Java设计模式之模板方法模式Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022_Java_E8_AE_BE_E8_AE_A1_c104_644658.htm 一、引子 这是一个很简单的模式，却被非常广泛的使用。之所以简单是因为在这个模式中仅仅使用到了继承关系。 继承关系由于自身的缺陷，被专家们扣上了“ 罪恶 ”的帽子。“ 使用委派关系代替继承关系 ”，“ 尽量使用接口实现而不是抽象类继承 ”等等专家警告，让我们这些菜鸟对继承“ 另眼相看 ”。 其实，继承还是有很多自身的优点所在。只是被大家滥用的似乎缺点更加明显了。合理的利用继承关系，还是能对你的系统设计起到很好的作用的。而模板方法模式就是其中的一个使用范例。 二、定义与结构 GOF给模板方法（Template Method）模式定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。这里的算法的结构，可以理解为你根据需求设计出来的业务流程。特定的步骤就是指那些可能在内容上存在变数的环节。 可以看出来，模板方法模式也是为了巧妙解决变化对系统带来的影响而设计的。使用模板方法使系统扩展性增强，最小化了变化对系统的影响。这一点，在下面的举例中可以很明显的看出来。 来看下这个简单模式的结构吧： 1) AbstractClass（抽象类）：定义了一到多个的抽象方法，以供具体的子类来实现它们；而且还要实现一个模板方法，来定义一个算法的骨架。该模板方法不仅调用前面的抽象方法，也可以调用其他的操作，只要能完成自身的使命。 2) ConcreteClass（具体类）：实现父类中的抽象方法以完成算法

中与特定子类相关的步骤。 下面是模板方法模式的结构图。直接把《设计模式》上的图拿过来用下： 三、举例 还是在我刚刚分析完源码的JUnit中找个例子吧。JUnit中的TestCase以及它的子类就是一个模板方法模式的例子。在TestCase这个抽象类中将整个测试的流程设置好了，比如先执行Setup方法初始化测试前提，在运行测试方法，然后再TearDown来取消测试设置。但是你将在 Setup、TearDown里面作些什么呢？鬼才知道呢！！因此，而这些步骤的具体实现都延迟到子类中去，也就是你实现的测试类中。 来看下相关的源代码吧。 这是TestCase中，执行测试的模板方法。你可以看到，里面正像前面定义中所说的那样，它制定了“ 算法 ”的框架先执行setUp方法来做下初始化，然后执行测试方法，最后执行tearDown释放你得到的资源。public void runBare() throws Throwable { setUp(). try { runTest(). } finally { tearDown(). }} 这就是上面使用的两个方法。与定义中不同的是，这两个方法并没有被实现为抽象方法，而是两个空的无为方法（被称为钩子方法）。这是因为在测试中，我们并不是必须要让测试程序使用这两个方法来初始化和释放资源的。如果是抽象方法，则子类们必须给它一个实现，不管用到用不到。这显然是不合理的。使用钩子方法，则你在需要的时候，可以在子类中重写这些方法。protected void setUp() throws Exception {}protected void tearDown() throws Exception {} 四、适用情况 根据上面对定义的分析，以及例子的说明，可以看出模板方法适用于以下情况： 1) 一次性实现一个算法的不变的部分，并将可变的行为留给子类来实现。 2) 各子类中公共的行为应被提取出来并集中到一个公共父类中以避免代码重复。其实这

可以说是一种好的编码习惯了。 3) 控制子类扩展。模板方法只在特定点调用操作，这样就只允许在这些点进行扩展。比如上面runBare（）方法就只在runTest前面适用setUp方法。如果你不愿子类来修改你的模板方法定义的框架，你可以采用两种方式来做：一是在API中不体现出你的模板方法；二、将你的模板方法置为final就可以了。 可以看出，使用模板方法模式可以将代码的公共行为提取出来，达到复用的目的。而且，在模板方法模式中，是由父类的模板方法来控制子类中的具体实现。这样你在实现子类的时候，根本不需要对业务流程有太多的了解。 