LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Chương 6: Kế thừa (p1)

Trình bày: ThS. Lê Thanh Trọng

NỘI DUNG

1. Quan hệ giữa các lớp đối tượng
2. Kế thừa
3. Kế thừa đơn
4. Ràng buộc ngữ nghĩa ở lớp con

NỘI DUNG

1. **Quan hệ giữa các lớp đối tượng**
2. Kế thừa
3. Kế thừa đơn
4. Ràng buộc ngữ nghĩa ở lớp con

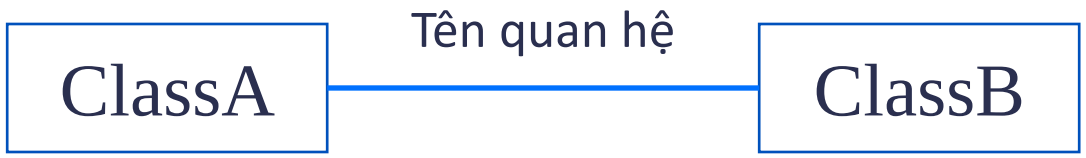
Quan hệ giữa các lớp đối tượng

- Giữa các lớp đối tượng có những loại quan hệ sau:
  - 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)

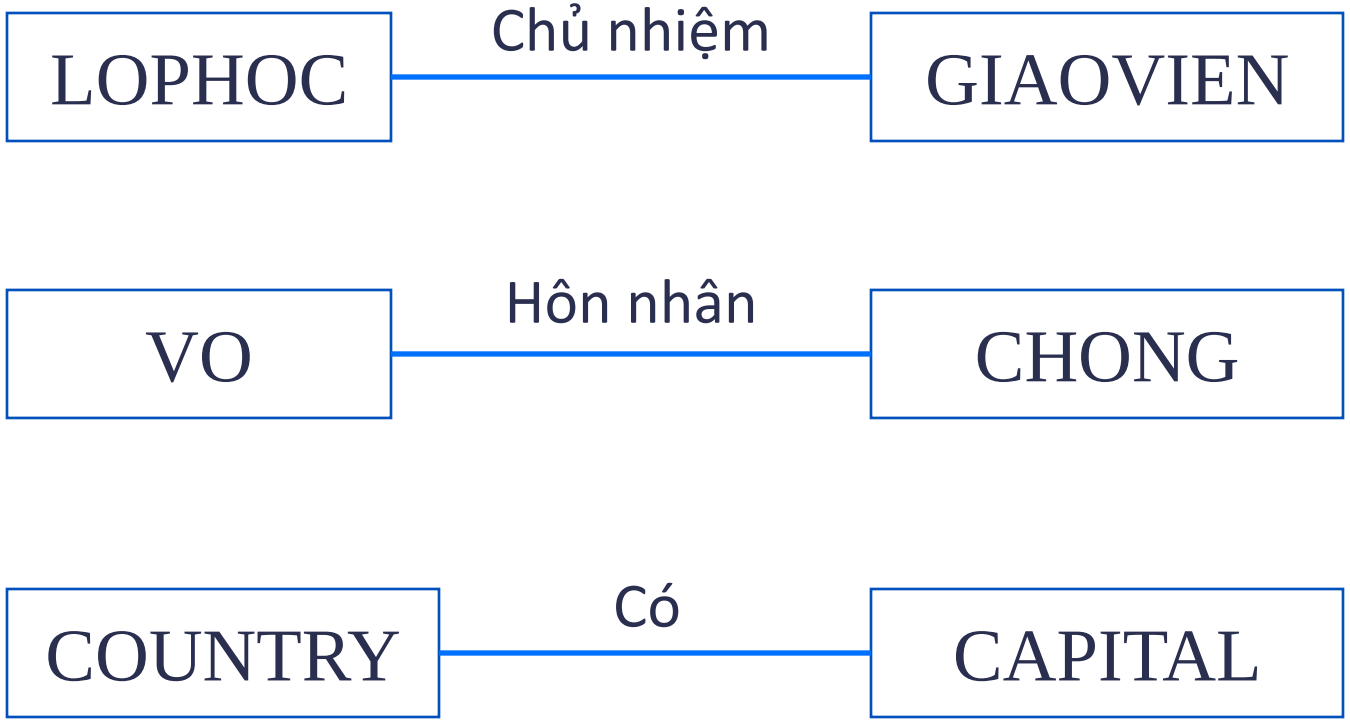
❖Khái niệm: Hai lớp đối tượng được gọi là có quan hệ một-một với nhau khi một đối tượng thuộc lớp này quan hệ với một đối tượng thuộc lớp kia và một đối tượng thuộc lớp kia có quan hệ duy nhất với một đối tượng thuộc lớp này.

