải object
- Nhận vào tham số **cls** trỏ đến trạng thái của class
- Chỉnh sửa được trạng thái của class
- Khai báo: thêm decorator @classmethod ở trên và nhận vào tham số đầu tiên là **cls**
- Sử dụng: ClassName.method(args)

STATIC METHOD TRONG PYTHON

- Là phương thức gắn với class, chứ không phải object
- Không nhận tham số ngầm định như cls hay self
- Không chỉnh sửa được trạng thái class
- Khai báo: thêm decorator @staticmethod ở trên
- Sử dụng: ClassName.method(args)

CLASS VÀ STATIC METHOD TRONG PYTHON

```python
class Math:
    __PI = 3.14

    @classmethod
    def getPI(cls):
        return cls.__PI

    @staticmethod
    def isBigger(a, b):
        return a > b
```

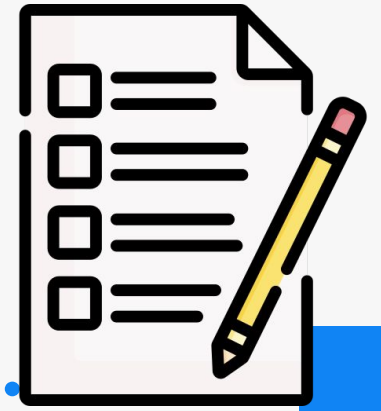
```python
print(Math.getPI())
print(Math.isBigger(4, 5))
```

```
3.14
False
```

