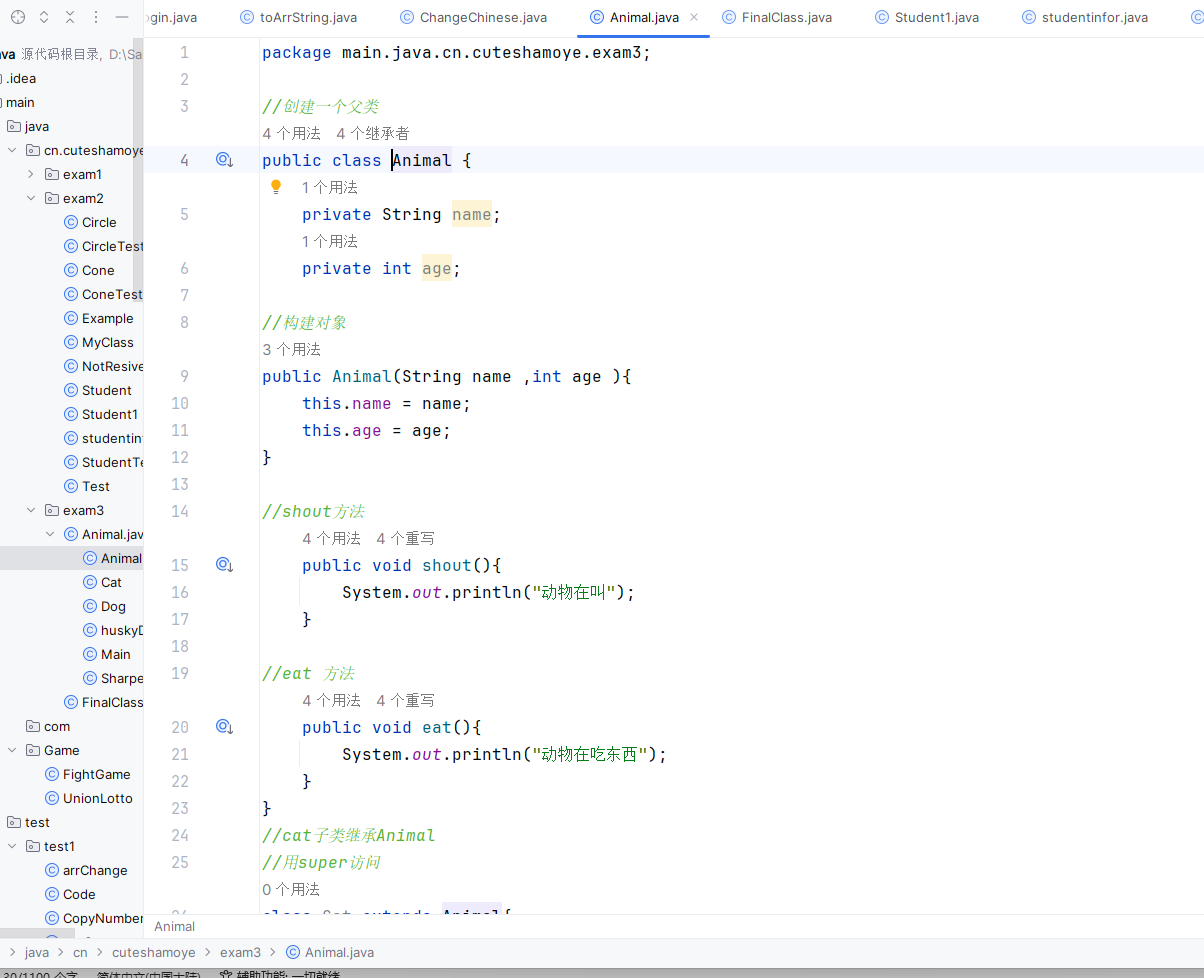
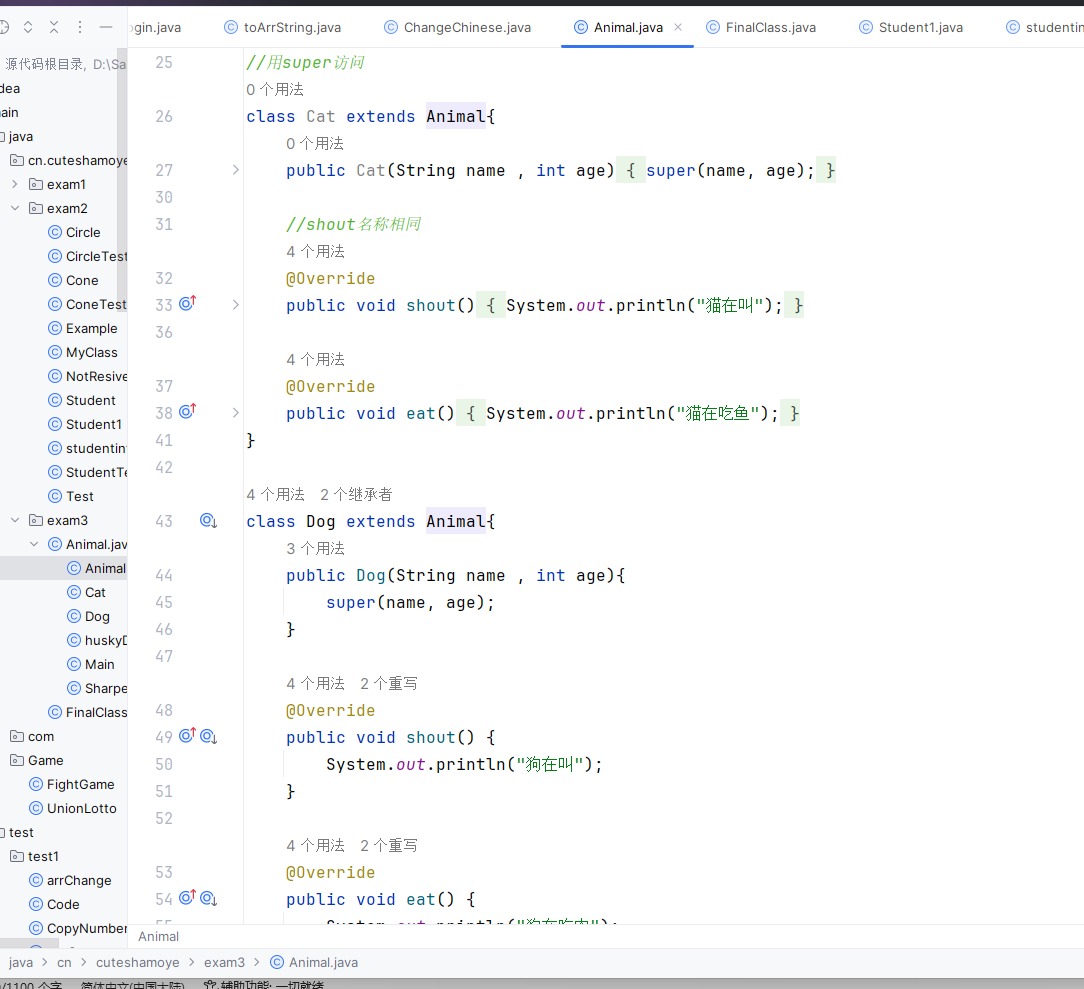
4.1-4.2

