第一篇：

设计问题没有没有标准答案，不要轻信一家之言，主动思考，积极讨论。

第二篇：

设计模式对于复杂功能系统的设计开发具有重要作用；让读源码、学框架事半功倍；为你的职场发展做铺垫

第三篇：

好代码：

1. 可维护性：
2. 可读性：符合编码规范，命名达意，注释详尽，函数长短合适、模块划分清晰、高内聚低耦合
3. 可扩展性：
4. 灵活性
5. 简洁性：思从深而行从简，真正的高手能云淡风轻的用最简单的方法解决最复杂的问题
6. 可复用性
7. 可测试性：能否比较容易写出单元测试

第四篇：

抽象类：不能实例化，只能继承；抽象类可以包含属性和方法，不包含代码实现的方法叫做抽象方法；子类继承抽象类，必须实现抽象类中的所有抽象方法。

Java中的接口：接口不能包含属性；接口只能声明方法，方法不能包含代码实现；类实现接口的时候，必须实现接口中声明的所有方法。

他们之间的区别：抽象类是一种特殊的类，这种类不能被实例化对象，只能被子类继承。继承关系是一种is-a的关系；而接口表示一种has-a的关系，表示具有某些功能，对于接口有一个更加形象的叫法，那就是协议。

第五篇：

基于接口而非实现编程，是一种非常高效的提高代码质量的手段。

原因： 可以将接口和实现分离，封装不稳定的实现，暴露稳定的接口。越抽象、越顶层、越脱离具体某一实现的设计，越能提高代码的灵活性，越能应对未来的需求变化。好的代码设计不仅能应对当下的需求，而且在将来需求发生变化的时候，仍然能够在不破坏原有代码设计的情况下灵活应对。而抽象就是提高代码扩展性、灵活性、可维护性最有效的手段之一。

（音频数据管理中的接口其实可以改成 接口类的实现，这样就会更加灵活）

第六篇：

组合优于继承。

继承层次过深、过复杂会影响到代码的可维护性。

有一些特殊的场景必须使用继承，如果你不能改变一个函数的入参类型，而入参又非接口，为了支持多态，只能采用继承实现，重写这个类中的方法。

结合具体场景使用。

第七篇：

贫血模式 和 充血模式

什么是贫血模式？什么是充血模式？

贫血模式: 业务与数据不在一个类中即业务与数据相分离，破坏了面向对象的封装特性，是一种典型的面向过程的编程风格。

充血模型（Rich Domain Model）：正好相反，数据和对应的业务逻辑被封装到同一个类中。因此满足面向对象的封装特性。

领域驱动设计即DDD,主要用来指导如何解耦业务系统，划分业务模块，定义业务领域模型及其交互。属于充血模型

为什么说贫血模式的传统开发模式违反OOP?

基于贫血模式的传统开发模式既然违反OOP，那又为什么如此流行？

什么情况下我们应该考虑使用基于充血模型的DDD开发模式？ 复杂系统，业务划分建模

第八篇：

面向对象设计：

1. 划分职责进而识别出有那些类
2. 定义类属性和方法
3. 定义类与类之间的交互关系
4. 将类组装起来并提供执行入口