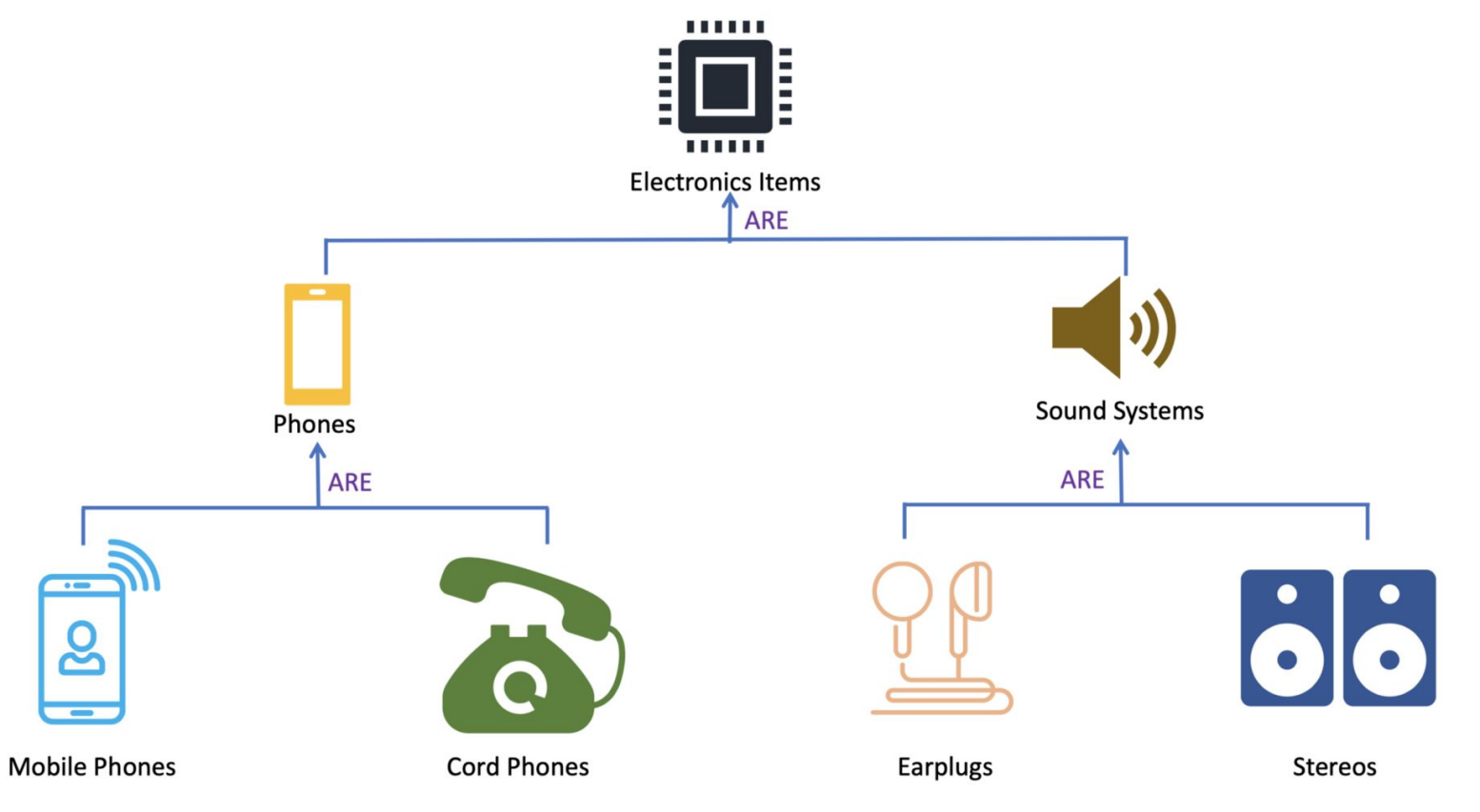
INHERITANCE

KHÁI NIỆM INHERITANCE

Inheritance (kế thừa) là một cơ chế mà một class (class con) kế thừa lại tất cả các thuộc tính và phương thức của một class khác (class cha)



VÍ DỤ VỀ KẾ THỪA



LỢI ÍCH CỦA KẾ THỪA

- Biểu diễn tốt mối quan hệ trong thế giới thực
- Khả năng tái sử dụng code. Thêm các tính năng vào trong lớp mà không cần sửa lại nó.
- Có tính chất bắc cầu. Lớp B kế thừa lớp A thì tất cả các lớp con của lớp B cũng kế thừa lớp A

KẾ THỪA TRONG C++

- Có 3 loại kế thừa là: public, protected, private. Chủ yếu chúng ta sử dụng kế thừa public
- Khai báo: class <ClassChild> : <loại kế thừa> <ClassFather>
- Ví dụ :

```
class Child : public Father {
private:
    // Thuộc tính riêng của lớp con

public:
    // Phương thức riêng của lớp con
};
```

BẢNG TẦM VỰC TRONG C++

Tầm vực	Kế thừa public	Kế thừa protected	Kế thừa private
public	public	protected	private
protected	protected	protected	private
private	Không truy xuất được	Không truy xuất được	Không truy xuất được

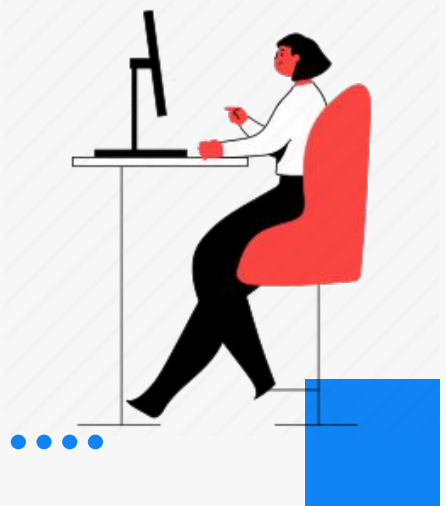
KẾ THỪA TRONG PYTHON

- Trong Python thì chỉ có kế thừa public
- Khai báo: class <ClassChild>(ClassFather):
- Ví dụ:

```
class Child(Father):
    def __init__(self):
        #thuộc tính của lớp con
        pass

    # các phương thức của lớp con
```

TÍNH CHẤT KẾ THỪA

- Trong kế thừa, lớp con kế thừa lại các phương thức và thuộc tính của lớp cha
- Có thể thêm thuộc tính và phương thức riêng cho class con
- Có thể sử dụng lại hoặc định nghĩa lại phương thức của lớp cha



ĐỊNH NGHĨA LẠI PHƯƠNG THỨC

- Phải khai báo lại hàm ở lớp con giống y hệt hàm của
- Có thể sử dụng hàm của lớp cha nếu cần. Trong C++ sử dụng cú pháp: **<ClassFather>::<method>**; trong Python sử dụng cú pháp: **super().<method>** hoặc **<classFather>.<method>(self, args)**

