BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ II NĂM HỌC 2022 – 2023

ĐOÀN KHOA
CÔNG NGHỆ PHẦN MỀM
BAN HỌC TẬP

Sharing is learning

BAN HỌC TẬP

Khoa Công nghệ Phần mềm

Trường Đại học Công nghệ Thông tin

Đại học Quốc gia thành phố Hồ Chí Minh

CONTACT

bht.cnpm.uit@gmail.com

fb.com/bhtcnpm

fb.com/groups/bht.cnpm.uit

TRAINING

OOP – Cuối kỳ

Thời gian: 9:30 thứ 2 ngày 19/06/2023

Địa điểm: Phòng C109

Trainers: Nguyễn Việt Khoa – KTPM2022.2

Lê Duy Nguyên – KTPM2022.2

Sharing is learning

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình

1. Lớp và đối tượng

- **Lớp** là một *mô tả trừu tượng* của nhóm các **đối tượng** cùng bản chất, ngược lại mỗi một **đối tượng** là một *thể hiện cụ thể* cho những **mô tả trừu tượng đó.**
- **Lớp** là cái ta thiết kế và lập trình
- **Đối tượng** là cái ta tạo (từ một lớp) tại thời gian chạy.

1. Lớp và đối tượng

Một lớp đối tượng bao gồm 2 thành phần chính:

- Thành phần dữ liệu (**data member**), hay còn được gọi là thuộc tính (**attribute**).
- Hàm thành phần (**member function**), còn có tên gọi khác là phương thức (**method**), là các hành động mà đối tượng của lớp có thể thực hiện.

1. Lớp và đối tượng

Khai báo Lớp:

```
class <tên_lớp>
{
    //Thành phần dữ liệu

    //Thành phần xử lý
};
```

1. Lớp và đối tượng

Khai báo và tạo đối tượng:

```
<tên_lớp>  <tên_đối_tượng>;
```

Gọi hàm thành phần của lớp

```
<tên_đối_tượng> . <tên_hàm_thành_phần> (<danh sách
các tham số nếu có>);
<tên_con_trỏ_đối_tượng> -> <tên_hàm_thành_phần>
(<danh sách các tham số nếu có>);
```

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình

2. Tính trừu tượng hóa và đóng gói

Trừu tượng hóa dữ liệu là một kỹ thuật lập trình và thiết kế dựa trên sự tách biệt của **Giao diện (Interface)** và **Thực thi (Implementation).**

- **Giao diện** của một Lớp đối tượng là các hoạt động mà người dùng của một Lớp có thể thao tác trên các đối tượng của Lớp đó.
- **Phần Thực thi** bao gồm các dữ liệu thành viên (thuộc tính), phần định nghĩa của các phương thức.

2. Tính trừu tượng hóa và đóng gói

Đóng gói chính là quá trình ẩn đi **phần Thực thi** khỏi người dùng bên ngoài và giới hạn quyền truy cập vào nó. Người dùng của một Lớp chỉ có thể sử dụng **Giao diện** mà *không* có quyền truy cập vào **phần Thực thi**

2. Tính trừu tượng hóa và đóng gói

Phạm vi truy xuất:

- Trong định nghĩa của lớp ta có thể xác định khả năng truy xuất thành phần của một lớp nào đó từ bên ngoài phạm vi lớp.
- `private`, `protected` và `public` là các từ khoá xác định phạm vi truy xuất
- Mọi thành phần được liệt kê trong phần `public` đều có thể truy xuất trong bất kỳ hàm nào.
- Những thành phần được liệt kê trong phần `private` chỉ được truy xuất bên trong phạm vi lớp.

2. Tính trừu tượng hóa và đóng gói

Ví dụ:

```
class TamGiac{
  private:
    float a,b,c;//độ dài ba cạnh
    int Loaitg();//cho biết kiểu của tam giác:
    float DienTich();//tính diện tích của tam giác*
  public:
    void Nhap();//nhập vào độ dài ba cạnh
    void In();//in ra các thông tin liên quan đến t
giác
};
```