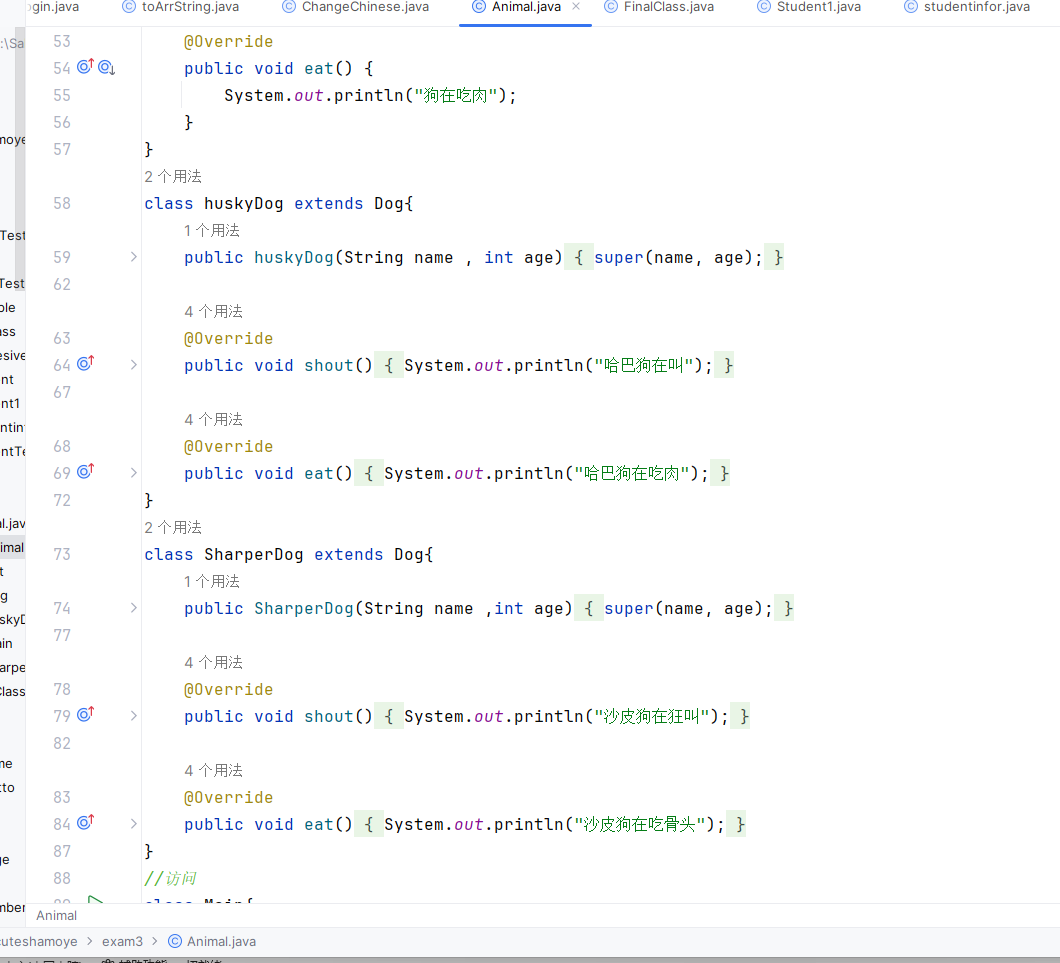
1. **参照书本，写一个Animal父类，有name，age属性，shout，eat方法。分别设计子类Cat和Dog继承Animal，再写哈巴狗类和沙皮狗类继承Dog类，重写相应的方法。给Cat和Dog添加构造方法，使用super()访问父类构造方法。**

