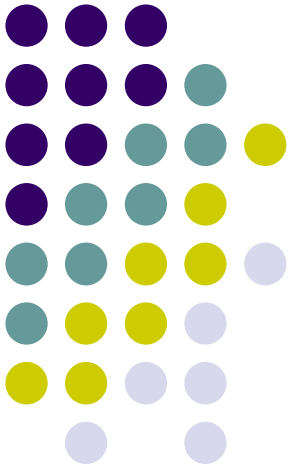
Chương 4

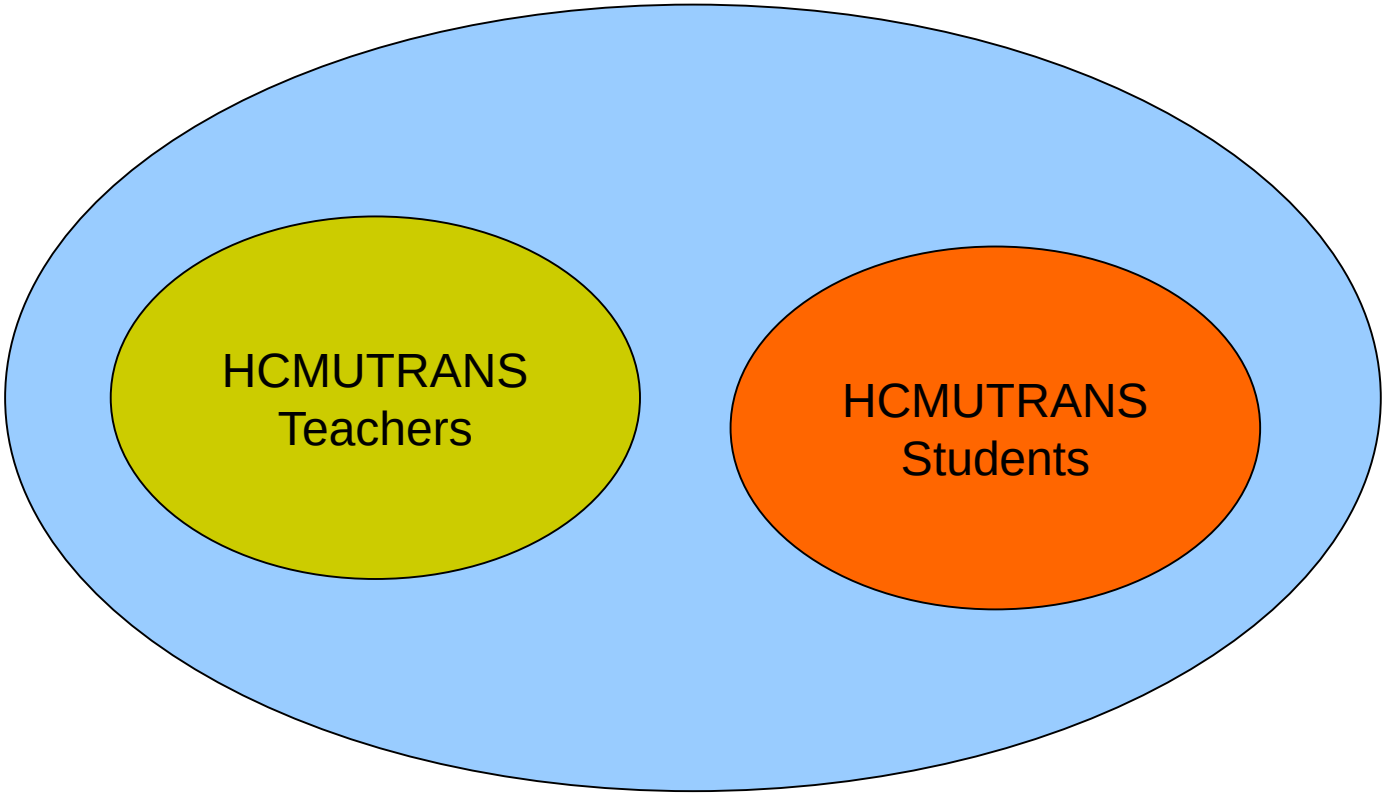
Kế thừa (Inheritance)



Nội dung

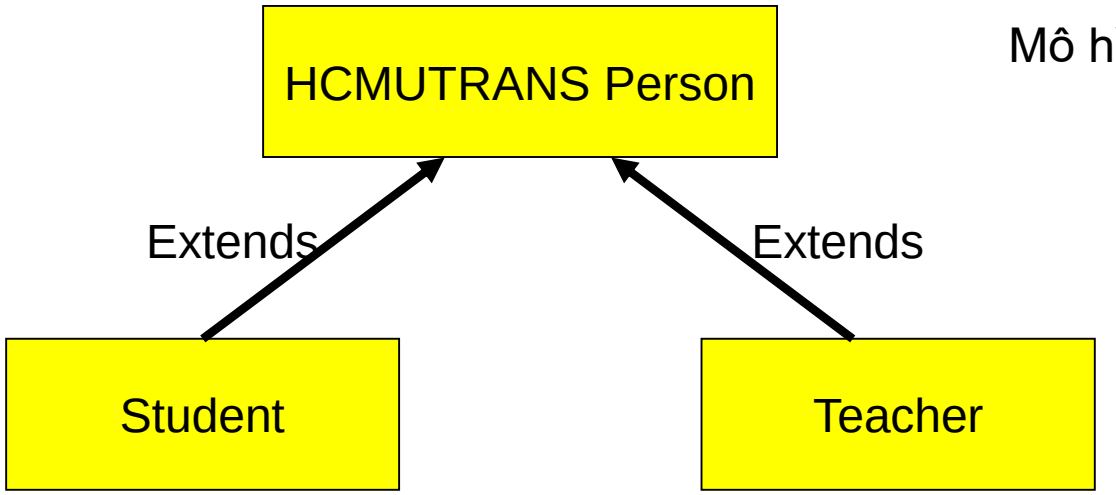
- Giới thiệu
- Lớp cơ sở và lớp dẫn xuất
- Xây dựng lớp dẫn xuất
- Thành viên Protected
- Các kiểu kế thừa
- Hàm tạo và hàm hủy trong lớp dẫn xuất
- Định nghĩa lại phương thức
- Đa kế thừa

Giới thiệu – Tính chất kế thừa



People at HCMUTRANS

Giới thiệu – Tính chất kế thừa

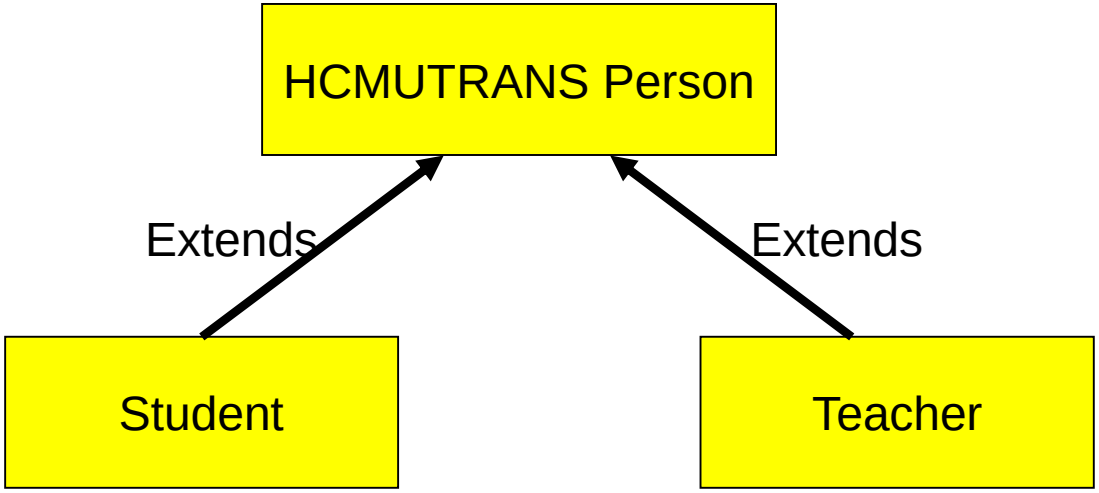


Mô hình phân cấp

Mọi người ở HCMUTRANS cần có những thuộc tính/hành vi chung nào?