❖Ký hiệu:



Quan hệ một một (1-1)

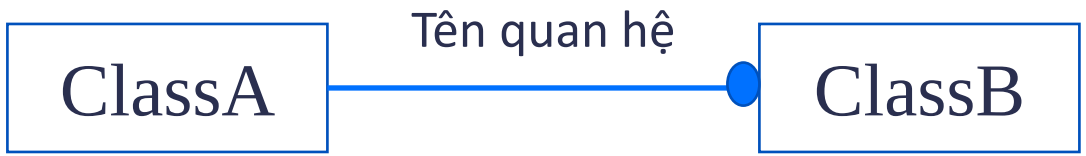
❖Ví dụ:



Quan hệ một nhiều (1-n)

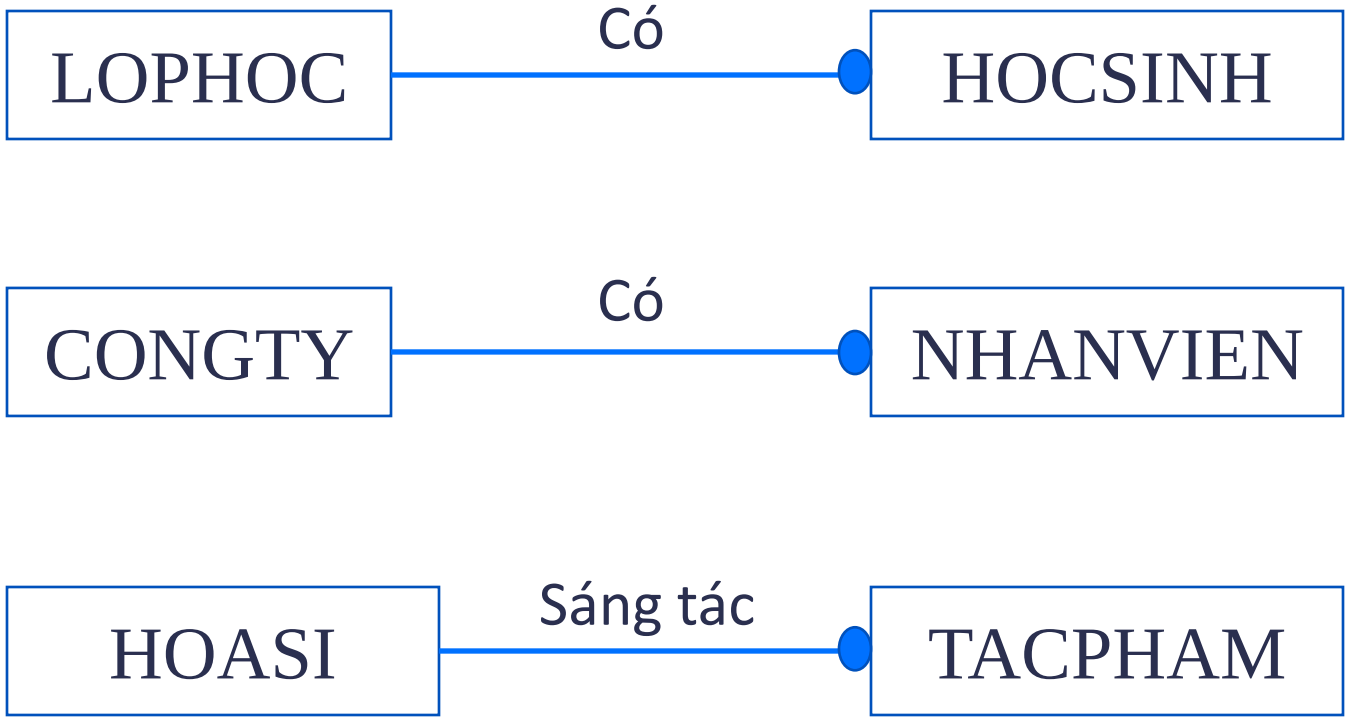
❖Khái niệm: Hai lớp đối tượng được gọi là có quan hệ **một-nhiều** với nhau khi một đối tượng thuộc lớp này quan hệ với nhiều đối tượng thuộc lớp kia và một đối tượng lớp kia có quan hệ duy nhất với một đối tượng thuộc lớp này.

❖Kí hiệu:



Quan hệ một nhiều (1-n)

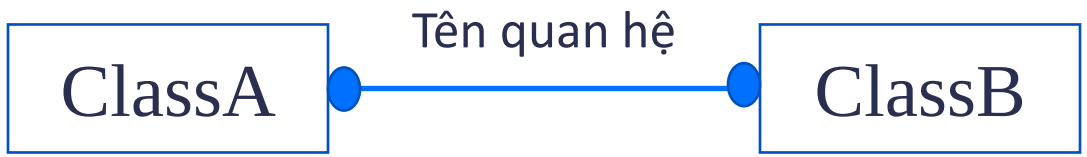
❖Ví dụ:



Quan hệ nhiều nhiều (n-n)