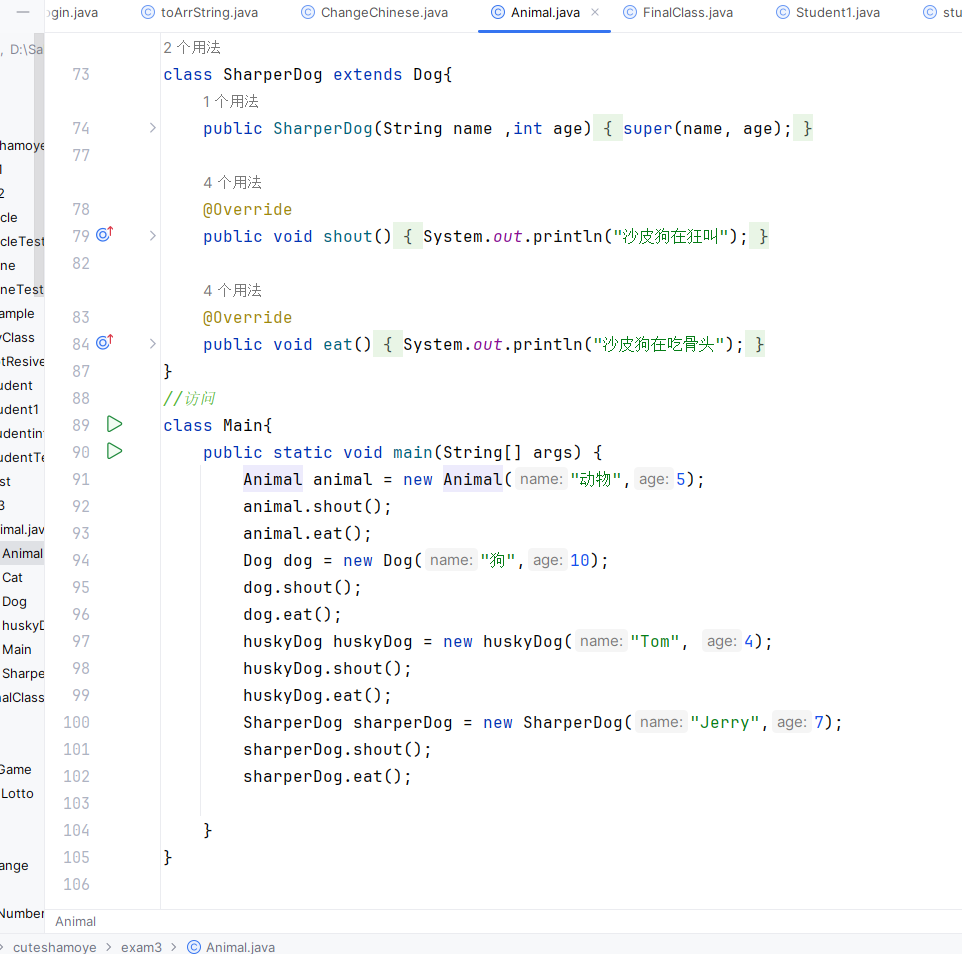
只能有一个类是public的，并且该类的名称必须与文件名相同

内部类（Inner Class）的方式在同一个文件中定义多个public类



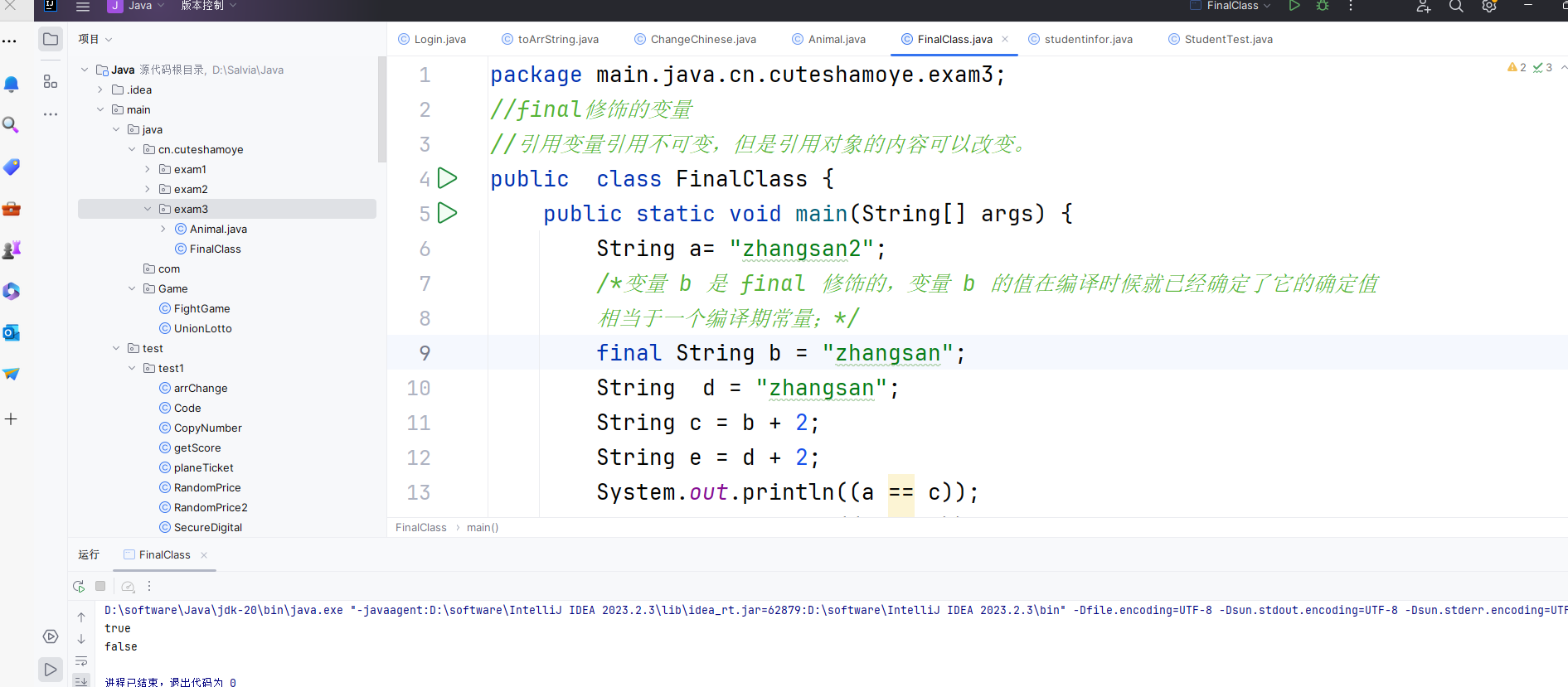






**2,用代码举例说明final关键字的三种用途。**

final作为Java中的关键字可以用于三个地方。用于修饰类、类属性和类方法。

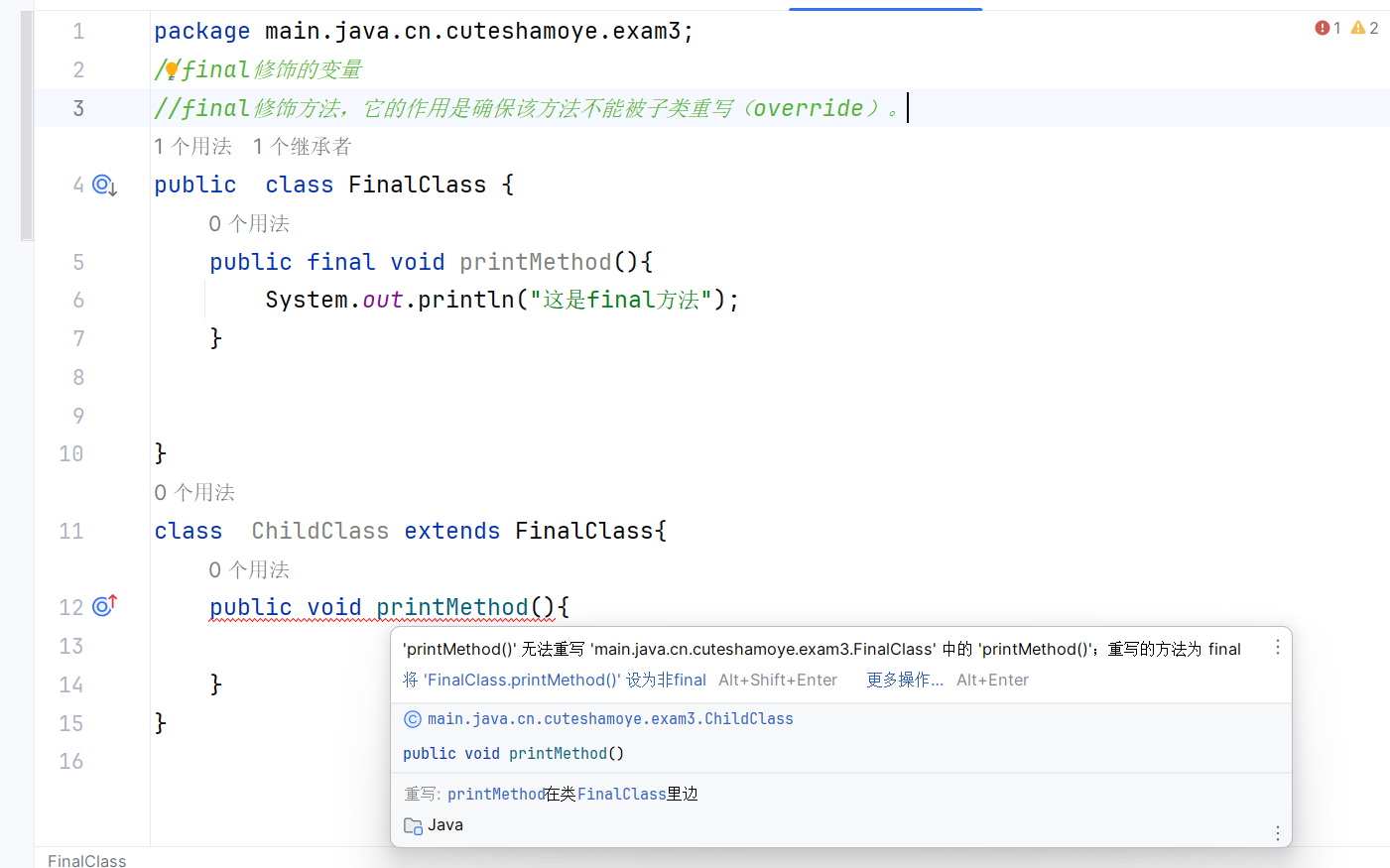


特征：凡是引用final关键字的地方皆不可修改！

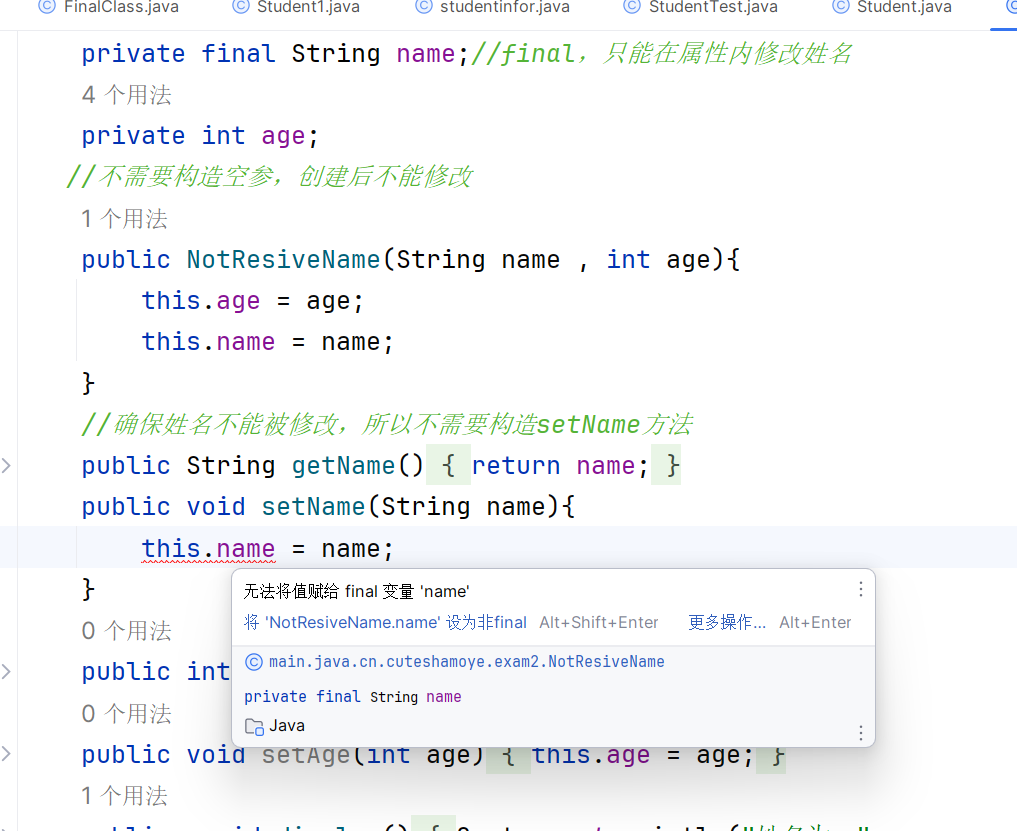
1,final修饰的类，不能被继承（String、StringBuilder、StringBuffer、Math，不可变类），



2，其中所有的方法都不能被重写(这里需要注意的是不能被重写，但是可以被重载)