- Thuộc tính: name, ID, address
- Hành vi: displayProfile(), changeAddr()

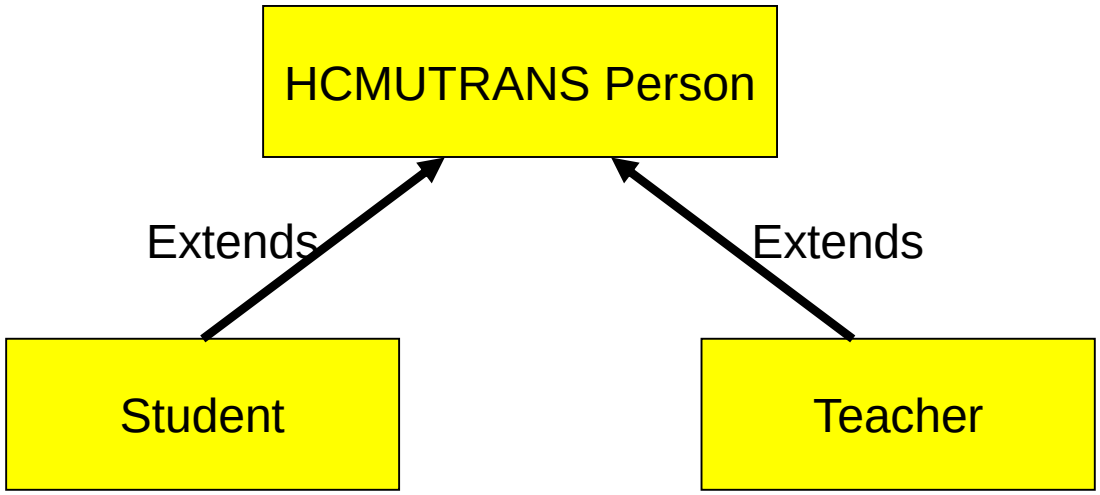
Giới thiệu – Tính chất kế thừa



Sinh viên cần có những thuộc tính/hành vi riêng nào?

- Thuộc tính: group, year, course
- Hành vi: changeGroup()

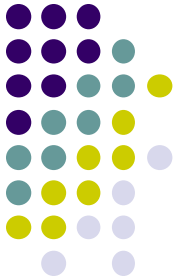
Giới thiệu – Tính chất kế thừa



Giảng viên cần có những thuộc tính/hành vi riêng nào?

- Thuộc tính: rank, year, subject, salary,….
- Hành vi: addSubj(), increaseSal()

Giới thiệu – Tính chất kế thừa

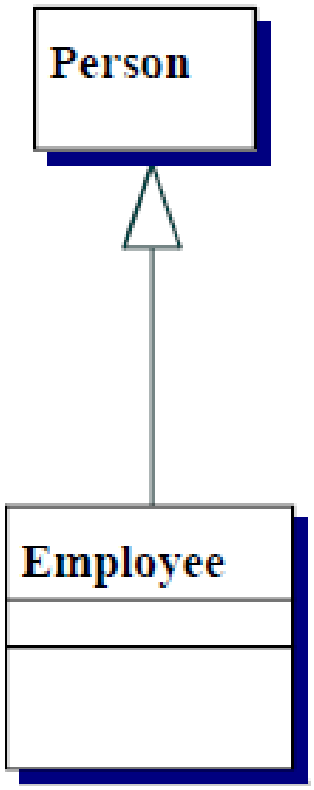


- Cho phép tạo lớp mới(**derived class**) từ lớp đã có(**base class**)
- Làm tăng khả năng tái sử dụng
- Cho phép tạo ra sự phân loại theo cấp bậc
- Là công cụ cho phép mô tả cụ thể hóa các khái niệm theo nghĩa

Base class và Derived class

- Mối quan hệ giữa Base class và Derived class là "is-A"
- Derived class: kế thừa tất cả các thành viên của Base class.

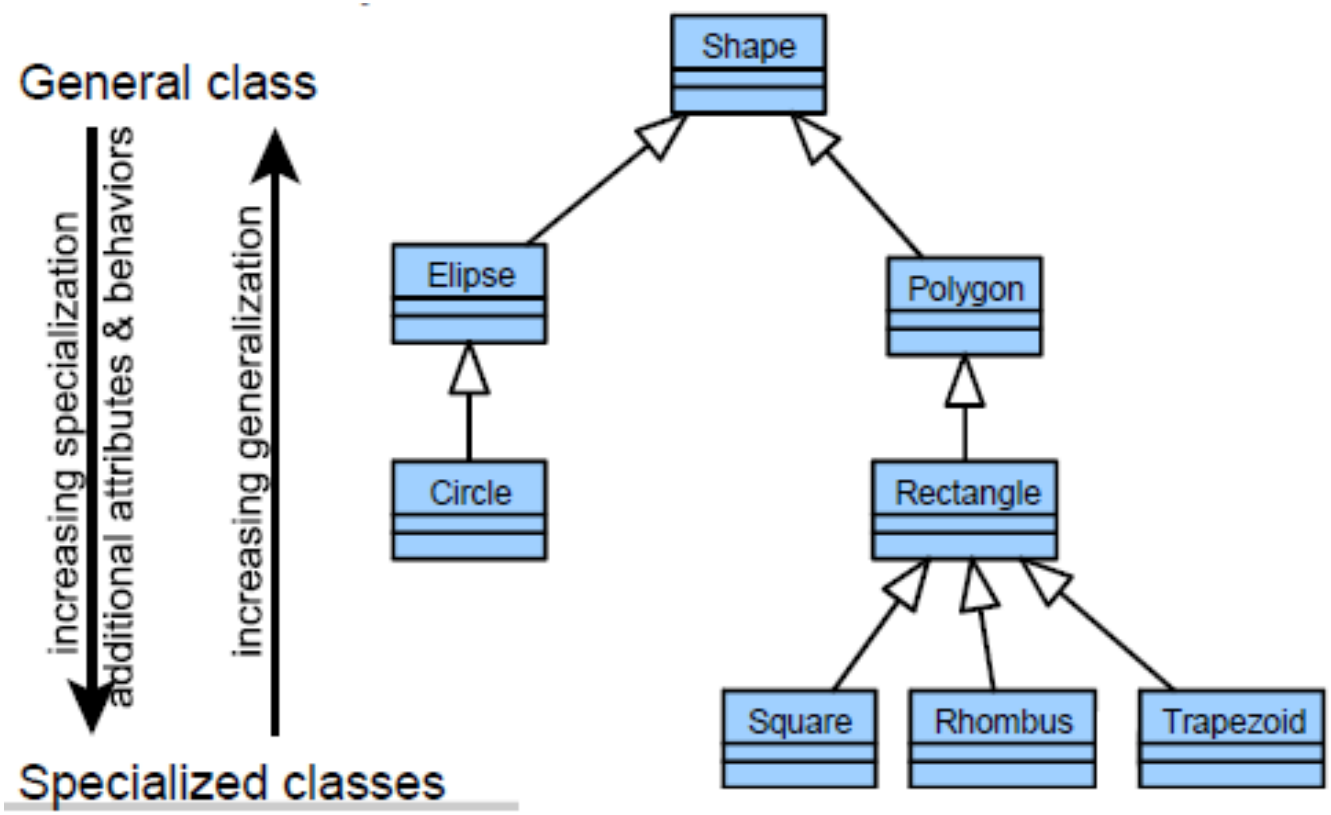
  Chú ý: hàm bạn không được kế thừa.
- Object của Derived class cũng chính là object của Base class
- Derived class có thể định nghĩa lại các phương thức của Base class



