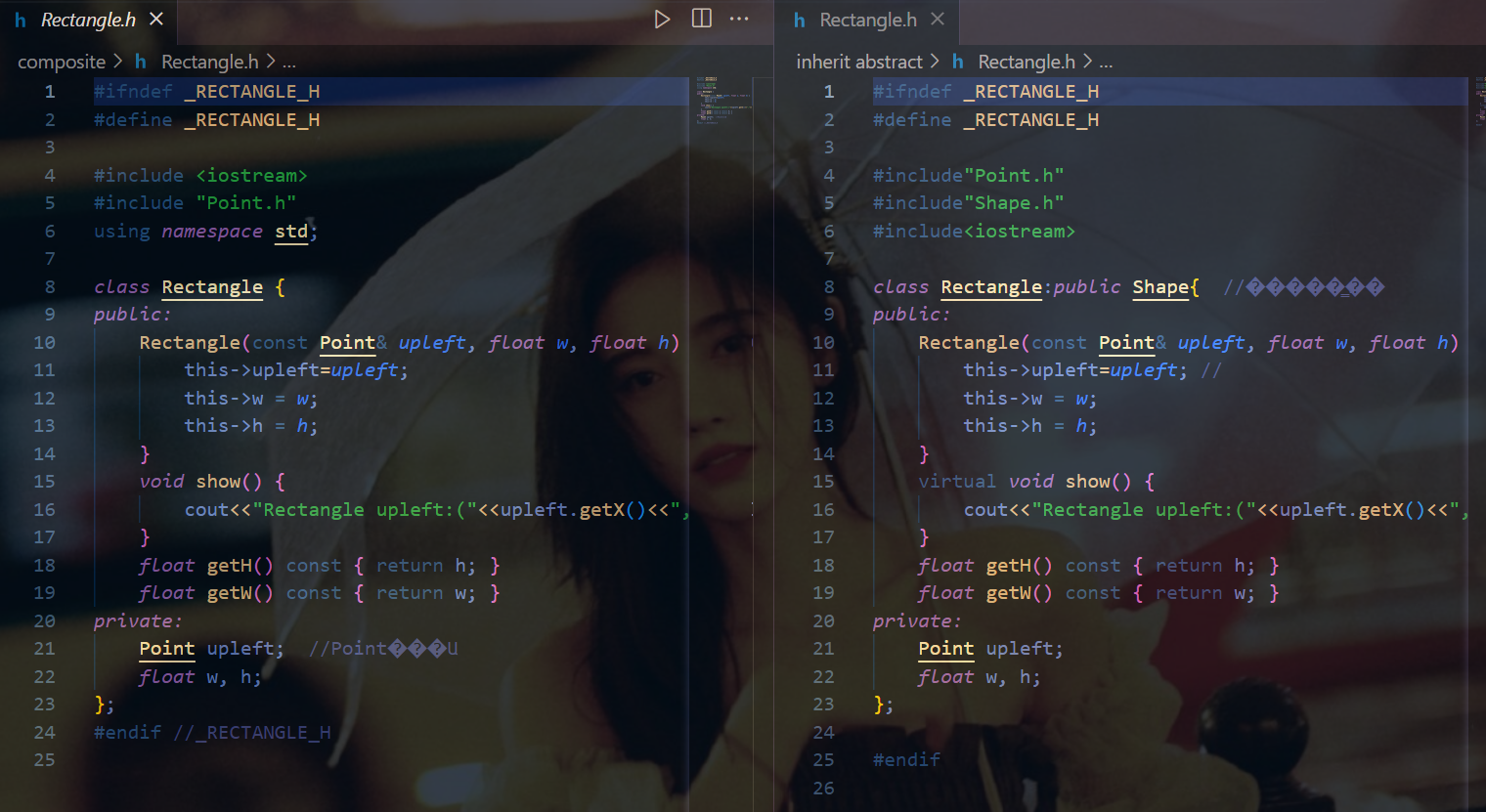
（1）附件13.zip是“设计显示一个UI界面中包含的几何图形元素”的C++例子，其中提供了两种设计，主调测试程序为TestUI.cpp。请分别分析评价这两种设计。如果需求改变，如给界面中增加圆形元素，请分别按原设计方法增加这个功能。对比所需修改的部分，分析哪种设计修改量少、通用性高、更适用于软件复用？

评价设计：



在Rectangle类的实现上，二者十分相似。组合：Rectangle类只使用Point类的接口，所以组合保持封装的特性没有表现出来；继承：Circle类几乎重写了Shape类的所有接口，所以继承选择性覆盖的特性也没有体现出来。



由上图可知，组合方法需要对UI类做出的修改，而继承方法不需要对UI类做出任何修改。继承能够发挥多态的优势，在许多类拥有相同的接口时能够抽象出他们的共同方法，从而在新增一个类时也能够直接调用这个共同的方法来完成任务，故使用继承的设计修改量少、通用性高、更适合复用。

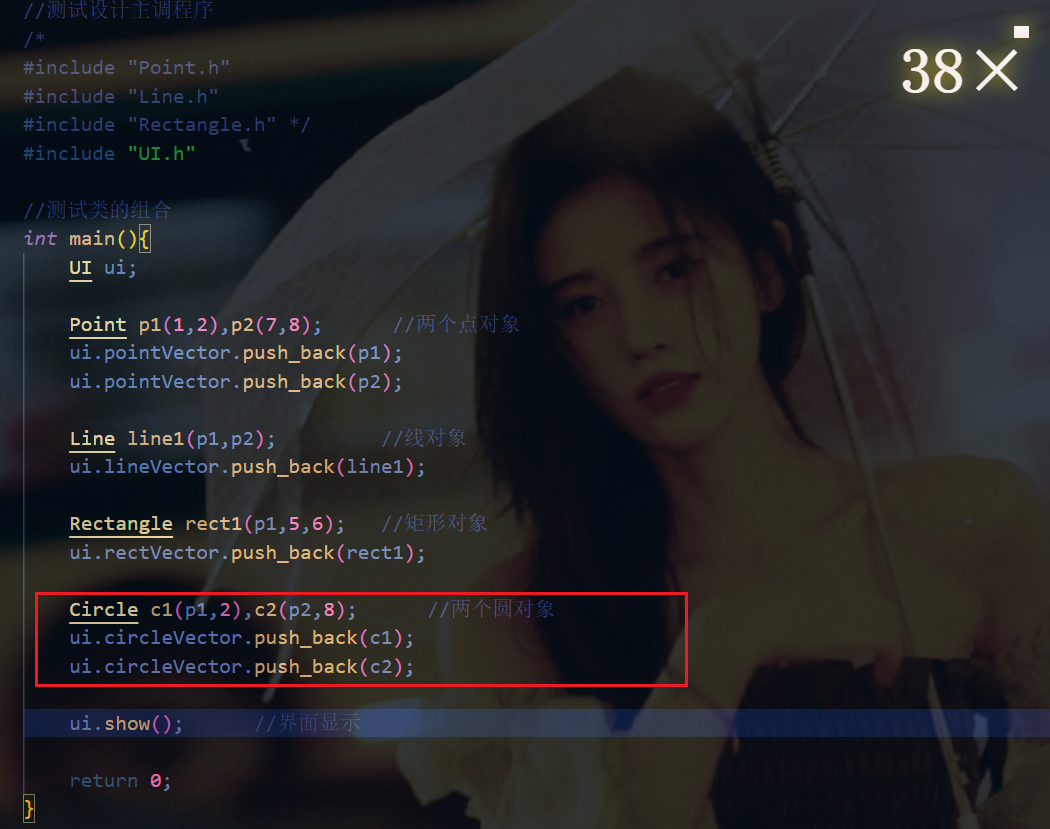
分析：本次修改中，继承中的设计修改量少，通用性高，更适合软件复用。因为继承可以通过选择性地覆载被继承的定义，可以改变和特化继承方法的行为。这个特性可以帮助我们快速创建具有新行为的、新的类型的对象。

