Tarena Technology
达内科技

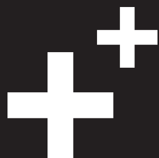
标准C++语言

PART 2

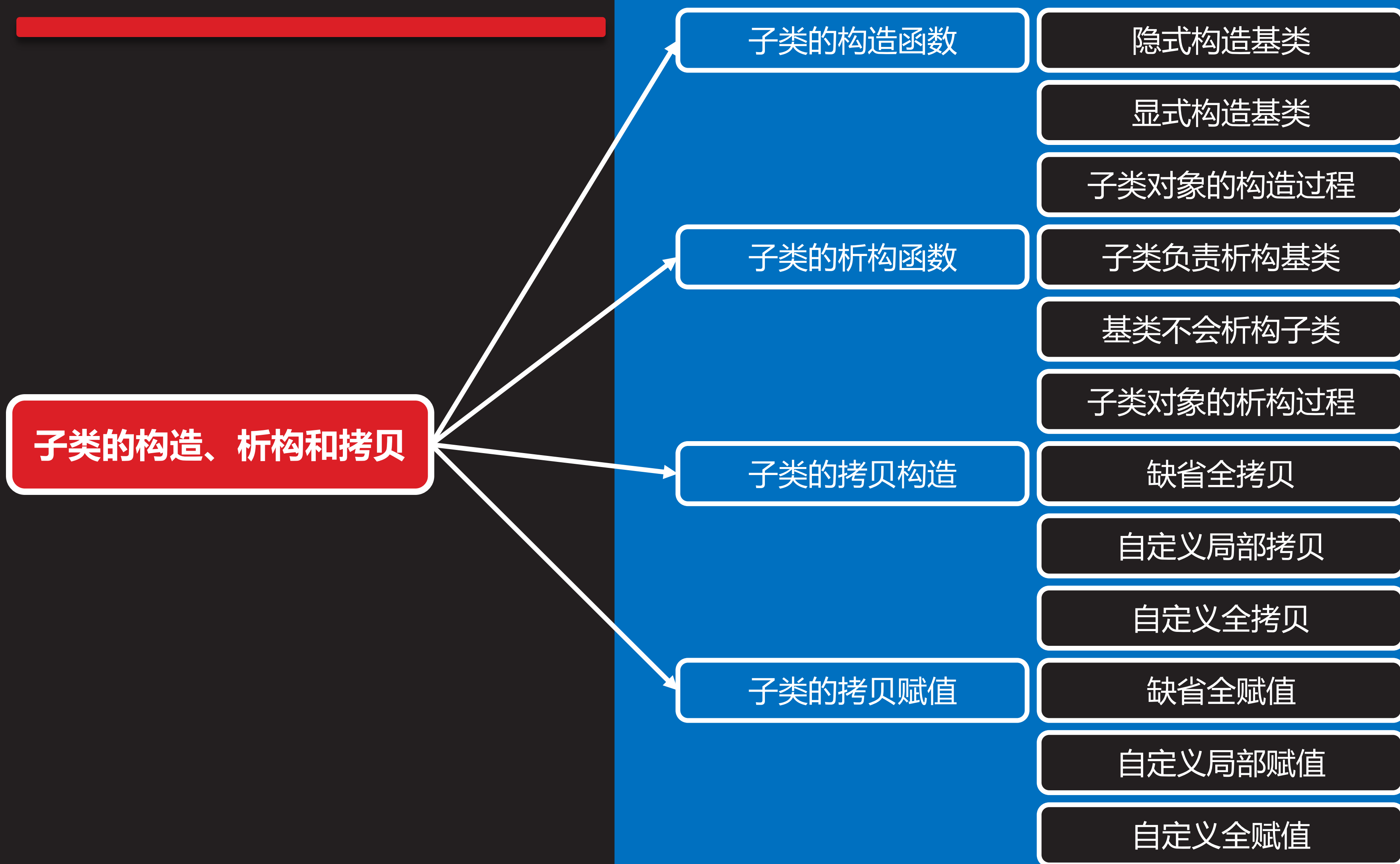
DAY02

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	子类的构造、析构和拷贝
	10:30 ~ 11:20	
	11:30 ~ 12:20	名字隐藏与重载
下午	14:00 ~ 14:50	私有继承与保护继承
	15:00 ~ 15:50	多重继承、钻石继承和虚继承
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑



子类的构造、析构和拷贝



Tarena Technology
达内科技

隐式构造基类

Tarena Technology 达内科技

知识讲解

- 如果子类的构造函数没有显式指明其基类部分的构造方式，那么编译器会选择其基类的缺省构造函数，构造该子类对象中的基类子对象
  - class Student : public Human {
    public:
        Student (int no) : m_no (no) {}
    };
- 但是请注意，只有在为基类显式提供一个无参构造函数，或者不提供任何构造函数(系统会提供一个缺省的无参构造函数)的情况下，基类才拥有无参构造函数

显式构造基类

Tarena Technology
达内科技

知识讲解

- 子类的构造函数可以在初始化表中显式指明其基类部分的构造方式，即通过其基类的特定构造函数，构造该子类对象中的基类子对象
  - class Human {
    public:
        Human (string const& name, int age) :
            m_name (name), m_age (age) {}
    };
  - class Student : public Human {
    public:
        Student (string const& name, int age, int no) :
            Human (name, age), m_no (no) {}
    };

子类对象的构造过程

Technology
Tarena
达内科技

- 子类的构造函数执行如下步骤：
  - 首先，按照继承表的顺序，依次调用各个基类的构造函数，构造子类对象中的基类子对象
  - 其次，按照声明的顺序，依次调用各个类类型成员变量相应类型的构造函数，构造子类对象中的成员子对象
  - 最后，执行子类构造函数体中的代码，完成整个构造过程
- 无论如何，子类的构造函数都一定会(显式或隐式地)调用其基类和类类型成员变量类型的构造函数

子类的构造过程

【参见：TTS COOKBOOK 】

课堂练习

- 子类的构造过程

Technology Tarena 达内科技

子类负责析构基类

Tarena Technology 达内科技

知识讲解

- 子类的析构函数，无论是自己定义的(自定义析构函数)还是系统提供的(缺省析构函数)，在执行完其中的析构代码，并析构完所有的类类型成员子对象以后，会自动调用其基类的析构函数，析构该子类对象中的基类子对象
  - class Human { ... };
  - class Student : public Human {
    public:
        ~Student (void) { ... /* ~Human () */ }
    };
  - Student* student = new Student (...);
    delete student;

基类不会析构子类

- 对一个指向子类对象的基类指针使用delete运算符，实际被调用的将是基类的析构函数，该函数不会调用子类的析构函数，其所析构的仅仅是子类对象中的基类子对象，而子类的扩展部分极有可能因此而形成内存泄漏
  - class Human {
    public: ~Human (void) { ... }
    };
  - class Student : public Human { ... };
  - Human* human = new Student (...);
    delete human;

Tarena Technology 达内科技

子类对象的析构过程

知识讲解

- 子类的析构函数执行如下步骤：
  - 首先，执行子类析构函数体中的代码，析构子类的扩展部分
  - 其次，按照声明的逆序，依次调用各个类类型成员变量相应类型的析构函数，析构子类对象中的成员子对象
  - 最后，按照继承表的逆序，依次调用各个基类的析构函数，析构子类对象中的基类子对象，完成整个析构过程
- 无论如何，子类的析构函数都一定会隐式地调用其类类型成员变量类型和基类的析构函数

Tarena Technology 达内科技

子类的析构函数

【参见：TTS COOKBOOK 】

课堂练习

- 子类的析构函数

Tarena Technology 达内科技

缺省全拷贝

Tarena Technology
达内科技

- 如果子类没有定义拷贝构造函数，那么编译器为子类提供的缺省拷贝构造函数，会自动调用其基类的(自定义或缺省)拷贝构造函数，拷贝构造子类对象中的基类子对象
  - class Student : public Human {

```
class Student : public Human {
public:
    Student (String const& name, int age, int no) :
        Human (name, age), m_no (no) {}
    /*
    Student (Student const& that) : Human (that) { ... }
    */
};
```

知识讲解

自定义局部拷贝

- 如果子类定义了拷贝构造函数，但没有显式指明以拷贝方式构造其基类部分，那么编译器会选择其基类的缺省构造函数，构造子类对象中的基类子对象
  - class Student : public Human {

```
class Student : public Human {
public:
    Student (String const& name, int age, int no) :
        Human (name, age), m_no (no) {}
    Student (Student const& that) : /* Human (), */
        m_no (that.m_no) {}
};
```

