

类的封装继承和多态

封装、继承和多态是面向对象编程的三大编程。

封装，把对象的属性和方法结合成一个独立的整体，隐藏实现细节，并提供对外访问的接口。

继承，从已知的一个类中派生出一个新的类，叫子类。子类实现了父类所有非私有化的属性和方法，并根据实际需求扩展出新的行为。

多态，多个不同的对象对同一消息作出响应，同一消息根据不同的对象而采用各种不同的方法。

类的封装

封装的目的是为了保障变量的安全性，使得使用者在使用的时候不必要在意实现的细节，只需要通过外部接口来访问类的成员，如果不进行封装，那么就可以通过创建类实例来创建实例，可能会修改影响到其他代码，所以说在编写类时就会将成员变量私有化，在外部通过Getter和Setter来查看和设置即可；

★将Cat类中的成员变量变为 `private` ,这样在外部就访问不到了

```
1 package com.company;
2
3 import sun.awt.windows.WPrinterJob;
4
5 public class Cat {
6     private String name;
7     private String sex;
8     private int age;
9     private static String info;
10
11     public Cat(String name,String sex,int age){
12         this.name = name;
13         this.sex = sex;
14         this.age = age;
15     }
16 }
```

```
public class Main {
    public static void main(java.lang.String[] args) {
        // write your code here
        Cat cat = new Cat( name: "小花猫", sex: "母", age: 3);
        out.println(cat.age);
    }
}

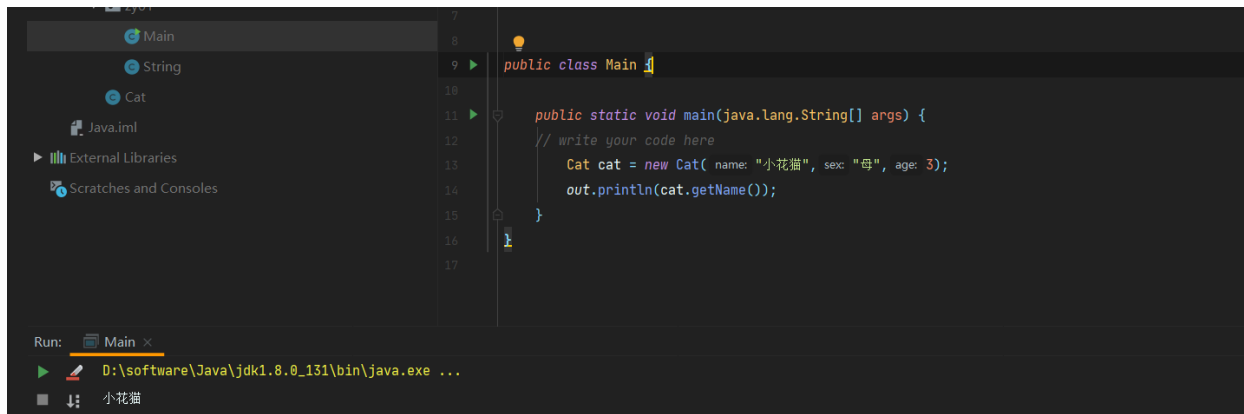
'age' 在 'com.company.Cat' 中具有 private 访问权限
使 'Cat.age' 为 public Alt+Shift+Enter More actions... Alt+Enter

com.company.Cat
private int age
```

这样在外部就不能直接通过类对象来获取和修改，但是可以使用get和set来获取和修改

```
1 package com.company;
```

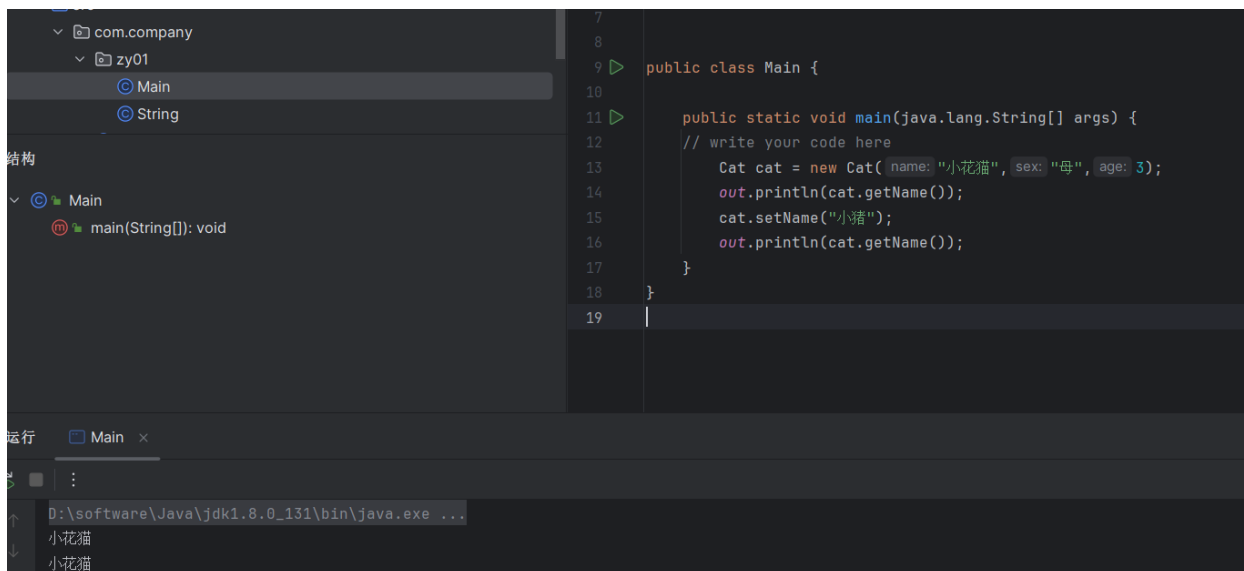
```
2
3 import sun.awt.windows.WPrinterJob;
4
5 public class Cat {
6     private String name;
7     private String sex;
8     private int age;
9     private static String info;
10
11     public Cat(String name,String sex,int age){
12         this.name = name;
13         this.sex = sex;
14         this.age = age;
15     }
16
17     public String getName() {
18         return name;
19     }
20
21     public void setName(String name) {
22         this.name = name;
23     }
24
25     public String getSex() {
26         return sex;
27     }
28
29     public void setSex(String sex) {
30         this.sex = sex;
31     }
32
33     public int getAge() {
34         return age;
35     }
36
37     public void setAge(int age) {
38         this.age = age;
39     }
40
41     public static String getInfo() {
42         return info;
43     }
44
45     public static void setInfo(String info) {
46         Cat.info = info;
47     }
48 }
49
```



★也可以通过修改set函数，对外界修改进行一些额外的操作，如下，使得外界调用修改 name 时不能修改包含有"猪"这个字

```
1 public void setName(String name) {
2     if(name.contains("猪")) return;
3     this.name = name;
4 }
```