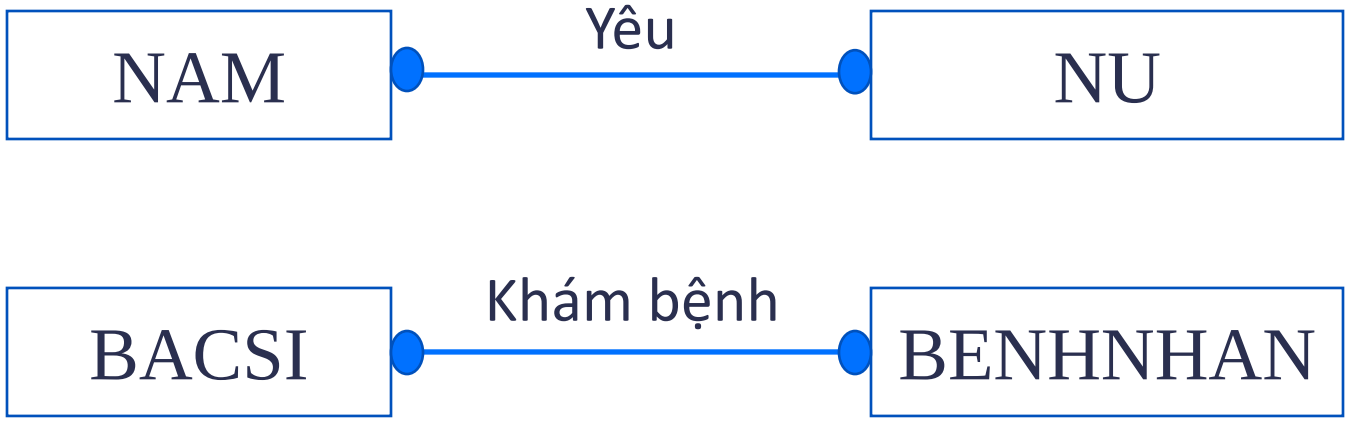
❖Khái niệm: hai lớp đối tượng được gọi là quan hệ nhiều-nhiều với nhau khi một đối tượng thuộc lớp này có quan hệ với nhiều đối tượng thuộc lớp kia và một đối tượng lớp kia cũng có quan hệ với nhiều đối tượng thuộc lớp này.

❖Kí hiệu



Quan hệ nhiều nhiều (n-n)

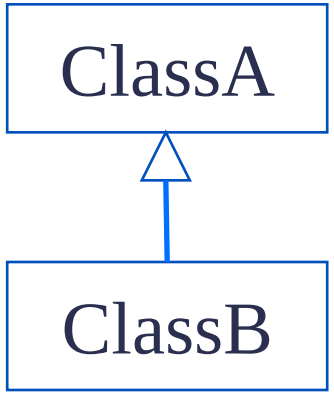
❖Ví dụ



Quan hệ đặc biệt hóa – tổng quát hóa

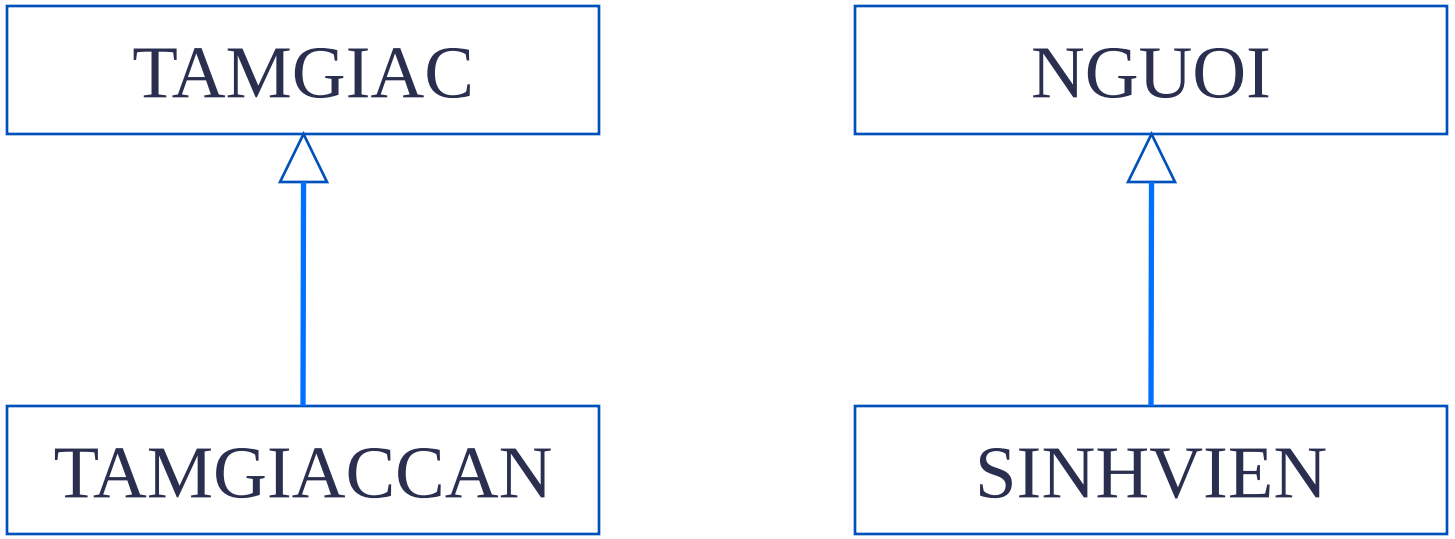
❖Khái niệm: hai lớp đối tượng được gọi là có quan hệ **đặc biệt hóa-tổng quát hóa** với nhau khi lớp đối tượng này là trường hợp đặc biệt của lớp đối tượng kia và lớp đối tượng kia là trường hợp tổng quát của lớp đối tượng này.

