

## KỸ THUẬT LẬP TRÌNH C/C++

DẪN XUẤT/THỪA KẾ VÀ ĐA HÌNH/ĐA XẠ

Giảng viên: TS. Nguyễn Thị Kim Thoa

Học kỳ: 20211

Năm học: 2021-2022



# Nội dung



**1. Dẫn xuất và thừa kế**

**2. Hàm ảo và cơ chế đa hình/đa xạ**

**3. Lớp thuần ảo**

**4. Kiểm soát truy nhập**

**5. Tương thích kiểu**



# 1. Một số khái niệm

- **Đối tượng là gì?**

- Mô hình đại diện của một đối tượng vật lý:
  - ✓ Person, student, employee, employer
  - ✓ Car, bus, vehicle,...
- Đối tượng logic:
  - ✓ Trend, report, button, window,...

- **Một đối tượng có:**

- Các thuộc tính
- Trạng thái
- Hành vi
- Cẩn cước
- Ngữ nghĩa



# 1. Dẫn xuất và thừa kế

**Nghiên cứu các thuộc tính và phương thức của các loại xe đạp sau**

- *Xe đạp (Bicycle)*



- *Tandem bicycle*

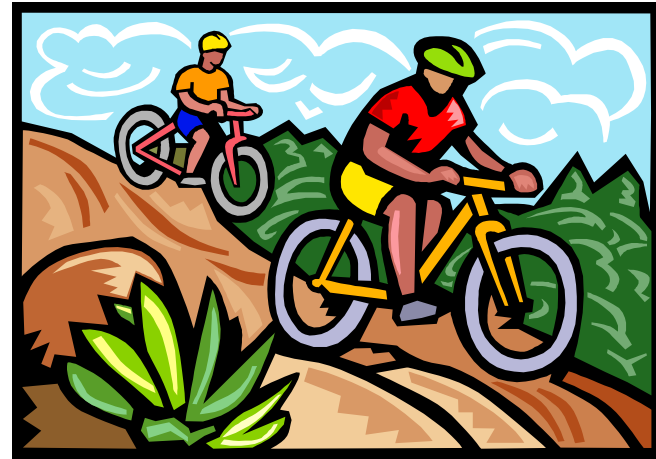


# @ 1. Dẫn xuất và kế thừa [...]

- *Racing Bike*



- *Mountain Bike*







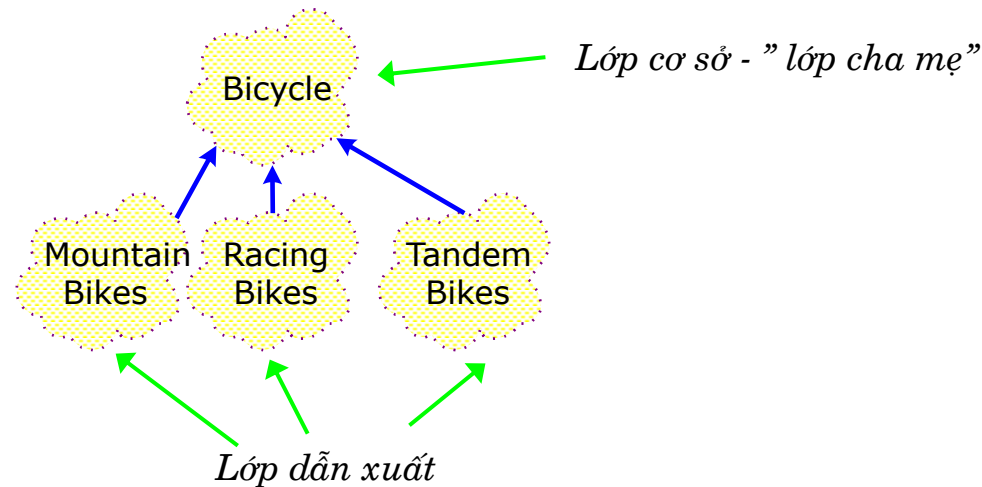
# 1. Dẫn xuất và kế thừa [...]

