1. 多态性
2. 向上类型转换

1、含义：基类类型兼容派生类类型，即派生类可以默认自动转换为基类类型

如：若有base b ; derived d;

则b=d; base &b=d; base \*p=&d都合法

1. 原理： 某一个指针最初被创建时，即与其类进行了静态绑定

由此，不管基类指针指向哪个对象，调用的总是基类的成员函数

1. 应用示例：作为普通函数编写的析构函数

Void del (base \*p){ delete p; }

//由于基类兼容派生类，故这样的析构函数可以析构base类家族的所有派生类

1. 虚函数
2. 声明：virtual void show()
3. 含义：与类进行**动态绑定**，通过基类的引用或者指针调用虚函数来实现动态绑定

//由此可以实现用基类指针指向派生类的地址

基类的成员函数

1. 注意： ①必须是成员函数

②构造函数不能是虚函数

③析构函数最好是虚函数

//当析构函数为虚函数时，析构时会临时进行动态绑定，调用的就是 派生类的析构函数而非基类的析构函数

④当基类的成员函数具有虚特性时，这样的虚特性是可以继承的

故基类的成员函数其实可以全部设置为虚特性，方便之后进行动态绑定

这样的好处在于此后可以使用一种基类指针操作各种派生类

1. 纯虚函数：基类中没有具体的实现代码的成员函数可设置为纯虚函数（设置为=0）， 这样的函数功能在派生类中实现（常用指针作为参数与返回值）
2. 覆盖检查：override与final
3. override：自动检查函数签名（函数的“身份证”）是否完全一致，

比如 有无带const结尾为不同的函数

1. final：将一个类设置为final之后它就不能再派生出派生类，用来避免类被继承，或 是基类的函数被改写
2. 抽象类
3. 含义：至少含有一个纯虚函数的类
4. 规则：
   1. 抽象类不能用来创建对象，只能用来创建新的派生类
   2. 不能直接用抽象类对象作为函数的参数与返回值
   3. 可以使用指向抽象类的指针或引用
5. 作用：规定类家族的公共行为