VÍ DỤ VỀ KẾ THỪA

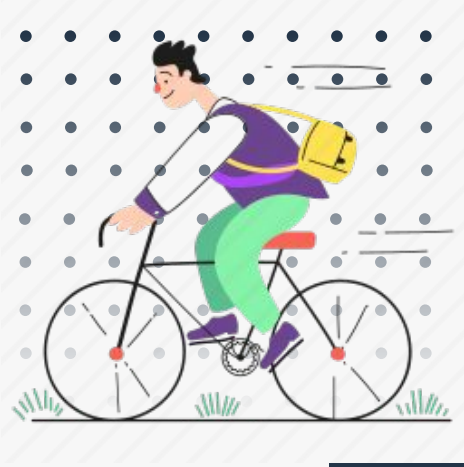
Person

- Attribute
  - name
  - birthday
- Method
  - getName()
  - getBirthday()
  - print()

Student

- Attribute
  - name
  - birthday
  - id, school
- Method
  - getName()
  - getBirthday()
  - getId()
  - getSchool()
  - print()

TRONG C++



```
class Person {
protected:
    string name;
    string birthday;
public:
    string getName();
    void setName(string);
    string getBirthday();
    void setBirthday(string);
    void print();
};
```

```
class Student : public Person {
private:
    string id;
    string school;
public:
    string getId();
    void setID(string);
    string getSchool();
    void setSchool(string);
    void print();
};
```

TRONG C++

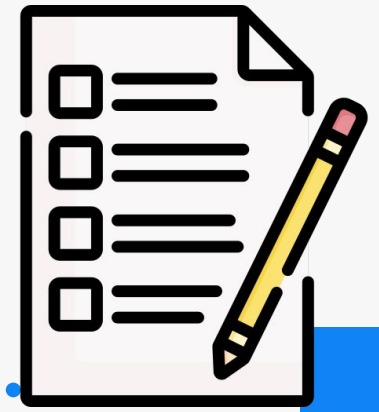
```cpp
string Person::getName() {return name;}
void Person::setName(string name) {
    this->name = name;
}
string Person::getBirthday() {
return birthday;
}
void Person::setBirthday(string birthday) {
    this->birthday = birthday;
}
void Person::print() {
    cout << "Name: " << name << endl;
    cout << "Birthday: " << birthday << endl;
}
```

```cpp
string Student::getId() {return id;}
void Student::setID(string id) {
    this->id = id;
}
string Student::getSchool() {
    return school;
}
void Student::setSchool(string school) {
    this->school = school;
}
void Student::print() {
    Person::print();
    cout << "ID: " << id << endl;
    cout << "School: " << school << endl;
}
```

TRONG C++

```
int main() {
    Person per1;
    Student std1;
    per1.setName("thi");
    per1.setBirthday("21/02/2001");
    std1.setName("thi");
    std1.setBirthday("21/02/2001");
    std1.setID("19120662");
    std1.setSchool("HCMUS");
    cout << "Person: " << endl;
    per1.print();
    cout << "Student: " << endl;
    std1.print();
    return 0;
}
```

```
Person:
Name: thi
Birthday: 21/02/2001
Student:
Name: thi
Birthday: 21/02/2001
ID: 19120662
School: HCMUS
```



TRONG PYTHON

```python
class Person:
    def __init__(self):
        self._name = ""
        self._birthday = ""
    def getName(self):
        return self._name
    def setName(self, name):
        self._name = name
    def getBirthday(self):
        return self._birthday
    def setBirthday(self, birthday):
        self._birthday = birthday
    def cout(self):
        print("Name:", self._name)
        print("Year:", self._birthday)
```

```python
class Student(Person):
    def __init__(self):
        super().__init__()
        self.__id = ""
        self.__school = ""
    def getId(self):
        return self.__id
    def setId(self, id):
        self.__id = id
    def getSchool(self):
        return self.__school
    def setSchool(self, school):
        self.__school = school
    def cout(self):
        super().cout() #hoặc Person.cout(self)
        print("ID:", self.__id)
        print("School:", self.__school)
```