❖Kí kiệu:

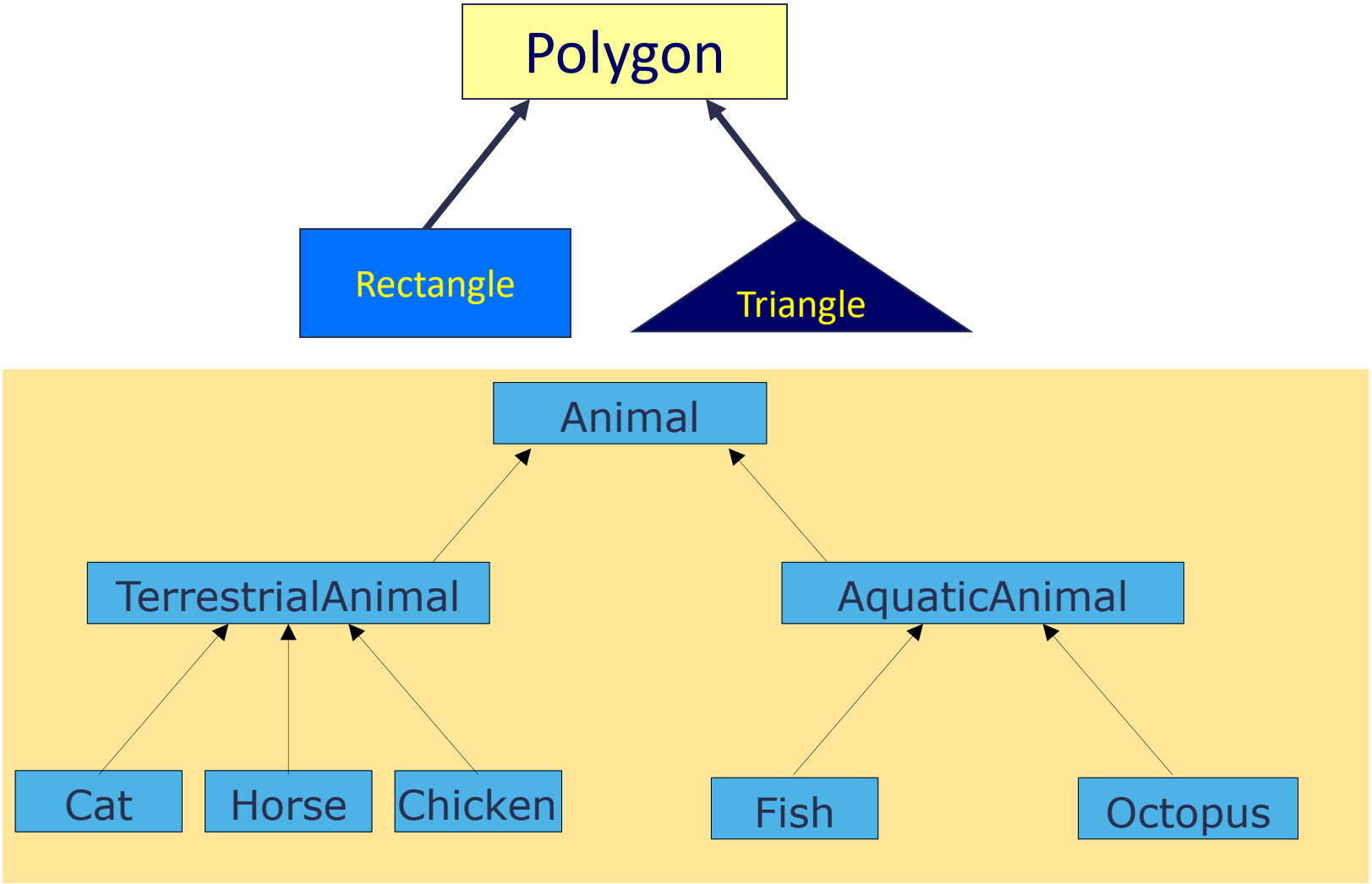


Quan hệ đặc biệt hóa – tổng quát hóa

❖Ví dụ:



Quan hệ đặc biệt hóa – tổng quát hóa



NỘI DUNG

1. Quan hệ giữa các lớp đối tượng
2. **Kế thừa**
3. Kế thừa đơn
4. Ràng buộc ngữ nghĩa ở lớp con

Kế thừa

- Kế thừa là một đặc điểm của ngôn ngữ dùng để biểu diễn mối quan hệ **đặc biệt hóa – tổng quát hóa** giữa các lớp
- Sự kế thừa là một mức **cao hơn của trừu tượng hóa**, cung cấp một cơ chế **gom chung các lớp có liên quan** với nhau thành một mức khái quát hóa đặc trưng cho toàn bộ các lớp

