**1. 模块设计的分析与评价**

**（1）评价设计**

**组合方法：**

①**具有一定的可重用性**。许多代码可以重复利用，稍微进行修改即可。

②**灵活性较高**。可以根据不同的要求与目的，灵活地进行修改不同模块或者类。

③**模块化较强**。组合方法可以将整个系统划分成多个独立的模块，各模块之间利用接口关联，便于维护和测试，同时降低了整个系统的复杂度。

④具**有一定的可拓展性**。可以根据需求的变化进行修改和拓展，新的模块、类或对象可以被添加到现有的组合中，而不会影响已有的功能。

⑤**高内聚低耦合**。组合设计强调模块之间的高内聚和低耦合，即每个模块都应该尽可能地独立，模块之间的依赖应该尽可能地少，这样可以提高系统的稳定性和可维护性。

**继承方法：**

①**代码重用性较好**。继承方法允许子类继承父类的代码和属性，避免了重复编写代码的问题，提高了代码的重用性。

②**具有一定的灵活性**。通过继承，子类可以从父类继承一些基本的属性和方法，然后在此基础上进行修改和扩展，从而实现更为灵活的设计。

③**可扩展性较好**。继承设计方法允许开发人员在已有的类的基础上添加新的方法和属性，以满足新的需求或者改进原有的设计，提高了系统的可扩展性。

④**具有多态性**。通过继承，子类可以覆盖父类的方法，实现多态性的效果，这种方式可以提高程序的可读性和可维护性。

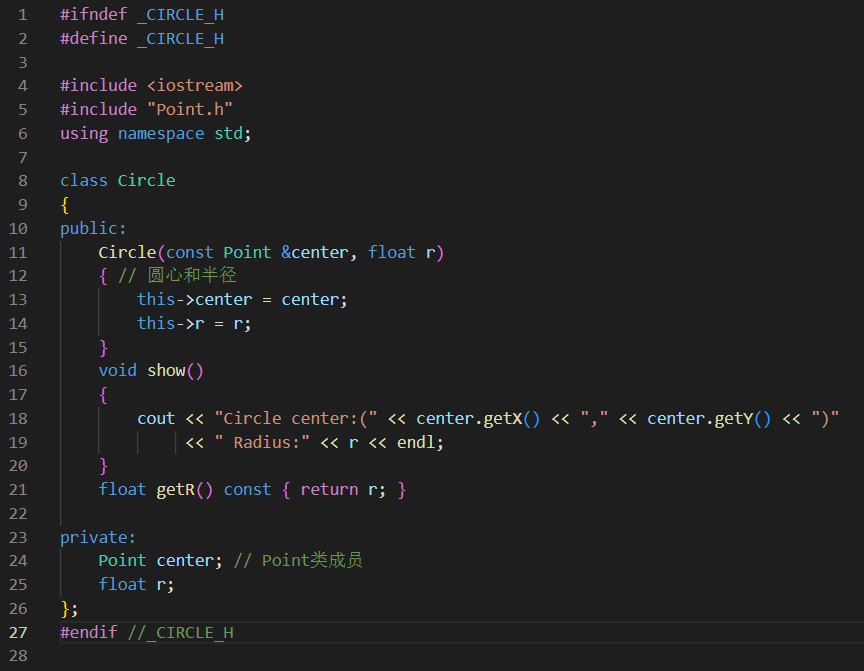
⑤**继承层次清晰**。通过继承设计方法，可以构建一个清晰的继承层次结构，方便开发人员理解和维护代码。

