编写高质量的代码

1. 面向对象
2. 什么是面向对象编程？**面向对象编程是一种编程范式或编程风格**。它以类或对象作为组织代码的基本单元，并将封装、抽象、继承、多态四个特性，作为代码设计和实现的基石 。
3. 什么是面向对象编程语言？**面向对象编程语言是支持类或对象的语法机制**，并有现成的语法机制，能方便地实现面向对象编程四大特性（封装、抽象、继承、多态）的编程语言。
4. 如何判定一个编程语言是否是面向对象编程语言？如果按照严格的的定义，需要有现成的语法支持**类、对象、四大特性**才能叫作面向对象编程语言。如果放宽要求的话，只要某种编程语言支持类、对象语法机制，那基本上就可以说这种编程语言是面向对象编程语言了，不一定非得要求具有所有的四大特性。
5. 面向对象编程和面向对象编程语言之间有何关系？面向对象编程一般使用面向对象编程语言来进行，但是，不用面向对象编程语言，我们照样可以进行面向对象编程。反过来讲，即便我们使用面向对象编程语言，写出来的代码也不一定是面向对象编程风格的，也有可能是面向过程编程风格的。
6. 什么是面向对象分析和面向对象设计？简单点讲，**面向对象分析就是要搞清楚做什么，面向对象设计就是要搞清楚怎么做**。两个阶段最终的产出是类的设计，包括程序被拆解为哪些类，每个类有哪些属性方法、类与类之间如何交互等等。
   1. 面向对象的四大特性：封装、抽象、继承、多态

**封装：**封装也叫作信息隐藏或者数据访问保护。类通过暴露有限的访问接口，授权外部仅能通过类提供的方式（或者叫函数）来访问内部信息或者数据。要实现这个功能需要依赖编程语言的访问权限控制。（Java 语言中的private、public 等关键字）。这就好比，如果一个冰箱有很多按钮，你就要研究很长时间，还不一定能操作正确。相反，如果只有几个必要的按钮，比如开、停、调节温度，你一眼就能知道该如何来操作，而且操作出错的概率也会降低很多。

**抽象：**封装主要讲的是如何隐藏信息、保护数据，而抽象讲的是如何隐藏方法的具体实现，让调用者只需要关心方法提供了哪些功能，并不需要知道这些功能是如何实现的。常用编程语言提供的接口类（比如 Java 中的 interface 关键字语法）或者抽象类（比如 Java 中的 abstract 关键字语法）这两种语法机制，来实现抽象这一特性。在面对复杂系统的时候，人脑能承受的信息复杂程度是有限的，所以我们必须忽略掉一些非关键性的实现细节。而抽象作为一种只关注功能点不关注实现的设计思路，正好帮我们的大脑过滤掉许多非必要的信息。很多设计原则都体现了抽象这种设计思想，比如基于接口而非实现编程、开闭原则（对扩展开放、对修改关闭）、代码解耦（降低代码的耦合性）等。我们在定义（或者叫命名）类的方法的时候，也要有抽象思维，不要在方法定义中，暴露太多的实现细节，以保证在某个时间点需要改变方法的实现逻辑的时候，不用去修改其定义。

**继承：**继承是用来表示类之间的 is-a 关系，比如猫是一种哺乳动物。从继承关系上来讲，继承可以分为两种模式，单继承和多继承。单继承表示一个子类只继承一个父类，多继承表示一个子类可以继承多个父类，比如猫既是哺乳动物，又是爬行动物。继承最大的一个好处就是代码复用。假如两个类有一些相同的属性和方法，我们就可以将这些相同的部分，抽取到父类中，让两个子类继承父类。我们通过继承来关联两个类，反应真实世界中的这种关系，非常符合人类的认知，而且，从设计的角度来说，也有一种结构美感。不过，过度使用继承，继承层次过深过复杂，就会导致代码可读性、可维护性变差。子类和父类高度耦合，修改父类的代码，会直接影响到子类。

**多态：**多态是指，子类可以替换父类，在实际的代码运行过程中，调用子类的方法实现。为了实现多态需要以下几种语法机制，第一个语法机制是编程语言要支持父类对象可以引用子类对象；第二个语法机制是编程语言要支持继承；第三个语法机制是编程语言要支持子类可以重写（override）父类中的方法。也就是说，只要两个类具有相同的方法，就可以实现多态，并不要求两个类之间有任何关系，这就是所谓的 duck-typing，是一些动态语言所特有的语法机制。而像 Java 这样的静态语言，通过继承实现多态特性，必须要求两个类之间有继承关系，通过接口实现多态特性，类必须实现对应的接口。多态特性能提高代码的可扩展性和复用性。多态也是很多设计模式、设计原则、编程技巧的代码实现基础，比如策略模式、基于接口而非实现编程、依赖倒置原则、里式替换原则、利用多态去掉冗长的 if-else 语句等等。

## 封装

What：隐藏信息，保护数据访问。

How：暴露有限接口和属性，需要编程语言提供访问控制的语法。

Why：提高代码可维护性；降低接口复杂度，提高类的易用性。

##抽象

What: 隐藏具体实现，使用者只需关心功能，无需关心实现。

How: 通过接口类或者抽象类实现，特殊语法机制非必须。

Why: 提高代码的扩展性、维护性；降低复杂度，减少细节负担。

##继承

What: 表示 is-a 关系，分为单继承和多继承。

How: 需要编程语言提供特殊语法机制。例如 Java 的 “extends”，C++ 的 “:” 。

Why: 解决代码复用问题。

##多态

What: 子类替换父类，在运行时调用子类的实现。

How: 需要编程语言提供特殊的语法机制。比如继承、接口类、duck-typing。

Why: 提高代码扩展性和复用性。

3W 模型的关键在于 Why，没有 Why，其它两个就没有存在的意义。从四大特性可以看出，面向对象的终极目的只有一个：可维护性。易扩展、易复用，降低复杂度等等都属于可维护性的实现方式。