自定义全拷贝

Tarena Technology 达内科技

- 如果子类定义了拷贝构造函数，同时显式指明了其基类部分以拷贝方式构造，那么子类对象中的基类部分和扩展部分将一起被复制
  - ```
    class Student : public Human {
    public:
        Student (String const& name, int age, int no) :
            Human (name, age), m_no (no) {}
        Student (Student const& that) :
            Human (that), m_no (that.m_no) {}
    };
    ```

知识讲解

    ```

Tarena Technology
达内科技

# 子类的拷贝构造

【参见：TTS COOKBOOK 】

课堂练习

- 子类的拷贝构造

Tarena Technology 达内科技

# 缺省全赋值

Tarena Technology 达内科技

- 如果子类没有定义拷贝赋值运算符函数，那么编译器为子类提供的缺省拷贝赋值运算符函数，会自动调用其基类的(自定义或缺省)拷贝赋值运算符函数，复制子类对象中的基类子对象
  - class Student : public Human {

```
class Student : public Human {
    /*
    Student& operator= (Student const& rhs) {
        ... Human::operator= (rhs) ...
    }
    */
};
```

知识讲解

Technology
Tarena
达内科技

自定义局部赋值

- 如果子类定义了拷贝赋值运算符函数，但没有显式调用其基类的拷贝赋值运算符函数，那么子类对象中的基类子对象将因得不到复制而保持原状
  - ```
    class Student : public Human {
    public:
        Student& operator= (Student const& rhs) {
            if (&rhs != this)
                m_no = rhs.m_no;
            return *this;
        }
    };
    ```

知识讲解

    ```

# 自定义全赋值

Tarena Technology 达内科技

- 如果子类定义了拷贝赋值运算符函数，同时显式调用了其基类的拷贝赋值运算符函数，那么子类对象中的基类部分和扩展部分将一起被复制
  - class Student : public Human {

```
class Student : public Human {
public:
    Student& operator= (Student const& rhs) {
        if (&rhs != this) {
            Human::operator= (rhs);
            m_no = rhs.m_no;
        }
        return *this;
    }
};
```

知识讲解

子类的拷贝赋值

【参见：TTS COOKBOOK 】

- 子类的拷贝赋值

名字隐藏与重载

Tarena Technology
达内科技

继承不会改变作用域

继承不会改变作用域

Tarena Technology 达内科技

知识讲解

- 继承不会改变类成员的作用域，基类的成员永远都是基类的成员，并不会因为继承而变成子类的成员
  - class Human {
    private:
    string m_name;
    int m_age;
    };
  - class Student : public Human {
    private:
    string m_name;
    int m_age;
    int m_no;
    };

Technology Tarena 达内科技

隐藏不是重载

Tarena Technology 达内科技

- 因为作用域的不同，分别在子类和基类中定义的同名成员函数(包括静态成员函数)，并不构成重载关系，相反是一种隐藏关系

```
– class Real {
      void add (Real const& that) {
          m_r += that.m_r; }
  };
– class Complex : public Real {
      void add (Complex const& that) {
          m_r += that.m_r;
          m_i += that.m_i; }
  };
```

知识讲解

作用域限定

作用域限定

Technology
Tarena
达内科技

知识讲解

- 任何时候，无论在子类的内部还是外部，总可以通过作用域限定操作符“::”，显式地调用那些在基类中定义却为子类所隐藏的成员函数

```
– class Complex : public Real {
      void add (Complex const& that) {
          Real::add (that);
          m_i += that.m_i;
      }
  };
– Complex c (1, 2);
  Real r (3);
  c.Real::add (r);
```

Tarena Technology 达内科技

using声明

using声明

- 通过using声明可将在基类中声明的标识符引入子类的作用域，就如同在子类中声明的一样
- 如果所引入的标识符是基类的成员函数，并且满足函数重载的条件，那么子类对基类的隐藏关系可以变为重载关系
  - class Complex : public Real {

```
class Complex : public Real {
    using Real::add;
    void add (Complex const& that) {
        add ((Real const&)that);
        m_i += that.m_i; }
};
```

- using声明只能针对标识符，而不能针对具体的函数版本
- using声明可能导致名字冲突，子类版本优先

知识讲解

隐藏与重载

【参见：TTS COOKBOOK 】

- 隐藏与重载

私有继承与保护继承

Technology Tarena 达内科技

防止扩散

- 私有继承亦称实现继承，旨在于子类中将其基类的公有和保护成员私有化，既禁止从外部通过该子类访问这些成员，也禁止在该子类的子类中访问这些成员
  - class DCT { public: void codec (void); };
  - class JPEG : private DCT {
    public:
        void render (void) { ... codec () ... };
    };
  - JPEG jpeg (...);
    jpeg.render ();
    jpeg.codec (); // 错误

