



**TRAINING CUỐI KÌ 2 - K17**

# **Lập trình hướng đối tượng**

**Trainer**

**Đặng Quang Khánh Linh  
Nguyễn Mai Thanh Thảo**



# **NỘI DUNG TÌM HIỂU**

- |                                     |                                |
|-------------------------------------|--------------------------------|
| <b>1. Các phương pháp lập trình</b> | <b>6. Con trỏ this</b>         |
| <b>2. Lớp và đối tượng</b>          | <b>7. Thành phần static</b>    |
| <b>3. Thuộc tính và phương thức</b> | <b>8. Phương thức khởi tạo</b> |
| <b>4. Phạm vi truy cập</b>          | <b>9. Phương thức phá hủy</b>  |
| <b>5. Cặp hàm get/set</b>           | <b>10. Friend</b>              |



# **1. Các phương pháp lập trình**

**1. Lập trình phi cấu trúc**

**2. Lập trình có cấu trúc**

**3. Lập trình hướng đối tượng**





## 1. Lập trình phi cấu trúc

Sử dụng trong các ngôn ngữ lập trình bậc thấp như Assembly, MIPS,...

Lập trình phi cấu trúc chỉ là lập trình để giải quyết vấn đề. Nó không tạo ra một cấu trúc logic.

## 2. Lập trình có cấu trúc

Sử dụng ở một số ngôn ngữ lập trình như C, Pascal,...

Chương trình được chia thành các hàm thực hiện các chức năng khác nhau

Chương trình = Cấu trúc dữ liệu + Giải thuật



### 3. Lập trình hướng đối tượng

Được xây dựng trên nền tảng của Lập trình có cấu trúc (hướng chức năng) và sự trừu tượng hóa dữ liệu.

Sử dụng ở một số ngôn ngữ lập trình như C#, Java, ..

Là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng chương trình.

Một chương trình hướng đối tượng sẽ bao gồm các tập đối tượng có quan hệ với nhau.



## **4 đặc tính cơ bản của lập trình hướng đối tượng**

- 1. Tính đóng gói (Encapsulation)**
- 2. Tính trừu tượng (Abstraction)**
- 3. Tính kế thừa (Inheritance)**
- 4. Tính đa hình (Polymorphism)**

## 1. Tính đóng gói (Encapsulation)

Nhóm những gì có liên quan với nhau vào làm một để có thể gọi đến bằng một tên và để che dấu một số thông tin và chi tiết cài đặt nội bộ.

## 2. Tính trừu tượng (Abstraction)

Tính trừu tượng giúp loại bỏ những thứ phức tạp, không cần thiết của đối tượng và chỉ tập trung vào những gì cốt lõi, quan trọng.



### 3. Tính kế thừa (Inheritance)

Tính kế thừa cho phép xây dựng một lớp mới (lớp Con), kế thừa và tái sử dụng các thuộc tính, phương thức dựa trên lớp cũ (lớp Cha) đã có trước đó.

### 4. Tính đa hình (Polymorphism)

Tính đa hình cho phép các đối tượng khác nhau thực thi chức năng giống nhau theo những cách khác nhau.



## 2. Lớp và đối tượng

### Lớp đối tượng là gì?

Các đối tượng có các đặc tính tương tự nhau được gom chung thành lớp đối tượng, là một mô tả trừu tượng của các nhóm đối tượng cùng bản chất

Lớp trong C++ thực chất là một kiểu dữ liệu do người dùng định nghĩa

Cú pháp khai báo lớp trong C++:

```
class <Tên lớp> {...};
```



## 2. Lớp và đối tượng

### Đối tượng là gì?

Đối tượng là một thể hiện cụ thể của một lớp

Trong C++, đối tượng thuộc một lớp thực chất là một biến mang

kiểu dữ liệu lớp đó

Cách khai báo đối tượng thuộc lớp:

<Tên lớp> <Tên đối tượng>;



### 3. Thuộc tính và phương thức

**Thuộc tính:** Là thành phần của đối tượng, có giá trị nhất định cho mỗi đối tượng trong hệ thống

**Thao tác:** Thể hiện hành vi của một đối tượng tác động qua lại với các đối tượng khác hoặc với chính nó.

Cách gọi **thuộc tính/phương thức** của đối tượng thuộc lớp:

<tên đối tượng>.<tên thuộc tính/phương thức>...;

<tên con trỏ đối tượng>-><tên thuộc tính/phương thức>...;



## 4. Phạm vi truy cập

**Private:** Các thành phần được khai báo **private** chỉ được truy xuất trong nội bộ lớp.

**Public:** Các thành phần được khai báo **public** được truy xuất ở mọi nơi trong chương trình.

**Protected:** Các thành phần được khai báo **protected** được truy xuất trong nội bộ lớp và lớp con.

Phạm vi truy cập mặc định trong C++ là **private**.