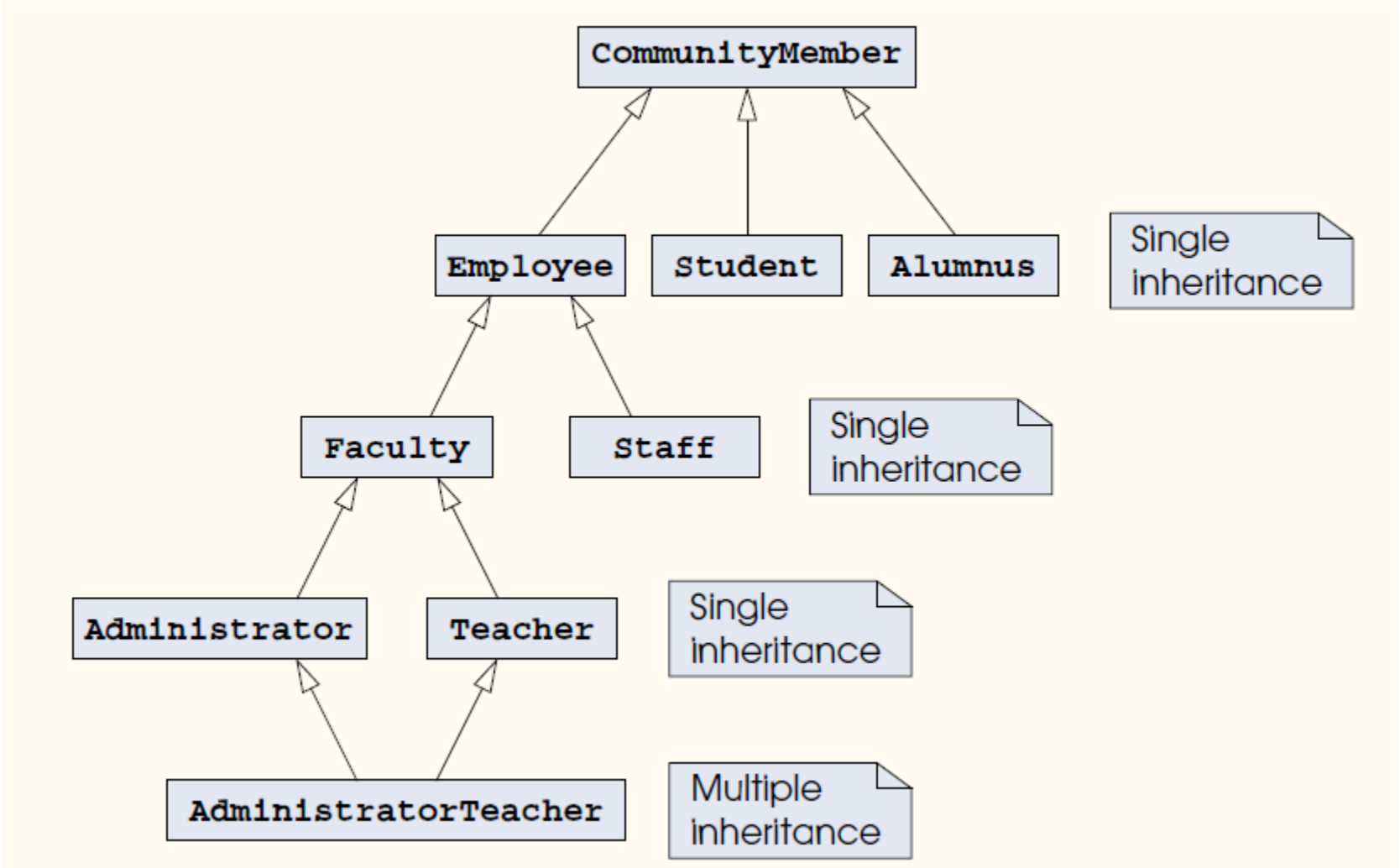
Base Class
Superclass
Parent Class
Generalization
Ancestor

Derived Class
Subclass
Child Class
Specialization
Descendant

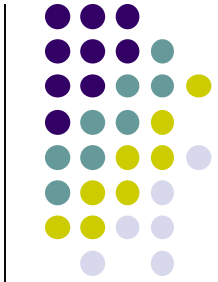
Base class và Derived class



Base class và Derived class



Xây dựng Derived class(lớp dẫn xuất)

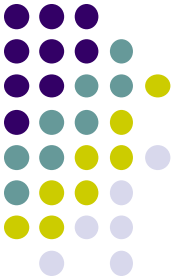


- Cú pháp

```
class <name> : <access-specifier> <basename> {

};
```

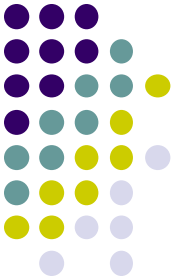
- <access-specifier>: kiểm soát truy xuất các thành viên được kế thừa
  - public
  - private (default)
  - Protected

Ví dụ



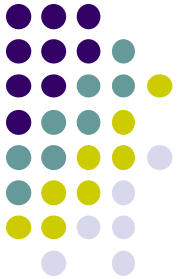
```
1  //Point.h
2  #pragma once
3  #include <iostream>
4  using namespace std;
5  class Point
6  {
7  private:
8      float x;
9      float y;
10 public:
11     //constructor and destructor
12     Point(float x = 0, float y = 0);
13     ~Point(void);
14     //setters and getters
15     void setX(float);
16     float getX() const;
17     void setY(float);
18     float getY() const;
19     //print
20     friend ostream& operator<<(ostream &output, const Point &p );
21
22 };
```

Ví dụ



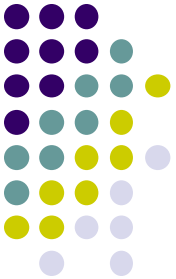
```
1  //Point.cpp
2  #include "Point.h"
3  Point::Point( float x, float y )
4  {
5      setX(x);
6      setY(y);
7  }
8
9  void Point::setX(float x) { ... }
14 float Point::getX() const { ... }
19 void Point::setY(float y) { ... }
24 float Point::getY() const { ... }
29 ostream& operator<<(ostream &output, const Point &p )
30 {
31     output << "[" << p.x << ", " << p.y << "]";
32     return output;
33 }
34
35 Point::~Point(void) { ... }
```