Sharing is learning

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình

3. Phương thức thiết lập, phá hủy

- Các phương thức thiết lập **(Constructors)**: có nhiệm vụ thiết lập các thông tin ban đầu của các đối tượng thuộc về lớp đối tượng khi đối tượng được khai báo.
- Bất kỳ một đối tượng nào được khai báo đều phải sử dụng một hàm thiết lập để khởi tạo các giá trị thành phần của đối tượng.
- Hàm thiết lập được khai báo giống như một phương thức với tên phương thức trùng với tên lớp và không có giá trị trả về (kể cả void).
- Constructor phải có thuộc tính public

3. Phương thức thiết lập, phá hủy

Phân biệt các loại **Constructor :**

Phương thức thiết lập mặc định (**Default Constructors**): là phương thức thiết lập các thông tin ban đầu cho đối tượng về lớp bằng những giá trị mặc định (do người lập trình quyết định).

```
PhanSo :: PhanSo()
{
    tu = 0;
    mau = 1;
}
```

3. Phương thức thiết lập, phá hủy

Phân biệt các loại **Constructor :**

Phương thức thiết lập sao chép (**Copy Constructors**): là phương thức nhận tham số đầu vào là 1 đối tượng cùng thuộc lớp .

Thông tin ban đầu của đối tượng sẽ giống hoàn toàn thông tin đối tượng tham số đầu vào.

```
PhanSo :: PhanSo( const PhanSo& a )
{
    tu = a.tu;
    mau = a.mau;
}
```

3. Phương thức thiết lập, phá hủy

Phân biệt các loại **Constructor :**

*Phương thức thiết lập nhận tham số đầu vào (***Paremeterired Constructors***):* Là thiết lập thông tin ban đầu cho các đối tượng thông qua các tham số đầu vào. **Constructor** nhận tham số đầu vào có thể sử dụng để ép kiểu.

```
PhanSo :: PhanSo(int a, int b)
{
    tu = a;  mau = b;
}
```

3. Phương thức thiết lập, phá hủy

Phân biệt các loại **Constructor :**

Phương thức thiết lập nhận tham số đầu vào (**Parameterized Constructors**)*:* Là thiết lập thông tin ban đầu cho các đối tượng thông qua các tham số đầu vào. **Constructor** nhận tham số đầu vào có thể sử dụng để ép kiểu.

```
PhanSo :: PhanSo(int a)
{
    tu = a;
    mau = 1;
}
```

3. Phương thức thiết lập, phá hủy

- Các phương thức phá hủy (**Destructor**): có nhiệm vụ thu hồi lại các tài nguyên cấp phát cho đối tượng khi đối tượng hết phạm vi hoạt động.
- Destructor, được gọi ngay trước khi một đối tượng bị thu hồi.
- Destructor thường được dùng để thực hiện việc dọn dẹp cần thiết trước khi một đối tượng bị hủy.
- Một lớp chỉ có ***duy nhất*** một Destructor
- Phương thức Destructor trùng tên với tên lớp nhưng có dấu **~** đặt trước
- Được tự động gọi thực hiện khi đối tượng hết phạm vi sử dụng.
- Destructor phải có thuộc tính **`public`**

3. Phương thức thiết lập, phá hủy

```
class vector{
      int n;     //số chiều
      float *v;        //vùng nhớ tọa độ
public:
      vector(); //Hàm thiết lập mặc định
      vector(int size); //Hàm thiết lập nhận tham số đầu vào
      vector(int size, float *a);
      ~vector();//Hàm hủy bỏ, luôn luôn không có tham số

};
```

Sharing is learning

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình

4. Thành viên static, hàm bạn, lớp bạn

Thành viên static

- Các thành phần tĩnh (static member) là các thành phần thuộc về cả một **lớp** chứ không thuộc về một **đối tượng cụ thể**. Điều này có nghĩa là tất cả các đối tượng của một lớp đều chia sẻ chung một thành phần tĩnh, và nó có thể được truy cập mà không cần thông qua một đối tượng.
- Có hai loại thành phần tĩnh:
  - Các thuộc tính (dữ liệu thành viên) tĩnh (Static data member)
  - Hàm thành viên (phương thức) tĩnh (Static member function)
- Thuộc tính tĩnh không phải là thuộc tính của một đối tượng nào cả
- Một hàm thành viên tĩnh không gắn với bất kì đối tượng nào

4. Thành viên static, hàm bạn, lớp bạn

Khởi tạo static member

Muốn khởi tạo một thành viên tĩnh chúng ta sẽ thêm từ khóa `static` vào trước dòng khai báo của nó. Như các thành viên khác thì thành viên tĩnh cũng có thể được khai báo như là một thành phần `private`, `public` hay là `protected`.