3,final修饰的变量，不能改变



Final的好处：安全性：使用final关键字可以确保变量、方法或类的不可变性，防止其被修改或重写。这可以提高代码的安全性，避免意外的修改或重写导致的错误。

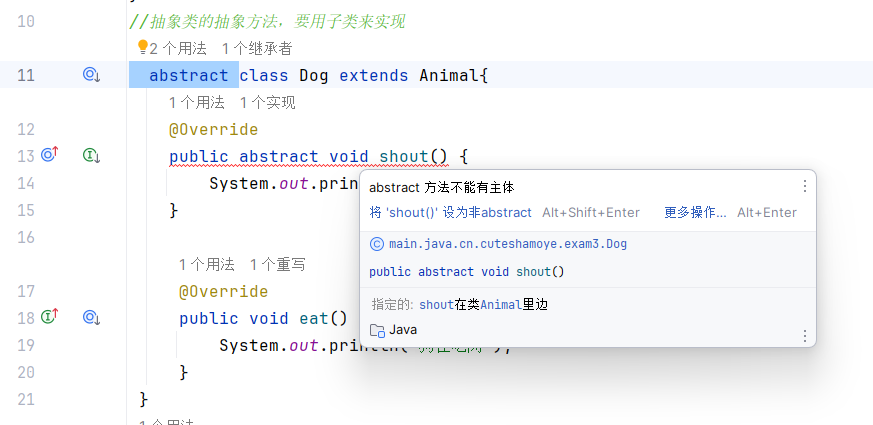
**4.3-4.4**

**1、把上面的Animal变成抽象类，把shout和eat设置成抽象方法，并在子类重写。**

如果子类选择保留抽象方法，那么该子类也必须被声明为抽象类



抽象类方法不能有主体

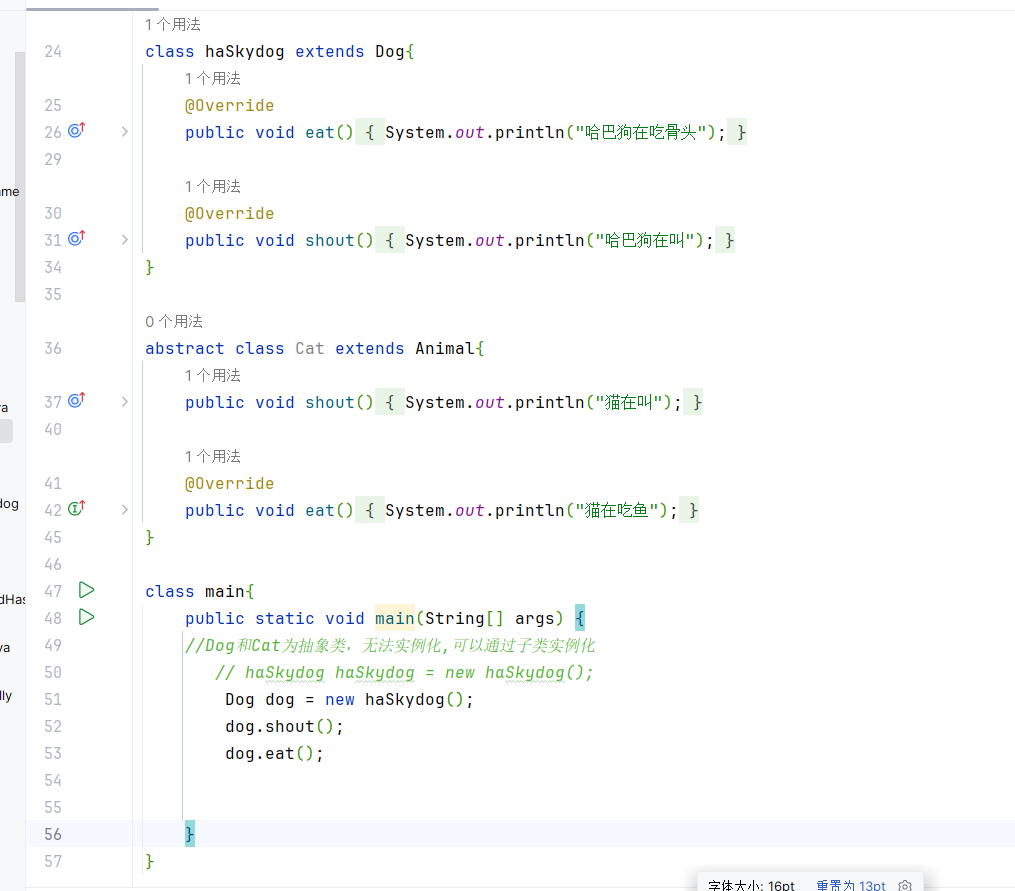


非抽象类不能存在抽象方法



以下为第一题完整代码：



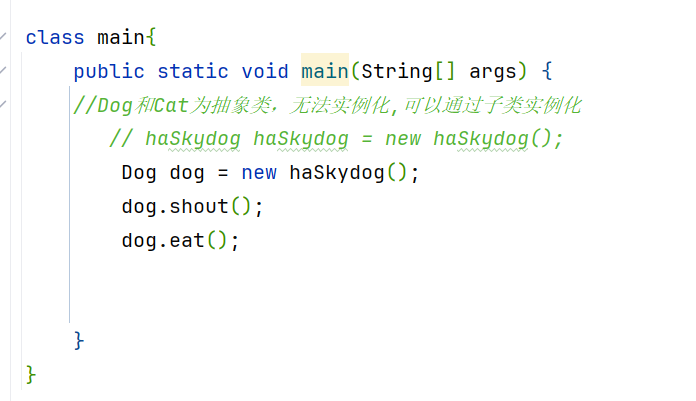


**2、写代码验证以下问题：**

**1）抽象类是否可以实例化（创建出对象）？**

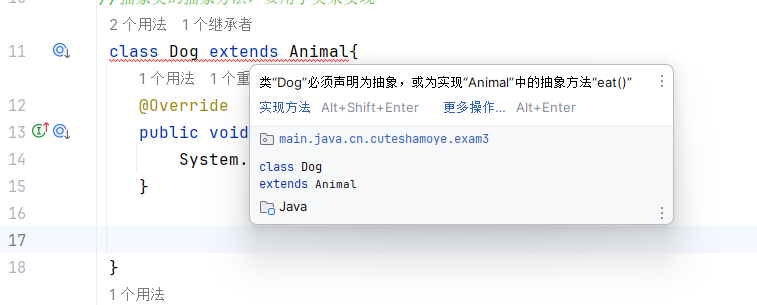
抽象类无法直接实例化，但可以通过具体子类来实例化





**2）一个类继承了抽象类，那么它是否一定要重写抽象方法？**

如果一个类继承了抽象类，并且没有被声明为抽象类，那么它必须重写抽象类中的所有抽象方法

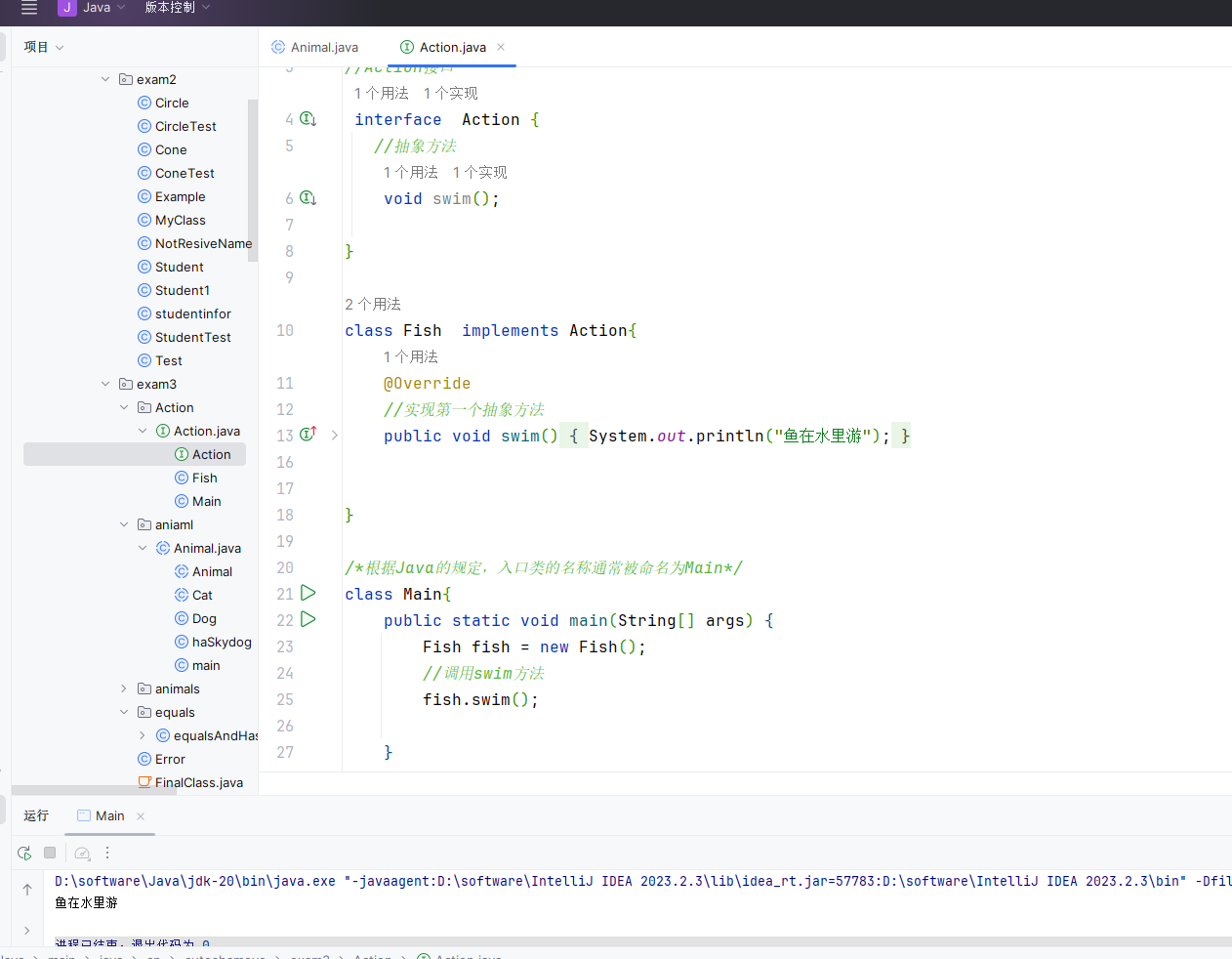


**3）一个抽象类是否可以没有抽象方法？**

一个抽象类可以没有抽象方法。抽象类可以包含普通的具体方法、成员变量、构造方法和其他非抽象成员。



1. **设计一个Action接口，里面有swim()方法。设计一个Fish类继承Animal类，并实现了Action接口，写main方法，在main方法创建Fish对象，并调用swim方法。**

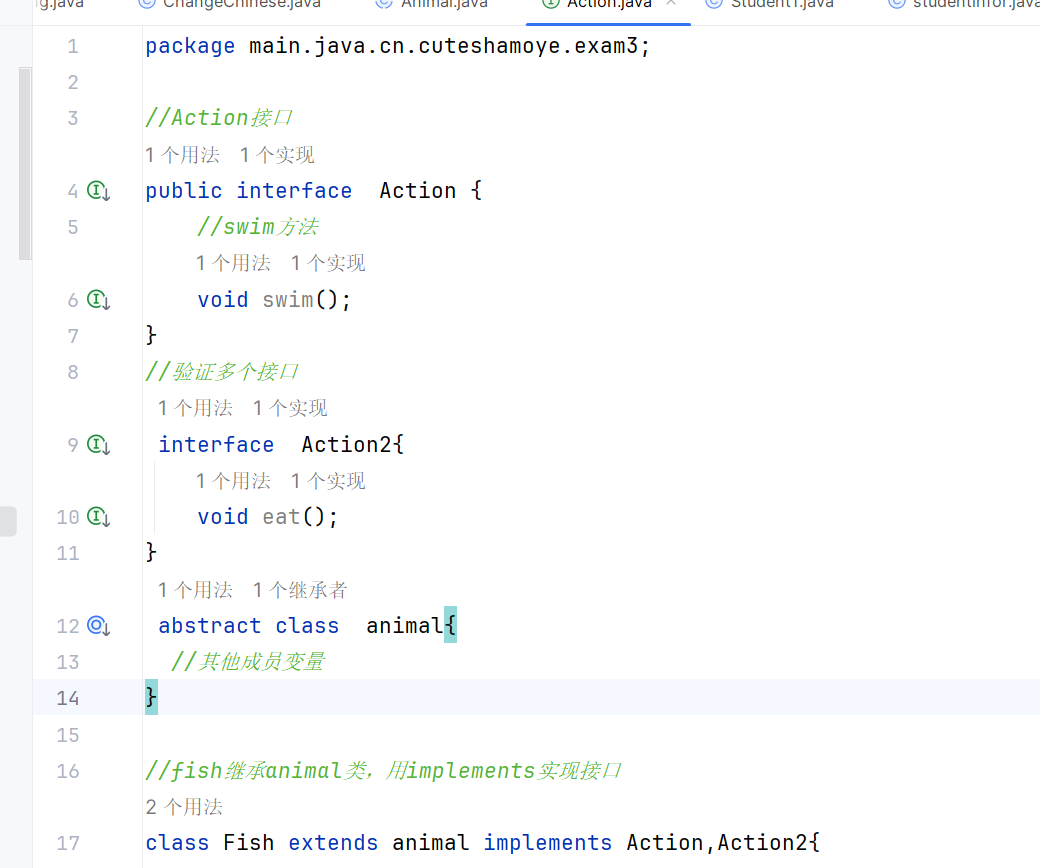


**4写代码验证以下问题：**

接口的作用：

1. 实现多继承：接口允许一个类实现多个接口，从而实现多继承。这使得类能够从多个不同的接口中继承行为，提供了更大的灵活性。
2. 接口可以继承其他接口，从而形成接口的继承关系。这使得接口能够组织和扩展行为，提供了更好的代码组织和复用。
3. 接口是一种完全抽象的类，它只包含抽象方法和常量
4. **一个类是否可以同时实现多个接口？**