Kế thừa

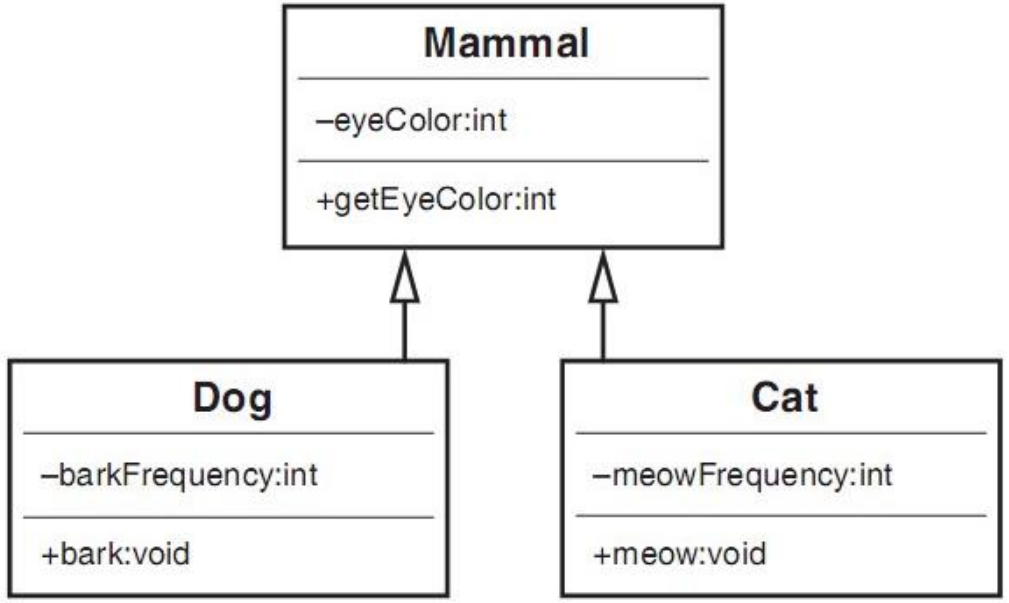
- Các lớp với các đặc điểm tương tự nhau có thể được tổ chức thành một sơ đồ phân cấp kế thừa (cây kế thừa)
- Quan hệ "**là 1**": Kế thừa được sử dụng thông dụng nhất để biểu diễn quan hệ "là 1"
  - Một sinh viên **là một** người
  - Một hình tròn **là một** hình ellipse
  - Một tam giác **là một** đa giác
  - …

Lợi ích 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
- Trong C++, kế thừa còn định nghĩa sự tương thích, nhờ đó ta có cơ chế chuyển kiểu tự động

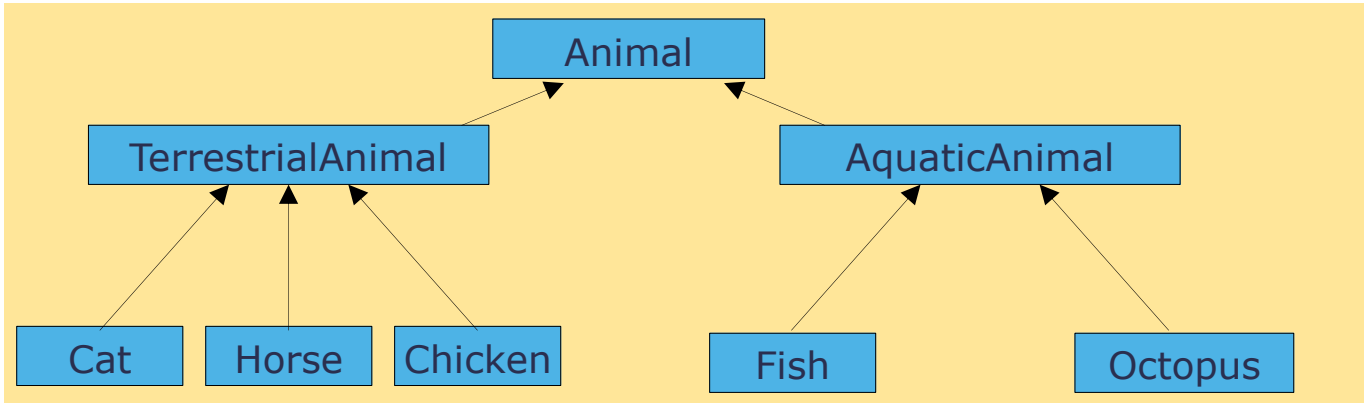
Đặc tính Kế thừa

- Cho phép định nghĩa lớp mới từ lớp đã có
  - Lớp mới gọi là lớp con (subclass) hay lớp dẫn xuất (derived class)
  - Lớp đã có gọi là lớp cha (superclass) hay lớp cơ sở (base class)



Đặc tính Kế thừa

- Thừa kế cho phép:
  - Nhiều lớp có thể dẫn xuất từ một lớp cơ sở
  - Một lớp có thể là dẫn xuất của nhiều lớp cơ sở
- Thừa kế không chỉ giới hạn ở một mức: Một lớp dẫn xuất có thể là lớp cơ sở cho các lớp dẫn xuất khác