4. Thành viên static, hàm bạn, lớp bạn

Khởi tạo static member

Ví dụ:

```
class Rectangle
{
private:
        int width;
        int length;
        static int count;
public:
        void set(int w, int l);
        int area();
}
```


Sharing is learning

4. Thành viên static, hàm bạn, lớp bạn

Hàm bạn (**Friend Function**): không thuộc lớp nhưng vẫn có quyền truy cập vào thành phần `private, protected`.

```
class MyInt {
private:
      int Data = 1;
public:
      friend int Get( MyInt );
};
```

4. Thành viên static, hàm bạn, lớp bạn

Hàm bạn (**Friend Function**): không thuộc lớp nhưng vẫn có quyền truy cập vào thành phần `private, protected`.

```
int Get(MyInt a)
{
   return a.Data;
}
```

4. Thành viên static, hàm bạn, lớp bạn

Lớp bạn (**Friend Class**): là lớp có thể truy cập các thành phần `private, protected` của lớp xem nó là bạn.

Ví dụ:

```
class TOM{
public:
	friend class JERRY; //Có lớp bạn là JERRY
private:
	int SecretTom;
};
class JERRY{
public:
	void Change(TOM T){
		T.SecretTom++; //Bạn nên có thể truy cập
	}
};
```

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình

5. Nạp chồng toán tử

- **Định nghĩa lại toán tử** đã có trên các **lớp đối tượng** do người dùng tự tạo để dể dàng thể hiện các câu lệnh trong chương trình.
- Ví dụ:

```
PhanSo ps1(1, 2);
PhanSo ps2(2, 3);
PhanSo ketQua;
ketQua = ps1.Tong(ps2); // Dùng hàm
ketQua = ps1 + ps2; // Dùng Overload operator
```

5. Nạp chồng toán tử

- **Bản chất:** Nạp chồng tức là định nghĩa các hàm có **cùng tên** nhưng **khác tham số** truyền vào.
- **Cú pháp:** Nạp chồng (overload) hàm có tên là "**operator@**", với @ là toán tử cần overload (+, -, *, /, ...).
- **Có 2 cách nạp chồng:**
  - Sử dụng phương thức của lớp.
  - Sử dụng hàm bên ngoài.

5.1. Nạp chồng bằng phương thức

- Cú pháp khai báo: `KDLTrảVề operator@(<DSThamSố>);`
- Ví dụ: câu lệnh `a + b` sẽ tương đương với `a.operator+(b)`
- Số lượng tham số đầu vào = Số toán hạng – 1 (do toán hạng đầu tiên sẽ là **đối tượng gọi phương thức**).

5.1. Nạp chồng bằng phương thức

- **Ví dụ:** Nạp chồng toán tử + bằng phương thức cho lớp PhanSo:

```
class PhanSo {
public:
    PhanSo operator+(const PhanSo& ps);
    //...
};
PhanSo::PhanSo operator+(const PhanSo& ps) {
    PhanSo kq;
    kq.tu = this->tu * ps.mau + ps.tu * this->mau;
    kq.mau = this->mau * ps.mau;
    return kq;
}
```

5.2. Nạp chồng bằng hàm bên ngoài

- Ví dụ: câu lệnh `a + b` sẽ tương đương với `operator+(a,b)`
- Số lượng tham số đầu vào = Số toán hạng
- Hàm bên ngoài phải là **hàm bạn** để có thể truy cập vào các thuộc tính riêng tư của lớp.

5.2. Nạp chồng bằng hàm bên ngoài

- **Ví dụ:** Nạp chồng toán tử + bằng hàm bạn cho lớp PhanSo:

```
class PhanSo {
    friend PhanSo operator+(const PhanSo& ps1,
                            const PhanSo& ps2)
    //...
};
PhanSo operator+(const PhanSo& ps1, const PhanSo& ps2){
    PhanSo kq;
    kq.tu = ps1.tu * ps2.mau + ps1.mau * ps2.tu;
    kq.mau = ps1.mau * ps2.mau;
    return kq;
}
```