Ví dụ



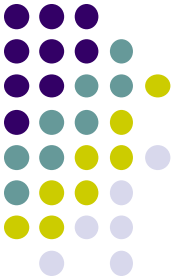
```
1  //circle.h
2  #pragma once
3  #include "Point.h"
4  class Circle: public Point //Lop Circle ke thua tu lop Point
5  {
6  //Lop Circle ke thua du lieu x,y tu lop Point
7  //va bo sung them du lieu radius
8  private:
9      float radius;
10 public:
11     Circle(float x = 0, float y = 0, float r = 0);
12     ~Circle(void);
13     void setR(float);
14     float getR() const;
15     float CalArea() const;
16 };
```

Ví dụ



```
1  //Circle.cpp
2  #include "Circle.h"
3  Circle::Circle(float x, float y, float r):Point(x,y)
4  {
5      setR(r);
6  }
7  void Circle::setR(float r)
8  {
9      radius = ( r > 0 ? r : 0 );
10 }
11 float Circle::getR() const { ... }
16 float Circle::CalArea() const
17 {
18     return 3.14159*radius*radius;
19 }
20
21 Circle::~Circle(void)
22 {
23 }
```

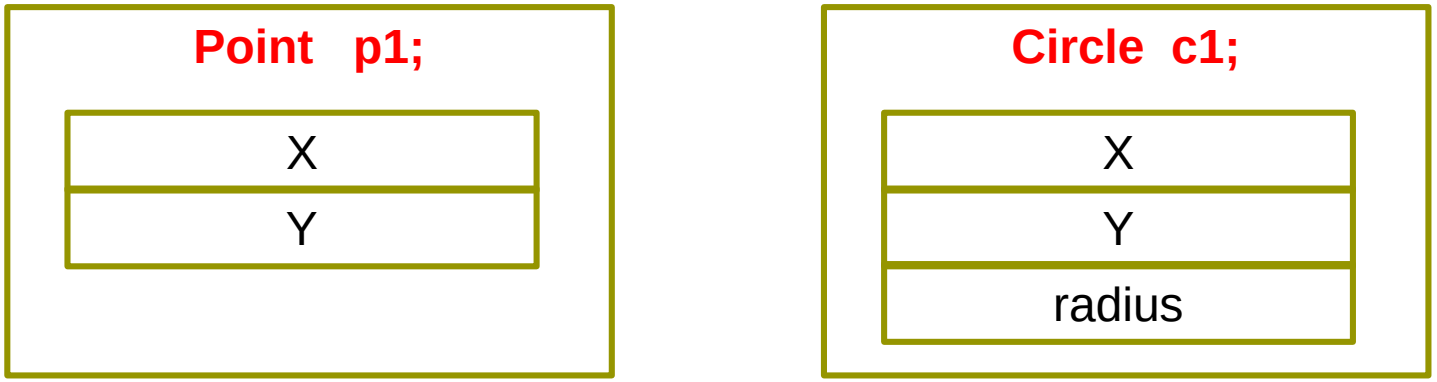
Ví dụ



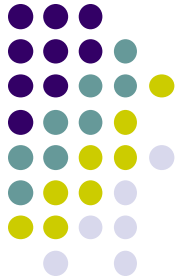
```
1  //main.cpp
2  #include "Point.h"
3  #include "Circle.h"
4  int main()
5  {
6      Point p1;
7      Circle c1;
8  //  p1.CalArea(); invalid
9      cout<<p1<<endl;
10     cout<<c1<<endl; //goi operator<< thua ke tu lop Point
11     return 0;
12 }
```

Ví dụ

- Bộ nhớ được cấp phát cho 2 đối tượng p1 và c1



Thành viên “Protected”



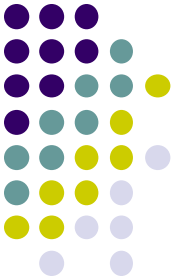
- Thành viên “private” không thể được truy xuất bên ngoài phạm vi lớp
- Derived class không thể truy xuất thành viên “private” của Base class

→ Sử dụng quyền truy xuất “Protected”

→ Thành viên “Protected” của Base class có thể được truy xuất bởi:

  - Thành viên và bạn của Base class
  - Thành viên và bạn của các lớp dẫn xuất từ Base class

Thành viên “Protected”



```
5  class Point
6  {
7  private:
8       float x;
9       float y;
10 public:
11      //constructor and destructor
12      Point(float x = 0, float y = 0);
13      ~Point(void);
```

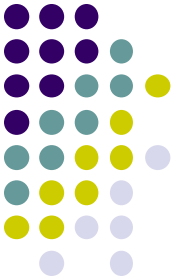
```
4  class Circle: public Point //Lop Circle ke thua tu lop Point
5  {
6  //Lop Circle ke thua du lieu x,y tu lop Point
7  //va bo sung them du lieu radius
8  private:
9       float radius;
10 public:
11      Circle(float x = 0, float y = 0, float r = 0);
12      ~Circle(void);
13      void setR(float);
14      float getR() const;
15      double CalArea() const;
16      friend ostream &operator<<( ostream &, const Circle & );
```

Thành viên “Protected”

```
1  //Circle.cpp
2  #include "Circle.h"
3  Circle::Circle(float x, float y, float r) { ... }
7  void Circle::setR(float r) { ... }
11 float class Circle e::getR() const { ... }
16 double Circle::CalArea() const { ... }
21 ostream &operator<<( ostream &output, const Circle &c )
22 {
23     output<<"Center of circle: ["<<c.x<<","<<c.y<<"]\n";
24     output<<"Radius of circle: "<<c.radius<<"\n";
25     return output;
26 }
27 Circle::~Circle(void) { ... }
```

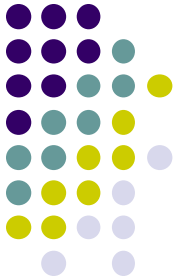
```
tput
ow output from: Build
>Compiling...
>Circle.cpp
>h:\trainning\oop-lap trinh huong doi tuong\bai giang\example_inheritance\circle.cpp(23) :
error C2248: 'Point::x' : cannot access private member declared in class 'Point'
```

Thành viên “Protected”



```
1  //Point.h
2  #pragma once
3  #include <iostream>
4  using namespace std;
5  class Point
6  {
7  protected:
8      float x;
9      float y;
10 public:
11     //constructor and destructor
12     Point(float x = 0, float y = 0);
13     ~Point(void);
14     //setters and getters
15     void setX(float);
16     float getX() const;
17     void setY(float);
18     float getY() const;
19     //print
20     friend ostream& operator<<(ostream &output, const Point &p );
```

Thành viên “Protected”