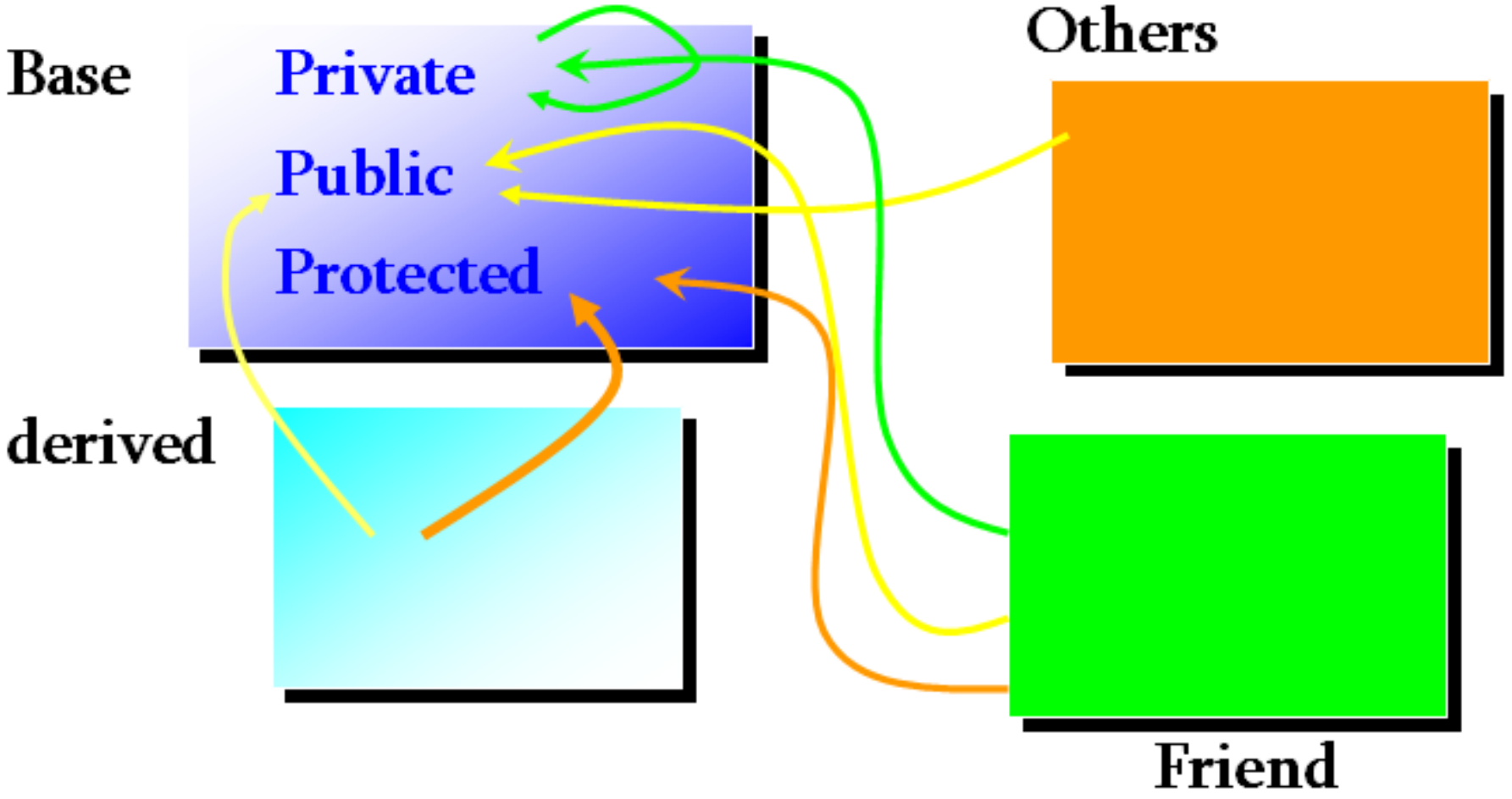


Cú pháp khai báo kế thừa

```
class SuperClass{
        //Thành phần của lớp cơ sở
};

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

Truy cập thành viên của lớp



NỘI DUNG

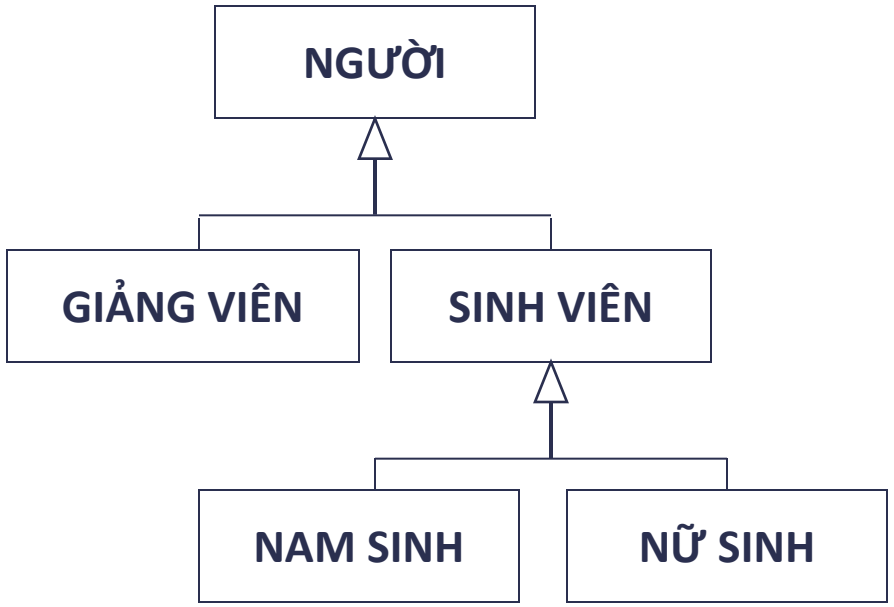
1. Quan hệ giữa các lớp đối tượng
2. Kế thừa
3. **Kế thừa đơn**
4. Ràng buộc ngữ nghĩa ở lớp con

Kế thừa đơn

- Xét hai khái niệm **Người** và **Sinh viên** với mối quan hệ tự nhiên: Một Sinh viên **là một** Người
- Trong C++, ta có thể biểu diễn khái niệm trên, một sinh viên là một người có thêm một số thông tin và một số thao tác (riêng biệt của sinh viên)
- Như vậy, ta tổ chức lớp Sinh viên kế thừa từ lớp Người

Kế thừa đơn

❖Ta có thể tổ chức hai lớp Nam sinh và Nữ sinh là hai lớp con (lớp dẫn xuất) của lớp Sinh viên. Trường hợp này, lớp Sinh viên trở thành ớp cha (lớp cơ sở) của hai lớp trên



Kế thừa đơn – Ví dụ

```
class Nguoi {
    char *HoTen;
    int NamSinh;
public:
    Nguoi();
    Nguoi( char *ht, int ns):NamSinh(ns) {HoTen=strdup(ht);}
    ~Nguoi() {delete [ ] HoTen;}
    void An() const { cout<<HoTen<<" an 3 chen com \n";}
    void Ngu() const { cout<<HoTen<< " ngu ngay 8 tieng \n";}
    void Xuat() const;
    friend ostream& operator << (ostream &os, Nguoi& p);
};
```

Kế thừa đơn – Ví dụ

```
class SinhVien : public Nguoi {
    char *MaSo;
public:
    SinhVien();
    SinhVien( char *ht, char *ms, int ns) : Nguoi(ht,ns) {
            MaSo = strdup(ms);
    }
    ~SinhVien() {
            delete [ ] MaSo;
    }
    void Xuat() const;
};
```

Kế thừa đơn – Ví dụ

```
void Nguoi::Xuat() const
{
    cout << "Nguoi, ho ten: " << HoTen;
    cout  << " sinh " << NamSinh;
    cout << endl;
}
void SinhVien::Xuat() const {
    Nguoi::Xuat();