可以看到修改失败



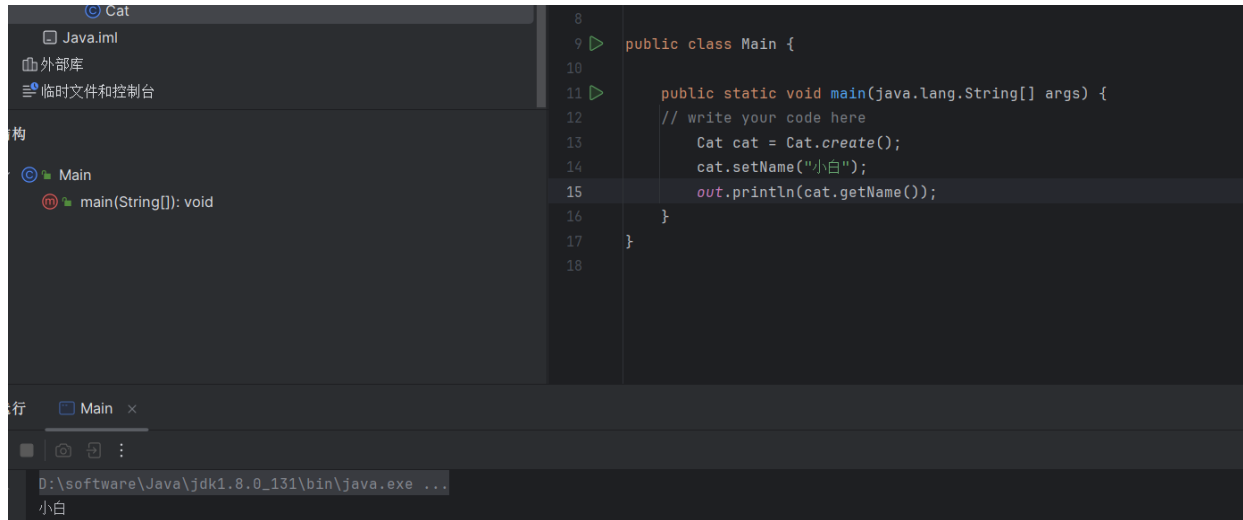
★还可以限制构造方法，将构造方法改为私有，只能通过内部的方式来构造对象

```
1 public class Cat {
2     private String name;
3     private String sex;
4     private int age;
5     private static String info;
6
7     private Cat() {
8
9     }
10
11     public static Cat create(){
12         return new Cat();
13     }
14 }
```

```

13     }
14
15     public String getName() {
16         return name;
17     }

```



类的继承

有时候定义很多类，它们都会有相同的属性，我们可以将这些相同的属性共同组成一个父类，**子类可以继承自父类但是不能访问父类的私有成员变量**

类的继承就相当于你和你老爸，你的很多是不是都是继承你的老爸的，性格啊、外貌啊、等等，当然有些东西是你爸爸自己独有的你继承不了，比如说你爸爸的年龄你肯定继承不了吧

还有一种就是人和我们自己，每个人的相同特征是不是都是有一个名字，有年龄，还有性别，这是人类共有的，但是你也会有属于你自己的东西，比如说你的职业，你的性格等等，就可以看作是两个类，一个类叫做人，一个类叫做学生，学生是继承人的特性的，那么学生也会有自己的成员变量，比如在哪里读书，成绩怎么样等等

这是一个父类

```

1 public class Person{
2     String name;
3     String sex;
4     int age;
5 }

```

★可以通过extends关键字来继承父类，创建了一个学生类继承了 Person

```

1 public class Student extends Person {
2
3 }

```

★类可以不断继续向下衍生，但是每个子类只能继承一个类，如果不想这个类被继承需要将类标记为 final

```

1 package com.company;
2
3 public final class Person{
4     String name;
5     String sex;
6     int age;
7 }
8

```



前面我们说过，子类无法访问到父类的私有变量，但是是已经继承了但是无法访问

★如果父类有一个全参构建方法，需要参数才可以创建对象，那么在子类中是无法直接继承的

```

1 public class Person{
2     String name;
3     String sex;
4     int age;
5
6     public Person(String name, String sex, int age) {
7         this.name = name;
8         this.sex = sex;
9         this.age = age;
10    }
11 }

```



★因为子类在构建的时候，不仅仅要初始化子类的属性，还需要初始化父类的属性，在默认的情况下子类调用了父类的构建方法，但是如果在全参的情况下，就需要手动指定使用 `super`，`super` 就表示父类 `this` 表示当前类

```

1 public class Student extends Person {
2
3     public Student(String name, String sex, int age) {
4         super(name, sex, age);
5     }
6 }
7

```

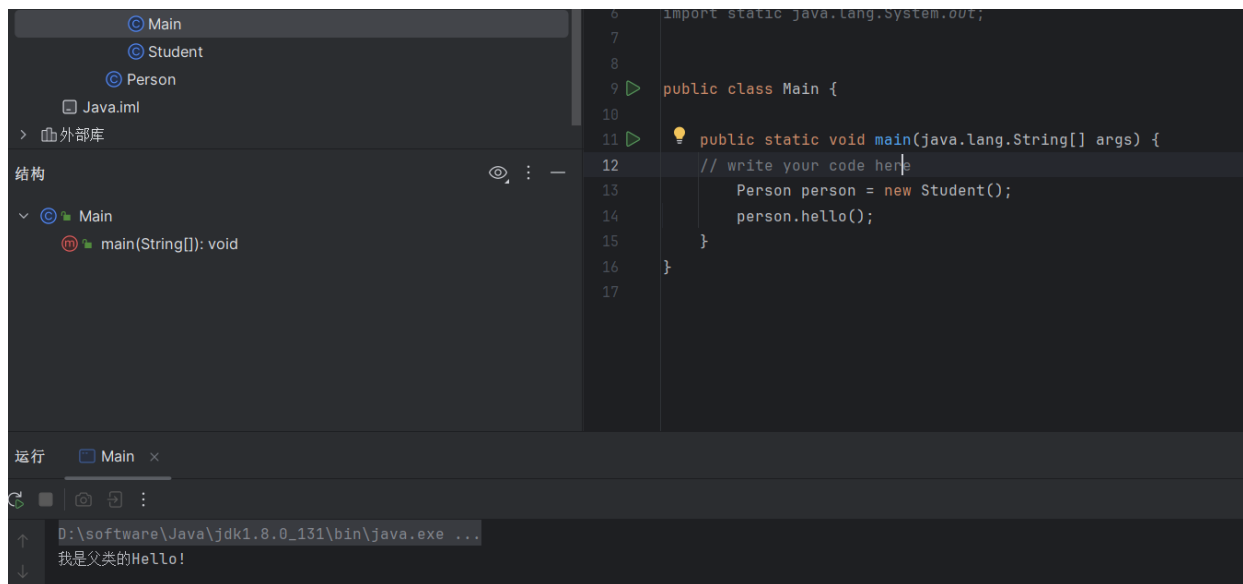
我们在使用子类时，是可以当作父类去使用的，在父类定义一个 `hello` 方法

```

1 package com.company;
2
3 public class Person{
4     String name;
5     String sex;
6     int age;
7
8     public void hello() {
9         System.out.println("我是父类的Hello!");
10    }
11 }
12

```

这里我用父类的类型，去应引用了一个子类的类型，但是这个子类的类型是访问到父类的东西的



★但是这里不是直接变成父类了，仅仅是当作父类使用而已，可以使用强制类型转换来转换成子类(但是我
觉得没有任何意义)

```

1 public class Main {
2
3     public static void main(java.lang.String[] args) {
4         // write your code here
5         Person person = new Student();
6         person.hello();
7         Student student = (Student) person;
8         student.hello();
9     }
10 }

```

★如果我们需要判断这个变量所引用的对象到底是什么类可以使用 `instanceof` 来判断

```

1 public class Main {
2
3     public static void main(java.lang.String[] args) {
4         // write your code here
5         Person person = new Student();
6         if (person instanceof Person) {
7             out.println("是Person类");
8         }
9         if (person instanceof Student) {
10            out.println("是Student类");
11        }
12    }
13 }
14

```

★如果子类 and 父类存在于同名的成员变量，那么就可以使用到 `super` 来访问

```

1 package com.company;
2
3 public class Person{
4     public String name = "白白白";
5     String sex;
6     int age;
7
8     public void hello() {
9         System.out.println("我是父类的Hello!");
10    }
11 }
12

