CODELEARN

54%

Object Oriented-Programming in C++

The course helps programmers learn programming techniques where all logic and practical requirements are built around objects.

cpp oop highlevelcoding programming c++

Free

Study now →

40 hours | 80 lectures | 73.991 students

Certificate | 4.4 (251 reviews)

Owner tuanlq7

73991 Learner

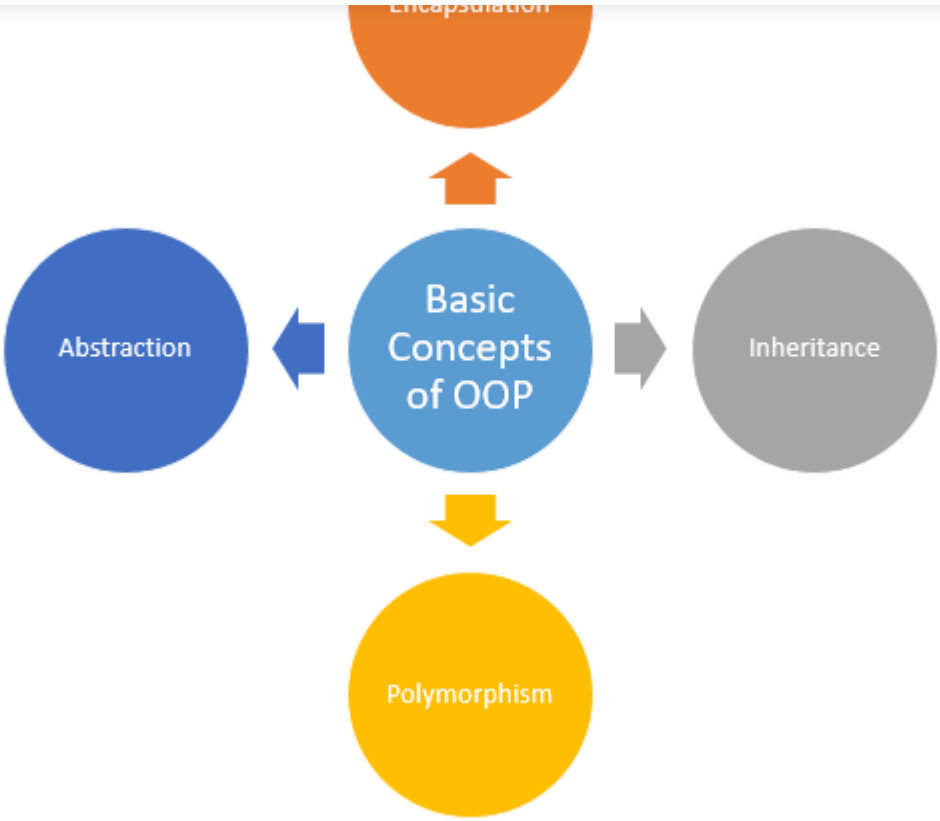
4.4

Introduce

Khi nhắc tới lập trình hướng đối tượng chắc bạn sẽ nghĩ ngay tới 4 tính chất là tính đóng gói, tính kế thừa, tính đa hình và tính trừu tượng. Thực chất thì 4 tính chất này chỉ giống như các nguyên liệu để xây dựng chương trình theo phương pháp hướng đối tượng, quan trọng nhất vẫn là cách mà bạn sử dụng các nguyên liệu này để xây dựng chương trình như thế nào.