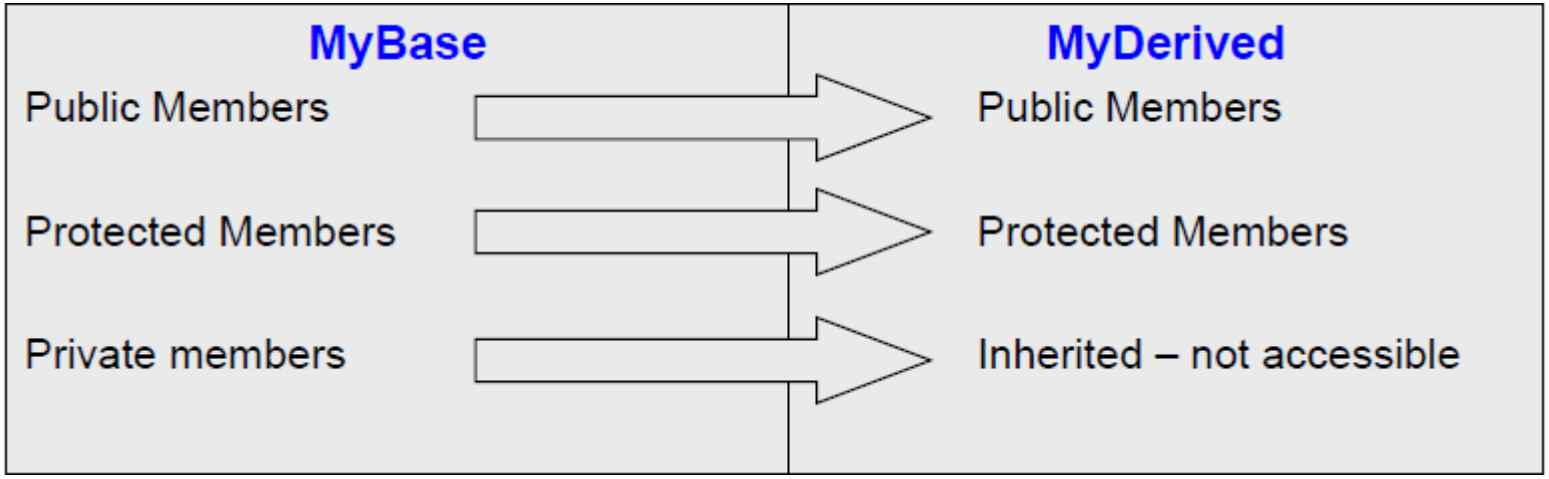
- Ưu điểm:
  - Derived class có thể truy xuất trực tiếp các dữ liệu thành viên của Base class
  - Giảm thiểu việc sử dụng get/set
- Nhược điểm:
  - Derived class có thể gây ra sự thay đổi không hợp lệ trên các dữ liệu thành viên của Base class
  - Phụ thuộc vào việc thực thi của lớp Base class

Các kiểu kế thừa

- Dựa trên quyền truy xuất
  - public Inheritance
  - protected Inheritance
  - private Inheritance
- Dựa trên cấu trúc của lớp
  - Single Inheritance
  - Multiple Inheritance
  - Multilevel Inheritance
  - Hierarchical Inheritance
  - Hybrid Inheritance

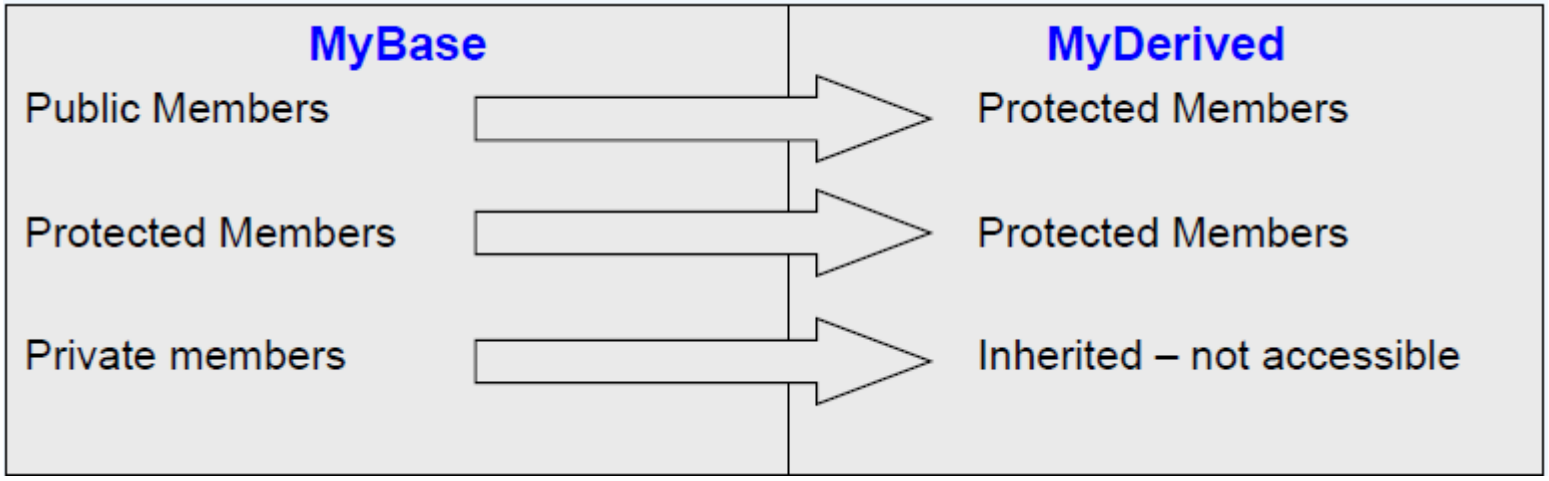
Public Inheritance

```
class  MyDerived : public  MyBase{
/*Lớp MyDerived sẽ kế thừa các thành viên “protected” và “public” của lớp MyBase */
};
```



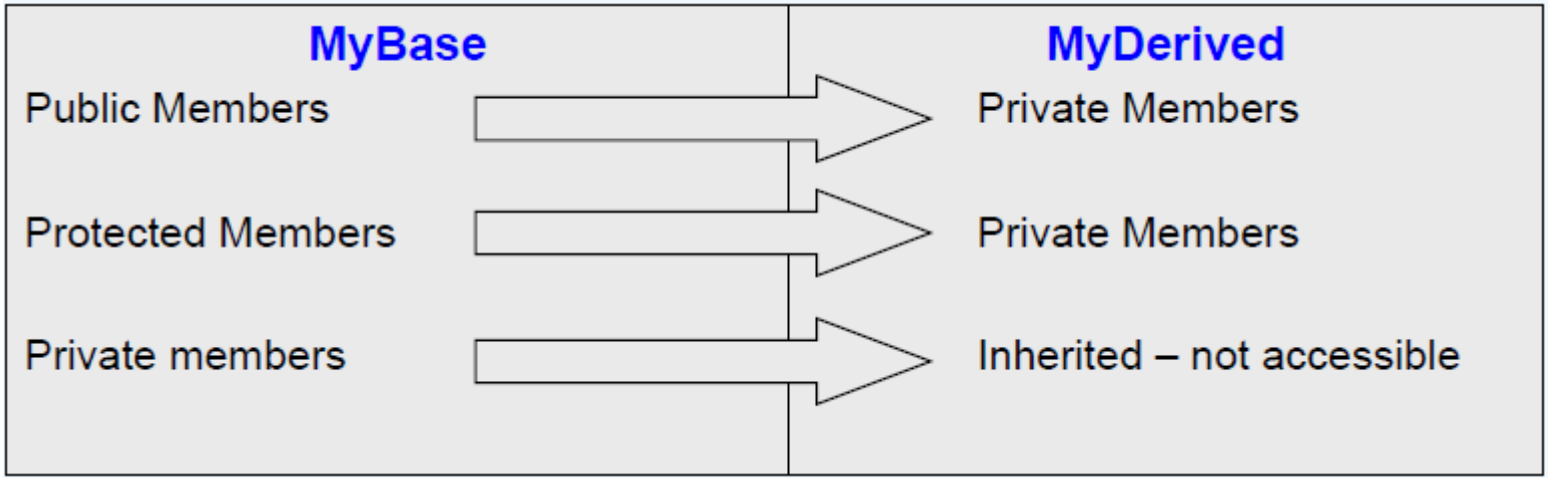
Protected Inheritance

```
class  MyDerived : protected  MyBase{
/*Lớp MyDerived sẽ kế thừa các thành viên “protected” và “public” của lớp MyBase */
};
```