- Tandem bicycle là một loại xe đạp
  - Xe đạp có hai yên
- Mountain bicycle là một loại xe đạp
  - Xe đạp có khả năng chống sốc (lốp dày và nhiều bánh răng)
- Racing bicycle là một loại xe đạp
  - Xe đạp có cấu tạo khí động lực học nhẹ
- Tandem, mountain, racing bicycle là những loại xe đạp chuyên dụng
  - Có các thành phần cơ bản của một chiếc xe đạp
  - Cùng nguyên lý hoạt động
  - Bổ sung thêm các thông tin khác



# 1. Dẫn xuất và kế thừa [...]

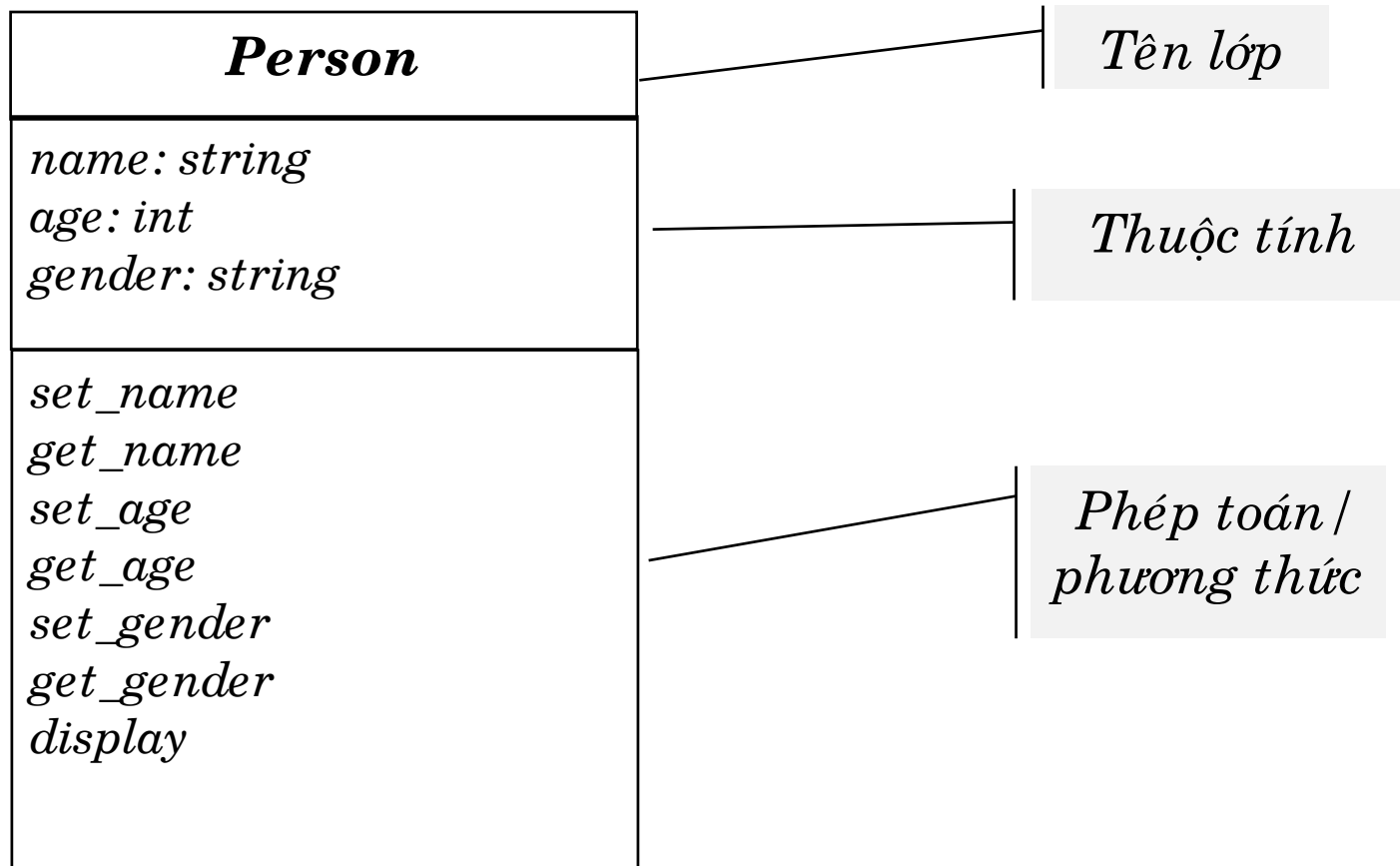
- Cơ chế dẫn xuất/ thừa kế là một kỹ thuật lập trình hướng đối tượng cho phép chuyên biệt hóa
- Dẫn xuất cho phép tạo ra một lớp mới (lớp dẫn xuất) của các đối tượng bằng cách sử dụng các lớp cũ như là các lớp cơ sở
  - Lớp dẫn xuất thừa hưởng các thuộc tính và hành vi của lớp “cha-mẹ” (lớp cơ sở)
  - Lớp dẫn xuất là một phiên bản chuyên biệt hóa của lớp “cha-mẹ”