5.3. Một vài lưu ý

- Các đối số truyền vào có thể là các **kiểu dữ liệu khác** nếu có phép **chuyển kiểu** tương ứng.
- Ví dụ:

```
// Chuyển kiểu bằng constructor
PhanSo(int a) {
    tu = a;
    mau = 1;
}
```

- Khi đó câu lệnh `a + 5` có thể được hiểu thành:

```
a.operator+(PhanSo(5))
```

5.3. Một vài lưu ý

- Nạp chồng bằng phương thức của lớp thì **toán hạng đầu tiên** phải là **đối tượng** thuộc lớp đó.
- Ví dụ:

```
PhanSo a(2,3);
5 + a; // Sai
```

- Câu lệnh trên sai vì không thể gọi phương thức `operator+` trên số nguyên 5.

5.3. Một vài lưu ý

- Vấn đề có thể được giải quyết bằng cách sử dụng **hàm bạn** và các phép **chuyển kiểu** phù hợp.
- Lúc này câu lệnh `5 + a` sẽ tương đương với:

```
operator+(PhanSo(5),a)
```

5.4. Cách nạp chồng các toán tử cơ bản

- Toán tử số học
- Toán tử gán
- Toán tử so sánh
- Toán tử số học, gán kết hợp
- Toán tử tăng một, giảm một
- Toán tử nhập, xuất

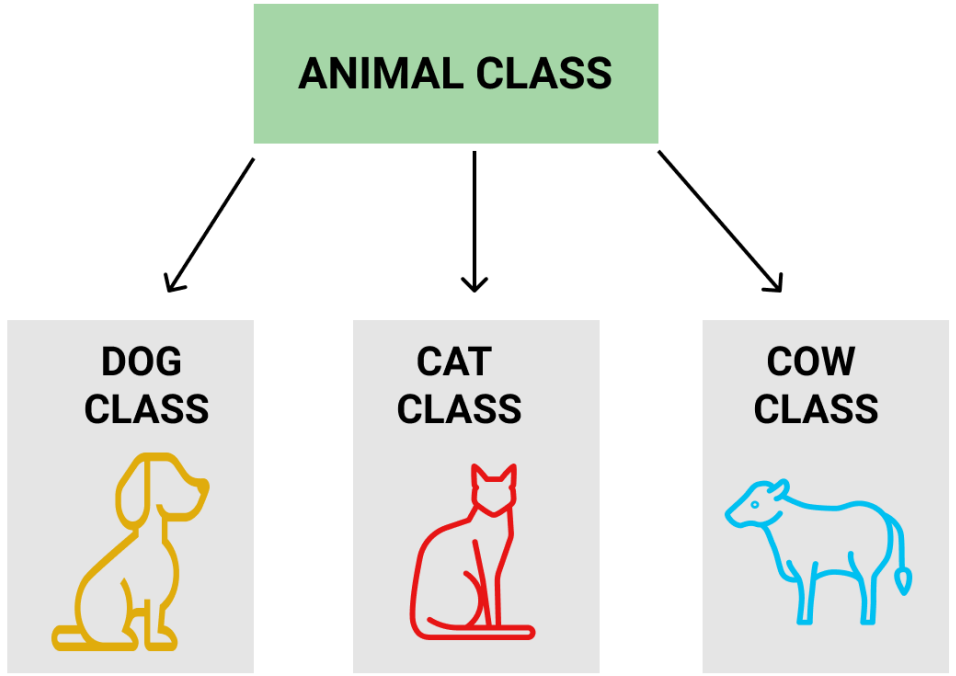
Sổ tay kiến thức OOP

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình

6. Kế thừa

- Một đặc điểm của ngôn ngữ lập trình dùng để biểu diễn mối quan hệ **đặc biệt hóa – tổng quát hóa** giữa các lớp (lớp này là trường hợp đặc biệt của lớp kia).



6. Kế thừa

- **Lợi ích** của kế thừa:
  - Kế thừa cho phép **xây dựng lớp mới** từ lớp đã có.
  - Kế thừa cho phép tổ chức các lớp **chia sẻ mã** chương trình chung, nhờ vậy có thể dễ dàng sửa chữa, nâng cấp hệ thống.
  - Định nghĩa sự tương thích giữa các lớp, nhờ đó ta có thể chuyển kiểu tự động.