知识讲解

Tarena Technology 达内科技

Tarena Technology 达内科技

有限地防止扩散

- 保护继承是一种特殊形式的实现继承，旨在于子类中将其基类的公有和保护成员进行有限的私有化，只禁止从外部通过该子类访问这些成员，但并不禁止在该子类的子类中访问这些成员
  - class DCT { public: void codec (void); };
  - class JPEG : protected DCT {
    public:
        void render (void) { ... codec () ... };
    };
  - class M-JPEG : public JPEG {
    public:
        void play (void) { ... codec () ... };
    };

Technology Tarena 达内科技

禁止向上造型

Tarena Technology 达内科技

- 私有子类和保护子类类型的指针或引用，不能隐式转换为其基类类型的指针或引用
  - DCT* dct = new JPEG (...); // 错误
  - void inverse (DCT const& dct) { ... }
    JPEG jpeg (...);
    inverse (jpeg); // 错误
  - class Inverse {
    public:
        DCT& impl (void) { return m_jpeg; } // 错误
    private:
        JPEG m_jpeg;
    };

知识讲解

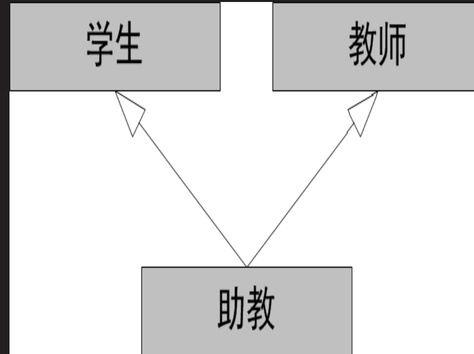
多重继承、钻石继承和虚继承

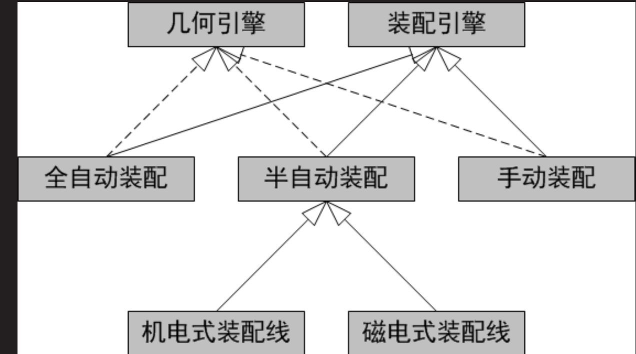
- 多重继承、钻石继承和虚继承
  - 多重继承
    - 多个基类
    - 内存布局与类型转换
    - 名字冲突
  - 钻石继承
    - 公共基类
    - 多个公共基类子对象
  - 虚继承
    - 共享公共基类子对象
    - 继承表中的virtual关键字
    - 末端子类负责构造虚基类
    - 虚基类子对象的拷贝
    - 虚基类子对象的赋值
    - 虚继承对象模型