# @ 1. Dẫn xuất và kế thừa [...]

Xây dựng các lớp biểu diễn về con người, sinh viên, giảng viên

- Các thuộc tính và phép toán cơ bản của lớp Person







# 1. Dẫn xuất và kế thừa [...]

- Các thuộc tính và phép toán cơ bản của lớp Student và Lecture

<i><b>Student</b></i>
<i>name: string</i> <i>age: int</i> <i>gender: string</i> <i>class: string</i> <i>id: int</i>
<i>set_name</i> <i>get_name</i> <i>set_age</i> <i>get_age</i> <i>set_gender</i> <i>get_gender</i> <i>set_class</i> <i>get_class</i> <i>set_id</i> <i>get_id</i> <i>display</i>

<i><b>Lecture</b></i>
<i>name: string</i> <i>age: int</i> <i>gender: string</i> <i>faculty: string</i> <i>telnumber: int</i>
<i>set_name</i> <i>get_name</i> <i>set_age</i> <i>get_age</i> <i>set_gender</i> <i>get_gender</i> <i>set_faculty</i> <i>get_faculty</i> <i>set_telnumber</i> <i>get_telnumber</i> <i>display</i>

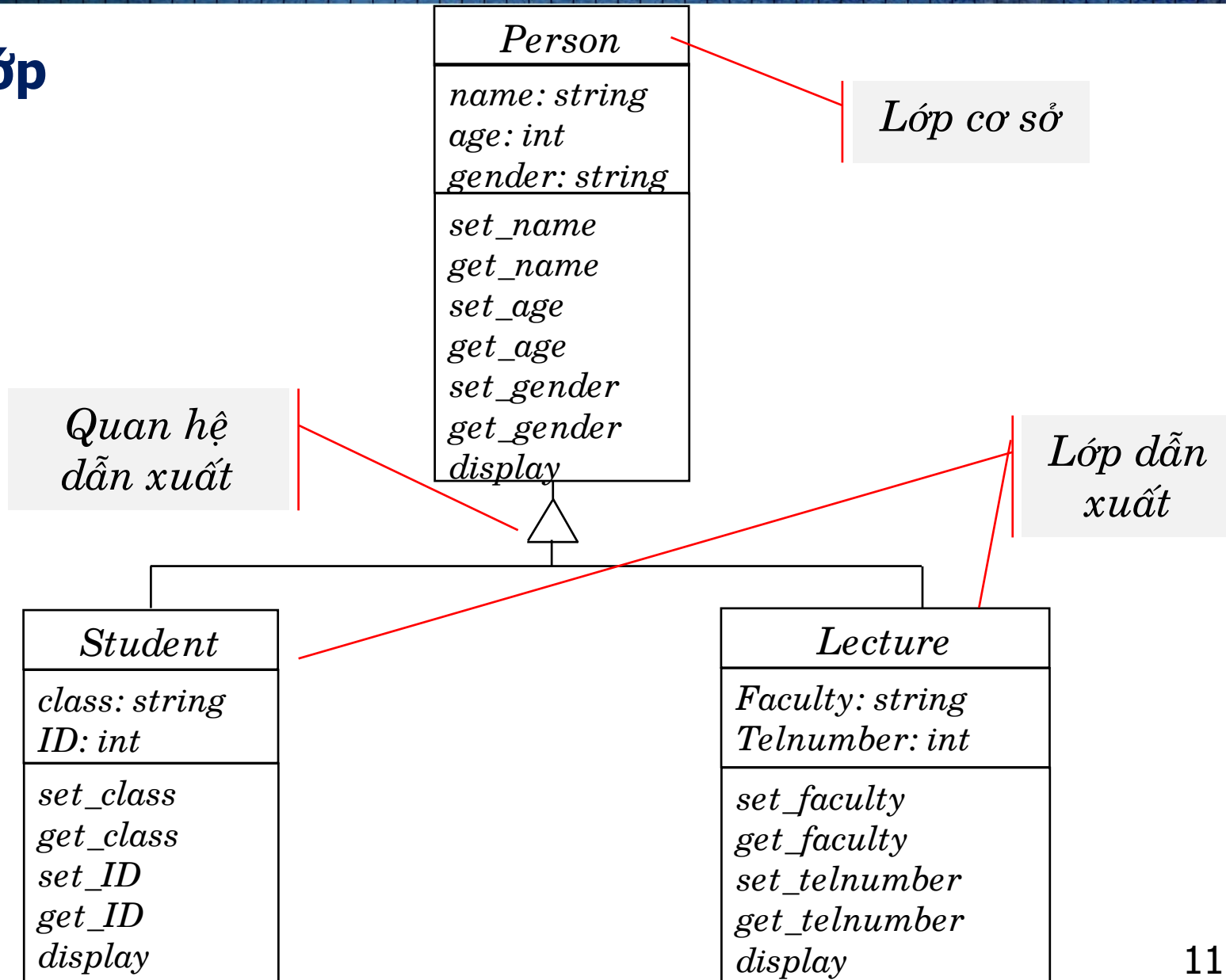


# 1. Dẫn xuất và kế thừa [...]

- Ba lớp trên giống nhau về:
  - Thuộc tính:
    - ✓ Name
    - ✓ Age
    - ✓ gender
  - Phương thức
    - ✓ set\_name, get\_name
    - ✓ set\_age, get\_age
    - ✓ set\_gender, get\_gender
- Khác nhau: lớp Student, Lecture có bổ sung thêm các thuộc tính và phương thức

# @ 1. Dẫn xuất và kế thừa [...]

## Quan hệ lớp





# 1. Dẫn xuất và kế thừa [...]

*//Khai báo lớp Person trong file person.h*

```
#pragma once
#include <string>
using namespace std;
class Person{
    string name;
    int age;
    string gender;
public:
    Person(string, int, string);
    string get_name();
    void set_name(string);
    int get_age();
    void set_age(int);
    string get_gender();
    void set_gender(string);
    void display();
};
```



# 1. Dẫn xuất và kế thừa [...]

*//Định nghĩa lớp Person trong file person.cpp*

```
#include "stdafx.h"
#include <iostream>
#include <string>
#include "person.h"
Person::Person(string _name, int _age, string _gender){
    name = _name;
    age = _age;
    sex = _gender;
}
string Person::get_name(){ return name;}
void Person::set_name(string _name){ name = _name;}
int Person::get_age(){ return age;}
void Person::set_age(int _age){ age = _age;}
string Person::get_gender(){ return gender;}
void Person::set_gender(string _gender){gender = _gender;}