⑥**便于代码重构**。继承设计方法也为代码重构提供了便利，可以将相似的代码抽象出来，形成一个基类，避免代码的重复。

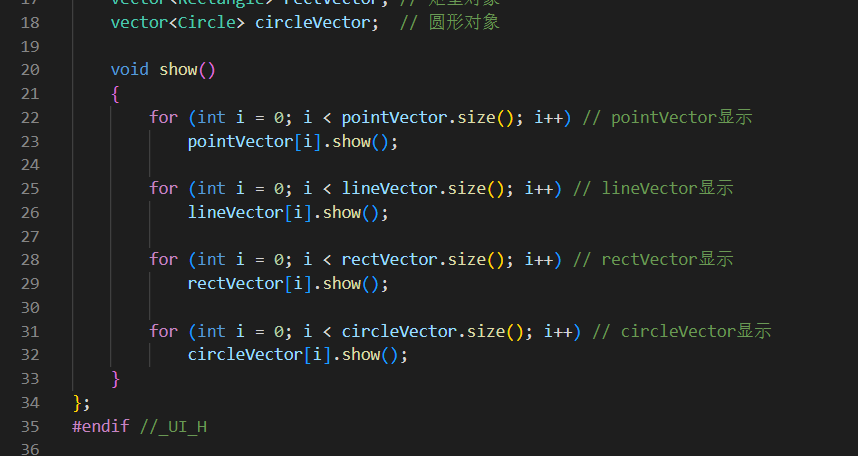
**（2）修改增加圆形**

**组合方法：**

①新加circle.h文件，增加Circle类，包括构造函数、展示函数和成员

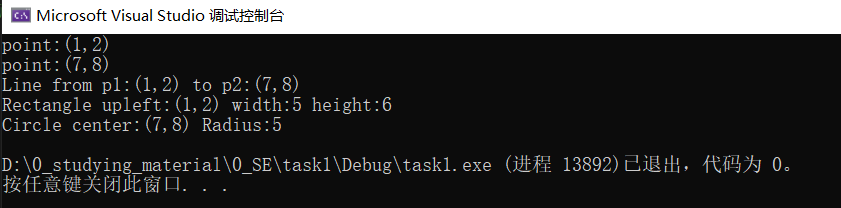


②在UI.h中增加圆形对象的数组，修改show函数，增加圆形对象的展示



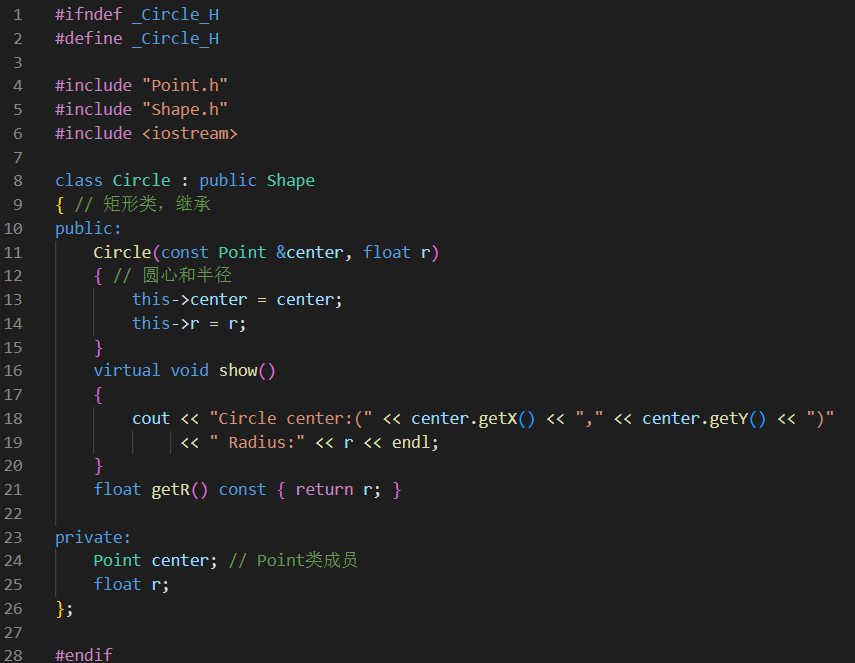
③在TestUI.cpp中，在main函数里，新加入圆形元素

④最终结果展示：



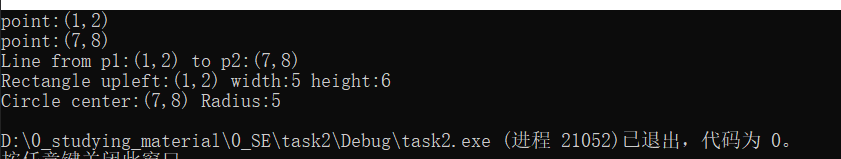
**继承方法：**

①新加circle.h文件，增加Circle类，包括构造函数、展示函数（为虚函数）和成员



②在TestUI.cpp中，在main函数里，新加入圆形元素

③最终结果展示：

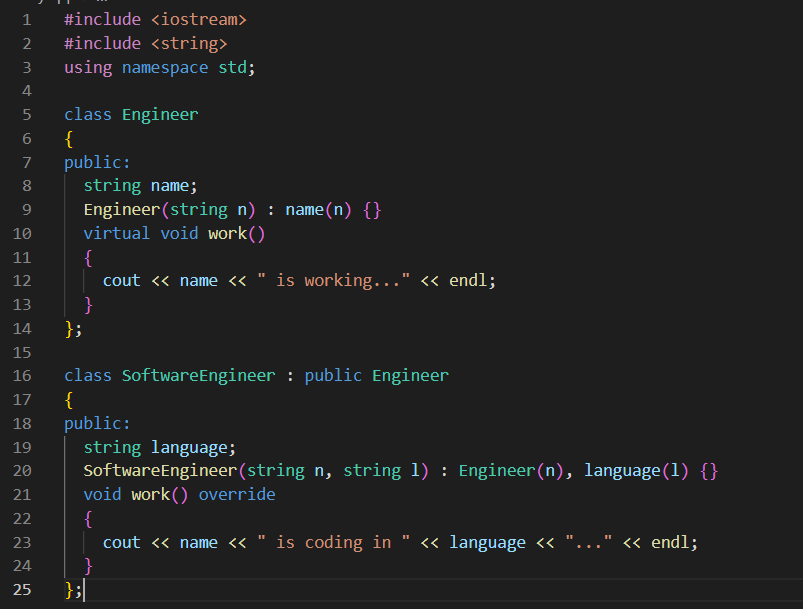


**（3）对比所需修改的部分，分析哪种设计修改量少、通用性高、更适用于软件复用**

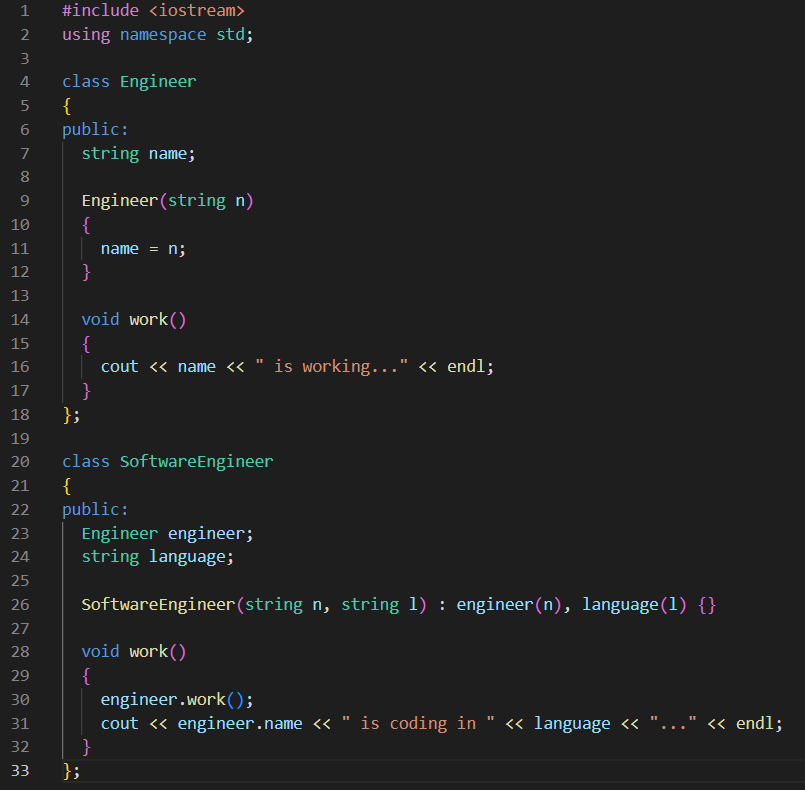
显然，继承设计的修改量较少，通用性较高，更适用于软件复用。因为继承方法主要是通过一个总的类作为父类，其他子类继承这个父类的框架，同时利用虚函数和多态函数实现了不同类之间利用相同函数名进行调用类似的功能，从而大大减少了修改量，更利于复用。

**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？**

继承方法：



组合方法：



若要利用组合来实现可替换性，需要将接口或抽象类定义为一个成员变量，并在构造函数中传入具体实现类的对象。例如，在上面的例子中，可以将Engineer定义为一个接口或抽象类，并将其作为SoftwareEngineer的成员变量，在构造函数中传入具体的Engineer对象。即在一个类中加入另一个类的对象，通过接口来实现替换和使用。

**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?**

假设电商平台中有以下几个类：