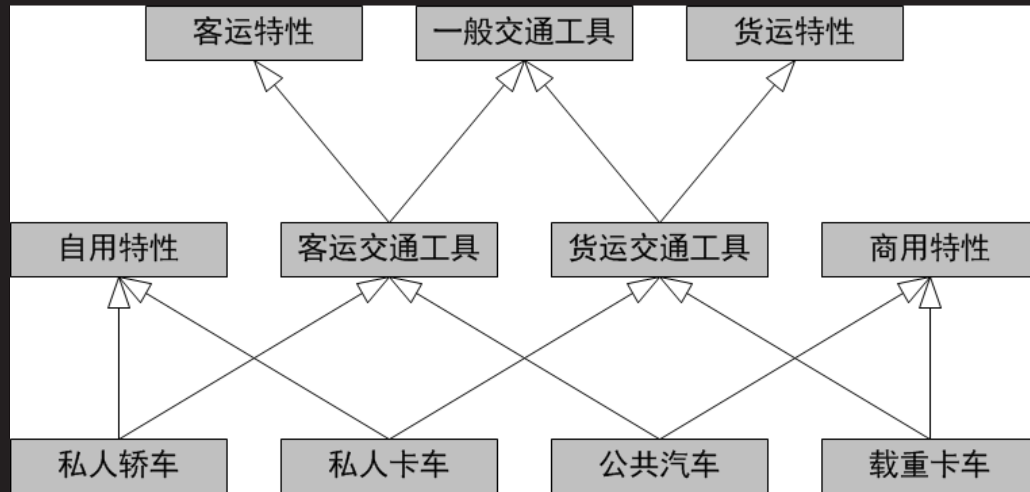
Technology Tarena
达内科技

多个基类

- 一个类可以同时从多个基类继承实现代码
  - 课堂教学系统
  - 交通工具系统
  - 零件装配系统
  - 智能手机系统







知识讲解

Tarena Technology 达内科技

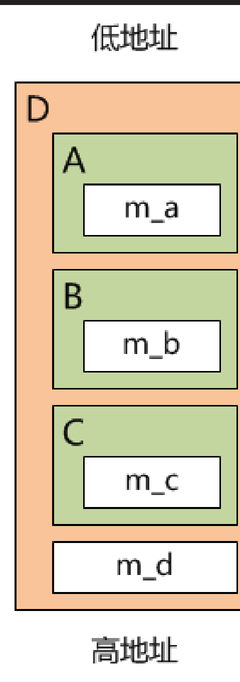
内存布局与类型转换

Technology Tarena 达内科技

- 子类对象中的多个基类子对象，按照继承表的顺序依次被构造，并从低地址到高地址排列，析构的顺序则与构造严格相反

知识讲解

```
class A {
public:
  int m_a;
};
class B {
public:
  int m_b;
};
class C {
public:
  int m_c;
};
class D : public A, public B, public C {
public:
  int m_d;
};
```



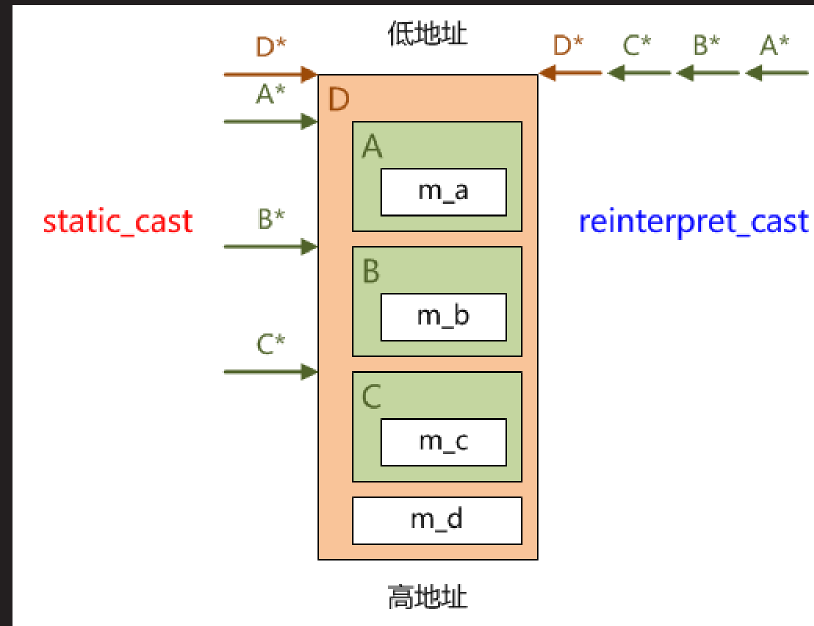
内存布局与类型转换（续1）

Tarena Technology 达内科技

- 将继承自多个基类的子类类型的指针，隐式或静态转换为它的基类类型，编译器会根据各个基类子对象在子类对象中的内存布局，进行适当的偏移计算，以保证指针的类型与其所指向目标对象的类型一致

知识讲解

- 反之，将该子类的任何一个基类类型的指针静态转换为子类类型，编译器同样会进行适当的偏移计算
- 无论在哪个方向上，重解释类型转换都不进行任何偏移计算



- 引用的情况与指针类似，因为引用的本质就是指针

名字冲突

Tarena Technology 达内科技

知识讲解

- 如果在子类的多个基类中，存在同名的标识符，而且子类又没有隐藏该名字，那么任何试图在子类中，或通过子类对象访问该名字的操作，都将引发歧义，除非通过作用域限定操作符“::”显式指明所属基类
  - class Real { public: void add (Real const& real) { ... } };
  - class Imag { public: void add (Imag const& imag) { ... } };
  - class Complex : public Real, public Imag { ... };
  - Real r (1); Imag i (2); Complex c (3, 4);
    c.add (r); // 错误
    c.add (i); // 错误
    c.Real::add (r);
    c.Imag::add (i);