## 5. Phương thức truy vấn

Đối với các truy vấn đơn giản, quy ước đặt tên phương thức như sau:

Tiền tố “**get**”, tiếp theo là tên của thành viên cần truy vấn.

VD:

```
int getX(); //Truy xuất X
```

```
int getSize(); //Truy xuất Size
```

Các loại truy vấn khác nên có tên có tính mô tả.

Truy vấn điều kiện nên có tiền tố “**is**”.



## 5. Phương thức truy vấn

Thường để thay đổi trạng thái của đối tượng bằng cách sửa đổi một hoặc nhiều thành viên dữ liệu của đối tượng đó. Dạng đơn giản nhất là gán một giá trị nào đó cho một thành viên dữ liệu.

Đối với dạng cập nhật đơn giản, quy ước đặt tên như sau: Dùng tiền tố “**set**” kèm theo tên thành viên cần sửa.

VD: void setX(int); **//cập nhật X**



## 6. Con trỏ this

Từ khóa **this** trong định nghĩa của các hàm thành phần lớp dùng để xác định địa chỉ của đối tượng dùng làm tham số ngầm định cho hàm thành phần **con trỏ this** tham chiếu đến đối tượng đang gọi hàm thành phần



## 7. Thành phần static

Thành phần **static** trong C++ là dữ liệu của lớp không phải là dữ liệu của đối tượng.

Thành phần **static** xuất hiện trước khi khởi tạo đối tượng của lớp, và nó chỉ tồn tại duy nhất.

Một hàm khai báo là **static** không được phép truy cập những thành phần **non-static** của lớp.





## 8. Phương thức khởi tạo

**Phương thức khởi tạo** là một phương thức đặc biệt được tự động gọi khi một đối tượng được khởi tạo, không có kiểu dữ liệu trả về, tên của phương thức **trùng với tên của lớp**.

**Cú pháp:** <Tên lớp>(Danh sách tham số nếu có);  
Trình biên dịch sẽ tự động trang bị một phương thức khởi tạo mặc định nếu người dùng không khai báo bất kì một phương thức khởi tạo nào khác.

## 9. Phương thức hủy

Phương thức hủy là một phương thức đặc biệt được **tự động gọi** khi một đối tượng không còn được sử dụng, không có kiểu dữ liệu trả về, tên của phương thức trùng với tên của lớp, không có tham số truyền vào.

Mỗi lớp có **duy nhất** một phương thức hủy.

**Cú pháp:** ~<Tên lớp>();



## 10. Thành phần friend

Thành phần **friend** là một thành phần không thuộc lớp nhưng nó được quyền truy xuất tất cả các thành phần **private** của lớp.

Hãy lưu ý khi sử dụng hàm **friend** vì nó sẽ phá hủy tính đóng gói và che giấu dữ liệu trong lập trình hướng đối tượng



# Overloading hàm và toán tử

1. **Overload là gì?**
2. **Operator là gì?**
3. **Một số lưu ý khi overload toán tử**
4. **Một số cú pháp overload toán tử cần phải nhớ**





# 1. Overload là gì?

**Overload ( nạp chồng):** là việc tạo ra nhiều phương thức có cùng tên thực hiện chức năng tương tự nhau nhưng khác nhau về tham số đầu vào (kiểu dữ liệu hoặc số lượng tham số đầu vào).

**Overload** là một yếu tố tạo nên tính đa hình trong OOP.

**Overload** thực hiện đa hình trong **compile time**.



## 2. Operator là gì?

Các toán tử cho phép ta sử dụng cú pháp toán học đối với các kiểu dữ liệu của C++ thay vì gọi hàm.

Có các loại toán tử một ngôi như ++, --, &, \*, ... và toán tử hai ngôi như +, -, \*, /, ...

Sự cài đặt phép toán chỉ cho phép tạo ra phép toán mới trên cơ sở kí hiệu phép toán đã có, không được quyền cài đặt các phép toán mới.

Một toán tử có thể dùng cho nhiều kiểu dữ liệu.