    cout << "Sinh vien, ma so: " << MaSo;
    cout << endl;
}
```

Kế thừa đơn – Ví dụ

```
void main() {
    Nguoi p1("Le Van Nhan",1980);
    SinhVien s1("Vo Vien Sinh", "200002541",1984);
    cout << "1.\n";
    p1.An();                    s1.An();
    cout << "2.\n";
    p1.Xuat();                  s1.Xuat();
    s1.Nguoi::Xuat();
    cout << "3.\n";
    cout << p1 << "\n";
    cout << s1 << "\n";
}
```

Kế thừa đặc tính của lớp cha

- Khai báo

```
class SinhVien : public Nguoi {
    //...
};
```

- Cho biết lớp Sinh viên kế thừa từ lớp Người. Khi đó Sinh viên thừa hưởng các đặc tính của lớp Người
- Về mặt dữ liệu: Mỗi đối tượng Sinh viên tự động có thành phần dữ liệu họ tên, năm sinh của người

Kế thừa đặc tính của lớp cha

- Về mặt thao tác: Lớp Sinh viên được tự động kế thừa các thao tác của lớp cha. Đây chính là khả năng sử dụng lại mã chương trình
- Riêng phương thức thiết lập không được kế thừa
- Khả năng thừa hưởng các thao tác của lớp cơ sở có thể được truyền qua “vô hạn mức”

NỘI DUNG

1. Quan hệ giữa các lớp đối tượng
2. Kế thừa
3. Kế thừa đơn
4. **Ràng buộc ngữ nghĩa ở lớp con**

Định nghĩa lại thao tác ở lớp con

❖Ta có thể định nghĩa lại các đặc tính ở lớp con đã có ở lớp cha, việc định nghĩa chủ yếu là thao tác

```
class SinhVien : public Nguoi {
    char *MaSo;
public:
    //...
    void Xuat() const;
};
void SinhVien::Xuat() const {
    cout << "Sinh vien, ma so: " << MaSo << ", ho ten: " << HoTen;
}
```

Ràng buộc ngữ nghĩa ở lớp con

- Có thể áp dụng quan hệ kế thừa mang ý nghĩa ràng buộc, đối tượng ở lớp con là đối tượng ở lớp cha nhưng có dữ liệu bị ràng buộc:
  - Hình tròn là Ellipse với ràng buộc **bán kính ngang dọc bằng nhau**
  - Số ảo là số phức với ràng buộc **phần thực bằng 0**
  - …

Ví dụ