名字冲突（续1）

Tarena Technology 达内科技

知识讲解

- 如果无法避免基类中的名字冲突，最简单的方法是在子类中隐藏这些标识符，或借助using声明令其在子类中重载
  - ```
    class Complex : public Real, public Imag {
    public:
        void add (Real const& real) { … }
        void add (Imag const& imag) { … }
    };
    ```
  - ```
    class Complex : public Real, public Imag {
    public:
        using Real::add;
        using Imag::add;
    };
    ```

Technology Tarena 达内科技

智能手机

【参见：TTS COOKBOOK 】

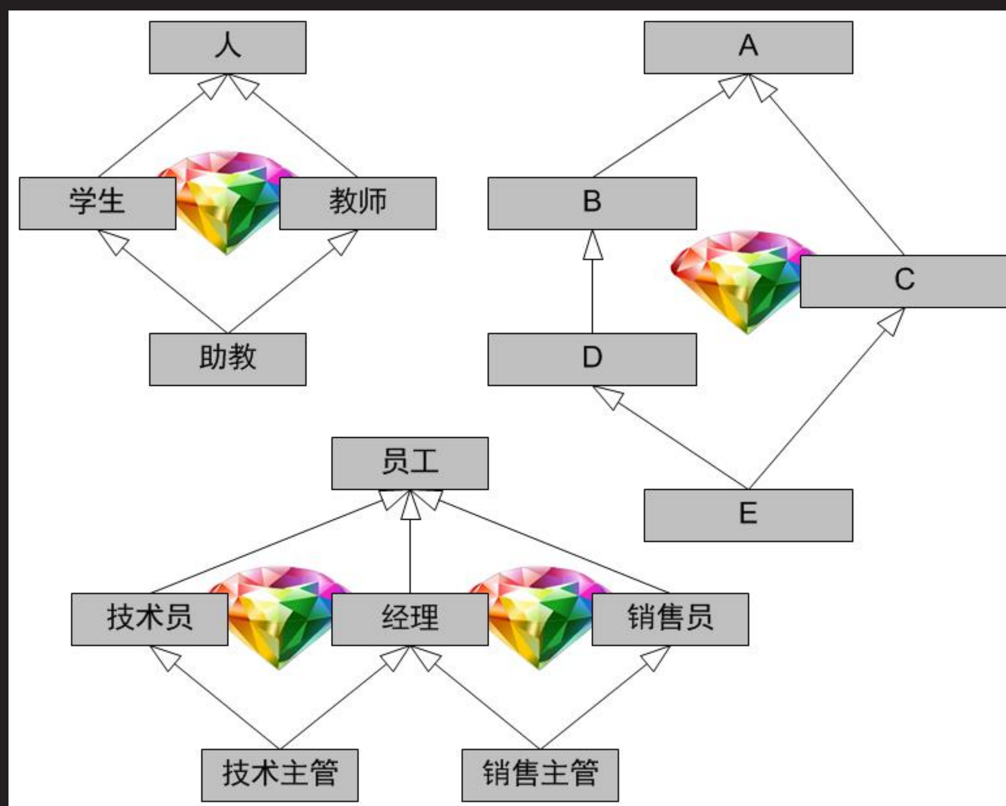
课堂练习

- 智能手机

Technology
Tarena
达内科技

公共基类

- 一个子类继承自多个基类，而这些基类又源自共同的祖先(公共基类)，这样的继承结构称为钻石继承

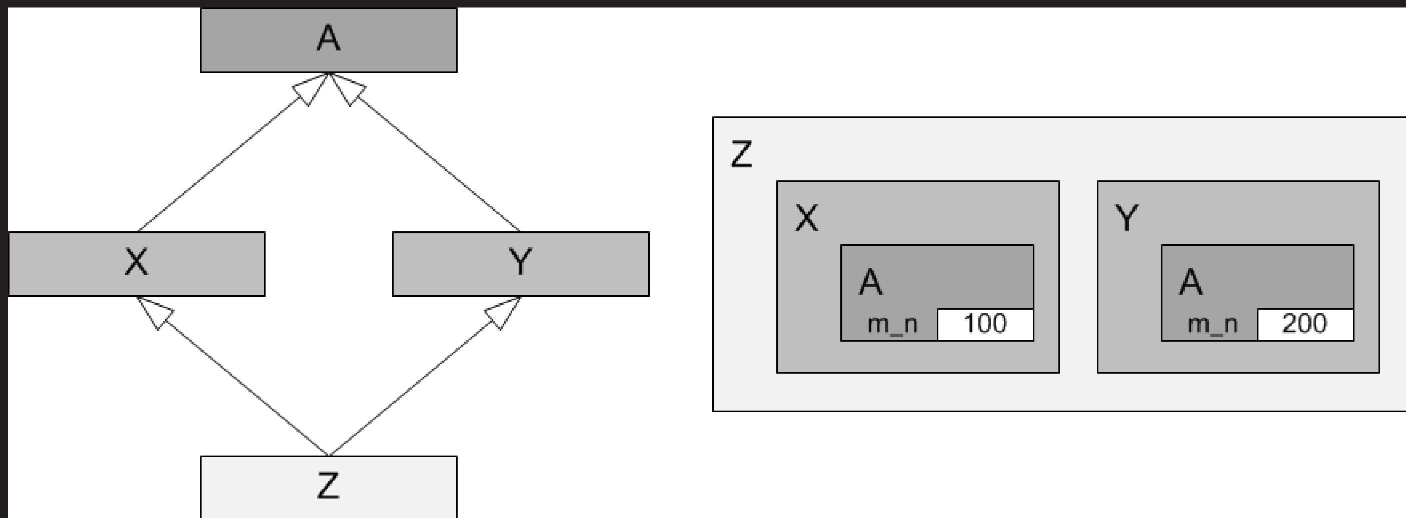


多个公共基类子对象