User：表示用户，包含用户的基本信息，如用户名、密码等。

Order：表示订单，包含订单的详细信息，如订单号、下单时间、购买商品列表等。

Product：表示商品，包含商品的详细信息，如商品名称、价格、库存等。

ShoppingCart：表示购物车，包含用户选择的商品列表，可以添加、删除商品，计算总价等。

Payment：表示支付，包含支付方式、金额等信息。

根据德米特法则，我们需要让各个类之间的依赖关系尽量松散，可以采用以下方式来实现：

User类只需要关心自己的信息，不需要直接访问Order、Product、ShoppingCart和Payment等其他类的信息。这可以通过将User类的属性和方法设置为私有或保护来实现，只能通过公共接口来访问。

Order类只需要关心与订单相关的信息，如订单号、下单时间、购买商品列表等，不需要直接访问User、Product、ShoppingCart和Payment等其他类的信息。这可以通过将Order类的属性和方法设置为私有或保护来实现，只能通过公共接口来访问。

Product类只需要关心与商品相关的信息，如商品名称、价格、库存等，不需要直接访问User、Order、ShoppingCart和Payment等其他类的信息。同样，这可以通过将Product类的属性和方法设置为私有或保护来实现，只能通过公共接口来访问。

ShoppingCart类只需要关心用户选择的商品列表，可以添加、删除商品，计算总价等，不需要直接访问User、Order、Product和Payment等其他类的信息。同样，这可以通过将ShoppingCart类的属性和方法设置为私有或保护来实现，只能通过公共接口来访问。

Payment类只需要关心支付方式、金额等信息，不需要直接访问User、Order、Product和ShoppingCart等其他类的信息。同样，这可以通过将Payment类的属性和方法设置为私有或保护来实现，只能通过公共接口来访问。

以上设计体现了德米特法则的思想，各个类之间的依赖关系更加松散，耦合度更低，具有以下特点：

①每个类都只关注自己的本职工作，不会涉及到其他类的实现细节，使得代码更加清晰简洁。

②通过公共接口来访问其他类的信息，降低了系统的复杂度，提高了系统的可维护性和扩展性。

③各个类之间的耦合度降低，可以更加方便地进行单元测试和模块化开发，提高了代码的质量和可重用性。