##### 开闭原则总结:

如果原有的类需要加新需求，我们尽量再创建一个类继承要修改的类，让要修改的类的子类去完成新需求，这样可以保证原有的功能模块不发生变化。

##### 单一职责原则总结:

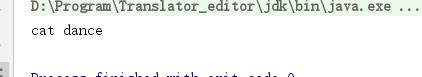
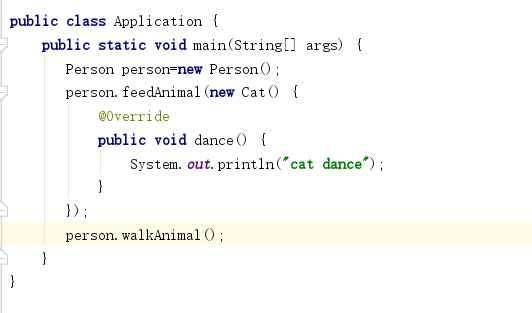
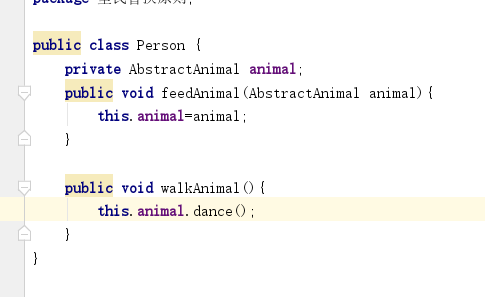
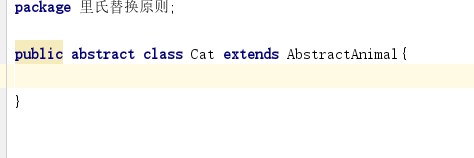
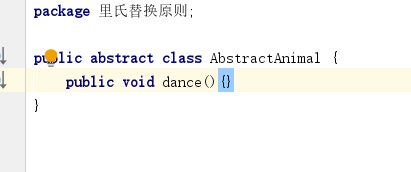
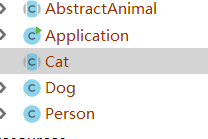
保证设计的类、接口、方法所做到的功能尽量单一。比如我们要修改用户的用户名、密码、年龄、家庭地址等等，如果设计成 updateUserMsg(String username,String password,int age,String address) 一个接口同时完成修改用户的若干个信息，如果换成其他业务，业务不会修改用户这么多信息，那么这个接口就变得不再通用。反之，我们把这个接口拆成

updateUsername、updatePassword、updateAge、updateAddress等，即把一个同时做很多功能的接口拆分，将其工作的粒度细化，那么我们在其他业务逻辑中就可以很好的复用这些接口。

**里氏替换原则：**

通俗讲，只要父类能出现的地方子类也可以出现，并且使用子类替换掉父类的话，不会产生任何异常或错误，使用者可能根本不知道是父类还是子类。反过来就不行，有子类的地方一定不能用父类替换。例如一个方法接受Map类型的参数，那么它一定可以接受HashMap、LinkedHashMap等参数，反过来的话是不行的。

例如：



**里氏替换原则总结：**

1. 里氏替换可以提高代码复用性，子类继承父类也继承到了父类的属性和方法。
2. 增加了代码的耦合性，因为子类必然有父类的方法和属性，如果父类的属性或者方法变更，需要考虑对其子类的影响。

**依赖倒置原则：**

定义：程序要依赖于抽象接口，不要依赖于具体实现。简单的说就是要求对抽象编程，不要对现实进行编程，这样就降低了客户与实现模块间的耦合。

依赖倒置原则要求我们在代码中传递参数时或在关联关系中，尽量引用高层次的抽象类，即使用接口和抽象类进行变量类型声明、参数类型声明、方法返回类型声明，以及数据类型的转换等，而不要用具体类来做这些事情。

其核心思想是:要面向接口编程，不要面向实现编程。

例如:

