一、三大特性

封装

利用抽象数据类型将数据和基于数据的操作封装在一起，使其构成一个不可分割的独立实体。数据被保护在抽象数据类型的内部，尽可能地隐藏内部的细节，只保留一些对外接口使之与外部发生联系。用户无需知道对象内部的细节，但可以通过对象对外提供的接口来访问该对象。

优点：

减少耦合：可以独立地开发、测试、优化、使用、理解和修改

减轻维护的负担：可以更容易被程序员理解，并且在调试的时候可以不影响其他模块

有效地调节性能：可以通过剖析确定哪些模块影响了系统的性能

提高软件的可重用性

降低了构建大型系统的风险：即使整个系统不可用，但是这些独立的模块却有可能是可用的

继承

继承实现了 IS-A 关系，例如 Cat 和 Animal 就是一种 IS-A 关系，因此 Cat 可以继承自 Animal，从而获得 Animal 非 private 的属性和方法。

继承应该遵循里氏替换原则，子类对象必须能够替换掉所有父类对象。

Cat 可以当做 Animal 来使用，也就是说可以使用 Animal 引用 Cat 对象。父类引用指向子类对象称为 向上转型 。

多态

多态分为编译时多态和运行时多态：

编译时多态主要指方法的重载

运行时多态指程序中定义的对象引用所指向的具体类型在运行期间才确定

运行时多态有三个条件：

继承

覆盖（重写）

向上转型

二、类图

泛化关系 (Generalization)

用来描述继承关系，在 Java 中使用 extends 关键字。

实现关系 (Realization)

用来实现一个接口，在 Java 中使用 implement 关键字。

聚合关系 (Aggregation)

表示整体由部分组成，但是整体和部分不是强依赖的，整体不存在了部分还是会存在。

组合关系 (Composition)

和聚合不同，组合中整体和部分是强依赖的，整体不存在了部分也不存在了。比如公司和部门，公司没了部门就不存在了。但是公司和员工就属于聚合关系了，因为公司没了员工还在。

关联关系 (Association)

表示不同类对象之间有关联，这是一种静态关系，与运行过程的状态无关，在最开始就可以确定。因此也可以用 1 对 1、多对 1、多对多这种关联关系来表示。比如学生和学校就是一种关联关系，一个学校可以有很多学生，但是一个学生只属于一个学校，因此这是一种多对一的关系，在运行开始之前就可以确定。

依赖关系 (Dependency)

和关联关系不同的是，依赖关系是在运行过程中起作用的。A 类和 B 类是依赖关系主要有三种形式：

A 类是 B 类方法的局部变量；

A 类是 B 类方法当中的一个参数；

A 类向 B 类发送消息，从而影响 B 类发生变化。

三、设计原则

S.O.L.I.D

简写 全拼 中文翻译

SRP The Single Responsibility Principle 单一责任原则

OCP The Open Closed Principle 开放封闭原则

LSP The Liskov Substitution Principle 里氏替换原则

ISP The Interface Segregation Principle 接口分离原则

DIP The Dependency Inversion Principle 依赖倒置原则

1. 单一责任原则

修改一个类的原因应该只有一个。

换句话说就是让一个类只负责一件事，当这个类需要做过多事情的时候，就需要分解这个类。

如果一个类承担的职责过多，就等于把这些职责耦合在了一起，一个职责的变化可能会削弱这个类完成其它职责的能力。

2. 开放封闭原则

类应该对扩展开放，对修改关闭。

扩展就是添加新功能的意思，因此该原则要求在添加新功能时不需要修改代码。

符合开闭原则最典型的设计模式是装饰者模式，它可以动态地将责任附加到对象上，而不用去修改类的代码。

3. 里氏替换原则

子类对象必须能够替换掉所有父类对象。

继承是一种 IS-A 关系，子类需要能够当成父类来使用，并且需要比父类更特殊。

如果不满足这个原则，那么各个子类的行为上就会有很大差异，增加继承体系的复杂度。

4. 稳定抽象原则

最稳定的包应该是最抽象的包，不稳定的包应该是具体的包，即包的抽象程度跟它的稳定性成正比。