void Person::display(){
    cout<<"Person:\n";
    cout<<"Name:\t"<<name<< endl;
    cout<<"Age:\t"<<age<< endl;
    cout<<"Gender:\t"<<gender<< endl;
}
```





# 1. Dẫn xuất và kế thừa [...]

*//Khai báo lớp Student trong file student.h*

```
#include "person.h"
#include <string>
using namespace std;
class Student: public Person
{
    string lop;
    int id;
public:
    Student(string, int, string, string,int);
    void set_class(string c);
    string get_class();
    void set_id(int i);
    int get_id();
    void display();
};
```

# @ 1. Dẫn xuất và kế thừa [...]

*//Định nghĩa lớp Student trong file student.cpp*

```
#include "stdafx.h"
#include <iostream>
#include "student.h"
using namespace std;
Student::Student(string _n, int _a, string _g, string _l,
int _id):Person(_n,_a,_g){
    lop = _l;
    id = _id;
}
void Student:: set_class(string c){lop = c;}
string Student:: get_class(){ return lop;}
void Student:: set_id(int i){id = i;}
int Student:: get_id(){return id;}
void Student:: display(){
    Person::display();
    cout<<"class:\t"<<lop<<endl;
    cout<<"ID:\t"<<id<<endl;
}
```

# @ 1. Dẫn xuất và kế thừa [...]

```
// Khai báo lớp Lecture trong file lecture.h
#include "person.h"
#include <string>
using namespace std;
class Lecture:public Person{
    string faculty;
    int telnumber;
public:
    Lecture(string,int, string, string,int);
    void set_faculty(string f);
    string get_faculty();
    void set_telnumber(int tel);
    int get_telnumber();
    void display();
};
```

# @ 1. Dẫn xuất và kế thừa [...]

*//Định nghĩa lớp Lecture trong file lecture.cpp*

```
#include "stdafx.h"
#include <iostream>
#include "Lecture.h"
using namespace std;
#include "lecture.h"
Lecture::Lecture(string _n, int _a, string _g, string _f, int
_t):Person(_n,_a,_g){
    faculty = _f;
    telnumber = _t;
}
void Lecture:: set_faculty(string f){faculty = f;}
string Lecture:: get_faculty(){ return faculty;}
void Lecture:: set_telnumber(int tel){telnumber = tel;}
int Lecture:: get_telnumber(){return telnumber;}
void Lecture:: display(){
    Person::display();
    cout<<"Faculty:\t"<<faculty<<endl;
    cout<<"Telephone Number:\t"<<telnumber<<endl;
}
```



# @1. Dẫn xuất và kế thừa [...]

## *Chương trình minh họa sử dụng 1*

### *//Thực hiện trong file main.cpp*

```
#include "stdafx.h"
#include "student.h"
#include "Lecture.h"
void main(){
    Person per("John",21,"man");
    Student stu("Marry",22,"woman","Electronics1-K53",20080001);
    Lecture lec("Michel",22,"man","Electronics
Engineering",123456789);
    cout<<"Person:\t"<<per.get_name()<<"\t"<<per.get_age()
        <<"\t"<<per.get_gender()<<"\n\n";
    cout<<"Student:\t"<<stu.get_name()<<"\t"<<stu.get_age()<<"\t"

    <<stu.get_gender()<<"\t"<<stu.get_class()<<"\t"<<stu.get_id()
    <<"\n\n";
    cout<<"Lecture:\t"<<lec.get_name()<<"\t"<<lec.get_age()<<"\t"
        <<lec.get_gender() <<"\t"<<lec.get_faculty()<<"\t"
        <<lec.get_telnumber()<<"\n\n";
}
```



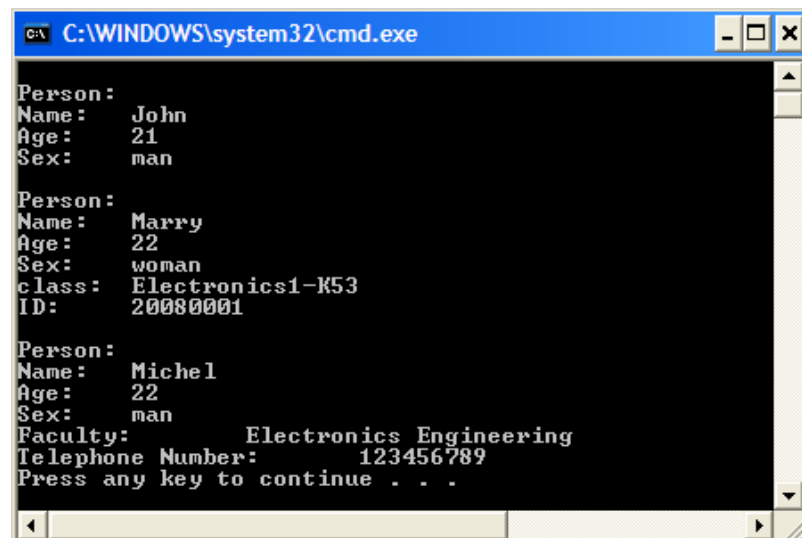
# @ 1. Dẫn xuất và kế thừa [...]

## Chương trình minh họa sử dụng 2

*//Thực hiện trong file main.cpp*

```
#include "stdafx.h"
#include "student.h"
#include "Lecture.h"
void main(){
    Person per("John",21,"man");
    Student stu("Marry",22,"woman","Electronics1-
    K53",20080001);
    Lecture lec("Michel",22,"man","Electronics
    Engineering",123456789);
    per.display();
    stu.display();
    lec.display();
}
```

- *Kết quả chạy chương trình*



```
C:\WINDOWS\system32\cmd.exe

Person:
Name:   John
Age:    21
Sex:    man

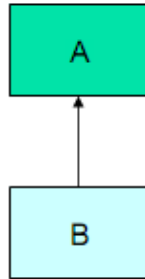
Person:
Name:   Marry
Age:    22
Sex:    woman
class:  Electronics1-K53
ID:     20080001

Person:
Name:   Michel
Age:    22
Sex:    man
Faculty: Electronics Engineering
Telephone Number: 123456789
Press any key to continue . . .
```