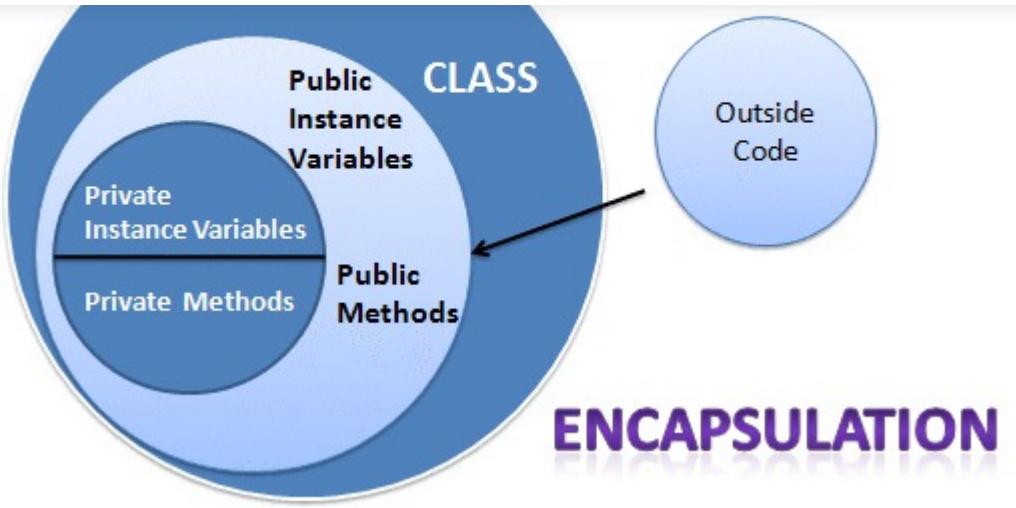
Vậy lập trình hướng đối tượng là gì?

Lập trình hướng đối tượng được hiểu đơn giản là một phương pháp để giải quyết bài toán lập trình mà khi áp dụng thì code sẽ trở nên dễ phát triển và dễ bảo trì hơn. Phương pháp này sẽ chia nhỏ chương trình thành các đối tượng và các mối quan hệ, mỗi đối tượng sẽ có các thuộc tính (dữ liệu) và hành vi (phương thức). Để có thể lập trình và thiết kế chương trình theo phương pháp này thì chắc chắn bạn cần hiểu rõ về 4 tính chất là là tính đóng gói, tính kế thừa, tính đa hình và tính trừu tượng.



1. Tính đóng gói (Encapsulation)

Đây là kỹ thuật giúp bạn che giấu đi những thông tin bên trong đối tượng bằng cách sử dụng phạm vi truy cập private cho các thuộc tính, muốn giao tiếp hay lấy ra các thông tin của đối tượng thì phải thông qua các phương thức public, từ đó sẽ hạn chế được các lỗi khi phát triển chương trình. Tính chất này cũng giống với trong thực tế, bạn không thể thấy được các thuộc tính thực của một người (tính cách, sở thích, các thông tin riêng tư khác, ...), những thứ mà bạn biết đều là thông qua các hành động của người đó. Ví dụ người đó nói cho bạn biết về sở thích, tuổi, ... nhưng các thông tin này chưa chắc đã thực sự là thuộc tính thật của người đó (giống với việc các getter không trả về giá trị thực của thuộc tính mà trả về một giá trị khác).



Các lợi ích chính mà tính đóng gói đem lại:

- Hạn chế được các truy xuất không hợp lệ tới các thuộc tính của đối tượng.
- Giúp cho trạng thái của các đối tượng luôn đúng. Ví dụ nếu thuộc tính `gpa` của lớp `Student` là `public` thì sẽ rất khó kiểm soát được giá trị, bạn có thể thay đổi `gpa` thành bất kỳ giá trị nào. Ngược lại, nếu bạn để thuộc tính `gpa` là `private` và cung cấp hàm `setGpa()` giống như sau:

```
void setGpa(double gpa) {
        if (gpa >= 0 && gpa <= 4) {
            this->gpa = gpa;
        } else {
            cout << "gpa is invalid";
        }
    }
```

thì lúc này giá trị của thuộc tính `gpa` sẽ luôn được đảm bảo là không âm và nhỏ hơn hoặc bằng `4` (do muốn thay đổi `gpa` thì phải thông qua hàm `setGpa()`).