```
class Complex {
    friend ostream& operator <<(ostream&, Complex);
    friend class Imag;
    double re, im;
public:
    Complex( double r = 0, double i = 0):re(r), im(i){ }
    Complex operator +(Complex b);
    Complex operator -(Complex b);
    Complex operator *(Complex b);
    Complex operator /(Complex b);
    double Norm() const { return sqrt(re*re + im*im);}
};
```

Ví dụ

```
class Imag: public Complex {
public:
    Imag(double i = 0):Complex(0, i){ }
    Imag(const Complex &c) : Complex(0, c.im){ }
    Imag& operator = (const Complex &c){
        re = 0; im = c.im;
        return *this;
    }
    double Norm() const {
        return fabs(im);
    }
};
```

Ví dụ

```
void main()
{
    Imag i = 1;
    Complex z1(1,1)
    Complex z3 = z1 - i;          // z3 = (1,0)
    i = Complex(5,2);             // i la so ao (0,2)
    Imag j = z1;                  // j la so ao (0,1)
    cout << "z1 = " << z1 << "\n";
    cout << "i = " << i << "\n";
    cout << "j = " << j << "\n";
}
```

Ràng buộc ngữ nghĩa ở lớp con

- Trong ví dụ trên, lớp số ảo (Imag) kế thừa hầu hết các thao tác của lớp số phức (Complex)
- Tuy nhiên, ta muốn ràng buộc mọi đối tượng thuộc lớp số ảo đều phải có phần thực bằng 0. Vì vậy, phải định nghĩa lại các hàm thành phần có thể vi phạm điều này
- Ví dụ phép toán gán phải được định nghĩa lại để đảm bảo ràng buộc này

Tóm tắt bài học kế thừa (p1)

- Kế thừa là một đặc điểm của ngôn ngữ dùng để biểu diễn mối quan hệ đặc biệt hóa – tổng quát hóa giữa các lớp (mối quan hệ **"là 1"**)
- Sự kế thừa là một mức cao hơn của trừu tượng hóa, cung cấp một cơ chế gom chung các lớp có liên quan với nhau thành một mức khái quát hóa đặc trưng cho toàn bộ các lớp
- Ưu điểm 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, giảm sự trùng lặp, nhờ vậy có thể dễ dàng sửa chữa, nâng cấp hệ thống,
  - Trong C++, kế thừa còn định nghĩa sự tương thích, nhờ đó ta có cơ chế chuyển kiểu tự động

Tóm tắt bài học kế thừa (p1)

- Cú pháp kế thừa

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

- Về mặt dữ liệu: Mỗi đối tượng Sinh viên tự động có thành phần dữ liệu họ tên, năm sinh của người
- Về mặt thao tác: Lớp Sinh viên được tự động kế thừa các thao tác của lớp cha. Đây chính là khả năng sử dụng lại mã chương trình
- Riêng phương thức thiết lập không được kế thừa
- Khả năng thừa hưởng các thao tác của lớp cơ sở có thể được truyền qua "vô hạn mức"

Tóm tắt bài học kế thừa (p1)

❖Đơn kế thừa(Single Inheritance):

- ❑Lớp con kế thừa từ một lớp cha duy nhất
- ❑Ví dụ: Lớp **Dog** kế thừa từ lớp **Animal**

❖Đa kế thừa (Multiple Inheritance):

- ❑Một lớp con kế thừa từ nhiều lớp cha
- ❑Ví dụ: Lớp **AmphibiousVehicle** kế thừa từ cả lớp **Car** và lớp **Boat**

❖Ràng buộc ngữ nghĩa ở lớp con

- ❑Có thể áp dụng quan hệ kế thừa mang ý nghĩa ràng buộc, đối tượng ở lớp con là đối tượng ở lớp cha nhưng có dữ liệu bị ràng buộc:
  - Hình tròn là Ellipse với ràng buộc **bán kính ngang dọc bằng nhau**
  - Số ảo là số phức với ràng buộc **phần thực bằng 0**

Quản lý nhân viên

Giả sử công ty có hai loại nhân viên: Nhân viên văn phòng và Nhân viên sản xuất. Mỗi nhân viên cần quản lý các thông tin sau: Họ tên, ngày sinh (ngày tháng năm). NV sản xuất có thêm thông tin: lương căn bản, số sản phẩm. NV văn phòng có thêm thông tin: số ngày làm việc.

- Công ty cần tính lương cho nhân viên như sau:
  - Đối với nhân viên sản xuất: **Lương = lương căn bản + số sản phẩm * 5.000**
  - Đối nhân viên văn phòng: **Lương = số ngày làm việc * 100.000**
- Viết chương trình cho phép nhập vào các nhân viên (N NV văn phòng và M NV sản xuất) và thực hiện các yêu cầu:
  - Xuất ra thông tin các nhân viên vừa nhập
  - Xuất tổng lương phải trả cho tất cả nhân viên
  - Xuất các nhân viên tăng dần theo số tuổi (tính đến năm hiện tại)