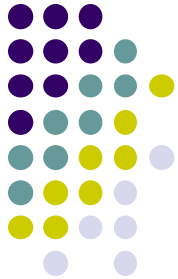


Private Inheritance

```
class  MyDerived : private  MyBase{
/*Lớp MyDerived sẽ kế thừa các thành viên “protected” và “public” của lớp MyBase */
};
```

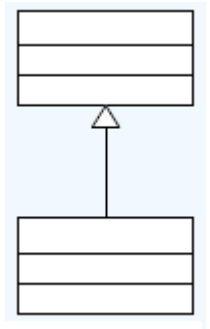


Public/Protected/Private Inheritance

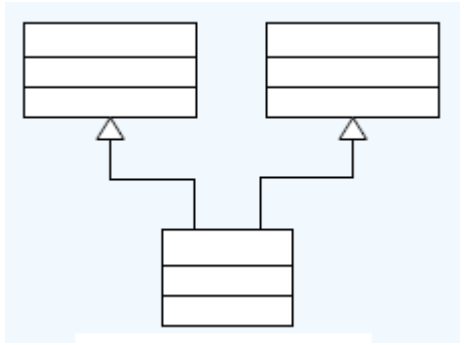


Type of Inheritance	Accessible inside Derived Class? i.e. In any of the methods of derived class			Accessible outside Derived Class? i.e. through derived class objects		
	Private member of Base class	Protected member of Base Class	Public member of Base class	Private member of Base class	Protected member of Base class	Public member of Base class
Private	**NO**	**YES**	**YES**	**NO**	**NO**	**NO**
Protected	**NO**	**YES**	**YES**	**NO**	**NO**	**NO**
Public	**NO**	**YES**	**YES**	**NO**	**NO**	**YES**

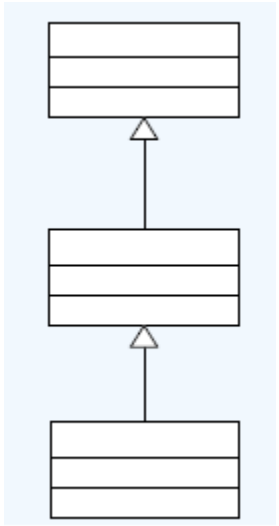
Kiểu thừa kế dựa trên cấu trúc



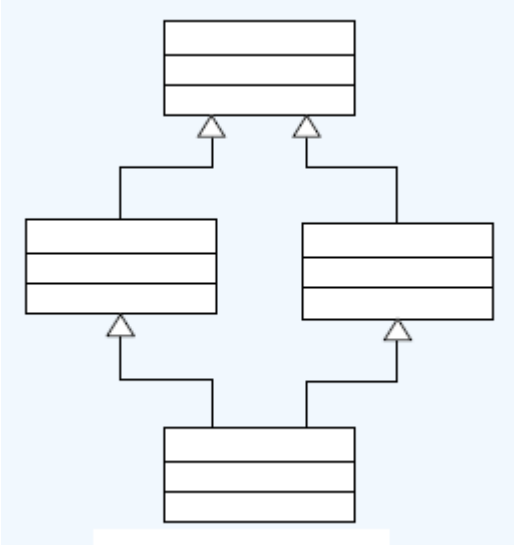
Single Inheritance



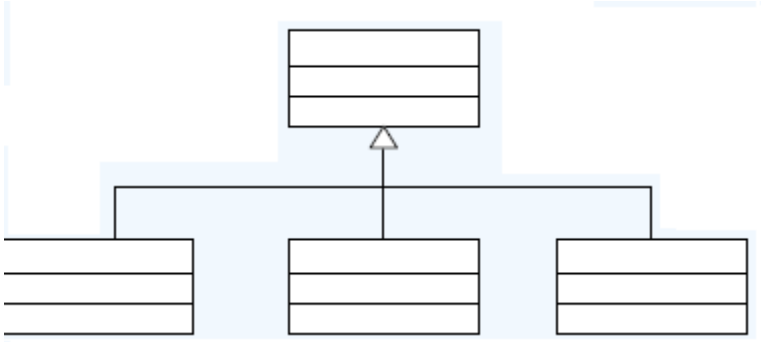
Multiple Inheritance



Multi-level Inheritance

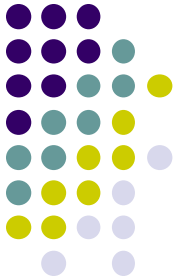


Hybrid Inheritance



Hierarchical Inheritance

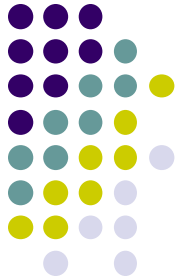
Hàm tạo và hàm hủy



- Hàm tạo
  - Không được kế thừa nhưng có thể được gọi trong lớp con
  - Khi tạo ra object của Derived class, thứ tự thực thi hàm tạo là:

    Hàm tạo của Base class → Hàm tạo của Derived class
  - Cách gọi hàm tạo của Base class trong Derived class:
    - Ngầm định(hàm tạo mặc định được gọi)
    - Tường minh

Hàm tạo và hàm hủy

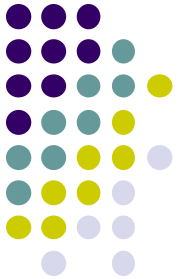


- Hàm tạo
  - **Cách gọi hàm tạo của Base class tường minh**:

```
<Hàm tạo lớp dẫn xuất>(tham số) :
 <Hàm tạo lớp cơ sở>(đối số)  //gọi hàm tạo của lớp cơ cở
{
   ....
}
```

  - **Multiple Inheritance**: hàm tạo lớp cha được thực thi theo trình tự được khai báo trong lớp con
  - **Multilevel inheritance**: hàm tạo được thực thi theo trình tự kế thừa

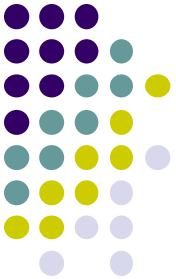
Hàm tạo và hàm hủy



- Hàm hủy
  - Không được kế thừa nhưng có thể được gọi trong lớp con
  - Được thực thi theo trình tự ngược lại với hàm tạo:

    Hàm hủy của Derived class → Hàm hủy của Base class

Ví dụ



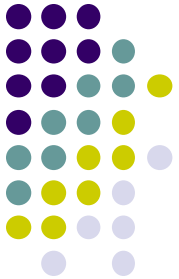
```
2   #include "Circle.h"
3   Circle::Circle(float x, float y, float r):Point(x,y)
4   {
5       cout<<"Constructor of Circle class is called\n";
6       setR(r);
7   }
8   void Circle::setR(float r) { ... }
12  float Circle::getR() const { ... }
17  double Circle::CalArea() const { ... }
22  ostream &operator<<( ostream &output, const Circle &c ) { ... }
28  Circle::~Circle(void)
29  {
30      cout<<"Destructor of Circle class is called\n";
31  }
```

Ví dụ

```
1  //main.cpp
2  #include "Point.h"
3  #include "Circle.h"
4  int main()
5  {
6      Point p1;
7      Circle c1;
8  //  p1.CalArea(); invalid
9      cout<<p1<<endl;
10     cout<<c1<<endl; //goi operator<< thua ke tu lop Point
11     return 0;
12 }
```

