**利斯科夫替换原则（里氏代换原则）**：如果对每一个类型为S的对象o1，都有类型为T的对象o2，使得以T定义的所有程序P在所有的对象o1都代换成o2时，程序P的行为没有变化，那么类型S是类型T的子类型。即所有引用基类（父类）的地方必须能透明地使用其子类的对象。可以通俗表述为：在软件中如果能够使用基类对象，那么一定能够使用其子类对象。把基类都替换成它的子类，程序将不会产生任何错误和异常，反过来则不成立，如果一个软件实体使用的是一个子类的话，那么它不一定能够使用基类。里氏代换原则是实现开闭原则的重要方式之一，由于使用基类对象的地方都可以使用子类对象，因此在程序中尽量使用基类类型来对对象进行定义，而在运行时再确定其子类类型，用子类对象来替换父类对象。

**单一职责原则**：在软件系统中，一个类只负责一个功能领域中的相应职责，就一个类而言，应该仅有一个引起它变化的原因。一个类（或者大到模块，小到方法）承担的职责越多，它被复用的可能性越小。而且如果一个类承担的职责过多，就相当于将这些职责耦合在一起，当其中一个职责变化时，可能会影响其他职责的运作。 类的职责主要包括两个方面：数据职责和行为职责，数据职责通过其属性来体现，而行为职责通过其方法来体现。单一职责原则是实现高内聚、低耦合的指导方针，在很多代码重构手法中都能找到它的存在，它是最简单但又最难运用的原则，需要设计人员发现类的不同职责并将其分离，而发现类的多重职责需要设计人员具有较强的分析设计能力和相关重构经验。

**开闭原则**：抽象化是开闭原则的关键。 绝大部分的设计模式都符合开闭原则，在对每一个模式进行优缺点评价时都会以开闭原则作为一个重要的评价依据，以判断基于该模式设计的系统是否具备良好的灵活性和可扩展性。

**德（迪）米特法则**： 迪米特法则(Law of Demeter, LoD)又称为最少知识原则(Least Knowledge Principle, LKP)，它有多种定义方法，其中几种典型定义如下：(1) 不要和“陌生人”说话。 (2) 只与你的直接朋友通信。 (3) 每一个软件单位对其他的单位都只有最少的知识，而且局限于那些与本单位密切相关的软件单位。迪米特法则可分为狭义法则和广义法则。在狭义的迪米特法则中，如果两个类之间不必彼此直接通信，那么这两个类就不应当发生直接的相互作用，如果其中的一个类需要调用另一个类的某一个方法的话，可以通过第三者转发这个调用。

**依赖转换原则**：高层模块不应该依赖低层模块，它们都应该依赖抽象。抽象不应该依赖于细节，细节应该依赖于抽象。即要针对接口编程，不要针对实现编程。实现开闭原则的关键是抽象化，并且从抽象化导出具体化实现，如果说开闭原则是面向对象设计的目标的话，那么依赖倒转原则就是面向对象设计的主要手段。常用实现方式之一是在代码中使用抽象类，而将具体类放在配置文件中。 依赖倒转原则要求客户端依赖于抽象耦合，以抽象方式耦合是依赖倒转原则的关键。

**合成复用原则**：又称为组合/聚合复用原则(Composition/ Aggregate Reuse Principle, CARP)，其定义如下：尽量使用对象组合，而不是继承来达到复用的目的。合成复用原则就是指在一个新的对象里通过关联关系（包括组合关系和聚合关系）来使用一些已有的对象，使之成为新对象的一部分；新对象通过委派调用已有对象的方法达到复用其已有功能的目的。简言之：要尽量使用组合/聚合关系，少用继承。在面向对象设计中，可以通过两种基本方法在不同的环境中复用已有的设计和实现，即通过组合/聚合关系或通过继承。继承复用：实现简单，易于扩展。破坏系统的封装性，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；只能在有限的环境中使用。（“白箱”复用 ）。组合/聚合复用：耦合度相对较低，选择性地调用成员对象的操作；可以在运行时动态进行。（“黑箱”复用）

在项目中如何应用：

**里氏代换原则：**在程序中尽量使用基类类型来对对象进行定义，而在运行时再确定其子类类型，用子类对象来替换父类对象。构造一个总的基类，对于不同地形的生成需求再构造对应子类。

**单一职责原则：**对类进行重构，将类尽量按不同的功能分开，例如地形生成应与植被生成分别属于两个不同的类。

**开闭原则：**设计类时考虑到之后的扩展性，尽量在不改变源代码的前提下实现对类的扩展。

**德（迪）米特法则：**减少类与类之间的调用关系，降低类的成员变量和函数的访问权限，把不会被访问的成员设置为私有变量，提高代码封装性。

**依赖转换原则：**在代码中使用抽象类，将具体类放在配置文件中。

**合成复用原则：**谨慎使用继承，避免破坏系统的封装性。