以Java和C++为例，说明OOPL中单继承和多继承的差异和优缺点。

C++支持多继承，其中派生类保留基类的所有数据成员（包括方法和属性）的副本，使用多继承时可能会出现重复继承，如继承结构中出现菱形，可以通过虚基类的使用来解决。然而，这样增加了C++中对多重继承使用上的复杂性，用户不仅要负责找出菱形继承的相同父类、在所有对该父类派生的继承树分支上使用虚基类，还要记住在派生类中对虚基类进行初始化。对于多继承中同一个层次上不同分支的名字冲突问题，用户需要使用基类名来限定有冲突的函数名。

Java禁止多重继承，采用“单继承加多接口”的方式来模拟多继承。从OOP的角度来看，类能够解决复用的问题，而接口纯粹是一种外部功能契约，不存在代码复用的能力。多继承的效果体现为可以让子类复用多个父类代码，并且所有父类型都可以接收子类实例。为了实现这样的效果，Java使用委派和聚合，即可以选择一个类进行继承，然后聚合其它类的实例，同时对各个类抽象出其接口，声明为对其实现。这样，从接口调用的方法都被明确地委派给了实现共同功能的对象，它们本来在多重继承中可以自动实现，现在要手工实现。Java虽然实现了外部功能契约的集成，但实际上和多继承所体现的多复用是不一样的。

我理解Java的模拟多继承是一种接口继承（interface inheritance），通过在类中手工实现从接口调用的方法，实际上达成了需要父类对象参数的函数可以接受子类对象参数、可以通过父类的接口使用子类对象（调用其方法）的效果，符合里氏替换原则。C++的真·多继承是一种实现继承（implementation inheritance），通过继承在子类中直接复用父类的代码（成员变量和成员函数）。Java中interface相当于由编程语言强制为类似C++中的纯虚函数，每个interface的方法都必须在成员中手动实现，也就回避了名字冲突和重复继承的问题。interface和虚基类都是解决多重继承中问题的工具，区别在于interface不能实例化，在派生类中也不保存“副本”，而虚基类的效果是即使在菱形结构中也只保留一个副本；C++中的虚基类被类继承，提供变量和方法的复用，Java中的interface不是被类继承而是要求类实现它们，interface不提供复用，而仅仅提供接口方法的规定约束。

单继承的局限在于在继承的层次结构中，重用只能发生在一个继承路径上，无法重用来自不同继承路径上的代码，如果想要用一个派生类实现对不同基类功能方法的集合，就不能使用单纯的继承，而必须像Java那样手动实现抽象接口中的方法，或者摒弃继承改用（作为成员）聚合；优点是实现起来比较简单明了，回避了菱形继承结构中可能出现的种种重复和冲突问题。像C++这样多重继承的优势是对象可以复用多个基类中的接口方法，缺点是需要处理来自不同超类的名字冲突，名字冲突可能导致的子类的二义性行为，以及如何重复的继承。