Tarena Technology 达内科技

- 派生多个中间子类的公共基类子对象，在继承自多个中间子类的汇聚子类对象中，存在多个实例
- 在汇聚子类中，或通过汇聚子类对象，访问公共基类的成员，会因继承路径的不同而导致不一致

知识讲解



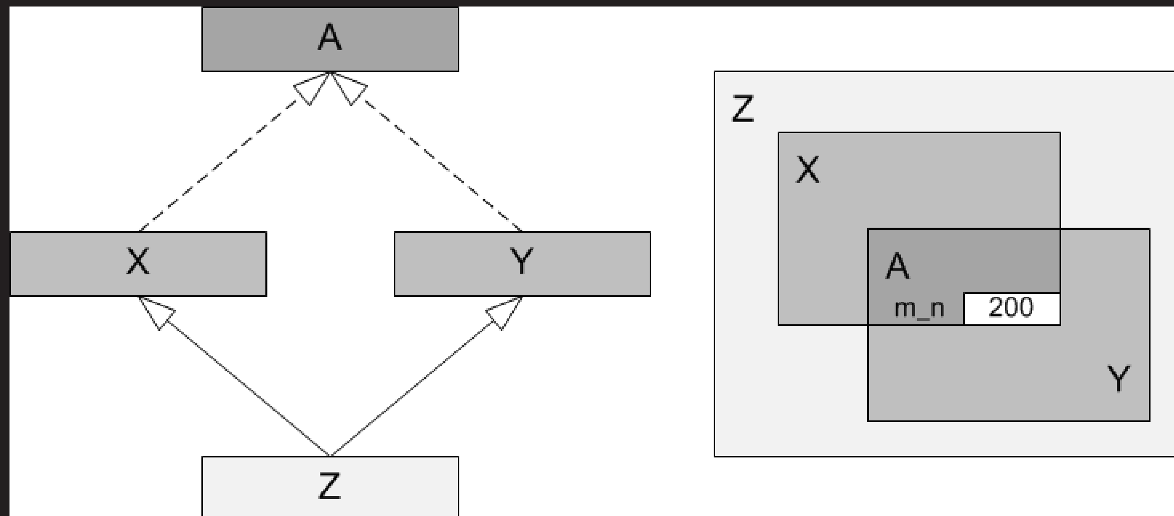
Technology
Tarena
达内科技

共享公共基类子对象

Tarena Technology 达内科技

- 通过虚继承，可以保证公共基类子对象在汇聚子类对象中，仅存一份实例，且为多个中间子类子对象所共享

知识讲解



继承表中的virtual关键字

Tarena Technology 达内科技

知识讲解

- 为了表示虚继承，需要在继承表中使用virtual关键字
  - class A { public: A (int data) : m_data (data) {} };
  - class X : virtual public A {
    public:
        X (int data) : A (data) {}
    };
  - class Y : virtual public B {
    public:
        Y (int data) : A (data) {}
    };
  - class Z : public X, public Y { ... };

