1. 开放-封闭原则：函数的参数直接设置为CommonThermometer和ElectricHeater，如果后续该调节软件使用其它类型的温度计或者其他类型的加热器，则必须修改源码中这些参数的定义，故无法做到“对扩展开放、对修改封闭”。利用Liskov替换原则思想，可以定义Thermometer和Heater抽象类，其中比如read、engage和disengage均定义成虚函数，Regulate函数的参数使用这些基类。CommonThermometer和ElectricHeater都是子类，后续如需使用其他类型的温度计或者加热器，分别添加新子类即可。

依赖倒置原则：程序依赖于具体的温度计和加热器类，持有指向具体类的引用，解决方法同上，分别定义抽象父类对Regulate函数的参数进行替换即可。

单一职责原则、接口隔离原则和迪米特法则：即对象应该包含单一的职责，不应该强迫客户依赖于它们不用的方法且应该控制最少的信息流量。Regluate函数每次调用的时候要指定四件事，分别是使用哪个温度计、哪个加热器，以及需要控制的炉温范围（上限和下限）。它包含了相对静态的温度范围设置和动态的炉温控制操作。而在实际使用场景中，可能选定了一对温度计和加热器之后就不会频繁更换了，即使确实需要更换也由专人管理，而操作员可能会经常改变温度范围设置。此外，操作员也不需要知道read、engage、disengage这些接口。考虑定义Regulator类对Thermometer和Heater进行聚合，提供接口void setTemp(double minTemp, double maxTemp)和void regulate()，则只有配置人员创建类时需要知道具体使用的温度计、加热器类型，操作员可以设置温度范围而不需要知道这些，实际运行时调用regulate，隐藏了内部的细节。

1. ①父类和子类之间的继承关系是一种“IS-A”关系，而“IS-A”关系是由类的行为定义的。根据Liskov替换原则，凡是引用基类的地方必须能透明地引用子类的对象。而正方形比长方形多出一条“长和宽应该相等”的要求，在原本引用长方形的地方引用正方形可能会出现错误。比如强行修改“长方形”的长或宽之一，则可能会违背正方形的定义，如果强制正方形每次同时修改长和宽，那么这个操作对“长方形”的调用者又是意料之外的。

② 1）封装变化点：体现了开放-封闭原则。对变化点的封装使得软件实体可以在不修改原有代码的情况下进行扩展，实现相对稳定的抽象层，有助于增加软件复用性。

2）对接口进行编程：体现了依赖倒置原则，即针对接口而非实现编程，客户不依赖也不需要知道与它无关的方法（接口），从而实现系统的高内聚和低耦合。

3）多使用组合，而不是继承：体现了合成/聚合复用原则，即优先使用对象组合而不是继承来达到复用的目的。这样可以实现运行时动态配置组件的功能，比静态的继承行为更灵活，而且还能防止类层次规模的爆炸性增长。