6. Kế thừa

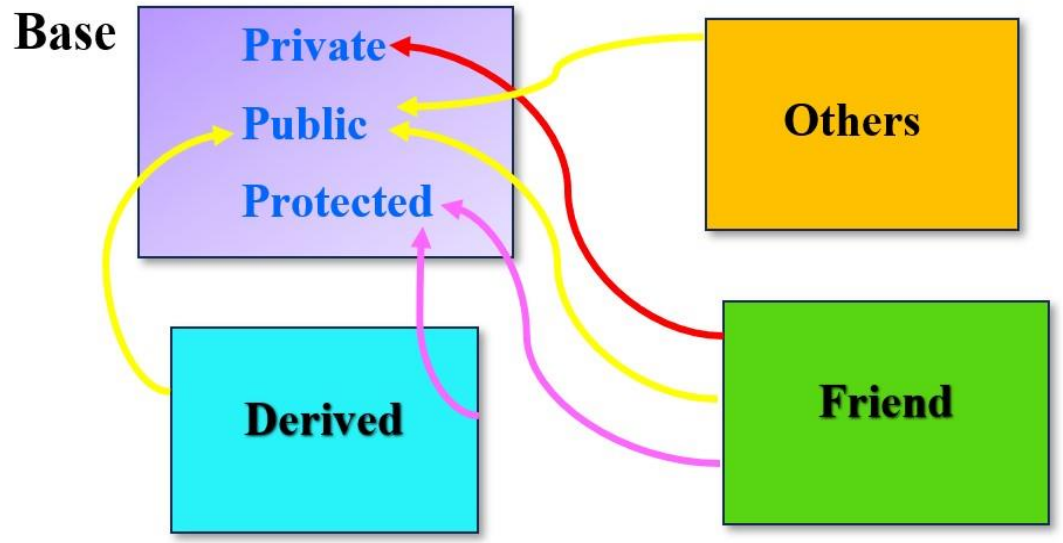
- Cú pháp khai báo:

```
class B : <từ khóa dẫn xuất> A {
        ...
};
```

- Lớp B kế thừa từ lớp A. B là lớp con (lớp dẫn xuất), A là lớp cha (lớp cơ sở).
- Lớp con sỡ hữu các **thuộc tính, phương thức** của lớp cha mà không cần phải khai báo.

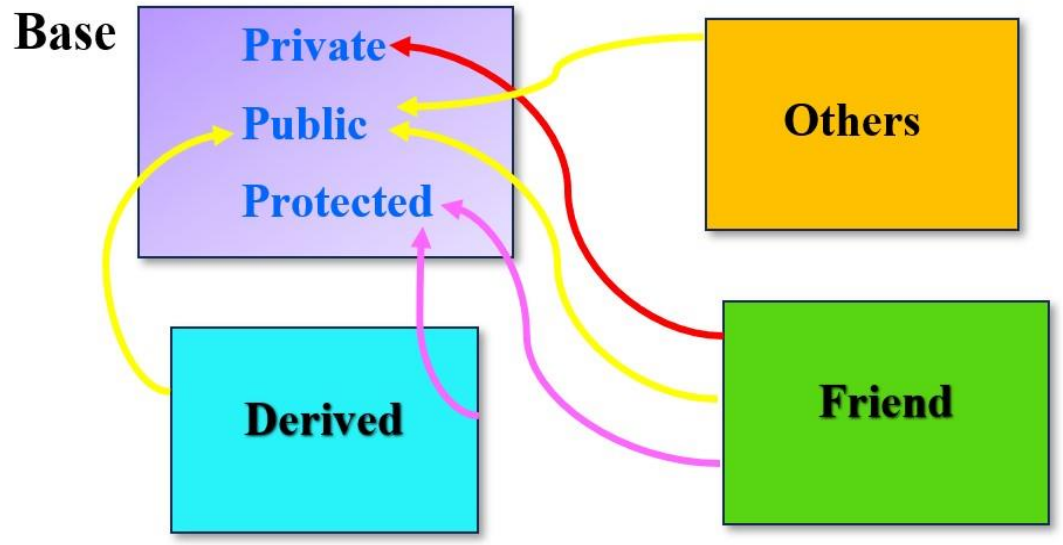
6.1. Phạm vi truy cập protected

- Sự kết hợp giữa `public` và `private:`
  - Giống với `private`, các thành viên `protected` không thể được truy cập từ bên ngoài lớp.