- Giúp ẩn đi những thông tin không cần thiết về đối tượng.
- Cho phép bạn thay đổi cấu trúc bên trong lớp mà không ảnh hưởng tới lớp khác. Ví dụ ban đầu bạn thiết kế lớp `Student` giống như sau:

```
class Student {
private:
    string firstName;
```

```
        ...
    }
    string getFullName() {
        return firstName + lastName;
    }
};
```

Sau này nếu bạn muốn gộp 2 thuộc tính `firstName` và `lastName` thành `fullName` thì lớp `Student` sẽ giống như sau:

```
class Student {
private:
    string fullName;
public:
    Student() {
        ...
    }
    string getFullName() {
        return fullName;
    }
};
```

Lúc này cấu trúc lớp `Student` đã bị thay đổi nhưng các đối tượng sử dụng lớp này vẫn không cần phải thay đổi do các đối tượng này chỉ quan tâm tới phương thức `getFullName()`. Nếu không có phương thức này thì bạn phải sửa tất cả những chỗ sử dụng thuộc tính `firstName` và `lastName` của lớp `Student`.

Lưu ý: hãy luôn nhớ rằng mục đích chính của tính đóng gói là để hạn chế các lỗi khi phát triển chương trình chứ không phải là bảo mật hay che giấu thông tin.

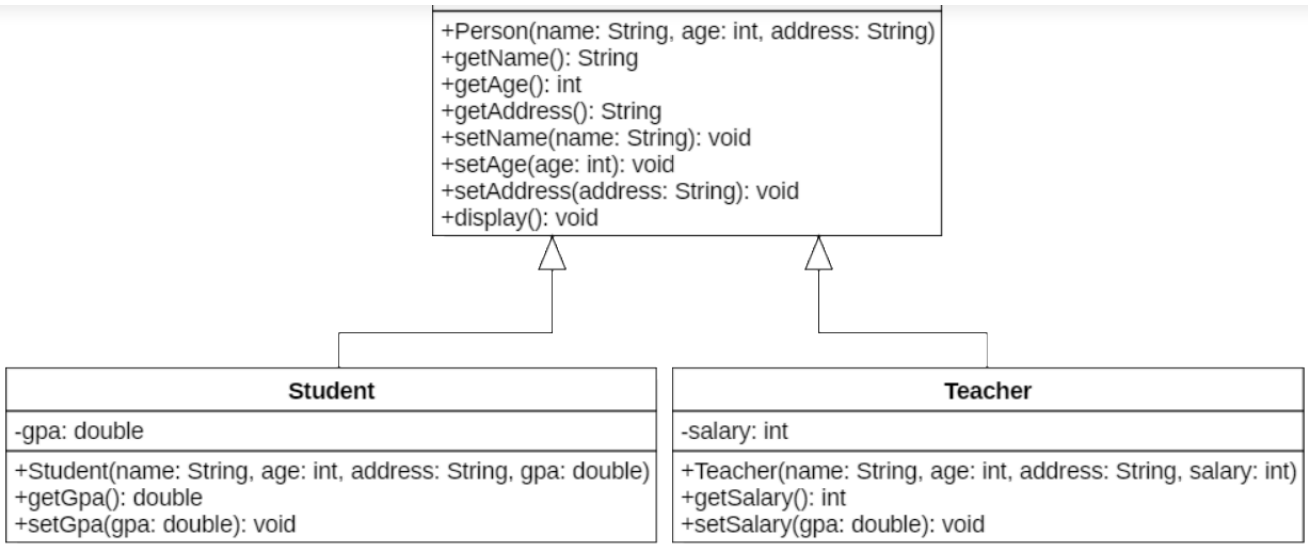
2. Tính kế thừa (Inheritance)