* 1. 面向对象编程与面向过程编程的区别和联系
  2. 面向对象分析、面向对象设计、面向对象编程
  3. 接口和抽象类的区别以及各自的应用场景
  4. 基于接口而非实现编程的设计思想
  5. 多用组合少用继承的设计思想
  6. 面向过程的贫血模型和面向对象的充血模型

1. 设计原则

2.1 SOLID 原则

* SRP 单一职责原则
* OCP 开闭原则
* LSP 里式替换原则
* ISP 接口隔离原则
* DIP 依赖倒置原则

2.2 DRY 原则、KISS 原则、YAGNI 原则、LOD 法则

1. 设计模式

3.1 创建型

常用的有：单例模式、工厂模式（工厂方法和抽象工厂）、建造者模式。

不常用的有：原型模式。

3.2 结构型

常用的有：代理模式、桥接模式、装饰者模式、适配器模式。

不常用的有：门面模式、组合模式、享元模式。

3.3 行为型

常用的有：观察者模式、模板模式、策略模式、职责链模式、迭代器模式、状态模式。

不常用的有：访问者模式、备忘录模式、命令模式、解释器模式、中介模式。

1. 编程规范

编程规范主要解决的是代码的可读性问题。编码规范相对于设计原则、设计模式，更加具体、更加偏重代码细节。即便你可能对设计原则不熟悉、对设计模式不了解，但你最起码要掌握基本的编码规范，比如，如何给变量、类、函数命名，如何写代码注释，函数不宜过长、参数不能过多等等。对于编码规范，考虑到很多书籍已经讲得很好了（比如《重构》《代码大全》《代码整洁之道》等）。而且，每条编码规范都非常简单、非常明确，比较偏向于记忆，你只要照着来做可以。它不像设计原则，需要融入很多个人的理解和思考。所以，在这个专栏中，我并没有花太多的篇幅来讲解所有的编码规范，而是总结了我认为的最能改善代码质量的 20 条规范。如果你暂时没有时间去看那些经典的书籍，看我这些就够了。除此之外，专栏并没有将编码规范单独作为一个模块来讲解，而是跟重构放到了一起。之所以这样做，那是因为我把重构分为大重构和小重构两种类型，而小重构利用的知识基本上就是编码规范。除了编码规范，我们还会介绍一些代码的坏味道，让你知道什么样的代码是不符合规范的，应该如何优化。参照编码规范，你可以写出可读性好的代码；参照代码的坏味道，你可以找出代码存在的可读性问题。

1. 代码重构

在软件开发中，只要软件在不停地迭代，就没有一劳永逸的设计。随着需求的变化，代码的不停堆砌，原有的设计必定会存在这样那样的问题。针对这些问题，我们就需要进行代码重构。重构是软件开发中非常重要的一个环节。持续重构是保持代码质量不下降的有效手段，能有效避免代码腐化到无可救药的地步。而重构的工具就是我们前面罗列的那些面向对象设计思想、设计原则、设计模式、编码规范。实际上，设计思想、设计原则、设计模式一个最重要的应用场景就是在重构的时候。我们前面讲过，虽然使用设计模式可以提高代码的可扩展性，但过度不恰当地使用，也会增加代码的复杂度，影响代码的可读性。在开发初期，除非特别必须，我们一定不要过度设计，应用复杂的设计模式。而是当代码出现问题的时候，我们再针对问题，应用原则和模式进行重构。这样就能有效避免前期的过度设计。对于重构这部分内容，你需要掌握以下几个知识点：重构的目的（why）、对象（what）、时机（when）、方法（how）；保证重构不出错的技术手段：单元测试和代码的可测试性；两种不同规模的重构：大重构（大规模高层次）和小重构（小规模低层次）。希望你学完这部分内容之后，不仅仅是掌握一些重构技巧、套路，更重要的是建立持续重构意识，把重构当作开发的一部分，融入到日常的开发中。

1. 五者之间的联系

面向对象编程因为其具有丰富的特性（封装、抽象、继承、多态），可以实现很多复杂的设计思路，是很多设计原则、设计模式等编码实现的基础。设计原则是指导我们代码设计的一些经验总结，对于某些场景下，是否应该应用某种设计模式，具有指导意义。比如，“开闭原则”是很多设计模式（策略、模板等）的指导原则。设计模式是针对软件开发中经常遇到的一些设计问题，总结出来的一套解决方案或者设计思路。应用设计模式的主要目的是提高代码的可扩展性。从抽象程度上来讲，设计原则比设计模式更抽象。设计模式更加具体、更加可执行。编程规范主要解决的是代码的可读性问题。编码规范相对于设计原则、设计模式，更加具体、更加偏重代码细节、更加能落地。持续的小重构依赖的理论基础主要就是编程规范。重构作为保持代码质量不下降的有效手段，利用的就是面向对象、设计原则、设计模式、编码规范这些理论。实际上，面向对象、设计原则、设计模式、编程规范、代码重构，这五者都是保持或者提高代码质量的方法论，本质上都是服务于编写高质量代码这一件事的。当我们追本逐源，看清这个本质之后，很多事情怎么做就清楚了，很多选择怎么选也清楚了。比如，在某个场景下，该不该用这个设计模式，那就看能不能提高代码的可扩展性；要不要重构，那就看重代码是否存在可读、可维护问题等。