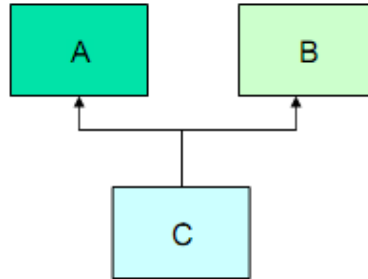


# 1. Dẫn xuất và kế thừa [...]

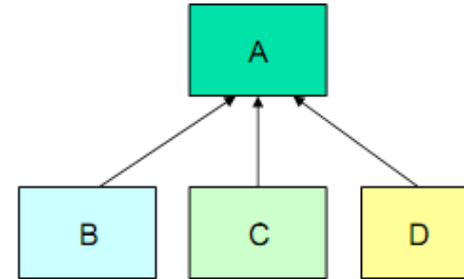
## *Các dạng dẫn xuất/ thừa kế*



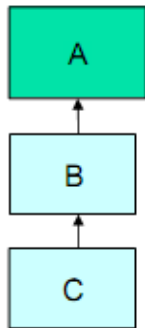
(a) Single Inheritance



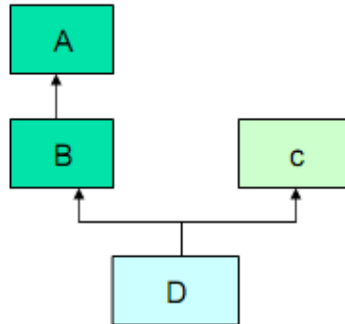
(b) Multiple Inheritance



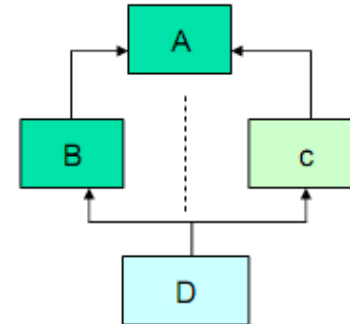
(c) Hierarchical Inheritance



(a) Multi-Level Inheritance



(b) Hybrid Inheritance



(b) Multipath Inheritance

- Lớp dẫn xuất có hai mục đích cơ bản
  - Mở rộng các tính năng của lớp cơ sở
  - Thừa hưởng các thuộc tính và phép toán của lớp cơ sở
- Ưu điểm của cơ chế thừa hưởng
  - Xây dựng một mô hình phần mềm hướng đối tượng dễ hiểu
  - Tiết kiệm được công việc thực hiện qua sử dụng lại các lớp cơ sở
  - Hạn chế lỗi qua cơ chế thừa hưởng



# 1. Dẫn xuất và kế thừa [...]

## *Hàm tạo và hàm tạo bản sao*

```
class A{
    int n;
    int *data;
public:
    A(int i = 0):n(i){
        data = new int[n];
    }
    A(const A& a){...}
    ~A(){ delete [] data;}
    ...
};
```

```
class B: public A{
    int m;
public:
    B(int i=0, int j=0):A(i),m(j){}
    B(const B& b):A(b), m(b.m){}
    ...
};
```

- Hàm tạo, hàm tạo bản sao không thừa hưởng được mà chỉ có thể gọi ở phần liệt kê khởi tạo (sau dấu :)
- B không định nghĩa thêm các biến thành viên thì B vẫn phải định nghĩa hàm tạo; chỉ trừ trường hợp A chỉ có hàm tạo mặc định (do compiler sinh ra) thì nó sẽ gọi hàm tạo mặc định của lớp cơ sở



# 1. Dẫn xuất và kế thừa [...]

## *Hàm tạo và hàm tạo bản sao[...]*

...

**B()** : m(j) {} *//không gọi hàm tạo của lớp cơ sở*

...

- Nếu không gọi hàm tạo của lớp cơ sở thì compiler sẽ bổ sung thêm lệnh gọi hàm tạo mặc định của lớp cơ sở. A()
- Nếu không định nghĩa hàm tạo bản sao ở lớp dẫn xuất thì hàm do compiler tạo ra sẽ gọi hàm tạo sao chép của lớp cơ sở và phần còn lại sẽ sao chép theo kiểu từng bit của các biến B định nghĩa thêm.
- Nếu trong B không định nghĩa thêm biến có sử dụng bộ nhớ động thì không cần thiết viết hàm tạo bản sao cho B.





# 1. Dẫn xuất và kế thừa [...]

## *Hàm hủy*

- Nếu không định nghĩa hàm hủy cho lớp dẫn xuất B thì compiler sẽ tự sinh ra và gọi hàm hủy của A
- Quá trình hủy ngược lại với quá trình tạo, phần tạo trước của lớp cơ sở sẽ được hủy sau.
- Sự cần thiết định nghĩa lại hàm hủy ở lớp dẫn xuất cũng giống như một lớp bình thường



# @ 1. Dẫn xuất và kế thừa [...]

## *Hàm toán tử gán*

```
...  
B& operator=(const B& b) {  
    A::operator=(b); //hàm toán tử gán của lớp cơ sở  
    m = b.m;          //biến B định nghĩa thêm  
    return (*this);  
}  
...
```