## Các loại toán tử

### 1) Toán tử số học:

`+` `-` `*` `/` `%` `++` `--`

### 2) Toán tử quan hệ:

`<` `<=` `>` `>=` `==` `!=`

### 3) Toán tử logic:

`&&` `||` `!`

### 4) Toán tử gán:

`=` `+=` `-=` `*=` `/=` `%=`

### 5) Toán tử thao tác trên bit:

`&` `|` `^(XOR)` `~(đảo bit)` `>>`  
`<<` và `&=` `|=` `^=` `>>=` `<<=`

### 6) Toán tử làm việc với con trỏ: `&` `*`

### 7) Toán tử khác:

`()` //gọi hàm

`[]` //truy xuất phần tử mảng

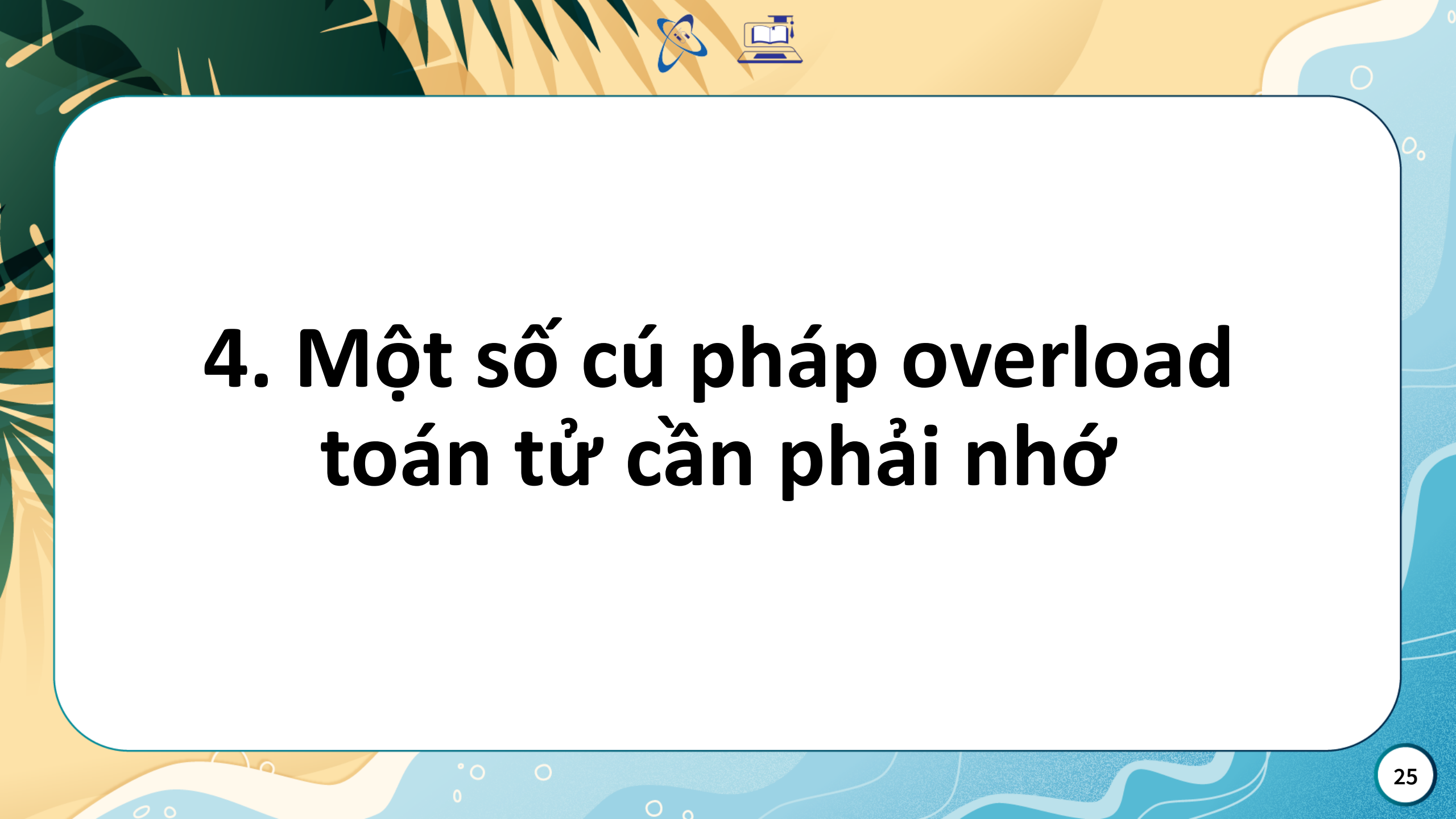
### 3. Một số lưu ý khi overload toán tử

Các hàm toán tử sau phải là hàm thành phần của lớp, khi đó đối tượng gọi hàm sẽ là toán hạng thứ nhất của toán tử.