末端子类负责构造虚基类

- 一般而言，子类的构造函数不能调用其间接基类的构造函数。但是，一旦这个间接基类被声明为虚基类，它的所有子类(无论直接的还是间接的)都必须显式地调用该间接基类的构造函数。否则，系统将试图为它的每个子类对象调用该间接基类的无参构造函数
  - class A { ... };
  - class X : virtual public A { ... };
  - class Y : virtual public A { ... };
  - class Z : public X, public Y {

```
public:
    Z (int data) : X (data), Y (data), A (data) { ... }
};
```

虚基类子对象的拷贝

Tarena Technology 达内科技

知识讲解

- 虚基类的所有子类(无论直接的还是间接的)都必须在其拷贝构造函数中显式指明以拷贝方式构造该虚基类子对象，否则编译器将选择以缺省方式构造该子对象
  - class A { ... };
  - class X : virtual public A { ... };
  - class Y : virtual public A { ... };
  - class Z : public X, public Y {

```
public:
    Z (Z const& that) : X (that), Y (that), A (that) { ... }
};
```

虚基类子对象的赋值

Tarena Technology 达内科技

知识讲解

- 与构造函数和拷贝构造函数的情况不同，无论是否存在虚基类，拷贝赋值运算符函数的实现没有区别
  - class Z : public X, public Y {

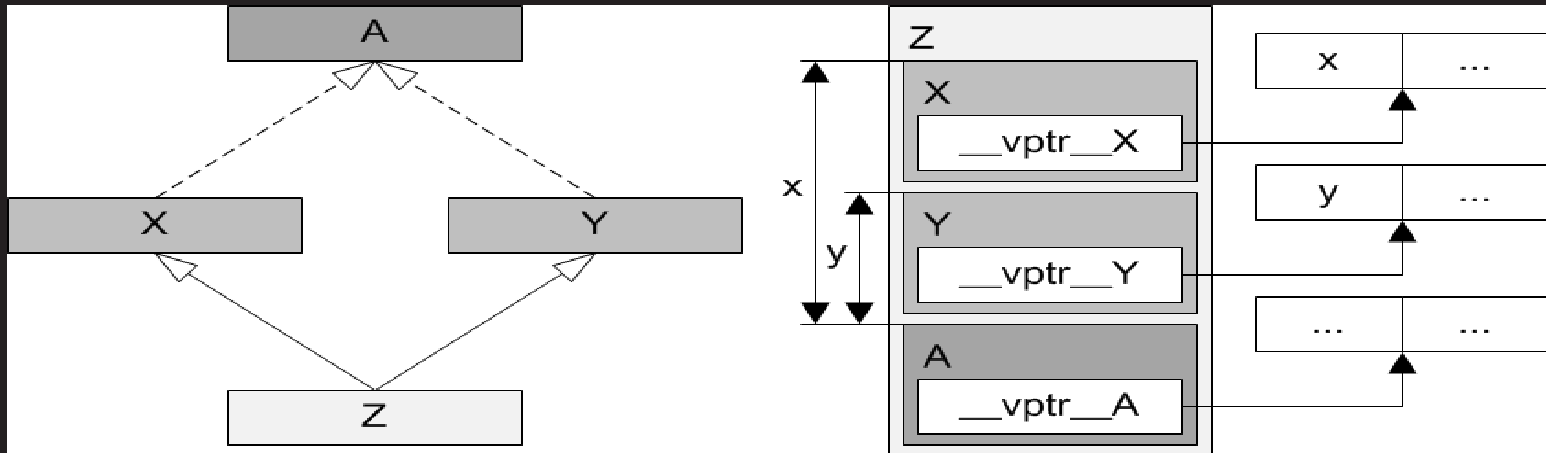
```
class Z : public X, public Y {
public:
    Z& operator= (Z const& rhs) {
        if (&rhs != this) {
            X::operator= (rhs);
            Y::operator= (rhs);
            // A::operator= (rhs); // 不需要
            ...
        }
        return *this; }
};
```

虚继承对象模型

Tarena Technology 达内科技

- 汇聚子类对象中的每个中间子类子对象都持有一个虚表指针，该指针指向一个被称为虚表的指针数组的中部，该数组的高地址侧存放虚函数指针，低地址侧存放虚基类子对象相对于每个中间子类子对象起始地址的偏移量
- 某些C++实现会将虚基类子对象的绝对地址直接存放在中间子类子对象中，而另一些实现(比如微软)则提供了单独的虚基类表，但它们的基本原理都是类似的

知识讲解



虚继承

【参见：TTS COOKBOOK 】

- 虚继承

总结和答疑