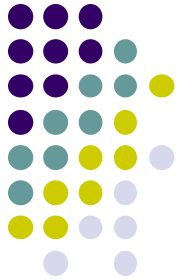
```
Constructor of Point class is called
Constructor of Point class is called
Constructor of Circle class is called
[0, 0]
Center of circle: [3,4]
Radius of circle: 0

Destructor of Circle class is called
Destructor of Point class is called
Destructor of Point class is called
Press any key to continue . . .
```

Định nghĩa lại phương thức (Method Overriding)



- Định nghĩa lại phương thức của Base class trong Derived class
- Xảy ra khi phương thức của Base class không còn phù hợp với Derived class
- Khác với định nghĩa chồng hàm

Định nghĩa lại phương thức (Method Overriding)



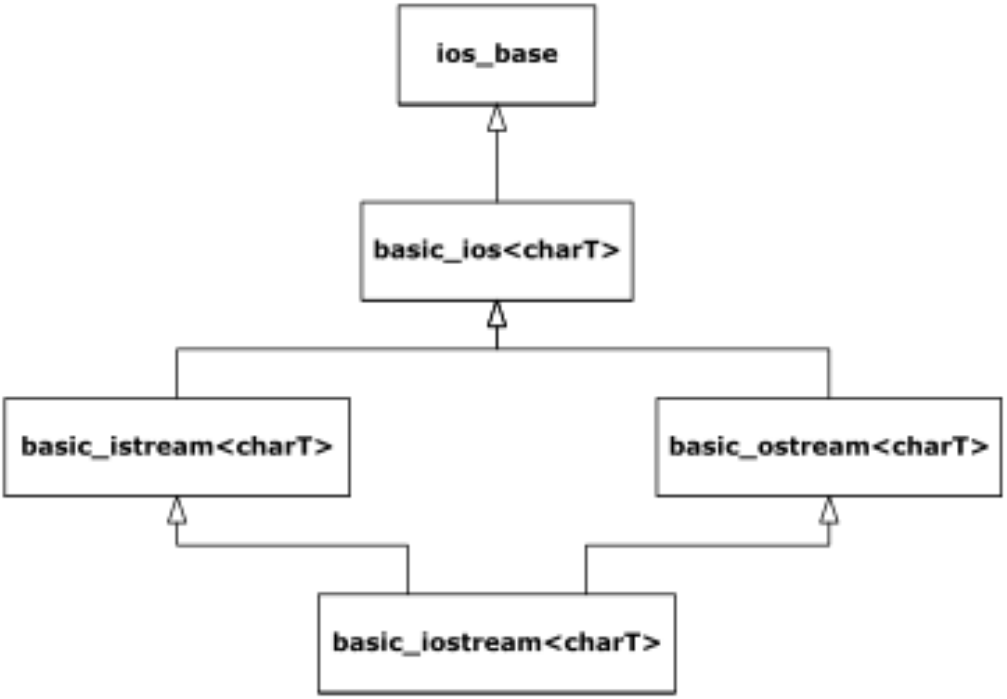
```
class Point
{
protected:
    float x;
    float y;
public:
    void Draw()
    {
        cout<<"Draw a Point:["<<x<<","<<y<<"]\n";
    }
```

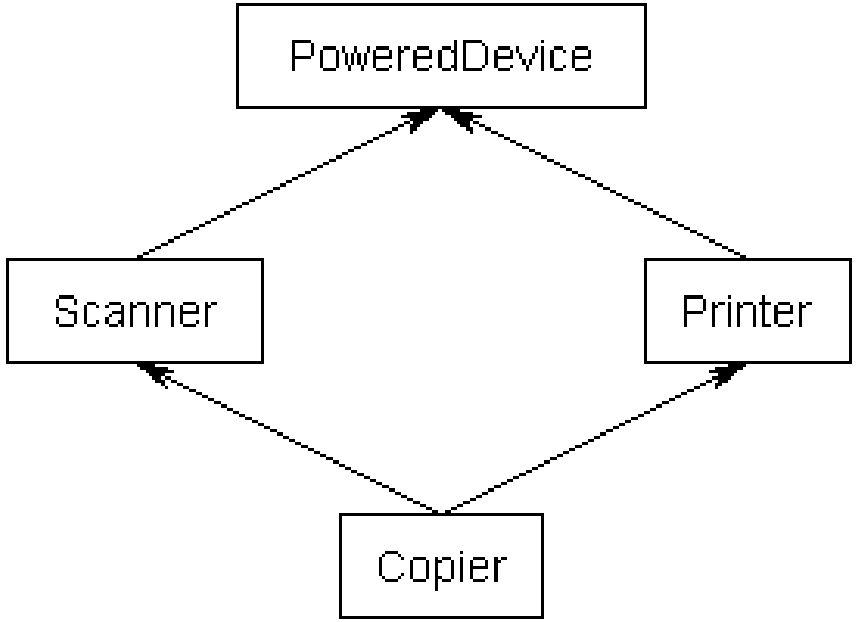
```
int main()
{
    Point p1;
    Circle c1(4,5,1);
    p1.Draw();
    c1.Draw();
```

```
class Circle: public Point //Lop Circle ke thua tu lop Point
{
private:
    float radius;
public:
    void Draw() //method overriding
    {
        cout<<"Draw a Circle with center:["<<x<<","<<y;
        cout<<"] and radius: "<<radius<<"\n";
    }
```

Đa kế thừa(Multiple Inheritance)

- **Khái niệm**: Là khả năng xây dựng lớp dẫn xuất kế thừa từ nhiều lớp cơ sở





Ví dụ

```
1  //Date.h
2  #pragma once
3  class Date
4  {
5  protected:
6      int day;
7      int month;
8      int year;
9  public:
10      Date(void);
11      Date(int d, int m, int y);
12      ~Date(void);
13     int getDay() const ;
14     void setDay(int d);
15     int getMonth() const;
16     void setMonth(int m);
17     int getYear() const;
18     void setYear(int y);
19 };
```

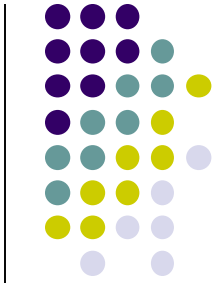
```
1  //Date.cpp
2  #include "Date.h"
3  Date::Date(void)
4  {
5       day = 1;
6       month = 1;
7       year = 1900;
8  }
9  Date::Date(int d, int m, int y)
10 {
11      setDay(d);
12      setMonth(m);
13      setYear(y);
14 }
15 int Date::getDay() const
16 {  return day; }
17
18 int Date::getMonth() const
19 {  return month; }
20
21 int Date::getYear() const
22 {  return year; }
23 void Date::setDay(int d) { ... }
26 void Date::setMonth(int m) { ... }
29 void Date::setYear(int y) { ... }
33 Date::~Date(void)
```

Ví dụ

```
1  //Time.h
2  #pragma once
3  class Time
4  {
5  protected:
6      int hour;
7      int min;
8      int sec;
9  public:
10     Time();
11     Time(int h, int m, int s);
12     ~Time(void);
13     int getHour() const;
14     void setHour(int h);
15     int getMin() const;
16     void setMin(int m);
17     int getSecond() const;
18     void setSecond(int s);
19 };
```

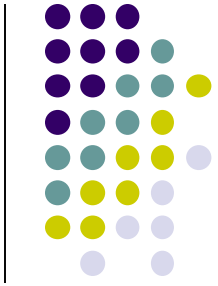
```
1  //Time.cpp
2  #include "Time.h"
3  Time::Time(void)
4  {
5      hour = 0;
6      min = 0;
7      sec = 0;
8  }
9  Time::Time(int h, int m, int s)
10 {
11     setHour(h);
12     setMin(m);
13     setSecond(s);
14 }
15 int Time::getHour() const { ... }
18 int Time::getMin() const { ... }
21 int Time::getSecond() const { ... }
24 void Time::setHour(int h) { ... }
28 void Time::setMin(int m) { ... }
32 void Time::setSecond(int s) { ... }
37 Time::~Time(void) { ... }
```