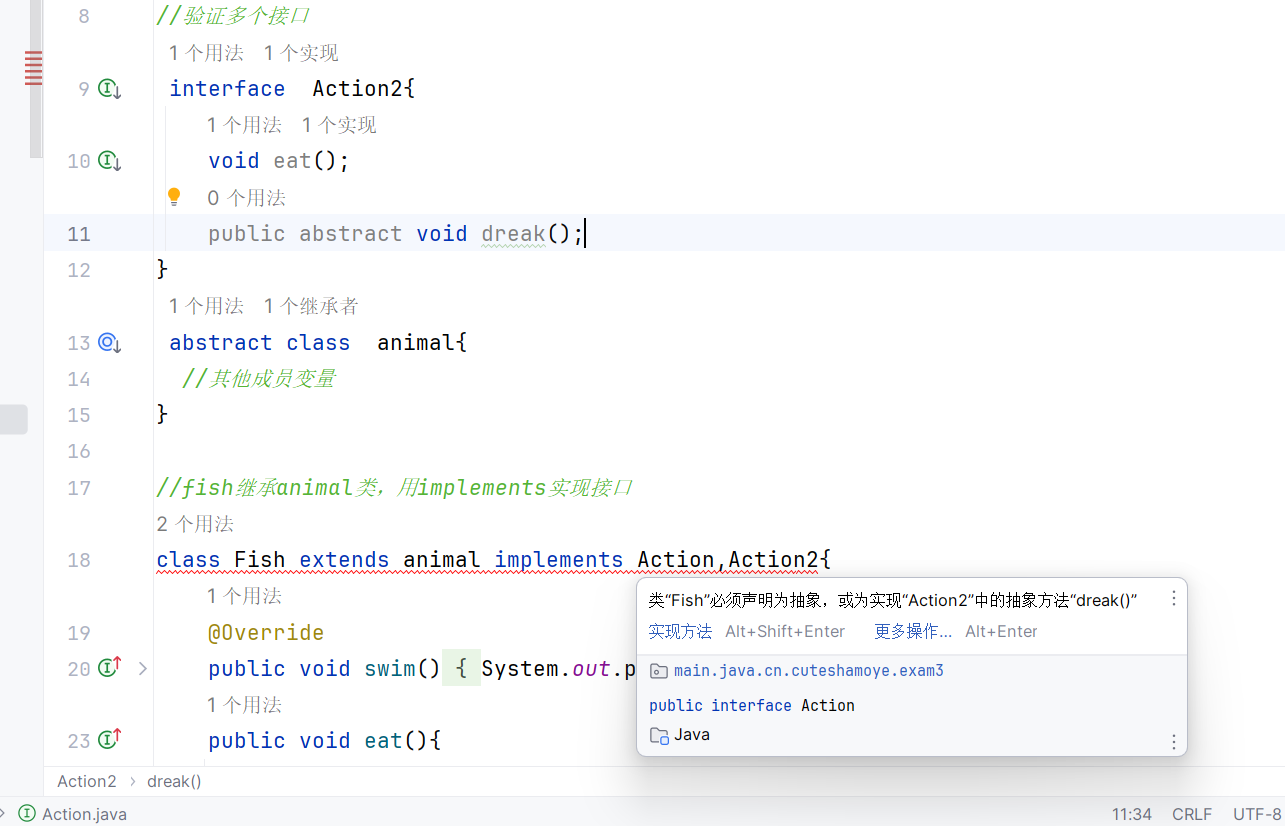
可以

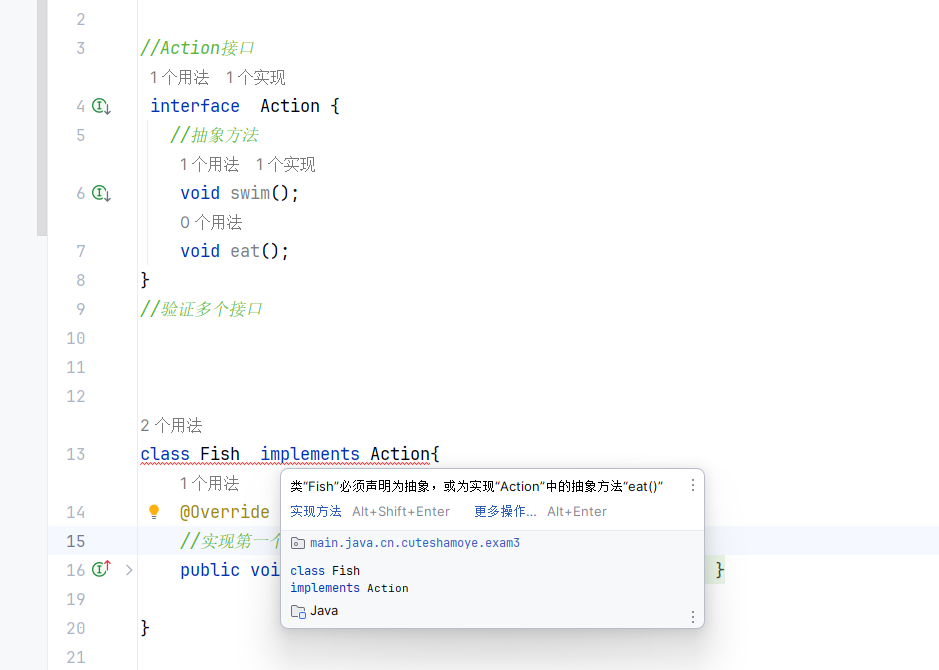




2）**实现接口时是否一定要实现接口里每个抽象方法？**

如果一个类实现了一个接口，那么它必须提供接口中所有抽象方法的实现，除非该类本身是抽象类。





5、**上面继承关系Animal，Dog，Cat，Fish的例子，写代码实现多态效果例子：当调用以下方法：**

public static void showEat(Animal animal){//实现多态效果

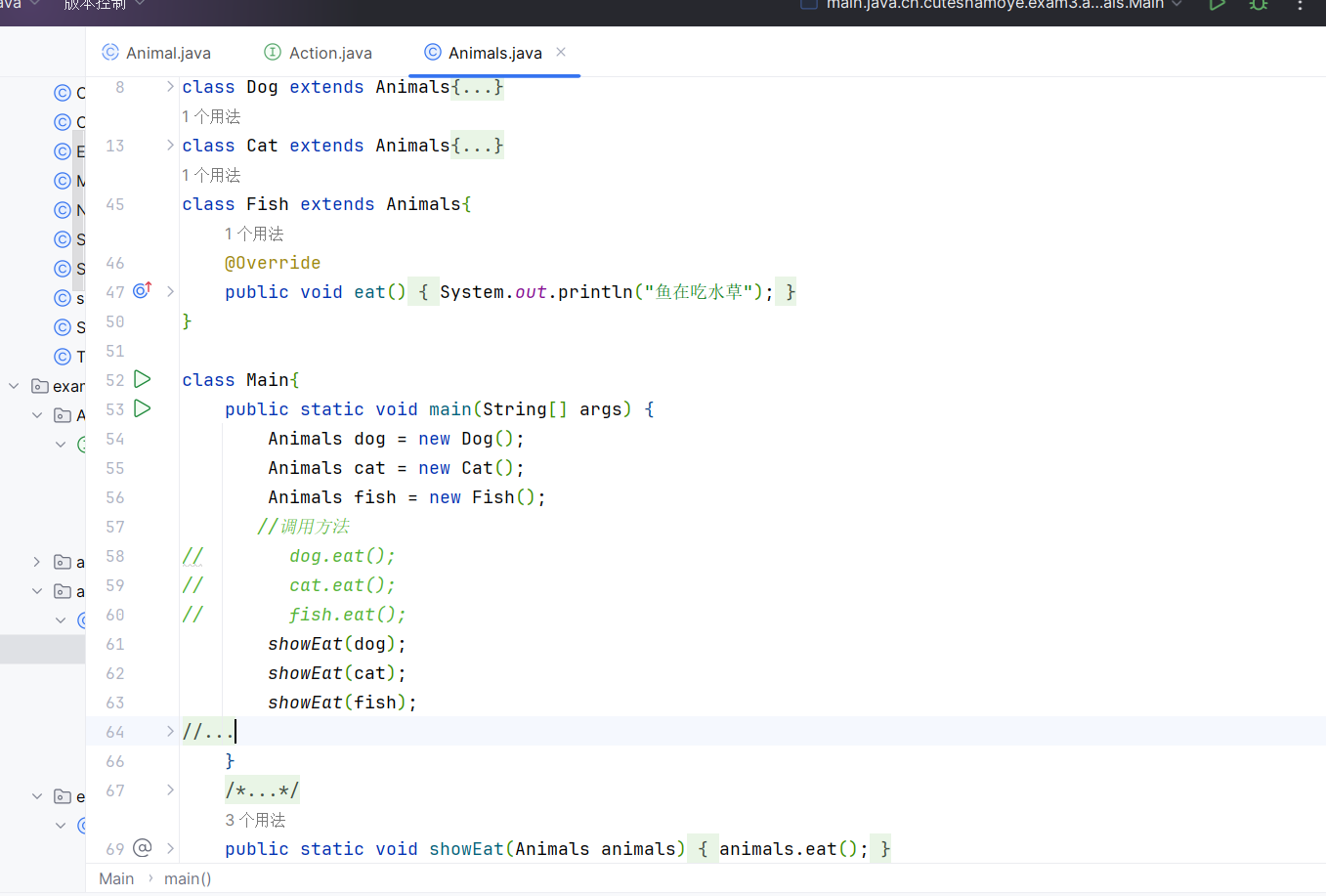
//需要补充代码

}

**如果参数是Dog对象，则输出“狗在吃骨头”；**

**如果参数是Cat对象，则输出“猫在吃鱼”；**

**如果参数是Fish对象，则输出“鱼在吃水草”**

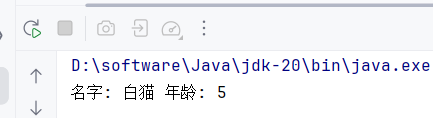




4.5-

1、**在上面的Cat类里重写toString()方法，返回姓名年龄字符串，例如：“名字：白猫”。**





2、（**可选）扩展题：研究如何重写类的hashCode()和equals(),实现判断两个对象相等。举例说明。**

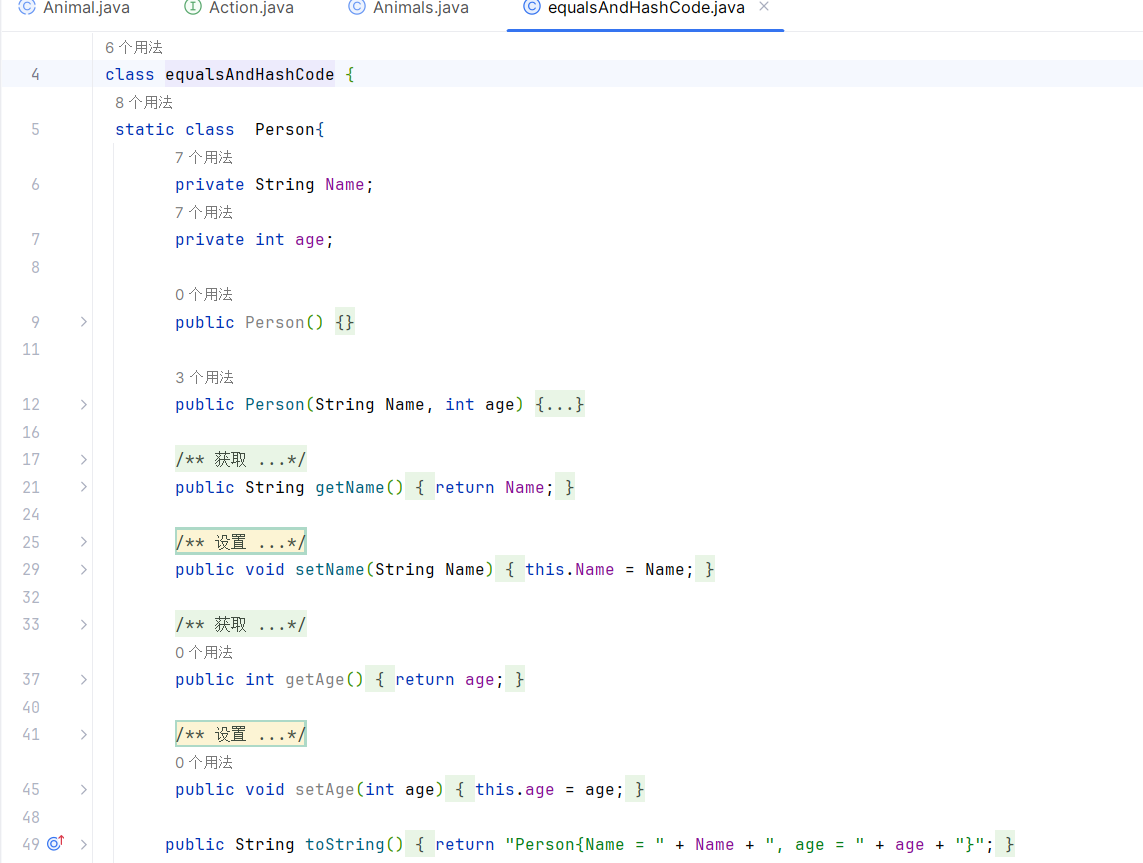
要重写类的hashCode()和equals()方法，以实现判断两个对象相等，需要遵循以下规则：