Nếu có sử dụng các toán tử gán ( $+=$ ,  $-=$ ,  $*=$ , ...) thì chúng phải được định nghĩa (là các hàm toán tử), cho dù đã định nghĩa toán tử gán ( $=$ ) và các toán tử số học ( $+$ ,  $-$ ,  $*$ , ...) cho lớp.

Cài đặt hàm toán tử đúng ý nghĩa của toán tử được overload.





## **4. Một số cú pháp overload toán tử cần phải nhớ**



## Các toán tử 2 ngôi

Thường đi kèm friend và một bộ constructor để thực hiện phép chuyển kiểu

Ví dụ: Các phép toán +, -, \*, /:

```
PhanSo(int Tu = 0, int Mau = 0);
```

```
friend PhanSo operator+(PhanSo a, PhanSo b);
```

Khi đó các phép toán `PhanSo + PhanSo`, `PhanSo + int` và `int + PhanSo` sẽ được thực hiện mà không cần cài đặt cả ba phiên bản.

# Toán tử nhập - xuất

Cú pháp:

```
friend istream& operator>>(istream& is, PhanSo& a){  
    //Thực hiện việc nhập  
    return is;  
}  
friend ostream& operator<<(ostream& os, PhanSo a){  
    //Thực hiện việc xuất  
    return os;  
}
```

# Toán tử ++,--

**Cú pháp:**

PhanSo& operator++(); //Dành cho ++a;

PhanSo operator++(int); //Dành cho a++;



# Phép toán lấy phần tử mảng

Ta có thể định nghĩa phép toán [ ] để truy xuất phần tử của một đối tượng có ý nghĩa mảng. Ví dụ DanhSachSinhVien

```
class DanhSachSinhVien {  
    int n;  
    SinhVien* List;  
    public:  
    SinhVien& operator[](int i) {  
        return List[i]; }  
    SinhVien operator[] (int i) const {  
        return List[i]; } //Dành cho đối tượng gọi hàm là một hằng  
};
```



# DẪN XUẤT VÀ THỪA KẾ

**1. Quan hệ giữa các lớp đối tượng**

**2. Kế thừa**



# QUAN HỆ GIỮA CÁC LỚP ĐỐI TƯỢNG

Giữa các lớp đối tượng có những loại quan hệ

- Quan hệ một một (**1-1**)
- Quan hệ một nhiều (**1-n**)
- Quan hệ nhiều nhiều (**n-n**)
- Quan hệ đặc biệt hóa, tổng quát hóa

## Quan hệ một một (1-1)

**Quan hệ một-một:** Một đối tượng thuộc lớp này quan hệ với một đối tượng thuộc lớp kia và một đối tượng thuộc lớp kia có quan hệ duy nhất với một đối tượng thuộc lớp này.

Kí hiệu:





## Quan hệ một nhiều (1-n)

**Quan hệ một-nhiều:** Một đối tượng thuộc lớp này quan hệ với nhiều đối tượng thuộc lớp kia và một đối tượng lớp kia có quan hệ duy nhất với một đối tượng thuộc lớp này.



## Quan hệ nhiều nhiều (n-n)

**Quan hệ nhiều-nhiều:** Một đối tượng lớp này có quan hệ với nhiều đối tượng lớp kia và một đối tượng lớp kia cũng có quan hệ với nhiều đối tượng lớp này.

**Kí hiệu:**



# Quan hệ đặc biệt hóa – tổng quát hóa

**Quan hệ đặc biệt hóa – tổng quát hóa:** Lớp đối tượng này là trường hợp đặc biệt của lớp đối tượng kia và lớp đối tượng kia là trường hợp tổng quát của lớp đối tượng này.

**Kí hiệu:**





# KẾ THỪA

**1.Đặc tính**

**2.Cú pháp**

**3.Kiểu dẫn xuất**

**4.Lợi ích**





# KẾ THỪA

Biểu diễn mối quan hệ **đặc biệt hóa – tổng quát hóa**.