Khi lập trình chắc chắn sẽ có những trường hợp mà các đối tượng có chung một số thuộc tính và phương thức. Ví dụ như khi bạn viết chương trình lưu thông tin về các học sinh và giáo viên. Với học sinh thì cần lưu thông tin về tên, tuổi, địa chỉ, điểm và với giáo viên thì cần lưu thông tin về tên, tuổi, địa chỉ, tiền lương => lúc này code sẽ bị trùng lặp khá nhiều (từ các thuộc tính cho tới các setter, getter, ...) và nó vi phạm một trong những nguyên tắc cơ bản nhất khi lập trình là DRY (Don't Repeat Yourself - đừng bao giờ lặp lại code). Để thấy rõ hơn thì bạn hãy xem sơ đồ lớp sau:

Student
-name: String -age: int -address: String -gpa: double
+Student(name: String, age: int, address: String, gpa: double) +getName(): String +getAge(): int +getAddress(): String +getGpa(): double +setName(name: String): void +setAge(age: int): void +setAddress(address: String): void +setGpa(gpa: double): void +display(): void

Teacher
-name: String -age: int -address: String -salary: int
+Teacher(name: String, age: int, address: String, salary: int) +getName(): String +getAge(): int +getAddress(): String +getSalary(): double +setName(name: String): void +setAge(age: int): void +setAddress(address: String): void +setSalary(gpa: double): void +display(): void

Với kế thừa thì vấn đề này sẽ được giải quyết, kế thừa trong lập trình hướng đối tượng chính là thừa hưởng lại những thuộc tính và phương thức của một lớp. Có nghĩa là nếu lớp A kế thừa lớp B thì lớp A sẽ có những thuộc tính và phương thức của lớp B. Do đó, ở sơ đồ trên bạn có thể tách các thuộc tính và phương thức trùng nhau ra một lớp mới là `Person` và cho lớp `Student` và `Teacher` kế thừa lớp này giống như sau:



Có thể thấy với sơ đồ này thì lớp `Student` và `Teacher` sẽ được thừa hưởng lại các thuộc tính chung từ lớp `Person` và code sẽ không còn bị trùng lặp. Đó chính là lợi ích của tính kế thừa.

3. Tính đa hình (Polymorphism)

Như bạn đã biết, lập trình hướng đối tượng là phương pháp tư duy và giải quyết bài toán lập trình theo hướng thực tế. Do đó, các tính chất của nó cũng sẽ gắn liền với thực tế nên trước hết bạn cần hiểu về tính đa hình trong thực tế. Đa hình được hiểu là trong từng hoàn cảnh, từng trường hợp khác nhau thì các đối tượng sẽ đóng các vai trò khác nhau. Ví dụ, cùng là một người nhưng khi ở công ty thì có vai trò là nhân viên, khi đi siêu thị thì có vai trò là khách hàng, hay khi ở trường thì lại có vai trò là học sinh, ... => cùng là một người nhưng có nhiều vai trò khác nhau nên đây chính là đa hình trong thực tế.
Trong lập trình thì khi một đối tượng hay một phương thức có nhiều hơn một hình thái thì đó chính là đa hình. Tính đa hình được thể hiện dưới 3 hình thức:

3.1. Đa hình với nạp chồng phương thức

Ví dụ: phương thức cộng sẽ có các hình thái là cộng 2 số nguyên, cộng 2 số thực, cộng 3 số nguyên, v/v. Có thể thấy cùng là phương thức cộng nhưng lại có nhiều hình thái khác nhau nên đây chính là biểu hiện của tính đa hình. Ví dụ về đa hình với nạp chồng phương thức:

```
#include <iostream>

using namespace std;

class Calculator {
```

```
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
};

int main() {
    Calculator c;
    cout << c.add(1, 2) << endl;
    cout << c.add(3.3, 4.2) << endl;
    cout << c.add(1, 2, 3) << endl;
    return 0;
}
```