composite的更改：

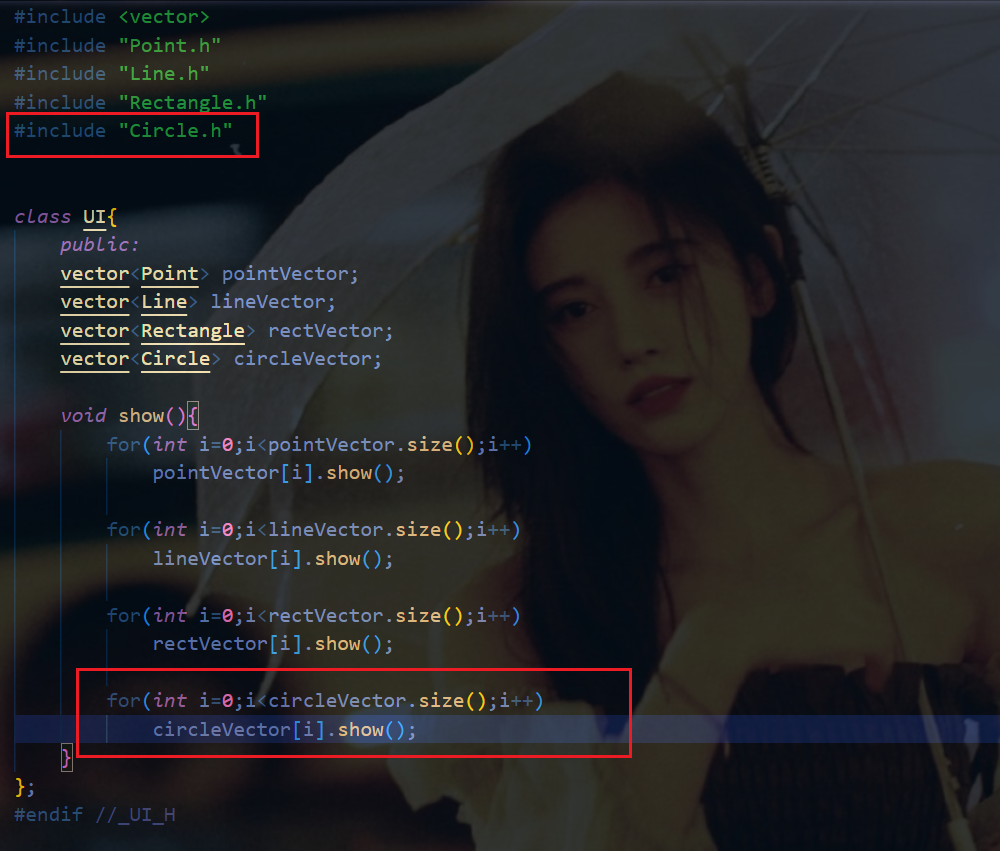
circle.h



TestUI.cpp



UI.h



inherit abstract的更改：

circle.h



TestUI.cpp



（2）实现教材图6-13中的 Engineer与Software Engineer的两种设计实例，并进行讨论。如果要利用组合来实现可替换性，需要怎样设计？

继承设计：



组合设计：



利用组合来实现可替换性的设计：



（3）参照课本P224图6-14，设计体现德米特法则的实例。分析该种设计的特点?

实例1：（不符合德米特法则的实例）



这是未遵循德米特法则的实例，无论是Item类修改还是Sale类修改，print函数都需要修改。同时，为了实现print函数，用户必须了解Item类和Sale类的内部状况。



这是遵循德米特法则的实例。customerAccount不直接依赖Item，改变Item只需要修改Sale，而无需改动customerAccount。

设计特点主要有以下几个方面：

1.限制对象之间的耦合：德米特法则要求对象之间的交互应该尽量减少，避免直接依赖其他对象的内部细节。这样可以降低对象之间的耦合度，提高代码的灵活性和可维护性。

2.提高代码的可复用性：德米特法则鼓励对象之间的通信应该通过接口进行，而不是直接调用其他对象的方法。这样可以提高代码的可复用性，使得代码更加模块化和易于维护。

3.优化代码的性能：德米特法则要求对象之间的通信应该尽量少，避免过多的对象之间的交互，从而优化代码的性能。

4.增强代码的可读性：德米特法则要求对象之间的交互应该尽量简单明了，避免过于复杂的依赖关系，从而增强代码的可读性和可理解性。