- Nếu không định nghĩa lại hàm toán tử gán thì compiler sẽ tự sinh ra cho ta hàm toán tử gán, hàm này sẽ gọi hàm toán tử gán của lớp cơ sở và sau đó sẽ gán từng bit một của các biến mà lớp dẫn xuất định nghĩa thêm.
- Nếu ta định nghĩa lại ở lớp dẫn xuất thì cũng cần gọi hàm toán tử gán của lớp cơ sở
- Việc cần định nghĩa lại hàm toán tử gán cũng giống như việc cần định nghĩa lại hàm tạo bản sao



## 2. Hàm ảo và cơ chế đa hình/đa xạ

**Nghiên cứu chương trình minh họa sử dụng các lớp Person, Student và Lecture ở trên**

*//Thực hiện trong file main.cpp*

```
void main(){
    Person *per = new Person ("John",21,"man");
    Person *stu = new Student ("Marry",22,"woman",
                               "Electronics1-K53",20080001);
    Person *lec = new Lecture ("Michel",22,"man",
                               "Electronics
                               Engineering",123456789);

    per->display();
    stu->display();
    lec->display();
    delete per; delete stu; delete lec;
}
```



## 2. Hàm ảo và cơ chế đa hình/đa xạ(...)

### Kết quả chạy chương trình

```
C:\WINDOWS\system32\cmd.exe

Person:
Name:   John
Age:    21
Sex:    man

Person:
Name:   Marry
Age:    22
Sex:    woman

Person:
Name:   Miche1
Age:    22
Sex:    man
Press any key to continue . . .
```

*Tại sao lại  
vậy?*



## 2. Hàm ảo và cơ chế đa hình/đa xạ(...)

- Nguyên nhân: Trong quá trình liên kết, lời gọi các hàm và hàm thành viên thông thường được chuyển thành các lệnh nhảy tới địa chỉ cụ thể của mã thực hiện hàm => "liên kết tĩnh"
- Giải pháp: sử dụng hàm ảo
- Hàm ảo là hàm thành viên của một lớp mà phần mã thực hiện nó được xác định đúng cho đối tượng định nghĩa nó trong khi chương trình chạy, kể cả trong trường hợp ta gọi hàm đó qua một con trỏ vào một lớp cơ sở (lớp cơ sở đã khai báo hàm ảo đó).





## 2. Hàm ảo và cơ chế đa hình/đa xạ(...)

### Sửa lại chương trình như sau:

- Chỉ cần khai báo hàm display của lớp person là hàm ảo bằng cách thêm chữ virtual vào trước hàm display trong phần khai báo lớp Person
- Các phần còn lại giữ nguyên

```
//Khai báo lớp Person trong file person.h  
#include <iostream>  
#include <string>  
using namespace std;  
class Person{  
    ... //giữ nguyên như cũ  
public:  
    ... //giữ nguyên như cũ  
    virtual void display()  
};
```



## 2. Hàm ảo và cơ chế đa hình/đa xạ(...)

### Kết quả chạy chương trình

```
C:\WINDOWS\system32\cmd.exe

Person:
Name: John
Age: 21
Sex: man

Person:
Name: Marry
Age: 22
Sex: woman
class: Electronics1-K53
ID: 20080001

Person:
Name: Michel
Age: 22
Sex: man
Faculty: Electronics Engineering
Telephone Number: 123456789
Press any key to continue . . .
```

*Như mong  
đợi chưa?*



### 3. Hàm thuần ảo, lớp thuần ảo

- Hàm thuần ảo (hàm trừu tượng) là hàm ảo có khai báo mà không có định nghĩa
- Lớp thuần ảo (lớp trừu tượng) là lớp có ít nhất một hàm thuần ảo
- Lớp thuần ảo chỉ là giao diện, không sử dụng được
- Bắt buộc phải định nghĩa lớp dẫn xuất
- Như vậy:
  - Phân biệt rõ phần giao diện và phần thực hiện
  - Có thể công khai phần giao diện cho người sử dụng, che giấu phần thực hiện
  - Có thể thay đổi phần thực hiện mà không ảnh hưởng đến cách sử dụng



### 3. Hàm thuần ảo, lớp thuần ảo

*//Ví dụ 1*

```
class A{
    int a;
public:
    virtual void f() = 0;
};
class B: public A{
    int b;
public:
    void f(){ b = 0;}
    void h(){}
};
void main(){
    A a; //Lỗi
    ...
}
```

*//Ví dụ 2*

```
class A{
    int a;
public:
    virtual void f() = 0;
};
class B: public A{
    int b;
public:
    void f(){ b = 0;}
    void h(){}
};
void main(){
    B b;    //OK
    b.f();  //OK
    ...
}
```



# Hàm hủy là hàm ảo

- Hàm hủy lớp cơ sở là ảo thì có thể dùng con trỏ lớp cơ sở để hủy đối tượng lớp dẫn xuất
- Nếu hàm hủy lớp cơ sở là ảo thì hàm hủy lớp dẫn xuất cũng tự động là ảo.