TRONG PYTHON

```
per1 = Person()
std1 = Student()
per1.setName("thi")
per1.setBirthday("21/02/2001")
std1.setName("thi")
std1.setBirthday("21/02/2001")
std1.setId("19120662")
std1.setSchool("HCMUS")
print("Person:")
per1.cout()
print("Student:")
std1.cout()
```

```
Person:
Name: thi
Year: 21/02/2001
Student:
Name: thi
Year: 21/02/2001
ID: 19120662
School: HCMUS
```

CONSTRUCTOR TRONG KẾ THỪA C++

- Lớp con không kế thừa constructor của lớp cha
- Khi khai báo một đối tượng của lớp con thì
  - Constructor của lớp cha được gọi trước
  - Constructor của lớp con được gọi sau
- Nếu không chỉ định constructor nào của lớp cha thì sẽ gọi default constructor

CONSTRUCTOR TRONG KẾ THỪA C++

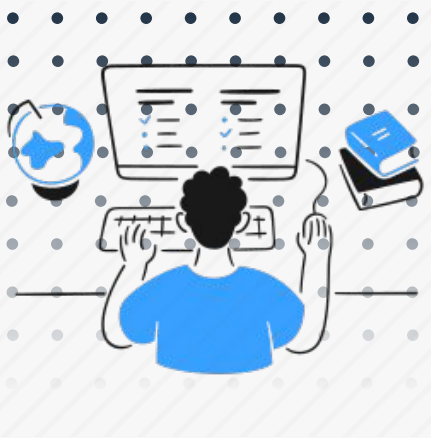
```
class Person {
protected:
    string name; string birthday;
public:
    Person(string, string);
    Person(); void print();
};
```

```
Person::Person() {
    name = ""; birthday = "";
}
Person::Person(string name, string birthday) {
    this->name = name;
    this->birthday = birthday;
}
```

```
class Student : public Person {
private:
    string id;string school;
public:
    Student(string, string, string, string);
    Student(); void print();
};
```

```
Student::Student(){//chỉ định default constructor
    id = ""; school = "";}
Student::Student
(string name, string bir, string id, string sch)
    : Person(name, bir) {//chỉ định constructor
    this->id = id; this->sch = school;
}
```

CONSTRUCTOR TRONG KẾ THỪA C++



```
int main() {
    Person per1;
    Student std1;
    Student std2("thi", "21/02/2001",
"19120662", "HCMUS");
    cout << "Student1:" << endl;
    std1.print();
    cout << "Student2:" << endl;
    std2.print();
    return 0;
}
```

```
Student1:
Name:
Birthday:
ID:
School:
Student2:
Name: thi
Birthday: 21/02/2001
ID: 19120662
School: HCMUS
```

CONSTRUCTOR TRONG KẾ THỪA PYTHON

- Không giống như C++, trong Python chỉ gọi constructor lớp con mà thôi
- Nếu không định nghĩa constructor cho lớp con thì lớp con sẽ kế thừa constructor của lớp cha

CONSTRUCTOR TRONG KẾ THỪA PYTHON

```
class Person:
    def __init__(self):
        print("constructor person")
class Student(Person):
    pass
std1 = Student()
```

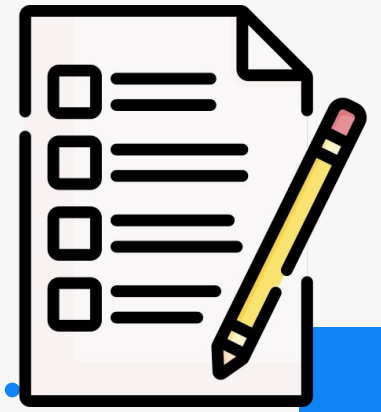
```
constructor person
```

```
class Person:
    def __init__(self):
        print("constructor person")
class Student(Person):
    def __init__(self):
        print("constructor student")
std1 = Student()
```

```
constructor student
```

