面向对象学习笔记

一、子类的访问权限与类型转化

问题场景；

```
template<typename T>
class X {
private:
    int member;
public:
    template<typename U>
    void Method(X<U>& y) {
        &y.member;
    }
};

int main() {
    X<int> x;
    X<float> y;
}
```

1 凡是public的成员都能访问
2 如果member是直接定义在Y里面的protected或private成员，只有当X和Y相同，才能够访问
3 如果member是定义在Y的某个直接基类Base里面的成员，那么当x和y相同时：member是Base的protectced成员，可以访问，private成员，不可以访问
4 如果member是定义在Y的某个直接基类Base里面的成员，那么当X和Y不相同时：
class Y : public Base: 仅当member是Base的public成员时，可以访问
class Y: protectced Base: 不可以访问，不仅如此，在X内部会将Base不当做Y的基类，无法做指针或引用的类型转换
class Y: private Base: 同上
5 如果member是定义在Y的某个直接基类Base里面的成员，那么当X继承自Y，但是X通过自己访问Y成员的时候：
class Y : public Base : 仅当member是Base的public或protected成员时可以访问
class Y : protectced Base : 同上
class Y : private Base : 不可以访问 不仅如此，在X内部会将Base不当做Y的基类，无法做指针或引用的类型转换
6 如果X是Y的内部类，那么Y能看到什么，X就能看到什么，不受以上约束
7 把y换成this指针，规则也成立，此时x与y相同

二、设计一个不可复制的类

> 要点：默认构造函数设为default 左值和右值复制构造函数设为delete 将2种引用的重载运算符=运算符也delete 最后private继承，因为，没有必要将子类再转化回去

```
class NotCopyable {
    NotCopyable() = default;
    NotCopyable(const NotCopyable &) = delete;
    NotCopyable(NotCopyable &&) = delete;
    NotCopyable &operator=(const NotCopyable &) = delete;
    NotCopyable &operator=(NotCopyable &&) = delete;
};
```

三、虚析构函数与内存泄漏

> 1. 场景：父类指针指向子类时释放
> 2 最好不要将父类指针指向子类数组

四、纯虚函数

> 访问者模式：思路，在基类中创建访问者基类和一个Accept()的纯虚函数，再创建针对不同子类的visit纯虚函数，然后，各子类中实现Accept方法，通过访问者更具传入参数调用不同子类的访问函数，最后实现一个具体需求的访问者类，实现visit纯虚函数将需求加进去