```

★子类在子类中使用 `super` 来调用父类的名字

```

1 package com.company.zy01;
2
3 import com.company.Person;
4
5 public class Student extends Person {

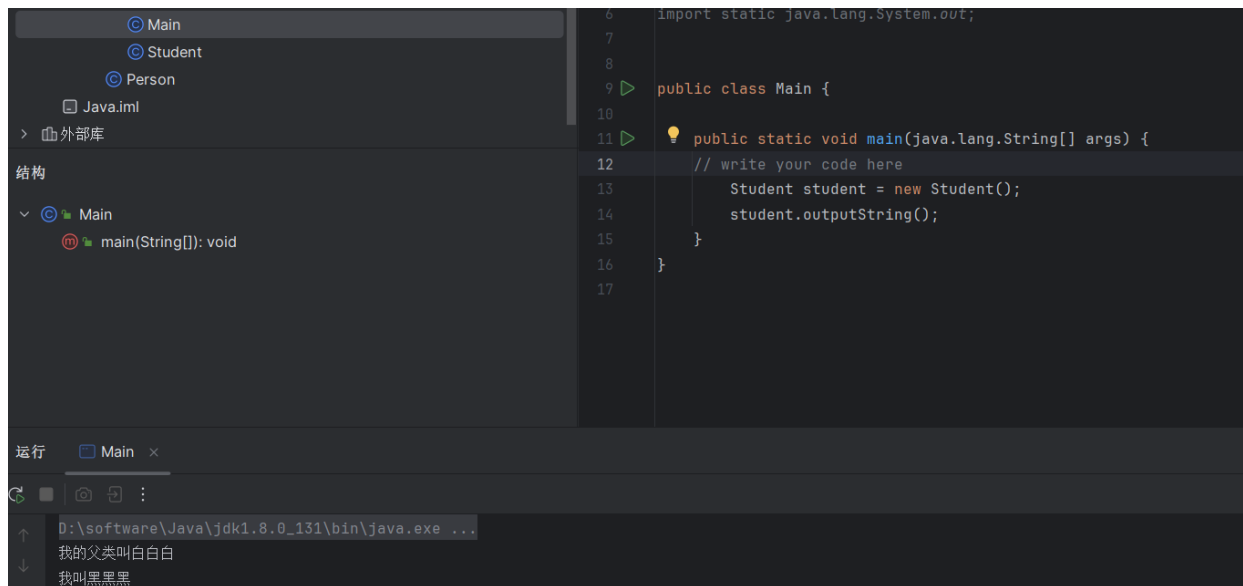
```

```

6      String name="黑黑黑";
7
8      public void outputString() {
9          System.out.println("我的父类叫"+super.name);
10         System.out.println("我叫"+this.name);
11     }
12
13 }
14

```

输出



顶层Object类

★所有的类默认都是默认继承了Object类，可以看到提示冗余



如下图所示，加粗的都是类本身自带的方法，而不加粗的都是继承自Object类



可以来看看这个类里面有哪些内容

```
1 public class Object {
2
3     private static native void registerNatives();    //标记为native的方法是本地方法，
底层是由C++实现的
4     static {
5         registerNatives();    //这个类在初始化时会对类中其他本地方法进行注册，本地方法不是
我们SE中需要学习的内容，我们会在JVM篇视频教程中进行介绍
6     }
7
8     //获取当前的类型Class对象，这个我们会在最后一章的反射中进行讲解，目前暂时不会用到
9     public final native Class<?> getClass();
10
11     //获取对象的哈希    值，我们会在第五章集合类中使用到，目前各位小伙伴就暂时理解为会返回对象
存放的内存地址
12     public native int hashCode();
13
14     //判断当前对象和给定对象是否相等，默认实现是直接等号判断，也就是直接判断是否为同一个对象
15     public boolean equals(Object obj) {
16         return (this == obj);
17     }
18
19     //克隆当前对象，可以将复制一个完全一样的对象出来，包括对象的各个属性
20     protected native Object clone() throws CloneNotSupportedException;
21
22     //将当前对象转换为String的形式，默认情况下格式为 完整类名@十六进制哈希值
23     public String toString() {
24         return getClass().getName() + "@" + Integer.toHexString(hashCode());
25     }
26
27     //唤醒一个等待当前对象锁的线程，有关锁的内容，我们会在第六章多线程部分中讲解，目前暂时不会
用到
28     public final native void notify();
29
30     //唤醒所有等待当前对象锁的线程，同上
```

```

31     public final native void notifyAll();
32
33     //使得持有当前对象锁的线程进入等待状态，同上
34     public final native void wait(long timeout) throws InterruptedException;
35
36     //同上
37     public final void wait(long timeout, int nanos) throws InterruptedException
38 {
39     ...
40 }
41
42 //同上
43 public final void wait() throws InterruptedException {
44     ...
45 }
46
47 //当对象被判定为已经不再使用的“垃圾”时，在回收之前，会由JVM来调用一次此方法进行资源释放之
48 //类的操作，这同样不是SE中需要学习的内容，这个方法我们会在JVM篇视频教程中详细介绍，目前暂时不会用
49 //到
50 protected void finalize() throws Throwable { }
51 }

```

方法的重写

如果在自定义类中重写了 `equals` 方法，那么在 `LinkedList`（或任何使用 `equals` 进行比较的集合类）中，如调用 `indexOf` 方法时，默认就会使用该类中重写的 `equals` 方法来进行对象间的比较。

原因在于Java的动态绑定机制（也称为晚期绑定或运行时绑定）。当你在一个类中重写了从父类继承来的方法，Java虚拟机（JVM）会在运行时根据对象的实际类型来决定调用哪个版本的 `equals` 方法。这意味着，即便是在像 `LinkedList` 这样的集合类内部调用 `equals`，它也会自动指向被重写过的那个版本，只要该调用发生在重写该方法的类的实例上。

在您之前提供的 `LinkedList.indexOf(Object o)` 源码中，可以看到这一行代码：

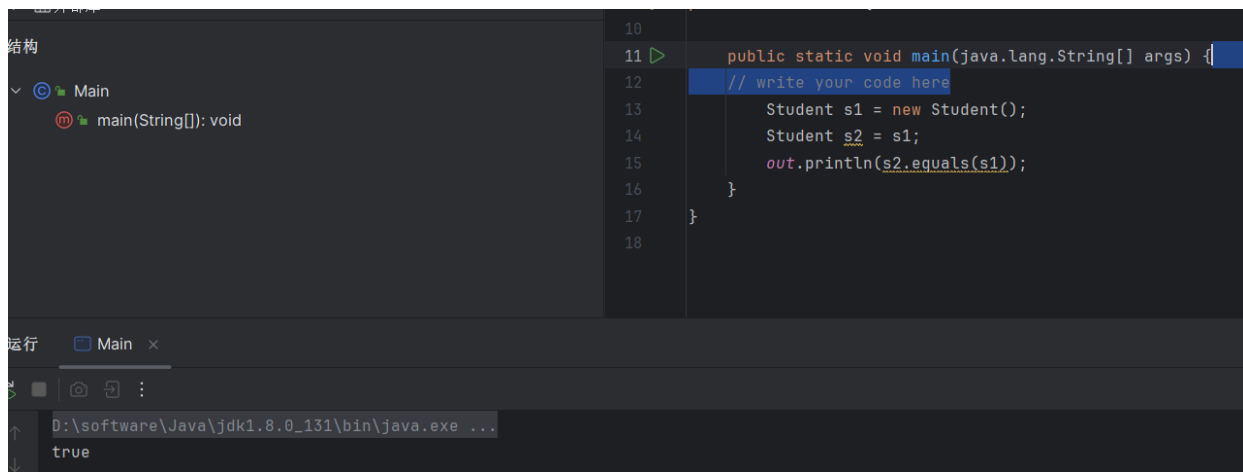
```

1 | if (o.equals(x.item))

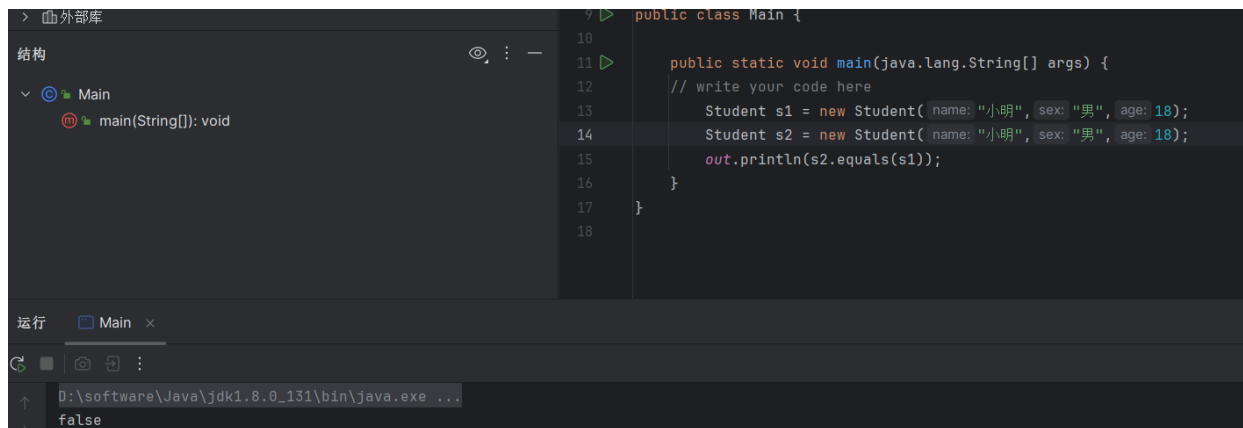
```

