1. 开闭原则

对接口开放，对修改关闭。也就是在不改变源码的基础上，对系统功能进行扩展开发。提高系统的可复用性和可维护性。

2. 里氏替换原则

继承类必须确保超类所拥有的性质在子类中仍然成立。

- 子类可以实现父类的抽象方法，但是不能覆盖父类的非抽象方法。
- 子类中可以增加自己的特有方法。
- 当子类方法重载父类的方法时，方法的前置条件（即方法的输入/入参）要比父类方法输入的参数更宽松。（参数范围大于父类）
- 当子类实现父类的方法（重载/重写/实现抽象方法），方法的后置条件（即方法的输出/返回值）要比父类更严格或者相等。（返回类型，父类的子类）

3. 依赖倒置原则

依赖倒置原则：`Dependence Inversion Principle`，简称为 `DIP`。其指的是在设计代码结构时，高层模块不应该依赖低层模块，而是都应该依赖其抽象。抽象不应该依赖细节，细节应该依赖抽象。通过依赖倒置原则可以减少类与类之间的耦合性，提高系统的稳定性，提高代码的可读性和可维护性，而且能够降低修改程序所带来的风险。

4. 单一职责原则

单一职责原则：`Single Responsibility Principle`，简称为 `SRP`。其指的是不要存在多于一个导致类变更的原因。假如我们有一个类里面有两个职责，一旦其中一个职责发生需求变更，那我们修改其中一个职责就有可能导致另一个职责出现问题，在这种情况应该把两个职责放在两个 Class 对象之中。

单一职责可以降低类的复杂度，提高类的可读性和系统的可维护性，也降低了变更职责引发的风险。

5. 接口隔离原则

接口隔离原则：`Interface Segregation Principle`，简称为 `ISP`。接口隔离原则符合我们所说的高内聚低耦合的设计思想，从而使得类具有很好的可读性、可扩展性和可维护性，在设计接口的时候应该注意以下三点：

- 一个类对其它类的依赖应建立在最小的接口之上。
- 建立单一的接口，不建立庞大臃肿的接口。
- 尽量细化接口，接口中的方法应适度。

6. 迪米特法则

迪米特法则：`Law of Demeter`，简称为 `LoD`，又叫作最少知道原则（`Least Knowledge Principle`，`LKP`）。是指一个对象对其它对象应该保持最少的了解，尽量降低类与类之间的耦合。

中介者模式，就遵守了该法则。

7. 合成复用原则

合成复用原则：`Composite Reuse Principle`，简称为`CRP`，又叫组合/聚合复用原则（`Composition/Aggregate Reuse Principle`，`CARP`）。指的是在软件复用时，要尽量先使用组合（has-a）或者聚合（contains-a）等关联关系来实现，这样可以使系统更加灵活，降低类与类之间的耦合度，一个类的变化对其它类造成的影响相对较少。

继承通常也称之为白箱复用，相当于把所有的实现细节都暴露给子类。组合/聚合也称之为黑箱复用，对类以外的对象是无法获取到实现细节的。

为什么要这样做？有一下两点原因：

1. 通过继承来进行复用的主要问题在于继承复用会破坏系统的封装性，因为继承会将基类的实现细节暴露给子类，由于基类的内部细节通常对子类来说是可见的，所以这种复用又称“白箱”复用，如果基类发生改变，那么子类的实现也不得不发生改变；从基类继承而来的实现是静态的，不可能在运行时发生改变，没有足够的灵活性；而且继承只能在有限的环境中使用（如类没有声明为不能被继承）。
2. 由于组合或聚合关系可以将已有的对象（也可称为成员对象）纳入到新对象中，使之成为新对象的一部分，因此新对象可以调用已有对象的功能，这样做可以使得成员对象的内部实现细节对于新对象不可见，所以这种复用又称为“黑箱”复用，相对继承关系而言，其耦合度相对较低，成员对象的变化对新对象的影响不大，可以在新对象中根据实际需要有选择性地调用成员对象的操作；合成复用可以在运行时动态进行，新对象可以动态地引用与成员对象类型相同的其他对象。