Ví dụ



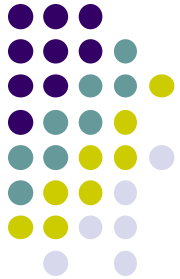
```
1  //Datetime.h
2  #pragma once
3  #include <iostream>
4  #include "Date.h"
5  #include "Time.h"
6  using namespace std;
7  const int SIZE = 20;
8  class DateTime: public Date, public Time
9  {
10 private:
11     char dateTimeString[SIZE];
12 public:
13     DateTime(void);
14     DateTime(int, int, int, int, int, int);
15     const char* getDateTime() const;
16     ~DateTime(void);
17     friend ostream& operator <<(ostream &, const DateTime &);
18 };
```

Ví dụ



```
1  //Datetime.cpp
2  #include "DateTime.h"
3  DateTime::DateTime(): Date(), Time()
4  {
5     strcpy(dateTimeString, "1/1/1900 0:0:0");
6  }
7  DateTime::DateTime(int dy, int mon, int yr, int hr, int mt, int sc):
8      Date(dy, mon, yr), Time(hr, mt, sc)
9  {
10    char temp[SIZE/2];
11    //Dinh dang MM/DD/YY
12    strcpy(dateTimeString, itoa(day, temp, SIZE/2));
13    strcat(dateTimeString, "/");
14    strcat(dateTimeString, itoa(month, temp, SIZE/2));
15    strcat(dateTimeString, "/");
16    strcat(dateTimeString, itoa(year, temp, SIZE/2));
17    strcat(dateTimeString, " ");
18    // Dinh dang HH:MM:SS
19    strcat(dateTimeString, itoa(hour, temp, SIZE/2));
20    strcat(dateTimeString, ":");
21    strcat(dateTimeString, itoa(min, temp, SIZE/2));
22    strcat(dateTimeString, ":");
23    strcat(dateTimeString, itoa(sec, temp, SIZE/2));
24 }
25 const char* DateTime::getDateTime() const { ... }
```

Một số vấn đề với đa kế thừa



- Đa kế thừa là công cụ cấu trúc khá mạnh tuy nhiên nó gây ra một số vấn đề sau:
  - Trùng tên
    - Những phương thức trùng nhau của các lớp cơ sở
    - Những dữ liệu trùng nhau của các lớp cơ sở
  - Một lớp được thừa kế nhiều lần(xung đột kế thừa)

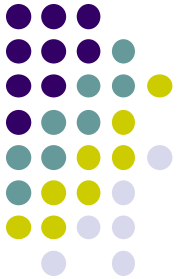
Vấn đề: Trùng tên

```
1  #pragma once
2  class Student
3  {
4  protected:
5       int id;
6  public:
7       Student(void);
8       int getID() const;
9       ~Student(void);
10 };
```

```
1  #pragma once
2
3  class Employee
4  {
5  protected:
6       int id;
7  public:
8       Employee(void);
9       int getID() const;
10      ~Employee(void);
11 };
```

class Employee

Vấn đề: Trùng tên



```
1  #pragma once
2  #include "Student.h"
3  #include "Employee.h"
4  class TeachingAssistant: public Student, public Employee
5  {
6  public:
7      TeachingAssistant(void);
8      ~TeachingAssistant(void);
9      void changeID(int _id);
10 };
```

```
1  #include "TeachingAssistant.h"
2
3  TeachingAssistant::TeachingAssistant(void): Student(), Employee()
4  {
5  }
6
7  void TeachingAssistant::changeID(int _id)
8  {
9      id = _id;
10 }
```

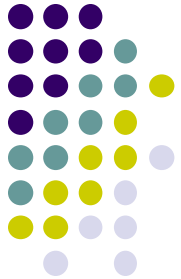
error C2385: ambiguous access of 'id'
could be the 'id' in base 'Student'
or could be the 'id' in base 'Employee'

Vấn đề: Trùng tên

```
3  //main.cpp
4  #include <iostream>
5  #include "TeachingAssistant.h"
6  using namespace std;
7  int main()
8  {
9      TeachingAssistant *mark = new TeachingAssistant();
10     cout<<mark->getID();
11     return 0;
12 }
```

error C2385: ambiguous access of 'getID'
could be the 'getID' in base 'Student'
or could be the 'getID' in base 'Employee'

Vấn đề: Trùng tên



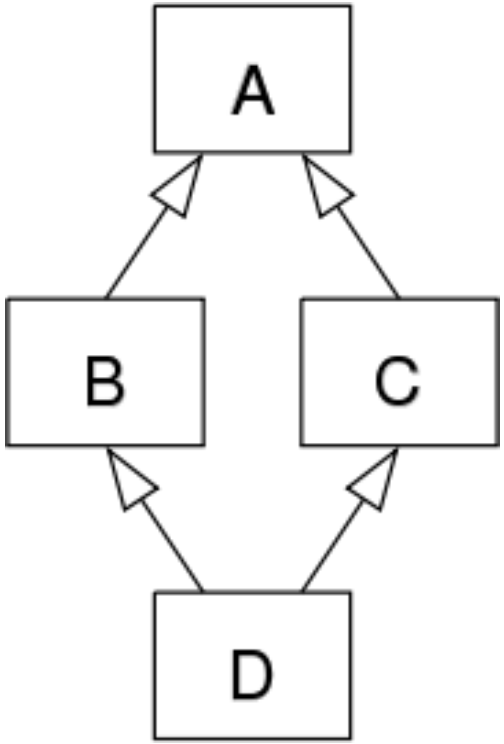
Giải pháp: Sử dụng toán tử phạm vi **::**

```
void TeachingAssistant::changeID(int _id){
    Employee::id = _id;
}

int main()
{
    TeachingAssistant *mark = new TeachingAssistant();
    cout<<mark->Employee::getID();
    return 0;
}
```

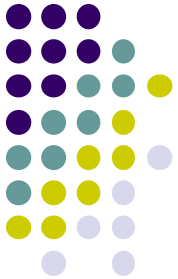
Vấn đề: Xung đột kế thừa

- Lớp D thừa kế lớp A hai lần

```
class Employee : public Person
 {..};
class Student: public Person
{..};
class TeachingAssistant:
public Student, public Employee
{..};
```



Vấn đề: Xung đột kế thừa



Giải pháp: Thừa kế ảo(virtual inheritance)

Tất cả các thành phần của lớp A chỉ tổ hợp một lần duy nhất trong lớp D.

```
class Employee : public virtual Person
 {..};
class Student: public virtual Person
{..};
class TeachingAssistant:
public Student, public Employee
{..};
```

Bài tập 1

- Xây dựng lớp Person gồm
  - Thuộc tính: id, name, birthday
  - Phương thức: hàm tạo,<<, get/set
- Xây dựng lớp Student kế thừa từ Person
  - Thuộc tính: dtb
  - Phương thức: hàm tạo,<<, get/set, Rank()
- Xây dựng lớp Teacher kế thừa tư Person
  - Thuộc tính: hsl
  - Phương thức: hàm tạo,<<, get/set, CalSalary()

Bài tập 2