# Hàm hủy là hàm ảo

```
class A{
    int n;
    int *data;

public:
    A(int _n):n(_n){ data = new int[n];}
    ~A(){ delete [] data;}
};

class B: public A{
    int m;
    int* data;

public:
    B(int _n,int _m):A(_n),m(_m){data = new int[m];}
    ~B(){ delete [] data; }
};

void main(){
    A *pb = new B(5,5);
    ...
    delete pb;
}
```

Gọi ~A(); không gọi ~B() →  
không hủy pb->data

**Giải pháp: thêm virtual trước ~A()**



# Hàm hủy là hàm ảo

```
class A{
    int n;
    int *data;

public:
    A(int _n):n(_n){ data = new int[n];}
    virtual ~A(){delete [] data;}
};

class B: public A{
    int m;
    int* data;

public:
    B(int _n,int _m):A(_n),m(_m){data = new int[m];}
    ~B(){ delete [] data; }
};

void main(){
    A *pb = new B(5,5);
    ...
    delete pb;
}
```

Gọi ~B(); sau đó gọi  
~A()



## @ 4. Kiểm soát truy nhập

### a) Kế thừa dạng public

```
class A{  
    int a;  
    void g();  
Public:  
    int n;  
    void f();  
Protected:  
    char c;  
    void h();  
};
```

```
class B: public A{  
    ...  
};
```

- Tất cả các thành viên public và protected của A giữ nguyên quyền kiểm soát truy nhập
  - public của A → public của B
  - protected của A → protected của B
  - Private của A → B thừa hưởng nhưng không truy nhập trực tiếp được
- Friend không thừa hưởng được; friend của lớp nào thì chỉ có ý nghĩa của lớp đó

## @ 4. Kiểm soát truy nhập

### b) Kế thừa dạng protected

```
class A{  
    ...  
};  
class B: protected A{  
    ...  
};
```

- public của A → protected của B
- protected của A → private của B
- B thừa hưởng các private của A nhưng không truy nhập trực tiếp được.

```
class C: public B{  
    void g2(){  
        h();    //ok; mac du h() trở thành private của B  
        f();    //OK; f(x) trở thành protected của B  
    }  
};  
void func(C c){  
    c.n = 1; //LỖI; n là protected của B  
    c.f();  //LỖI; f() là protected của B  
}
```



## 4. Kiểm soát truy nhập

### c) Kế thừa dạng private

```
class A{  
    ...  
};  
class B: private A{  
    ...  
};
```

- Mặc định private ta có thể viết

```
class A{  
    ...  
};  
class B: A{  
    ...  
};
```

- Tất cả các thành viên public và protected của A trở thành private của B
- B thừa kế các private của A nhưng không truy nhập trực tiếp được





## 5. Tương thích kiểu

```
class A{  
    ...  
};  
class B: public A{  
    ...  
};
```

- Ví dụ 1

```
B b;  
A a = b; //OK  
void f(A a){...}  
f(b);    //OK
```

Gọi hàm tự  
sao chép  
A::A(const&)

- Ví dụ 2

```
A a;  
B b1 = a;  
//LỖI
```

## @ 5. Tương thích kiểu

- Ví dụ 2

```
void g(A *pa){...}  
void main(){  
    B b;  
    g(&b);    // OK  
}
```

Đ/c của b có kiểu là con trỏ B (B\*).  
Con trỏ vào B sẽ được tự động chuyển đổi kiểu sang con trỏ A\*

- Kiểu con trỏ hoặc tham chiếu vào một kiểu dẫn xuất có thể tự động chuyển đổi thành con trỏ/tham chiếu vào kiểu cơ sở, ngược lại không đúng.

```
A a;  
B b;  
A *pa = &b; //OK; tự động chuyển đổi kiểu  
B *pb = pa; //LỖI;  
pb = (B*)pa; // khi biên dịch thì compiler không phát hiện ra lỗi;  
khi chạy có lỗi
```

Thực hiện lại trên máy tính các lớp Person, Student, Lecture theo sườn bài giảng.