如果两个对象相等（根据业务逻辑判断），则它们的hashCode()方法应该返回相同的值。

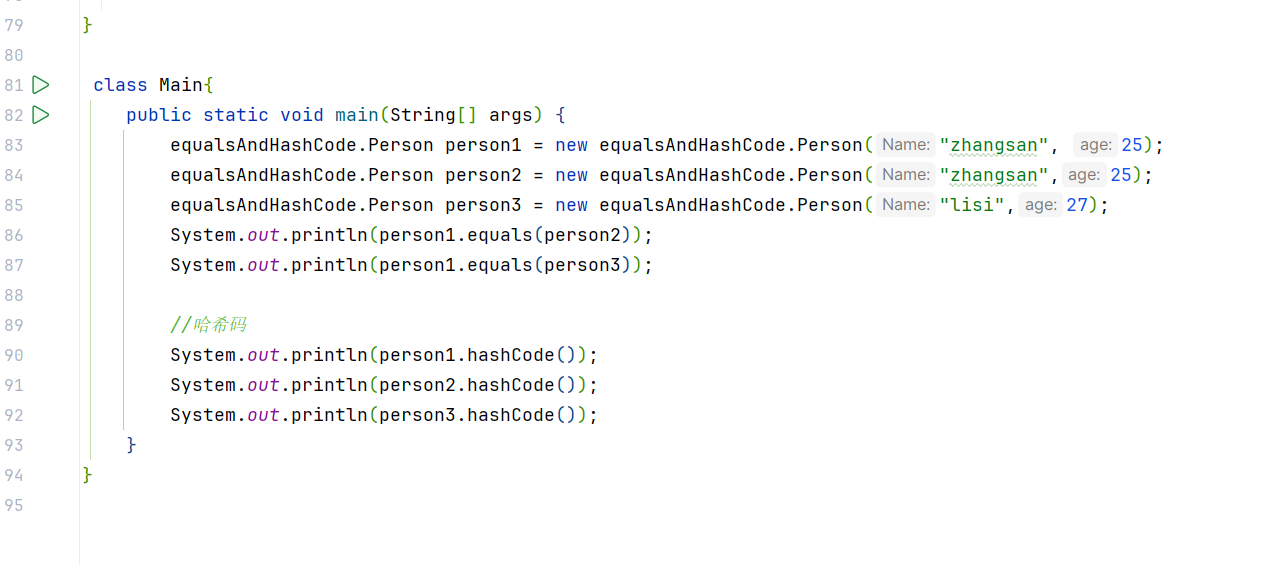
如果两个对象不相等，则它们的hashCode()方法可以返回不同的值，但最好是不同的对象返回不同的哈希值，以提高哈希表的性能。