6.1. Phạm vi truy cập protected

- Sự kết hợp giữa `public` và `private:`
  - Giống với `public`, các thành viên `protected` của lớp cơ sở có thể được truy cập bởi bạn và các lớp dẫn xuất của nó.



6. Kế thừa

- Cú pháp khai báo:

```
class B : <từ khóa dẫn xuất> A {
        ...
};
```

- Lớp B kế thừa từ lớp A. B là lớp con (lớp dẫn xuất), A là lớp cha (lớp cơ sở).
- Lớp con sỡ hữu các **thuộc tính, phương thức** của lớp cha mà không cần phải khai báo.

6.2. Các kiểu kế thừa

- Từ khóa dẫn xuất dùng để **thay đổi mức độ truy cập** của các thành viên ở lớp cha khi chúng được kế thừa xuống lớp con:

6.2. Các kiểu kế thừa

Phạm vi truy xuất được chỉ định ở lớp cơ sở:	Từ khóa dẫn xuất khi khai báo lớp con:	Phạm vi truy xuất ở lớp con được chuyển thành:
Public	public	public
Protected		protected
Private		Không truy cập được
Public	protected	protected
Protected		protected
Private		Không truy cập được
Public	private	private
Protected		private
Private		Không truy cập được

6.2. Các kiểu kế thừa

Phạm vi truy xuất được chỉ định ở lớp cơ sở:	Từ khóa dẫn xuất khi khai báo lớp con:	Phạm vi truy xuất ở lớp con được chuyển thành:
Public	public	public
Protected		protected
Private		Không truy cập được
Public	protected	protected
Protected		protected
Private		Không truy cập được
Public	private	private
Protected		private
Private		Không truy cập được

6.2. Các kiểu kế thừa

- Từ khóa dẫn xuất dùng để **thay đổi mức độ truy cập** của các thành viên ở lớp cha khi chúng được kế thừa xuống lớp con:
  - Lớp con kế thừa kiểu `public` từ lớp cha thì các thành phần `protected` của lớp cha trở thành `protected` của lớp con, các thành phần `public` của lớp cha trở thành `public` của lớp con.

6.2. Các kiểu kế thừa

Phạm vi truy xuất được chỉ định ở lớp cơ sở:	Từ khóa dẫn xuất khi khai báo lớp con:	Phạm vi truy xuất ở lớp con được chuyển thành:
Public	public	public
Protected		protected
Private		Không truy cập được
Public	protected	protected
Protected		protected
Private		Không truy cập được
Public	private	private
Protected		private
Private		Không truy cập được

6.2. Các kiểu kế thừa

- Từ khóa dẫn xuất dùng để **thay đổi mức độ truy cập** của các thành viên ở lớp cha khi chúng được kế thừa xuống lớp con:
  - Lớp con kế thừa kiểu `protected` từ lớp cha thì các thành phần `protected` và `public` của lớp cha trở thành `protected` của lớp con.

6.2. Các kiểu kế thừa

Phạm vi truy xuất được chỉ định ở lớp cơ sở:	Từ khóa dẫn xuất khi khai báo lớp con:	Phạm vi truy xuất ở lớp con được chuyển thành:
Public	public	public
Protected		protected
Private		Không truy cập được
Public	protected	protected
Protected		protected
Private		Không truy cập được
Public	private	private
Protected		private
Private		Không truy cập được

6.2. Các kiểu kế thừa

- Từ khóa dẫn xuất dùng để **thay đổi mức độ truy cập** của các thành viên ở lớp cha khi chúng được kế thừa xuống lớp con:
  - Lớp con kế thừa kiểu `private` từ lớp cha thì các thành phần `protected` và `public` của lớp cha trở thành `private` của lớp con.

6.2. Các kiểu kế thừa

Phạm vi truy xuất được chỉ định ở lớp cơ sở:	Từ khóa dẫn xuất khi khai báo lớp con:	Phạm vi truy xuất ở lớp con được chuyển thành:
Public	public	public
Protected		protected
Private		Không truy cập được
Public	protected	protected
Protected		protected
Private		Không truy cập được
Public	private	private
Protected		private
Private		Không truy cập được

6.2. Các kiểu kế thừa

- Lớp con được sỡ hữu các thuộc tính `private` của lớp cha.
- Tuy nhiên lớp con **không thể truy cập trực tiếp** đến các thuộc tính này mà phải thông qua các **phương thức truy vấn, cập nhật** được thừa hưởng từ lớp cha.