Các lớp được trừu tượng hóa và được tổ chức thành một sơ đồ phân cấp lớp.

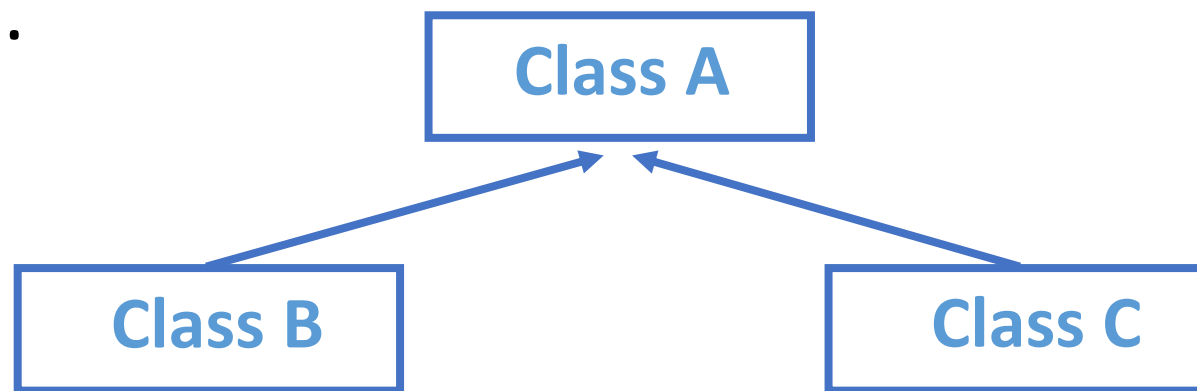
Kế thừa là mức cao hơn của trừu tượng hóa, cung cấp cơ chế gom chung các lớp có liên quan thành một mức khái quát hóa đặc trưng cho toàn bộ các lớp nói trên.



# 1. Đặc tính

Cho phép định nghĩa lớp mới từ lớp đã có  
Lớp đã có gọi là **Lớp Cơ Sở** (Base Class) hoặc **Lớp Cha** (Superclass).

Lớp mới gọi là **Lớp Dẫn Xuất** (Derived Class) hoặc **Lớp Con** (Subclass).



## 2. Cú pháp

```
class SuperClass{  
    //Thành phần của lớp cơ sở  
};  
class DerivedClass : public/protected/private SuperClass{  
    //Thành phần bổ sung của lớp dẫn xuất  
};
```

### 3. Kiểu dẫn xuất

Kiểu Dẫn Xuất / Kiểu Kế Thừa

Thành phần Lớp con

	Private	Protected	Public
Private	-	-	-
Protected	Private	Protected	Protected
Public	Private	Protected	Public





## 4. Lợi ích

Xây dựng lớp mới từ lớp đã có.

Chia sẻ mã chương trình chung  $\Rightarrow$  Dễ sửa chữa, nâng cấp.

Cơ chế chuyển kiểu tự động trong C++.

Dễ dàng bổ sung hoặc định nghĩa lại các thành phần.

## Không sử dụng Kế Thừa

```
1 class Person {  
2     private:  
3         string name;  
4         int age;  
5 };  
6  
7 class Staff {  
8     private:  
9         string name;  
10        int age;  
11        float salary;  
12 };
```

## Sử dụng Kế Thừa

```
1 class Person {  
2     private/protected:  
3         string name;  
4         int age;  
5 };  
6  
7 class Staff : public Person {  
8     private:  
9         float salary;  
10 };
```



# ĐA HÌNH (Polymorphism)

1. Khái niệm
2. Phân loại
3. Overloading/Overriding
4. Phương thức ảo
5. Phương thức thuần ảo
6. Mối liên hệ giữa phương thức thuần ảo và lớp trừu tượng



Có những **phương thức tổng quát** cho mọi lớp dẫn xuất nên có mặt ở **lớp cơ sở** nhưng **nội dung** của nó chỉ được **xác định** ở các **lớp dẫn xuất** cụ thể.

Ví dụ: Phương thức tính diện tích của lớp hình, hình tam giác, tứ giác,...





# 1. Khái niệm

Tính đa hình/đa xạ (polymorphism) trong OOP cho phép **các đối tượng khác nhau** nhưng **chung một lớp cơ sở** thực hiện một **hành động** (phương thức) theo những cách khác nhau.

