Java中消除实现继承和面向接口编程 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/278/2021_2022_Java_E4_B8_AD_E6_B6_88_c97_278615.htm 继承是面向对象中很重要的概念。如果考虑到Java语言特性，继承分为两种：接口继承和实现继承。这只是技术层面的问题，即便C 中不存在接口的概念，但它的虚基类实际上也相当于接口。对于oo的初学者来说，他们很希望自己的程序中出现大量的继承，因为这样看起来很OO。但滥用继承会带来很多问题，尽管有时候我们又不得不使用继承解决问题。 相比于接口继承，实现继承的问题要更多，它会带来更多的耦合问题。但接口继承也是有问题的，这是继承本身的问题。实现继承的很多问题出于其自身实现上，因此这里重点讨论实现继承的问题。 举个例子(这个例子实在太老套了)。我要实现一个Stack类，我想当然地选择Stack类继承于ArrayList类(你也可以认为我很想OO些或者出于本性的懒惰).现在又有了新的需求，需要实现一个线程安全的Stack，我又定义了一个ConcurrentStack类继承于Stack并覆盖了Stack中的部分代码。 因为Stack继承于ArrayList，Stack不得不对外暴露出ArrayList所有的public方法，即便其中的某些方法对它可能是不需要的.甚至更糟的是，可能其中的某些方法能改变Stack的状态，而Stack对这些改变并不知情，这就会造成Stack的逻辑错误。 如果我要在ArrayList中添加新的方法，这个方法就有可能在逻辑上破坏它的派生类Stack、ConcurrentStack。因此在基类(父类)添加方法(修改代码)时，必须检查这些修改是否会对派生类产生影响.如果产生影响的话，就不得不对派生类做进一步的修改。如果类的继承体系

不是一个人完成的，或者是修改别人的代码的情况下，很可能因为继承产生难以觉察的BUG。 问题还是有的。我们有时会见到这样的基类，它的一些方法只是抛出异常，这意味着如果派生类支持这个方法就重写它，否则就如父类一样抛出异常表明其不支持这个方法的调用。我们也能见到它的一个变种，父类的方法是抽象的，但不是所有的子类都支持这个方法，不支持的方法就以抛出异常的方式表明立场。这种做法是很不友好和很不安全的，它们只能在运行时被“ 侥幸捕捉 ”，而很多漏网的异常方法可能会在某一天突然出现，让人不知所措。 引起上面问题的很重要的原因便是基类和派生类之间的耦合。往往只是对基类做了小小的改动，却不得不重构它们的所有的派生类，这就是臭名昭著的“ 脆弱的基类 ”问题。由于类之间的关系是存在的，因此耦合是不可避免的甚至是必要的。但在做OO设计时，当遇到如基类和派生类之间的强耦合关系，我们就要思量思量，是否一定需要继承呢?是否会有其他的更优雅的替代方案呢?如果一定要学究的话，你会在很多书中会看到这样的原则：如果两个类之间是IS-A关系，那么就使用继承.如果两个类之间是Has-A的关系，那么就使用委派。很多时候这条原则是适用的，但IS-A并不能做为使用继承的绝对理由。有时为了消除耦合带来的问题，使用委派等方法会更好地封装实现细节。继承有时会对外及向下暴露太多的信息，在GOF的设计模式中，有很多模式的目的就是为了消除继承。 关于何时采用继承，一个重要的原则是确定方法是否能够共享。比如DAO ，可以将通用的CRUD 方法定在一个抽象DAO 中，具体的DAO 都派生自这个抽象类。严格的说，抽象DAO 和派生的DAO 实现并不

具有IS -A 关系，我们只是为了避免重复的方法定义和实现而作出了这一技术上的选择。可以说，使用接口还是抽象类的原则是，如果多个派生类的方法内容没有共同的地方，就用接口作为抽象.如果 多个派生类 的方法含有共同的地方，就用抽象类作为抽象。当这一原则不适用于接口继承，如果出现接口继承，就会相应地有实现继承(基类更多的是抽象类)。 现在说说面向接口编程。在众多的敏捷方法中，面向接口编程总是被大师们反复的强调。面向接口编程，实际上是面向抽象编程，将抽象概念和具体实现相隔离。这一原则使得我们拥有了更高层次的抽象模型，在面对不断变更的需求时，只要抽象模型做的好，修改代码就要容易的多。但面向接口编程不意味着非得一个接口对应一个类，过多的不必要的接口也可能带来更多的工作量和维护上的困难。 相比于继承，OO中多态的概念要更重要。一个接口可以对应多个实现类，对于声明为接口类型的方法参数、类的字段，它们要比实现类更易于扩展、稳定，这也是多态的优点。假如我以实现类作为方法参数定义了一个方法void doSomething(ArrayList list)，但如果领导哪天觉得 ArrayList不如LinkedList更好用，我将不得不将方法重构为void doSomething(LinkedList list)，相应地要在所有调用此方法的地方修改参数类型(很遗憾地，我连对象创建也是采用ArrayList list = new ArrayList()方式，这将大大增加我的修改工作量)。如果领导又觉得用list存储数据不如set好的话，我将再一次重构方法，但这一次我变聪明了，我将方法定义为void doSomething(Set set)，创建对象的方式改为Set set = new HashSet()。但这样仍不够，如果领导又要求将set改回list怎么办?所以我应该将方法重构为void

doSomething(Collection collection)， Collection的抽象程度最高，更易于替换具体的实现类。即便需要List或者Set固有的特性，我也可以做向下类型转换解决问题，尽管这样做并不安全。 面向接口编程最重要的价值在于隐藏实现，将抽象的实现细节封装起来而不对外开放，封装这对于Java EE 中的分层设计和框架设计尤其重要。但即便在编程时使用了接口，我们也需要将接口和实现对应起来，这就引出如何创建对象的问题。在创建型设计模式中，单例、工厂方法(模板方法)、抽象工厂等模式都是很好的解决办法。现在流行的控制反转(也叫依赖注入)模式是以声明的方式将抽象与实现连接起来，这既减少了单调的工厂类也更易于单元测试。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com