6.3. Phép gán với con trỏ trong kế thừa

- Một biến con trỏ kiểu lớp cha có thể giữ địa chỉ của của một đối tượng thuộc lớp con.
- Tuy nhiên ngược lại thì sai.
- Ví dụ:

```
Cha cha;
Con con;
Cha* chaPtr = &con; // Đúng
Con* conPtr = &cha; // Sai
```

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình

7. Đa hình

- Ngoài việc được thừa hưởng các thuộc tính và phương thức từ lớp cha, lớp con còn có thể **định nghĩa thêm** các thuộc tính, phương thức mới và **định nghĩa lại** một vài phương thức ở lớp cha.
- **Phương thức ảo** là những phương thức lớp cha muốn lớp con định nghĩa một **phiên bản riêng** của nó.

7. Đa hình

```
class Cha {
public:
    virtual void Xuat() {
        cout << "Cha";
    }
};
```

```
class Con : public Cha {
public:
    void Xuat() {
        cout << "Con";
    }
};
```


Sharing is learning

7. Đa hình

- Phân biệt giữa nạp chồng (overload) và ghi đè (override):
  - **Nạp chồng:** Các phương thức có **cùng tên** nhưng danh sách **tham số khác nhau**, kiểu dữ liệu trả về có thể giống hoặc khác nhau.
  - **Ghi đè:** Các phương thức **giống nhau hoàn toàn** về tên, danh sách tham số, kiểu dữ liệu trả về.

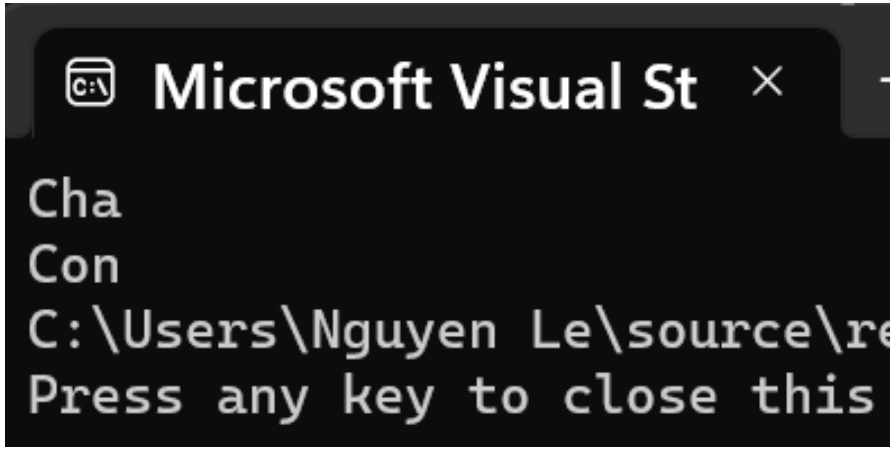
7. Đa hình

- Khi gọi một **phương thức ảo** thông qua một **con trỏ** kiểu lớp cha, tùy thuộc vào **kiểu dữ liệu** của đối tượng mà biến con trỏ đang giữ địa chỉ, phiên bản của phương thức ảo trong **lớp tương ứng** sẽ được thực hiện -> tính đa hình.

```
Cha base; Con derived;
Cha* basePtr;
basePtr = &base;
basePtr->Xuat();
basePtr = &derived;
basePtr->Xuat();
```

