1：java什么时候用重载，什么时候用重写？

**TOM：**

大家都知道重载是多态的一种实现，一种形式，在Java中的重载其实我们可以这样去思考：

主要解决还是从代码应用层即用户调用者这个角度来解决问题，也是遵循开闭原则，首先，举个例子，在Spring中有个getBean方法，getBean方法名字相同，大家都知道名字相同意味着这个方法的功能是相同的，但是在getBean里面有很多场景，我们可以传Bean的name，也可以传Bean的Class，等等非常多的情况，那么为什么会有这么多的情况呢，是因为用户在不同的应用场景使用的时候，可能会根据不同的需求来使用，传的参数不一定直接拿到，所以这时候用户在调用方法的时候，如果拿不到就没法调用，所以我们在顶层设计的时候，可以开发多种应用场景供用户去选择，

那么最底层我们通过去看Spring源码也知道最底层的实现最终都会归根到一个方法，也就是说在类的内部，重载方法的最终逻辑是一样的，外部只是一种形式。

什么时候用重写，重写也是遵循开闭原则，发生在父子类之间，像抽象类与子类之间，抽象类通常用在模板模式中 ，主要是用于扩展，像包装器模式，父类可能已经实现了已有的功能，但是子类对父类已有的功能不满意，或者想要扩展，那么就可以使用重写，重写也是开闭原则的一种体现，即在不修改原来的代码之上，再去增加功能，或者是覆盖原来的功能，这样的话就不会给原来的类带来新的问题，新开发的类只需要单独去维护，修改，测试，就OK了。

2：举例一个更倾向于使用抽象类，而不是使用接口的设计场景；

**TOM：**

接口和抽象类其实他们的外层差不多，都是定义了一个执行的规范，就是固定方法名，固定参数，但是抽象类和接口的根本区别在于抽象类可以增加实现，而接口不能增加实现，为什么抽象类能增加实现呢，因为抽象类可以去完成一些公共的功能，而接口没有办法去实现，所以接口通常只能定义标准，定义返回值的标准，定义策略模式的顶层规范，而我们的抽象类不仅可以作为策略模式的顶层规范，还可以作为模板模式的公共实现，这才是我们真正的底层设计逻辑。