Kết quả khi chạy chương trình:

```
3
7.5
6
```

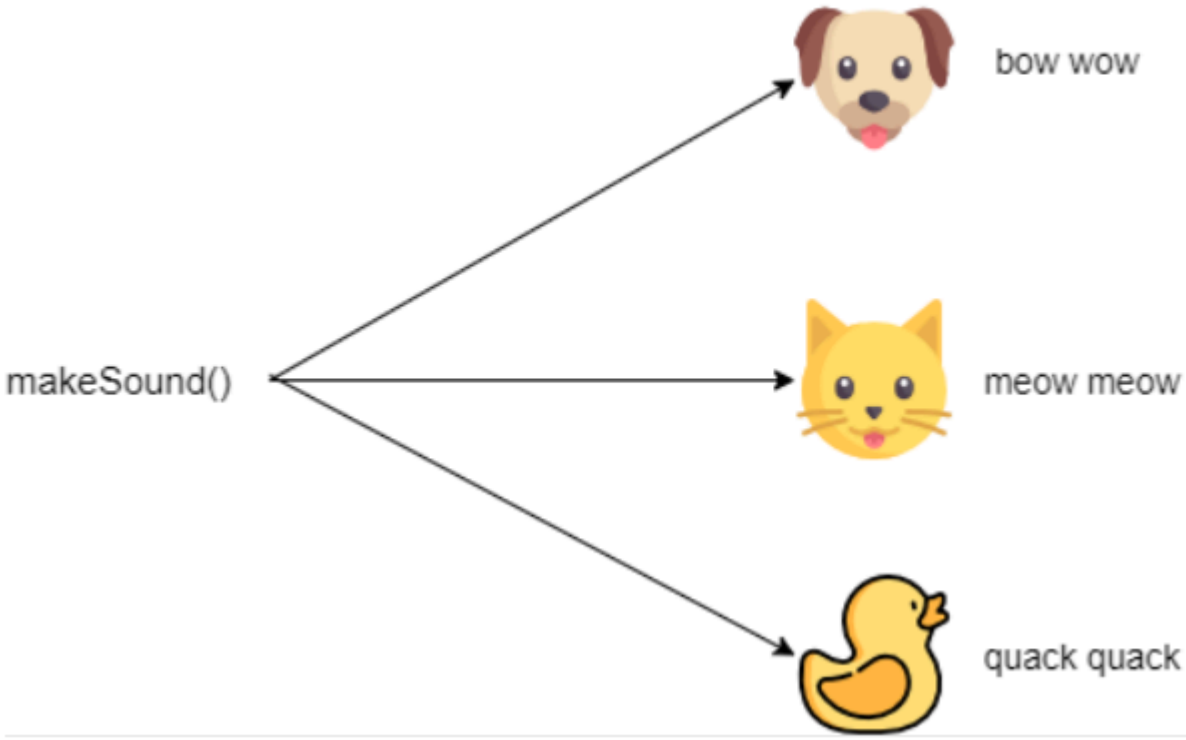
3.2. Đa hình với ghi đè phương thức

Ví dụ phương thức `getSalary()` dùng để tính lương sẽ có các hình thái là tính lương cho quản lý, tính lương cho nhân viên:

```
class Employee {
private:
    string name;
    int salary;

public:
    Employee(string name, int salary) {
        this->name = name;
        this->salary = salary;
    }
```

```
    void setName(string name) {
        this->name = name;
    }

    int getSalary() {
        return salary;
    }

    void setSalary(int salary) {
        this->salary = salary;
    }

    void display() {
        cout << "Name: " << getName() << endl;
        cout << "Salary: " << getSalary() << endl;
    }
};

class Manager : Employee {
private:
    int bonus;
public:
    Manager(string name, int salary, int bonus) : Employee(name, salary)
        this->bonus = bonus;
    }

    int getBonus() {
        return bonus;
    }

    void setBonus(int bonus) {
        this->bonus = bonus;
    }

    int getSalary() {
        return Employee::getSalary() + bonus;
    }
};
```

3.3 Đa hình thông qua các đối tượng đa hình (polymorphic objects)