这里调用了 `equals` 方法，如果 `o` 是一个自定义类的实例，并且这个类重写了 `equals` 方法，那么这里实际上会执行您自定义的 `equals` 逻辑，而非 `Object` 类中的默认实现。这就是为什么重写 `equals` 会影响 `indexOf` 行为的原因。

方法的重写是为了将之前原本的方法实现，来重写方法，使得达到自己想要的效果，上面我们提到过，任何类都会继承 `Object` 类，那我们可以使用重写的方法来修改一下这个父类的方法，接下来我们使用代码来重写一下 `equals` 方法，在 `Object` 中提到了 `equals` 方法是检测两个对象是否相等



但是如果是以下情况就不能，他们两名字一样，性别一样，年龄也一样，我们可以重写 `equals` 方法是判断成员变量是否相等，如果相等就表示他们相等



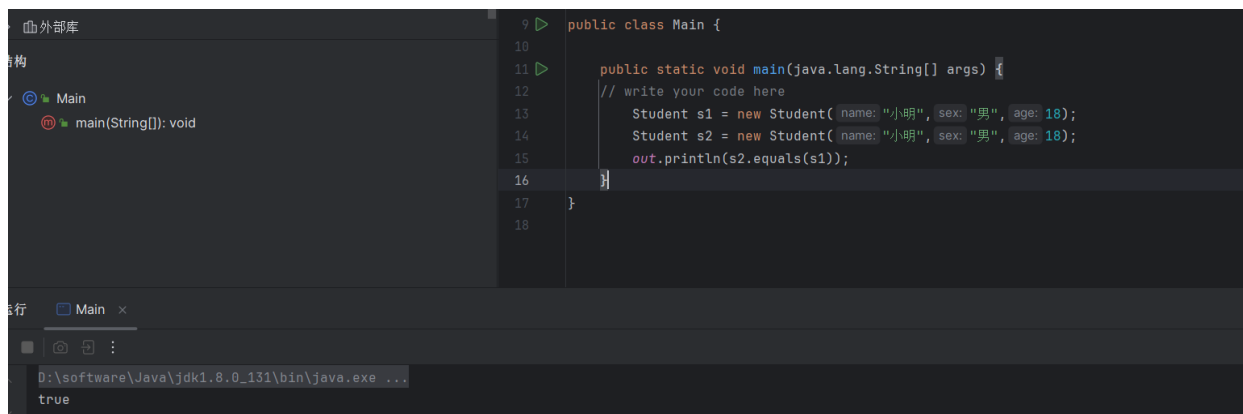
重写方法，在 `Person` 类中

```

1 public class Person{
2     ...
3
4     @Override //重写方法可以添加 @Override 注解，有关注解我们会在最后一章进行介绍，这个
        注解默认情况下可以省略
5     public boolean equals(Object obj) { //重写方法要求与父类的定义完全一致
6         if(obj == null) return false; //如果传入的对象为null，那肯定不相等
7         if(obj instanceof Person) { //只有是当前类型的对象，才能进行比较，要是都不
            是这个类型还比什么
8             Person person = (Person) obj; //先转换为当前类型，接着我们对三个属性挨个
                进行比较
9             return this.name.equals(person.name) && //字符串内容的比较，不能使用
                ==，必须使用equals方法
10                this.age == person.age && //基本类型的比较跟之前一样，直接
                ==
11                this.sex.equals(person.sex);
12        }
13        return false;
14    }
15 }

```

再次使用就是返回 `true` 了



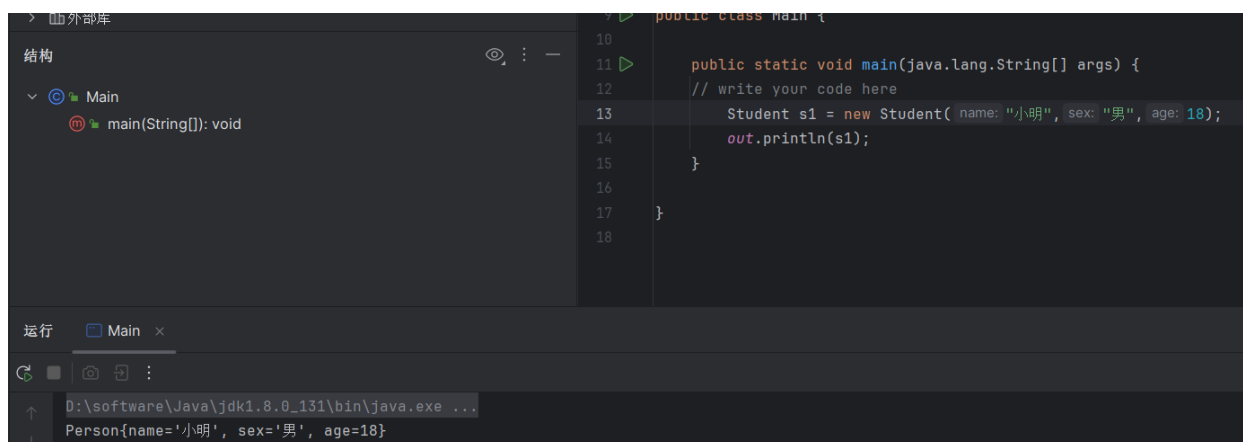
```
public class Main {
    public static void main(java.lang.String[] args) {
        // write your code here
        Student s1 = new Student( name: "小明", sex: "男", age: 18);
        Student s2 = new Student( name: "小明", sex: "男", age: 18);
        out.println(s2.equals(s1));
    }
}
```

运行 Main x

0:\software\Java\jdk1.8.0_131\bin\java.exe ...
true

★在类中，有时候为了方便我们查看各个成员变量对应的值，就可以重写 `toString` 方法，这个是可以通 `ide` 快捷生成的

```
1  @Override
2  public String toString() {
3      return "Person{" +
4          "name='" + name + '\'' +
5          ", sex='" + sex + '\'' +
6          ", age=" + age +
7          "'}";
8  }
```



```
public class Main {
    public static void main(java.lang.String[] args) {
        // write your code here
        Student s1 = new Student( name: "小明", sex: "男", age: 18);
        out.println(s1);
    }
}
```

运行 Main x

0:\software\Java\jdk1.8.0_131\bin\java.exe ...
Person{name='小明', sex='男', age=18}