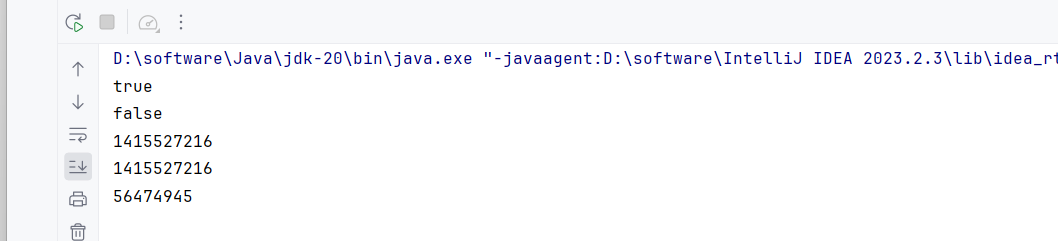
如果两个对象相等，则它们的equals()方法应该返回true。

在hashCode()方法中，使用了一种常用的计算哈希值的方式，通过将每个属性的哈希值乘以一个质数，并将结果相加，最后返回结果。



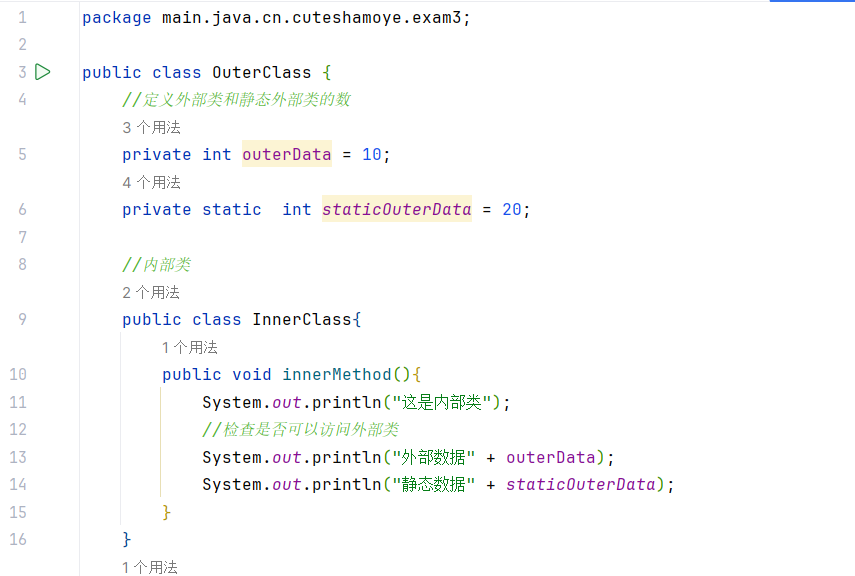




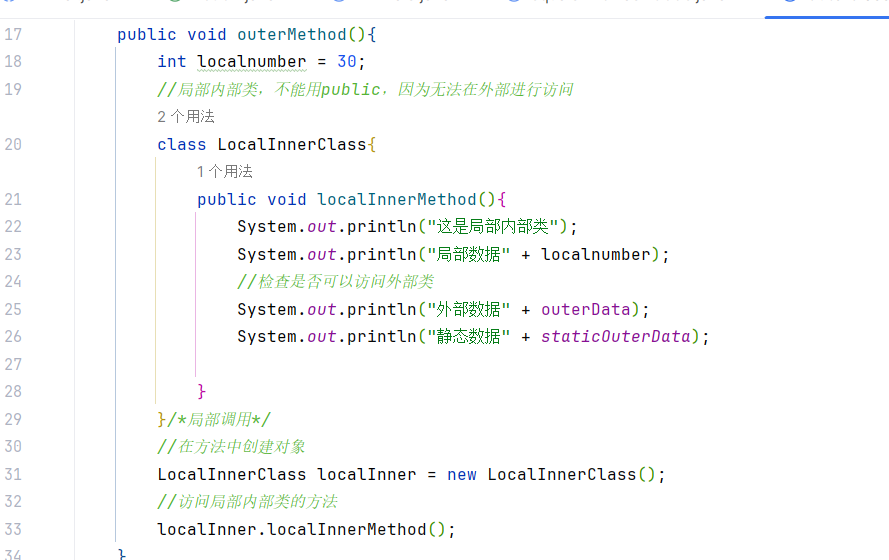


* 1. **举一个内部类、局部内部类、静态内部类、匿名内部类的代码例子。**

内部类是定义在另一个类内部的类。它们可以访问外部类的成员和方法，并且可以被外部类的实例化对象创建。



局部内部类是定义在方法或代码块内部的类。它们只能在所在的方法或代码块中使用，并且不能被外部类的实例化对象创建。

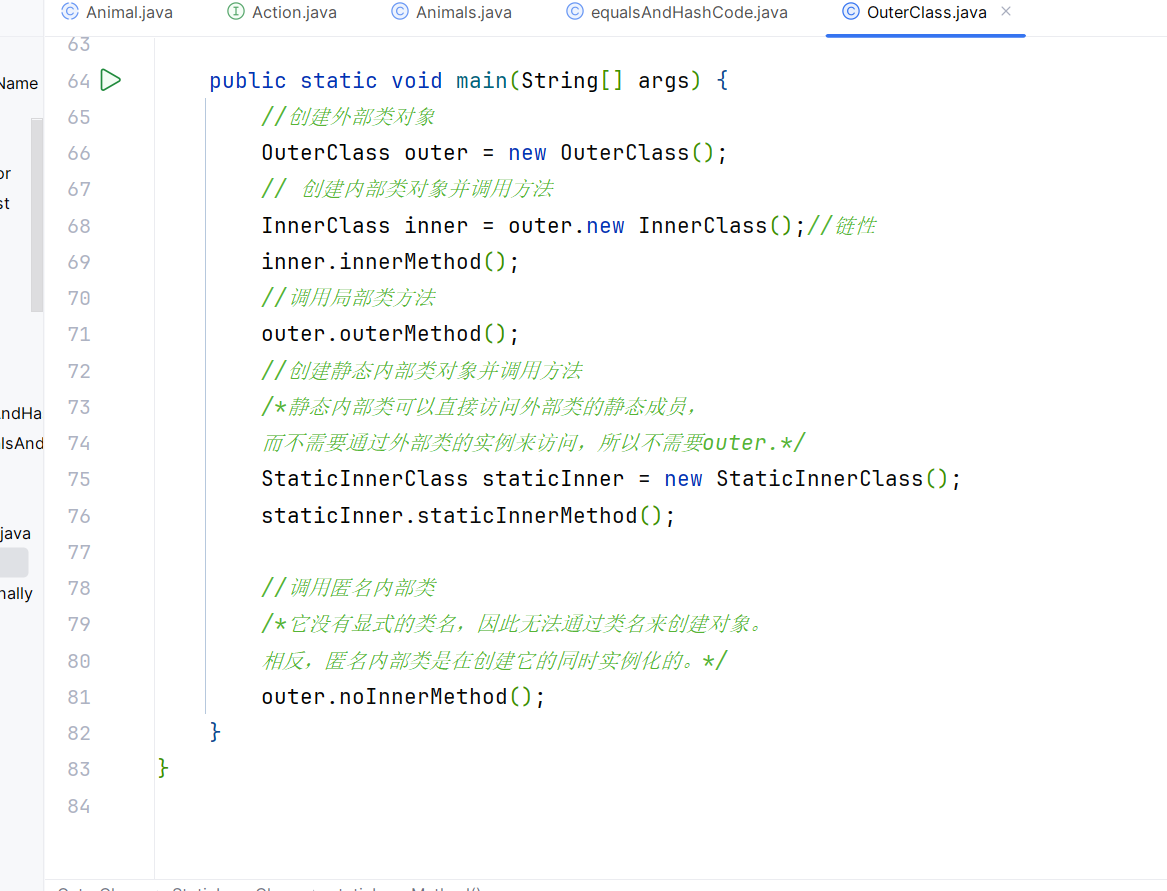


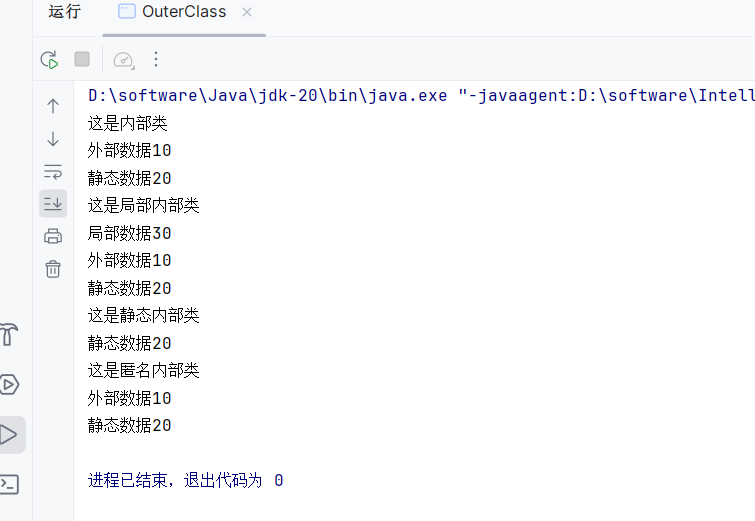
静态内部类是定义在另一个类内部的静态类。它们与外部类的实例化对象无关，可以直接通过外部类的类名访问。



匿名内部类是没有名字的内部类，它们通常用于创建只需要使用一次的类的实例。它们没有构造函数，可以直接在创建对象时定义类的实现。







* 1. **举一个使用try….catch…finally的例子。**

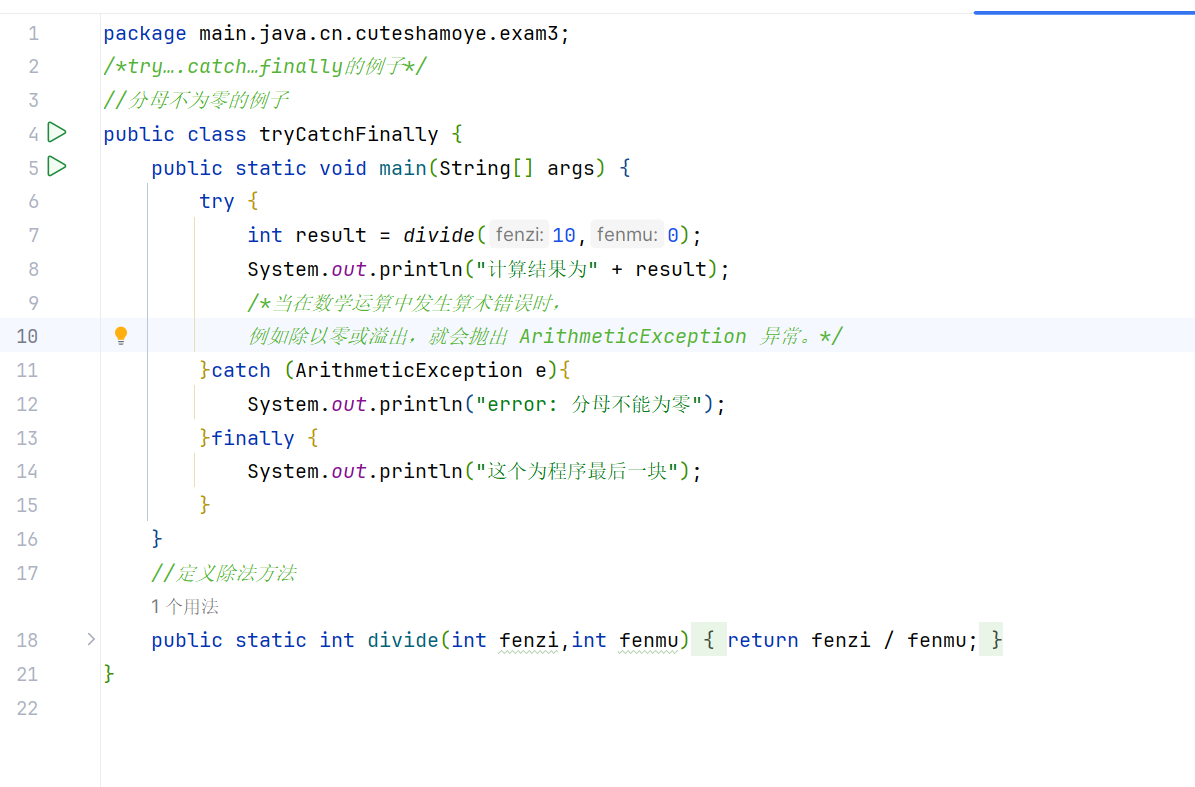
try...catch...finally 的执行流程：

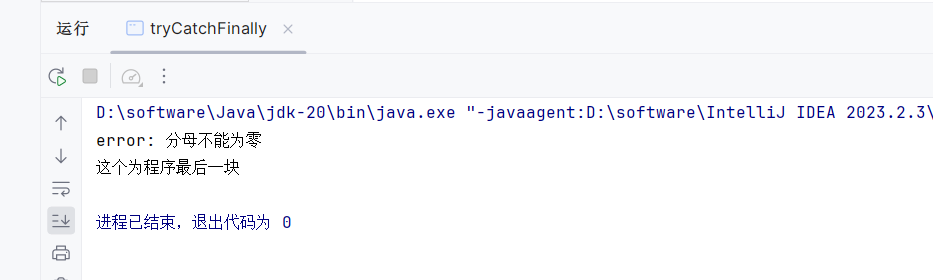
执行 try 块中的代码。

如果在 try 块中发生了异常，程序将跳转到与异常类型匹配的 catch 块，并执行其中的代码。

如果存在 finally 块，无论是否发生异常，都会执行其中的代码。

如果在 catch 块中的代码或 finally 块中的代码中发生了异常，该异常将被传播到上一级的异常处理机制。





* 1. **举一个自定义异常类的使用例子。**

throws 关键字用于声明方法可能抛出的异常类型。在这种情况下，validateAge 方法可能会抛出一个 InvalidAgeException 异常。这意味着在调用 validateAge 方法时，调用者必须要么使用 try...catch 块来捕获并处理这个异常，要么继续将这个异常向上抛出。