DESTRUCTOR TRONG KẾ THỪA C++

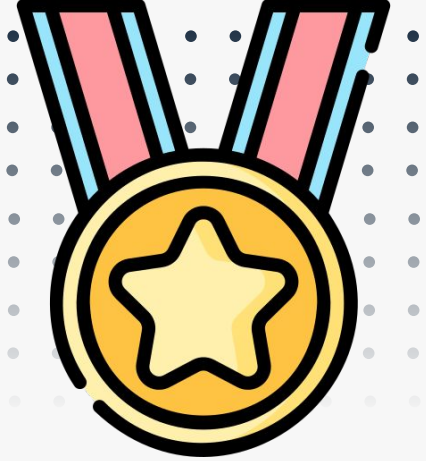
- Lớp con không kế thừa destructor của lớp cha
- Khi một đối tượng bị hủy
  - Gọi destructor lớp con trước
  - Gọi destructor lớp cha sau
- Kiểu dữ liệu con trỏ
  - Của lớp cha: lớp cha giải quyết
  - Của lớp con: lớp con giải quyết



DESTRUCTOR TRONG KẾ THỪA PYTHON

Destructor trong python tương tự với constructor

- Khi đối tượng bị hủy chỉ gọi destructor lớp con
- Nếu không định nghĩa destructor thì sẽ kế thừa của lớp cha



POLYMORPHISM

KHÁI NIỆM ĐA HÌNH

Đa hình (polymorphism) được hiểu là một thông điệp có thể được hiển thị ở nhiều dạng khác nhau

VD: một người lúc ở trường là sinh viên, lúc ở công ty là một lập trình viên, lúc ở nhà là một người con

Đa hình giúp cho việc viết chương trình ngắn gọn hơn và dễ mở rộng

ĐA HÌNH TRONG C++

Trong C++ cho phép chúng ta khai báo đối tượng lớp con là kiểu dữ liệu của lớp cha.

Ví dụ: Person* std1 = new Student();

Tuy nhiên khi sử dụng phương thức thì lại gọi phương thức của lớp cha => không thể hiện được tính đa hình

=> Giải pháp: hàm ảo

HÀM ẢO TRONG C++

- Hàm ảo:
  - Là một hàm của lớp
  - Mang tính chất ảo
  - Khai báo: thêm từ khóa **virtual** trước khai báo hàm
- VD: virtual void print();

ĐA HÌNH TRONG C++

Vậy nếu muốn sử dụng đa hình, phải

- Khai báo kế thừa
- Khai báo hàm ảo của lớp cha
- Sử dụng con trỏ

ĐA HÌNH TRONG C++

```
class Person {
protected:
    string name;
    string birthday;
public:
    Person(string, string);
    virtual void print(); //
sử dụng hàm ảo
};
```

```
class Teacher : public Person {
private:
    string major;
public:
    Teacher(string, string, string);
    void print();
};
```

```
class Teacher : public Person {
private:
    string major;
public:
    Teacher(string, string, string);
    void print();
};
```

ĐA HÌNH TRONG C++

```
int main() {
    vector<Person*> membersList;
    membersList.push_back(new Teacher("A", "19/09/1980", "SE"));
    membersList.push_back(new Teacher("B", "20/05/1970", "CS"));
    membersList.push_back(new Student("C", "09/07/2001", "19120060", "HCMUS"));
    membersList.push_back(new Student("D", "15/08/2001", "19120070", "HCMUS"));
    for (int i = 0; i < membersList.size(); i++) {
        cout << "------------" << endl;
        cout << "Member " << i + 1 << endl;
        membersList[i]->print();
    }
    for (Person* member : membersList) {
        delete member;
    }
    return 0;
}
```

ĐA HÌNH TRONG C++

```
------------
Member 1
Name: A
Birthday: 19/09/1980
Major: SE
------------
Member 2
Name: B
Birthday: 20/05/1970
Major: CS
```

```
------------
Member 3
Name: C
Birthday: 09/07/2001
ID: 19120060
School: HCMUS
------------
Member 4
Name: D
Birthday: 15/08/2001
ID: 19120070
School: HCMUS
```

⇨ Đã trỏ đúng hàm của lớp con

ĐA HÌNH TRONG PYTHON

- Trong Python, chúng ta không cần khai báo con trỏ hay hàm ảo gì hết mà vẫn có thể đa hình được, ta chỉ cần định nghĩa lại hàm ở lớp con
- Một điều lưu ý là trong python không có nạp chồng hàm, nghĩa là không có 2 hàm trùng tên trong 1 class

ĐA HÌNH TRONG PYTHON

```python
class Person:
    def __init__(self, name, birthday):
        self._name = name
        self._birthday = birthday
    def cout(self):
        print("Name:", self._name)
        print("Birthday:", self._birthday)
```

```python
class Teacher(Person):
    def __init__(self, name, birthday, major):
        super().__init__(name, birthday)
        self.__major = major
    def cout(self):
        super().cout()
        print("Major:", self.__major)
```

```python
class Student(Person):
    def __init__(self, name, birthday, id,
school):
        super().__init__(name, birthday)
        self.__id = id
        self.__school = school
    def cout(self):
        super().cout()
        print("ID:", self.__id)
        print("School", self.__school)
```

ĐA HÌNH TRONG PYTHON

```
members = []
members.append(Teacher("A", "19/09/1980", "SE"))
members.append(Teacher("B", "20/05/1970", "CS"))
members.append(Student("C", "09/07/2001", "19120060",
"HCMUS"))

members.append(Student("D", "15/08/2001", "19120070",
"HCMUS"))

for member in members:
    print("---------------")
    member.cout()
```

```
---------------
Name: A
Birthday:
19/09/1980
Major: SE
---------------
Name: B
Birthday:
20/05/1970
Major: CS
```

```
---------------
Name: C
Birthday:
09/07/2001
ID: 19120060
School HCMUS
---------------
Name: D
Birthday:
15/08/2001
ID: 19120070
School HCMUS
```

HÀM THUẦN ẢO TRONG C++

- Trong class cha đôi lúc mình sẽ không biết nên định nghĩa hàm như thế nào => giải pháp: hàm thuần ảo
- Khai báo: virtual <method> = 0;
- Khi đó lớp có phương thức thuần ảo được gọi là lớp trừu tượng

LỚP TRỪU TƯỢNG TRONG C++

- Là lớp chứa ít nhất một hàm thuần ảo
- Không thể tạo đối tượng từ lớp trừu tượng
- Chỉ dùng để kế thừa
- Lớp con kế thừa lớp trừu tượng
  - Cần cài đặt lại tất cả các hàm thuần ảo
  - Nếu không sẽ trở thành lớp trừu tượng

LỚP TRỪU TƯỢNG TRONG C++

```cpp
class Vehicle {
public:
    virtual void move() = 0;
};
```

```cpp
class Bike: public Vehicle {
    void move() {
        cout << "Cycling" << endl;
    }
};
```

```cpp
class Car : public Vehicle {
public:
    void move() {
        cout << "Driving" << endl;
    }
};
```

```cpp
int main() {
    // Vehicle* vehicle1 = new Vehicle; Sai
    Vehicle* vehi1 = new Car;
    Vehicle* vehi2 = new Bike;
    vehi1->move(); vehi2->move();
    delete vehi1; delete vehi2;
    return 0;
}
```

LỚP TRỪU TƯỢNG TRONG PYTHON

- Mặc định trong Python không cung cấp lớp trừu tượng cho chúng ta sử dụng.
- Nhưng chúng ta cũng có thể import module abstract class base và abstract method để làm điều đó
- Cú pháp: from abc import ABC, abstractmethod

LỚP TRỪU TƯỢNG TRONG PYTHON

```python
class Vehicle(ABC):
    @abstractclassmethod
    def move(self):
        pass
```

```python
class Bike(Vehicle):
    def move(self):
        print("Cycling")
```

```python
class Car(Vehicle):
    def move(self):
        print("Driving")
```

```python
# vehicle1 = Vehicle() Sai
vehi1 = Car()
vehi2 = Bike()
vehi1.move()
vehi2.move()
```

TÀI LIỆU THAM KHẢO

https://www.geeksforgeeks.org/python-programming-language/?ref=ghm

https://www.geeksforgeeks.org/c-plus-plus/?ref=shm

https://www.tutorialsteacher.com/python/

https://openplanning.net/11417/python-inheritance-polymorphism

https://www.w3schools.com/python/python_inheritance.asp

https://drive.google.com/drive/folders/1hPI1kqHwFi83wDMJp988KMFoA3UXHWd6

VIDEO BÀI GIẢNG

https://drive.google.com/file/d/1USC8vBQbKnJvpkc39ZO7kGHOClxYTsrJ/view?usp=sharing