VD: Chó, mèo, chuột cùng thuộc lớp cơ sở là **thú** thực hiện cùng một hành động là kêu nhưng mỗi con phát ra một âm thanh khác nhau.



## 2. Phân loại

Tính đa hình chủ yếu được chia làm 2 loại:

**Compile time Polymorphism:** gồm có **nạp chồng hàm** (Function Overloading) và **nạp chồng toán tử** (Operator Overloading). Tức là các hàm/toán tử cùng tên nhưng nhận nhiều đối tượng khác nhau.

**Ví dụ:** choAn(Cho c) và choAn(Meo m) là 2 hàm cùng tên nhưng nhận 2 đối tượng khác nhau.



## 2. Phân loại

**Runtime Polymorphism:** gồm có Virtual function được sử dụng để ghi đè và định nghĩa đúng chức năng của phương thức.

**Ví dụ:** lớp Thu có phương thức keu() và lớp Cho là lớp con của lớp Thu. Khi đó chúng ta phải định nghĩa lại phương thức keu() cho lớp Cho và ghi đè lên phương thức ở lớp Thu.



### 3. Overloading/Overriding

**OVERLOADING:** Một kỹ thuật cho phép trong một class có thể có nhiều phương thức trùng tên nhưng khác nhau về danh tham số.

**Đặc điểm:** phương thức A overload phương thức B thì A và B chỉ được phép trùng tên, phải khác nhau về danh sách tham số (thứ tự, kiểu dữ liệu,..) và kiểu dữ liệu trả về.





### 3. Overloading/Overriding

**OVERRIDING:** Một kỹ thuật sử dụng trong trường hợp lớp con kế thừa từ lớp cha và muốn định nghĩa lại một phương thức đã có mặt ở lớp cha.

**Đặc điểm:** phương thức A override phương thức B thì A và B phải cùng tên, danh sách tham số, kiểu dữ liệu trả về.



## 4. Phương thức ảo

## Khái niệm

Là cách thể hiện tính đa hình trong C++.

Các phương thức có tính đa hình phải được định nghĩa là một phương thức ảo.

## Cú pháp

Ta quy định một hàm thành phần là Phương thức ảo bằng cách thêm từ khóa `virtual` vào trước khai báo hàm.

```
virtual <Kiểu_Dữ_Liệu>  
<Tên_Phương_Thức>();
```



## Lưu ý

Phương thức ảo chỉ hoạt động thông qua **con trỏ**.

Phương thức ảo chỉ hoạt động nếu các phương thức ở lớp cơ sở và lớp con có **nghi thức giao tiếp giống hệt nhau**.

Nếu ở lớp con không định nghĩa lại phương thức ảo thì sẽ gọi phương thức ở lớp cơ sở (gần nhất có định nghĩa).



## 5. Phương thức thuần ảo

### Khái niệm

Là phương thức ảo và không có định nghĩa bên trong.

### Cú pháp

```
virtual <Kiểu_Dữ_Liệu> <Tên_Phương_Thức>()=0;
```





# Lớp trừu tượng

## Khái niệm

Là lớp chỉ được dùng làm cơ sở cho các lớp khác.  
Bắt buộc phải có ít nhất một phương thức thuần ảo.

## Đặc điểm

Dùng để định nghĩa một số khái niệm tổng quát,  
chung cho các lớp khác.  
Không có đối tượng nào thuộc Lớp trừu tượng.



# Mối liên hệ giữa phương thức thuần ảo và lớp trừu tượng

Các phương thức thuần ảo **thường được chứa** trong lớp trừu tượng.

Theo **quan điểm chung**, lớp trừu tượng không nhất thiết phải chứa phương thức thuần ảo.

Theo **quan điểm C++**, lớp trừu tượng phải chứa ít nhất một phương thức thuần ảo.



## Lưu ý

Lớp dẫn xuất từ lớp trừu tượng **phải định nghĩa lại tất cả các phương thức thuần ảo của lớp trừu tượng.**

Hoặc tiếp tục là phương thức thuần ảo.

Hoặc được cài đặt chi tiết trong lớp dẫn xuất.

Có ý nghĩa trong việc **tổ chức phân cấp các lớp: phương thức thuần ảo** trong lớp cơ sở sẽ là **phương thức chung** cho các lớp dẫn xuất thừa kế và cài đặt.



**CẢM ƠN CÁC BẠN ĐÃ LẮNG NGHE!**  
**QUÉT MÃ QR ĐIỂM DANH**