⚠ 静态类是不能被重写但是可以被继承

基于这种方法，在一个父类中定义了一个行为(方法)，不同的子类会表现出不同的方法，比如在人这个类中定义了一个收入，学生和工人的收入应该是会不一样的，所以说: **对于一个类的行为，不同的子类可以出现不同的行为**

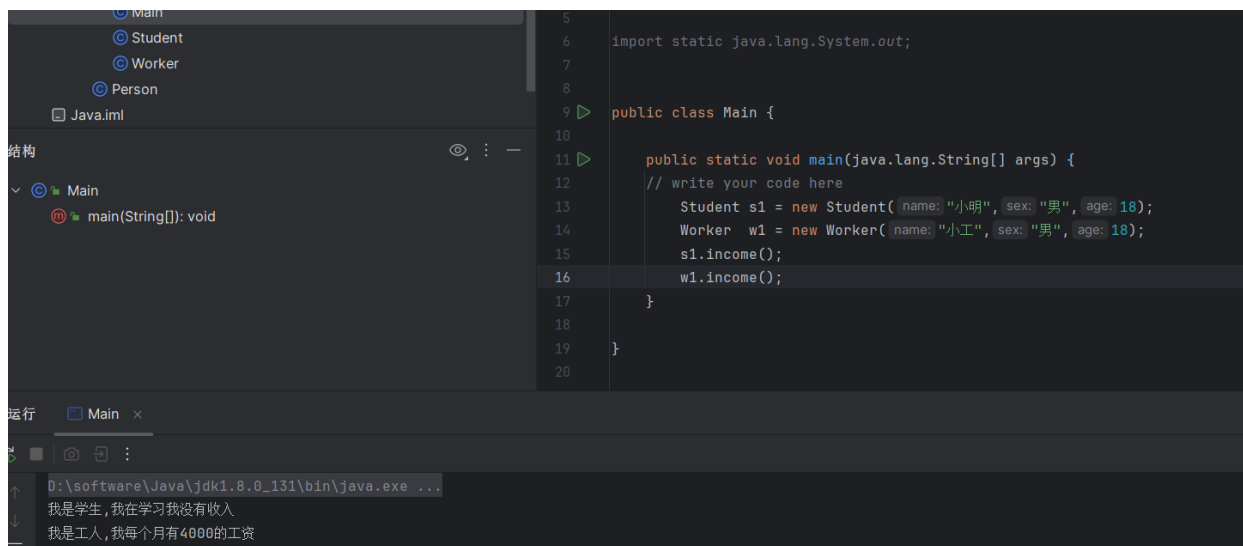
这其实就是多态特性的一种体现

```
1  public class Person {
2      ....
3      public void income(){ //定义了收入行为
4      }
5  }
6  }
```

在两个类中重写 income 方法

```
1  @Override
2  public void income() {
3      system.out.println("我是学生,我在学习我没有收入");
4  }
5
6
7  @Override
8  public void income() {
9      system.out.println("我是工人,我每个月有4000的工资");
10 }
```

在 Main 函数调用



The screenshot shows an IDE with a project structure on the left containing Main, Student, Worker, and Person classes. The Main class is selected, and its code is displayed in the editor. The code imports System.out and defines a main method that creates Student and Worker objects and calls their income() methods. The output window at the bottom shows the results of the execution: "我是学生,我在学习我没有收入" followed by "我是工人,我每个月有4000的工资".

```
5
6 import static java.lang.System.out;
7
8
9 public class Main {
10
11     public static void main(java.lang.String[] args) {
12         // write your code here
13         Student s1 = new Student( name: "小明", sex: "男", age: 18);
14         Worker w1 = new Worker( name: "小工", sex: "男", age: 18);
15         s1.income();
16         w1.income();
17     }
18
19 }
20
```

运行 Main x

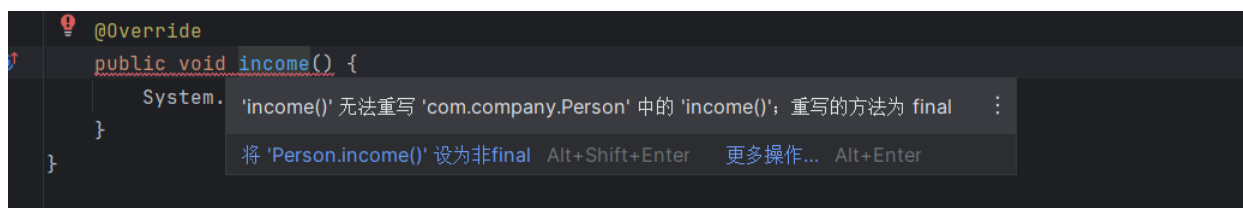
D:\software\Java\jdk1.8.0_131\bin\java.exe ...

我是学生,我在学习我没有收入

我是工人,我每个月有4000的工资

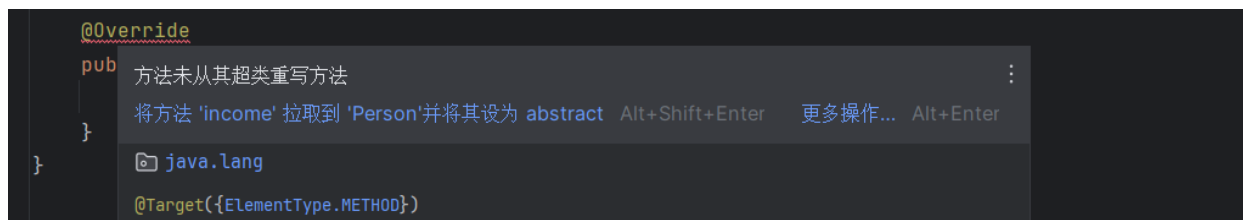
★那么如果不希望重写这个方法,就可以假如 final 关键字,表示这个方法已经是最终形态了;

```
1  public final void income(){
2  }
```

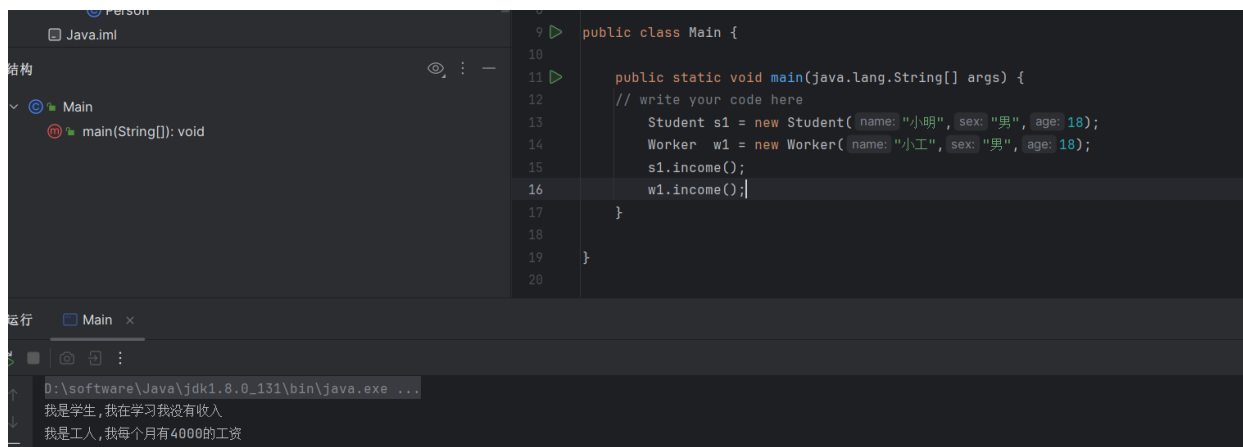
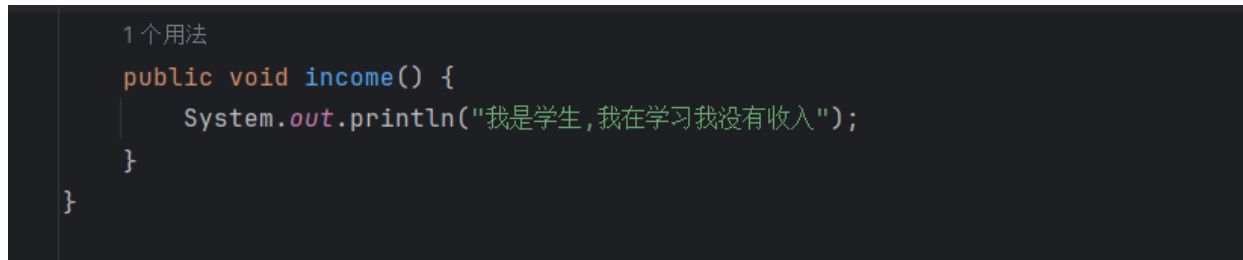


