1. **继承中的指针和引用：**

我们知道只要有了继承关系，基类指针或引用就可以指向派生类对象。

base \*p = new Derive;

实际上p指向的是Derive对象中Base组件部分的地址，引用也一样，这样在多重继承中会引发问题，详见多重继承。

1. **虚析构函数：**

一般有继承关系的时候，将析构函数设为虚函数，例如以下代码：

Base \* p = new Derive;

Delete p;

如果析构函数不为虚，那么只会调用Base的析构函数；

如果析构函数为虚，那么先调用Derive的析构函数，然后调用Base的析构函数

1. **抽象基类：**

含有纯虚函数的类，这种类只能用作基类，不能生成对象。

注意与多重继承中的虚基类区别。

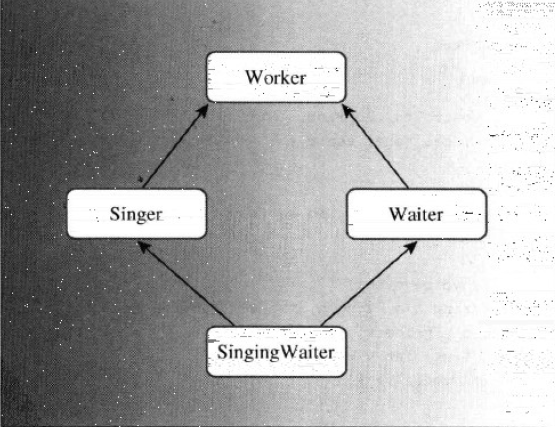
1. **各种继承方式：**

隐式向上转换意味着无需进行显示类型转换，就可以将基类指针或引用指向派生类对象。

各种继承方式：

|  |  |  |  |
| --- | --- | --- | --- |
| **特征** | **公有继承** | **保护继承** | **私有继承** |
| 公有成员变成 | 派生类的公有成员 | 派生类的公有成员 | 派生类的私有成员 |
| 保护成员变成 | 派生类的保护成员 | 派生类的公有成员 | 派生类的私有成员 |
| 私有成员变成 | 只能通过基类接口访问 | 只能通过基类接口访问 | 只能通过基类接口访问 |
| 能否隐式向上转换 | 是 | 是（但只能在派生类中）  *说明：只能在派生类中指的是在派生类的作用域可以，比如这样用：*  *Base \*p = this;* | 否  *说明：经过实测私有继承与保护继承一样，在派生类中也可以隐式向上转换，这可能与编译器有关* |

1. **多重继承和虚基类（虚继承）：**

****

1. 有多少个worker？

Class SingingWaiter : public Singer, public Waiter {…}

一个SingingWaiter对象有两个Worker组件，这将引起问题：例如如下代码：

SingingWaiter ed;

Worker \*pw = &ed; //报错，二义性

这种问题可以使用类型转换来解决：

Worker \*pw1 = (Singer \*) &ed;

Worker \*pw2 = (Waiter \*) &ed;

但是，真正的问题是，一个SingingWaiter对象有必要包含两个Worker组件么？为了解决这个问题，引入虚基类：

//virtual 和 public 次序无关紧要

Class Singer : virtual public Worker {…}

Class Waiter : public virtual Worker {…}

Class SingingWaiter : public Singer, public Waiter {…}

使用虚基类有新的构造函数准则：

旧的构造函数：

SingingWaiter(const Worker & wk, int p = 0, int v = Singer::other) : Waiter(wk, p),

Singer(wk, v) {}

这种构造函数，将wk通过2条不同的途径（waiter和Singer）将wk传递给Worker组件，为了避免这种冲突，C++规定这种情况禁止传递，上述构造函数仅仅初始化成员panache和 voice。然而，编译器在构造派生对象之前必须完成基类组件的构造，在上述情况，编译器将使用Worker的默认构造函数。

若不希望使用默认构造函数构造虚基类组件，则需要显示调用所需的基类构造函数。新的构造函数：

SingingWaiter(const Worker & wk, int p = 0, int v = Singer::other) : Worker(wk),

Waiter(wk, p), Singer(wk, v) {}

这种构造函数对于非虚基类是非法的，因为其跨越了一层继承关系直接使用Worker的构造函数，但是对于虚基类则合法。

1. 哪个方法？

SingingWaiter sw;

sw.show();

若SingingWaiter类没有重新定义（隐藏）show，那么根据继承关系，将调用基类的show方法，但是对于多重继承，将引发二义性。

可以使用作用域解析符：

sw.Singer::show();

更好的方法是在SingingWaiter中重定义show，并指出使用哪个show

void SingingWaiter::show()

{

Singer::show();

}

但是上述方法忽略了Waiter组件，改为：

void SingingWaiter::show()

{

Singer::show();

Waiter::show();

}

然而上述改动之后又存在打印多次Worker组件的问题，如何解决呢？

使用模块化：

void Worker::ShowData() const

{

cout << fullname << endl;

cout << ID << endl:

}

void Waiter::ShowData() const

{

cout << panache << endl;

}

void Singer::ShowData() const

{

cout << pv << endl;

}

void SiningWaiter::Show() const

{

Worker::ShowData();

Waiter::ShowData();

Singer::ShowData();

}