**继承**



1. Void print() 没有语法错误。

2. 派生类构造函数的执行顺序：

**1）调用基类的构造函数**

**2）调用内嵌对象的构造函数**

**3）执行派生类的构造函数**

3. 派生类定义了与基类相同的成员，此时基类的同名成员在派生类内不可见，也就是派生类成员隐藏了同名的基类成员，这种现象称为**继承时的同名成员隐藏规则**。

4. 派生类对象可以直接赋值给基类对象。

A a; // 基类对象

B b; // 派生类对象

a = b; // 直接赋值

注意，赋值后a的数据类型依然是A，因此不要企图使用a去访问派生类的成员，因此下面的语句是错误的：

a.j = 3; // 错误，A中不具有j成员

5. 派生类对象可以初始化基类的引用。

同样，一个函数的形式参数如果是一个基类的引用，在实际调用该函数的时候，可以传递一个派生类对象来代替基类对象。

6. 派生类对象的地址可以赋值给基类的指针。

**多态**

1. virtual只能使用在类定义的函数原型声明中，不能在成员函数实现的时候使用，也不能用来限定类外的普通函数。
2. virtual具有继承性，在派生类覆盖基类虚成员函数时，既可以使用virtual，也可以不用virtual来限定，二者没有差别，默认派生类中的重写函数是具有virtual的。
3. 对虚函数来说，基类与派生类同名函数的参数列表是完全相同的，唯一不同的就是函数所属的类不同，也就是调用函数的对象不同。
4. 声明虚函数后，系统会根据以下规则来判断函数的调用是否实现了动态多态：

1）派生类的函数是否**覆盖**了基类的虚函数

相同的函数名

相同的参数列表

相同的函数类型

1. 虚函数调用时，应通过基类的指针或者引用调用。
2. 一个具有纯虚函数的基类称为**抽象类(abstract class)。**
3. 抽象类中不仅包括纯虚函数，也可包括其他函数。
4. **抽象类不能被实例化，但可以声明抽象类的指针或引用。**

*shape x ;* *// error，抽象类不能建立对象*

*shape \*p ;* *// ok，可以声明抽象类的指针*

*shape f ( ) ;* *// error, 抽象类不能作为返回类型*

***void g ( shape ) ;* *// error, 抽象类不能作为参数类型***

***void g( shape & ); //ok, 抽象类的引用可以作为参数类型***

shape\* h ( shape &) ; *// ok，可以声明抽象类的引用*