★还有一种方法就是将父类的方法直接设置为private,那么子类也同样无法访问,也不能重写了,但是子类可以不加注解,直接创建一个新的同名方法

```
1  public class Person {
2      ....
3      private void income(){
4      }
5  }
```

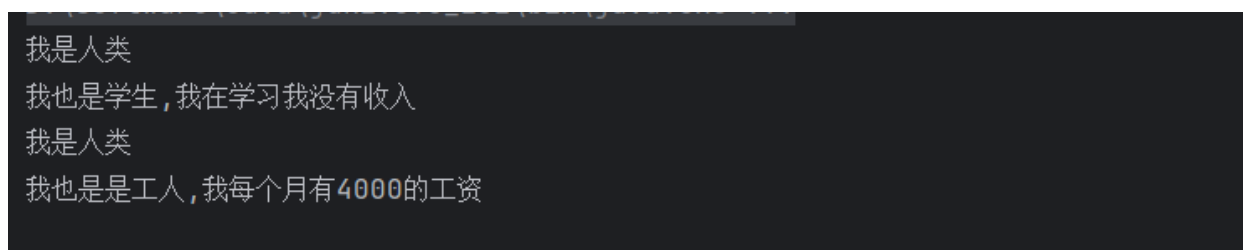


将 override 注解去除



★如果需要子类中调用父类的方法，还是可以使用 super 字段

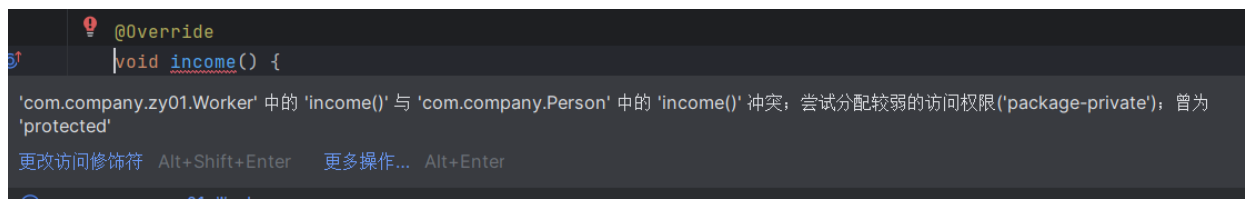
```
1  @Override
2  public void income() {
3      super.income();
4      System.out.println("我也是学生,我在学习我没有收入");
5  }
6  }
```



★子类的重写权限可以别父类大，但是不能比父类低

```
1  protected void income(){ //父类中的方法是protected权限
2      System.out.println("我是人类");
3  }
```

修改子类中的方法，可以看到如果我们将子类变成默认权限，就会提示权限报错



抽象类

★在Java中，抽象是指无法直接实例化的类和没有具体实现的方法。

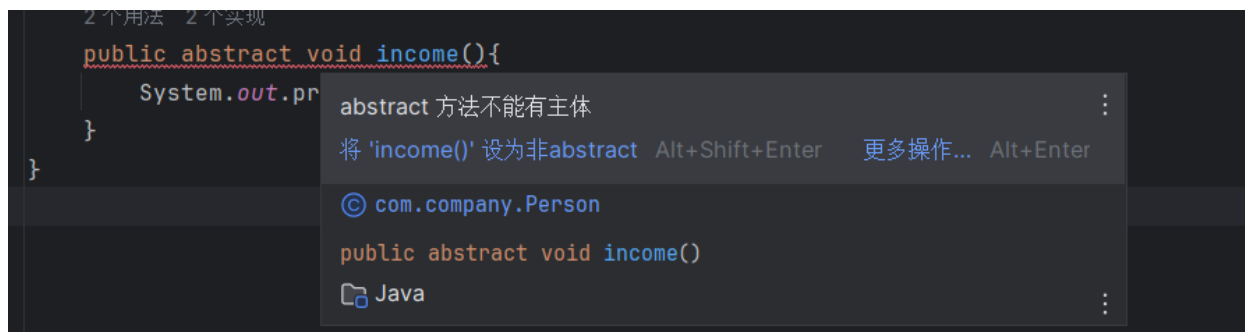
- **抽象类 (Abstract Class)**：抽象类是一种特殊的类，它不能被直接实例化。它的目的是为了被其他类继承，并提供一组公共的接口或部分实现供子类使用或扩展。抽象类通常包含抽象方法和具体方法：
 - **抽象方法**：在抽象类中定义但没有具体实现的方法，它们只有声明而没有方法体。这些方法必须在任何直接或间接继承该抽象类的子类中被重写（实现）。
 - **具体方法**：抽象类也可以包含已经实现了的的具体方法，这些方法可以在子类中直接使用，或者根据需要进行重写。
- **抽象方法 (Abstract Method)**：抽象方法是一种没有实现的方法，只有方法签名，即方法名、参数列表和返回类型。它们必须在继承了抽象类的任何非抽象子类中被实现。

★之前我们说过多态就是一个父类里面有一个行为方法，但是在继承它的不同子类的方法就会有不同的结果，这个行为方法是一点会被重写在子类中实现的，这个行为方法完全就是可以在子类中进行实现，父类不需要提供实现的，就可以使用抽象类，使用 `abstract` 定义为抽象类

```
1 public abstract class Person { //将Person类定义为抽象类
2
3 }
```

★ 抽象类和普通类的区别在于，抽象类可以拥有抽象方法，前面说过抽象方法不需要在父类中去实现，而是要在子类中去实现，所以说抽象类中的抽象方法是不能有主体的；

```
1 public abstract class Person { //将Person类定义为抽象类
2     public abstract void income(); //定义抽象类
3 }
```

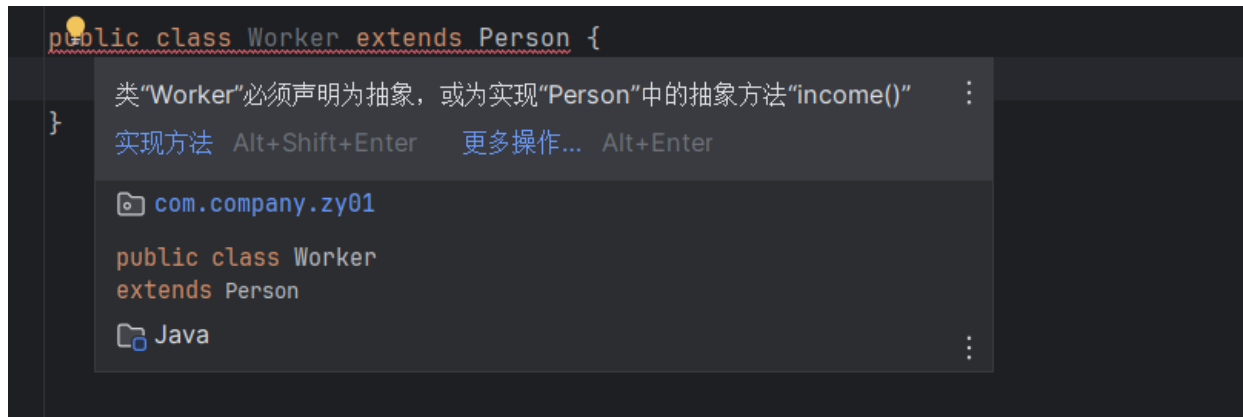


★ 子类如果继承的是一个抽象类，则必须实现抽象类中的抽象方法

```

1 public class Worker extends Person {
2     @Override
3     public void income() { //实现抽象方法
4         System.out.println("我也是工人,我每个月有4000的工资");
5     }
6 }