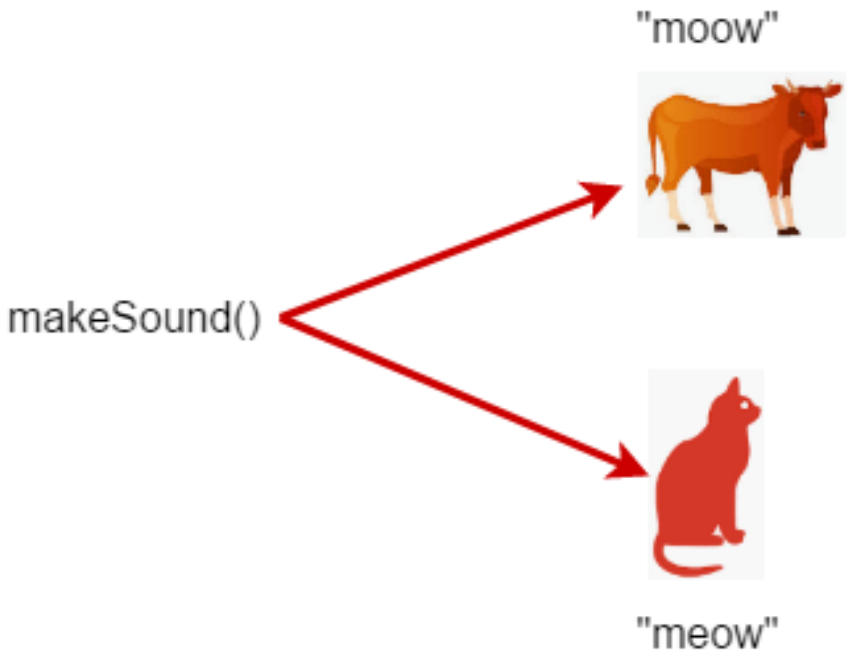
7. Đa hình

- Khi gọi một **phương thức ảo** thông qua một **con trỏ** kiểu lớp cha, tùy thuộc vào **kiểu dữ liệu** của đối tượng mà biến con trỏ đang giữ địa chỉ, phiên bản của phương thức ảo trong **lớp tương ứng** sẽ được thực hiện -> tính đa hình.



7. Đa hình

- Khi gọi một **phương thức ảo** thông qua một **con trỏ** kiểu lớp cha, tùy thuộc vào **kiểu dữ liệu** của đối tượng mà biến con trỏ đang giữ địa chỉ, phiên bản của phương thức ảo trong **lớp tương ứng** sẽ được thực hiện -> tính đa hình.



7. Đa hình

- **Hàm thuần ảo**: Một hàm được **tạo ra mà không có nội dung gì** để **phục vụ cho các lớp dẫn xuất** kế thừa từ lớp cơ sở ghi đè lại.
- **Lớp trừu tượng**: Một **lớp có ít nhất một hàm thuần ảo** được gọi là lớp trừu tượng. Mục đích lớp trừu tượng được tạo ra là để **các lớp dẫn xuất kế thừa lại**

7. Đa hình

- **Ví dụ:**

```
class Animal {
public:
    // phương thức thuần ảo:
    virtual void makeSound() = 0;
    // phương thức makeSound của Animal
    // không có hành động cụ thể
};
```

7. Đa hình

- **Lưu ý:** Khi gọi phương thức thông qua con trỏ lớp cha, ta chỉ có thể **gọi các phương thức mà lớp cha sỡ hữu**, cho dù con trỏ đang giữ địa chỉ của đối tượng thuộc lớp cha hay con.
- Ví dụ:

```
Con derived;
// PhuongThucCon do lớp Con định nghĩa thêm
derived.PhuongThucCon();  // Đúng
Cha* basePtr = &derived;
basePtr->PhuongThucCon(); // Sai!
```

7. Đa hình

- Có thể sử dụng **ép kiểu** để gọi các hàm của lớp con, tuy nhiên hành động này **dễ gây lỗi**, cần sử dụng cẩn thận.
- Ví dụ:

```
((Con*)basePtr)->PhuongThucCon(); // Đúng
```

Sổ tay kiến thức OOP

Sổ tay kiến thức OOP

Sharing is learning

Form điểm danh

ĐOÀN KHOA CÔNG NGHỆ PHẦN MỀM
BAN HỌC TẬP
Sharing is learning

BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ II NĂM HỌC 2022 – 2023

ĐOÀN KHOA CÔNG NGHỆ PHẦN MỀM
BAN HỌC TẬP

Sharing is learning

HẾT

CẢM ƠN CÁC BẠN ĐÃ THEO DÕI
CHÚC CÁC BẠN CÓ KẾT QUẢ THI THẬT TỐT!

BAN HỌC TẬP

Khoa Công nghệ Phần mềm

Trường Đại học Công nghệ Thông tin

Đại học Quốc gia thành phố Hồ Chí Minh

CONTACT

bht.cnpm.uit@gmail.com

fb.com/bhtcnpm

fb.com/groups/bht.cnpm.uit