Biến thuộc lớp cha có thể tham chiếu tới đối tượng của các lớp con, vậy biến thuộc lớp cha cũng có nhiều hình thái nên đây cũng là đa hình. Ví dụ:

```
#include <iostream>

using namespace std;

class Animal {
public:
    virtual void sound() {
        cout << "some sound" << endl;
    }
};

class Dog : public Animal {
public:
    void sound() {
        cout << "bow wow" << endl;
    }
};
```

```
        cout << "meow meow" << endl;
    }
};

class Duck : public Animal {
public:
    void sound() {
        cout << "quack quack" << endl;
    }
};

int main() {
    Animal* animal = new Animal();
    animal->sound();
    animal = new Dog();
    animal->sound();
    animal = new Duck();
    animal->sound();
    animal = new Cat();
    animal->sound();
    return 0;
}
```

Kết quả khi chạy chương trình:

```
some sound
bow wow
quack quack
meow meow
```

4.Tính trừu tượng (Abstraction)

Trừu tượng là tính chất mà đơn giản hóa đi những thông tin bên trong đối tượng, nó cho phép ta giao tiếp với các thành phần của đối tượng mà không cần phải biết về cách mà các thành phần này được xây dựng (chính xác hơn là không cần biết các thành phần được code như thế nào mà chỉ cần biết các thành phần này được dùng để làm gì). Trước hết, hãy cùng xem một ví dụ thực tế về tính trừu tượng:

Khi bạn đi rút tiền ở cây ATM thì bạn không cần quan tâm tới cách mà cây ATM hoạt

tiền, trừ tiền trong tài khoản, gửi dữ liệu về máy chủ đã được ẩn đi. Cái mà bạn nhìn thấy về đối tượng cây ATM chính là rút tiền => cây ATM đã ẩn đi những chi tiết không cần thiết và đó chính là tính trừu tượng.

Tương tự trong lập trình cũng vậy, khi gọi tới các phương thức của một đối tượng thì bạn chỉ cần quan tâm tới phương thức đó được dùng để làm gì chứ không cần quan tâm tới phương thức đó được code như thế nào. Tính chất này rất có ích khi làm việc nhóm, bạn chỉ cần quan tâm tới chức năng của các phương thức mà đồng nghiệp code chứ không cần biết nó được cài đặt như thế nào. Để thực hiện tính trừu tượng thì bạn có thể sử dụng các abstract class và interface vì nó chỉ chứa phần khai báo chứ không có phần cài đặt (ở một số ngôn ngữ không có khái niệm về interface nên nếu bạn chưa biết về interface thì có thể hiểu interface chính là abstract class với các phương thức đều là trừu tượng).

Trong thực tế, khi đi làm bạn sẽ sử dụng tới interface rất nhiều, với mỗi lớp bạn thường tạo ra 1 interface riêng để thể hiện các tính năng của lớp đó và sử dụng interface này để giao tiếp với đối tượng. Ví dụ lớp Customer sẽ có interface ICustomer, các đối tượng khác muốn giao tiếp với lớp Customer thì đều phải thông qua interface trên..

Kết luận

Lập trình hướng đối tượng không chỉ gói gọn trong 4 tính chất trên, để viết được một chương trình tốt thì bạn còn phải biết thêm rất nhiều nguyên liệu khác như OOP design, Software Architecture, ... trong bài này mình chỉ tóm tắt về lập trình hướng đối tượng và 4 tính chất chính, nếu muốn học chi tiết hơn thì bạn có thể tham khảo thêm tại khóa học C++ OOP và Java OOP trên hệ thống. Còn về các chủ đề khác trong OOP thì mình sẽ giới thiệu trong các bài viết tiếp theo.

What you will learn

- ✓ Object-Oriented-Programming (Object-Oriented-Programming) is an object-based programming method to find out the nature of the problem. This cour...

Course content

8 sections • 80 lectures • 40h total length