```



★ 抽象类不同于普通类可以直接创建类对象，因为抽象类可能会存在某些方法没有实现(比如抽象方法需要在子类中实现)，所以说无法直接通过 `new` 来创建对象,所以说要使用抽象类就需要使用子类来创建



★ 抽象类一般是做继承使用，子类也可以是一个抽象类，如果子类也是一个抽象类，那么它可以不去实现父类的抽象方法，可以看到下图并没有报错


```
public abstract class Student extends Person {
|
}

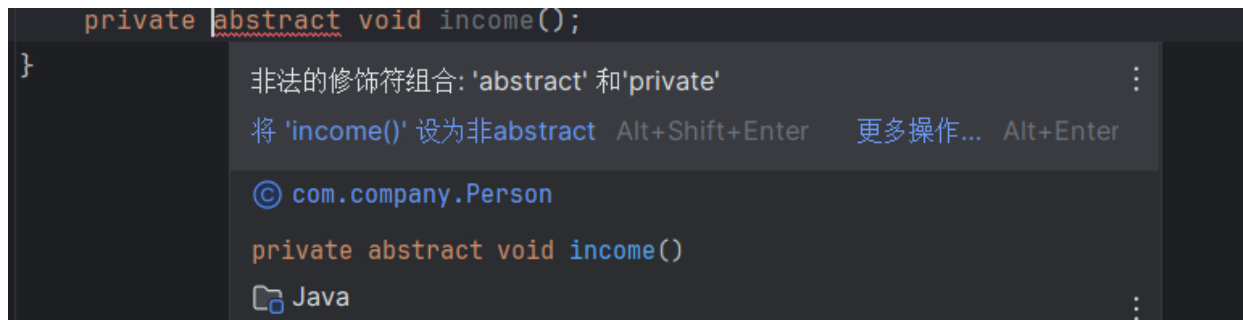
```

★ 抽象类中不是只能由抽象方法，也可以有正常方法

```
1 public abstract class Student extends Person {
2
3     public void hello(){
4         System.out.println("Hello 我是Student类的正常方法");
5     }
6
7 }

```

★ 抽象方法不能为 `private`, 抽象方法一定也是需要子类实现的，如果子类都实现不了，那就没有什么意义了



接口

接口和类不一样，它是一些方法的集合，接口包含了一类方法的定义，类可以实现这个接口，表示类支持接口代表的功能（类似于一个插件，只能作为一个附属功能加在主体上，同时具体实现还需要由主体来实现）

比如对于人的不同的职业，比如老师和学生，他们都具有学习的能力，就可以将学习作为一个结构来进行使用，只要是这个接口的类就都回有学习的能力

```
1 public interface Study {
2     public void study(); //接口内的方法也不可以创建本体，具体和抽象方法类似，但是可以使用默认方法，在下面会说到
3 }

```

之后就可以让类来调用这个接口了(需要在创建一个 `Teacher` 类)，使用关键字 `implements` 来继承类，★ 如果接口类有方法的话，接口和继承抽象类拥一样必须需要实现否则会报错，如果不想实现，可以将类设置为抽象类，或者

```
1 public class Teacher implements Study{
2 }

```



```
1 public class Teacher implements Study{
2     @Override
3     public void study() {
4         System.out.println("我是老师，我会在教书的过程中边教边学");
5     }
6 }
7
8
9 public class Student extends Person implements Study{
10
11     @Override
12     public void study(){
13         System.out.println("我是学生，我在读书的过程中学习");
14     }
15 }
```

★接口不同于继承，接口可以连接多个

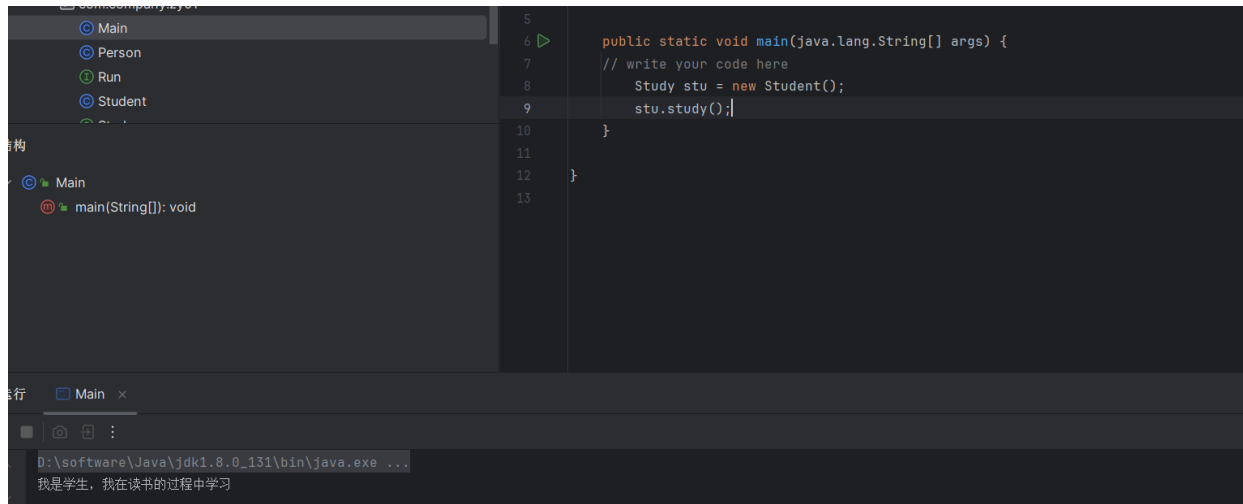
```
1 public interface Run { //在创建一个Run接口
2     public void run();
3 }
```

学生类在继承一个 Run 接口

```
1 public class Student extends Person implements Study,Run{
2
3     @Override
4     public void study(){
5         System.out.println("我是学生，我在读书的过程中学习");
6     }
7
8     public void run(){
9         System.out.println("我会跑步");
10    }
11 }
12
```

★ 学到这里我觉得这个和继承差不多，但是要比继承更加强大，如果是普通继承的话，就不是必须要实现父类中的行为，而抽象类和接口则必须实现,但是接口可以连接多个，而继承则只能继承一个

★ 接口和抽象类一样不可以直接创建对象，但是可以将接口的实现类的对象(就是连接了接口的类)，以接口的形式去使用



★ 接口也支持向连接了它的类对象强转，如下代码所示，但是我觉得非常多余，当然只需要知道就好

```
1 public class Main {
2
3     public static void main(String[] args) {
4         // write your code here
5         Study stu = new Student();
6         if(stu instanceof Student){
7             Student s1 = (Student) stu;
8             stu.study();
9         }
10    }
11
12 }
```

★ 从 Java8 开始，接口就可以存在方法的默认实现,默认方法就不需要强制要求实现

```
1 public interface Run {
2     public void run();
3     default void test1(){
4         System.out.println("我是默认实现方法1");
5     }
6     default void test2(){
7         System.out.println("我是默认实现方法2");
8     }
9 }
```

```

1 public class Main {
2
3     public static void main(String[] args) {
4         // write your code here
5         Student stu = new Student();
6         Teacher tea = new Teacher(); //这里需要将Teacher类连接Run接口
7         stu.test1();
8         tea.test2();
9     }
10 }
11

```

★ 接口不允许存在成员变量和成员方法，但是可以存在静态变量和静态方法

```

1 public interface Study {
2     String course_1 = "物理"; //我这里没有写static,默认就是static
3
4     public void study();
5     public static void sta(){
6         System.out.println("我是接口的静态方法");
7     }
8 }

```

可以看到默认就是 public static final

```

2个用法 2个实现
public interface Study {
    0个用法
    public static final String course_1 = "物理";
    0个用法 2个实现
    public void study();
}

```

和普通的类中的一样，可以直接通过接口名来调用

```

1 public class Main {
2
3     public static void main(String[] args) {
4         // write your code here
5         Study.sta();
6         System.out.println("Study.course_1");
7
8     }
9 }

```

```
public static void main(String[] args) {
    // write your code here
    Study.sta();
    Study.course_1 = "数学";
}
}
```

无法将值赋给 final 变量 'course_1'

com.company.zy01

public interface Study

Java

★ 接口是可以继承于接口的，但是接口的继承可以继承多个，接口继承接口相当于两个接口的继承

```
1 public interface Run extends Study{
2     ...
3 }
```

★ 在上面提到 object 类提供了一个克隆方法，它是需要接口才可以使用的，需要创建 Cloneable 接口

```
1 public interface Cloneable { //不需要定义任何方法
2
3
4 }
```

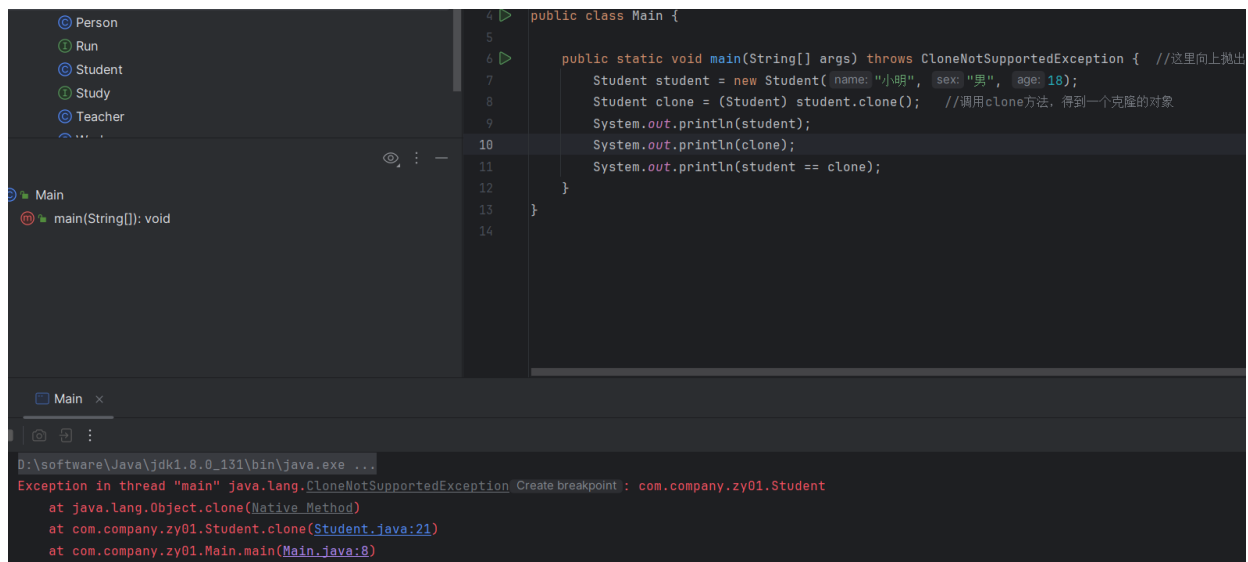
由类连接接口

```
1 public class Student extends Person implements Study,Run,Cloneable{ //连接接口
2
3
4 }
```

还需要将 clone 方法的权限范围修改一下，这就需要使用到重写

```
1 package com.company.zy01;
2
3 public class Student extends Person implements Study,Run,Cloneable{
4     @Override
5     public Object clone() throws CloneNotSupportedException { //提升clone方法的访问权限
6         return super.clone(); //因为底层是C++实现，我们直接调用父类的实现就可以了
7     }
8 }
```

然后使用，报错了



枚举类

一个学生肯定会有，它的一个状态的，比如睡觉，吃饭，学习这几个状态，那么这个状态就可以被定义为一个枚举类，主要是枚举类它可以提示你支持什么；

```

1 public class Student extends Person implements Study,Run,Cloneable{
2     private String status ;
3
4
5 }

```

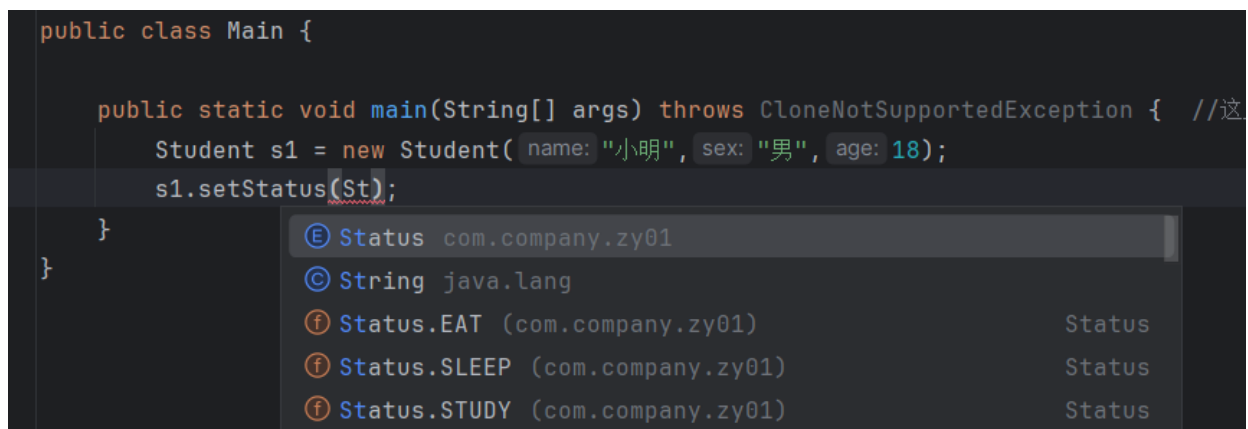
然后创建这个成员变量的枚举类

```

1 package com.company.zy01;
2
3 public enum Status {
4     STUDY,SLEEP,EAT
5 }
6

```

这样就可以知道我们这个成员变量支持哪些参数了



枚举类型使用起来就非常方便了，其实枚举类型的本质就是一个普通的类，但是它继承自 `Enum` 类，我们定义的每一个状态其实就是一个 `public static final` 的 `Status` 类型成员变量：

```
1 //这里使用javap命令对class文件进行反编译得到 Compiled from "Status.java"
2 public final class com.test.Status extends java.lang.Enum<com.test.Status> {
3     public static final com.test.Status RUNNING;
4     public static final com.test.Status STUDY;
5     public static final com.test.Status SLEEP;
6     public static com.test.Status[] values();
7     public static com.test.Status valueOf(java.lang.String);
8     static {};
9 }
```

★ 也可以给枚举的普通类

```
1 public enum Status {
2     RUNNING("睡觉"), STUDY("学习"), SLEEP("睡觉");    //无参构造方法被覆盖，创建枚举需要
3     //添加参数（本质就是调用的构造方法）
4     private final String name;    //枚举的成员变量
5     Status(String name){    //覆盖原有构造方法（默认private，只能内部使用！）
6         this.name = name;
7     }
8
9     public String getName() {    //获取封装的成员变量
10        return name;
11    }
12 }
```

这样，枚举就可以按照我们想要的中文名称打印了：

```
1 public static void main(String[] args) {
2     Student student = new Student("小明", 18, "男");
3     student.setStatus(Status.RUNNING);
4     System.out.println(student.getStatus().getName());
5 }
```

..