实验目的：

1.掌握Java构造方法的定义与使用，理解默认构造方法和重载构造方法的作用。

2.熟悉访问修饰符（public、private、protected、default）和非访问修饰符（static、final等）的应用场景及效果。

3.理解继承的基本概念，学会通过extends实现类继承，掌握方法重写（@Override）和super关键字的使用。

4.掌握接口的定义与实现（implements），理解接口的多继承特性及其在程序设计中的作用。

5.熟悉多态的实现机制（方法重写、向上转型、动态绑定），理解其在代码扩展性和灵活性中的意义。

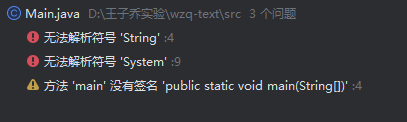
6.学习成员内部类、静态内部类、局部内部类和匿名内部类的定义与使用场景，理解其访问外部类成员的能力。

**二、实验环境：**

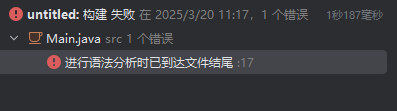
**IntelliJ IDEA Community Edition 2024.3.5**

**JDK**

**四、实验遇到的语逻辑问题：**

解决：换电脑

然后文件名和主方法永远不变

（中途换了台电脑。。）

**1：c语音和java分不清容易搞混如print，printf**

**解决问题1：熟悉java各个基础**

**2：实验二计算素数时不会用平方根计算来求出素数**

**解决问题2：通过豆包来查询math.sqrt函数的用法以绝对实验二计算素数的问题**

**3：实验4代码写完发现无法运行代码写完后少些括号，用ai帮助进行优化**

**4：实验4时发现if循环运行不正确**

**解决问题4：if（i%j == 0）， == 写成了 =使得程序发生错误**

**问题5：实验1创建i和j等于几时发现原先创建的i与j用于for循环不适用**

**解决问题5：根据for循环，令i=0，j=1.**

**问题6:在计算字符串长度使用lenth时，没有用x.length 而用了length(x)导致错误**

**解决问题:改用x.length**

**问题7:在创建setCPU方法时未能设置cpu的初始值**

**解决问题:将cpu的speed=2200;**

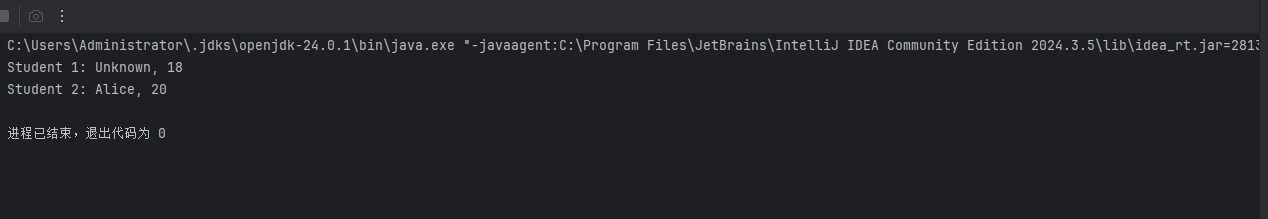
**问题8:无法调用show()方法**

**解决问题:检查了show方法发现show方法没有写，重新写了show方法**

#### ****题目1：默认构造方法与自定义构造方法****

定义一个 Student 类，包含以下属性：name（String 类型）age（int 类型），编写一个**默认构造方法**，初始化 name 为 "Unknown"，age 为 18。编写一个**带参数的构造方法**，允许传入 name 和 age 进行初始化。在main 方法中分别使用两种构造方法创建 Student 对象，并输出它们的属性。

运行结果：



源代码：

public class Main {  
 public static void main(String[] args) {  
 Student s1 = new Student(); // 使用默认构造方法  
 Student s2 = new Student("Alice", 20); // 使用自定义构造方法  
  
 System.*out*.println("Student 1: " + s1.name + ", " + s1.age);  
 System.*out*.println("Student 2: " + s2.name + ", " + s2.age);  
 }  
}

class Student {  
 String name;  
 int age;  
  
 // 默认构造方法  
 Student() {  
 this.name = "Unknown";  
 this.age = 18;  
 }  
  
 // 自定义构造方法  
 Student(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
}

class Student {  
 String name;  
 int age;  
  
 // 默认构造方法  
 Student() {  
 this.name = "Unknown";  
 this.age = 18;  
 }  
  
 // 自定义构造方法  
 Student(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
}

**题目2：构造方法的重载**

定义一个 Book 类，包含以下属性：title（String 类型）author（String 类型）price（double 类型）

编写**三个构造方法**：无参构造方法，初始化 title="未命名"，author="匿名"，price=0.0。接收 title 和 author 的构造方法，price 默认为 29.9。接收 title、author 和 price 的构造方法。在 main 方法中分别使用这三个构造方法创建 Book 对象，并输出它们的属性。

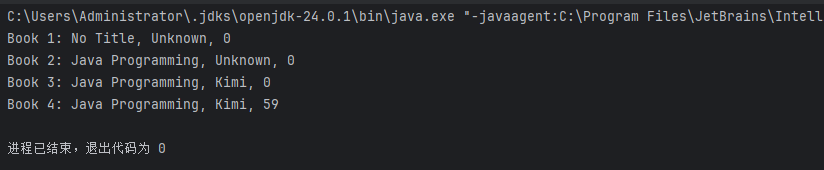
源代码：

public class Main {  
 public static void main(String[] args) {  
 Book b1 = new Book();  
 Book b2 = new Book("Java Programming");  
 Book b3 = new Book("Java Programming", "Kimi");  
 Book b4 = new Book("Java Programming", "Kimi", 59);  
  
 System.*out*.println("Book 1: " + b1.title + ", " + b1.author + ", " + b1.price);  
 System.*out*.println("Book 2: " + b2.title + ", " + b2.author + ", " + b2.price);  
 System.*out*.println("Book 3: " + b3.title + ", " + b3.author + ", " + b3.price);  
 System.*out*.println("Book 4: " + b4.title + ", " + b4.author + ", " + b4.price);  
 }  
}

class Book {  
 String title;  
 String author;  
 int price;  
  
 // 构造方法重载  
 Book() {  
 this.title = "No Title";  
 this.author = "Unknown";  
 this.price = 0;  
 }  
  
 Book(String title) {  
 this.title = title;  
 this.author = "Unknown";  
 this.price = 0;  
 }  
  
 Book(String title, String author) {  
 this.title = title;  
 this.author = author;  
 this.price = 0;  
 }  
  
 Book(String title, String author, int price) {  
 this.title = title;  
 this.author = author;  
 this.price = price;  
 }  
}

class Book {  
 String title;  
 String author;  
 int price;  
  
 // 构造方法重载  
 Book() {  
 this.title = "No Title";  
 this.author = "Unknown";  
 this.price = 0;  
 }  
  
 Book(String title) {  
 this.title = title;  
 this.author = "Unknown";  
 this.price = 0;  
 }  
  
 Book(String title, String author) {  
 this.title = title;  
 this.author = author;  
 this.price = 0;  
 }  
  
 Book(String title, String author, int price) {  
 this.title = title;  
 this.author = author;  
 this.price = price;  
 }  
}

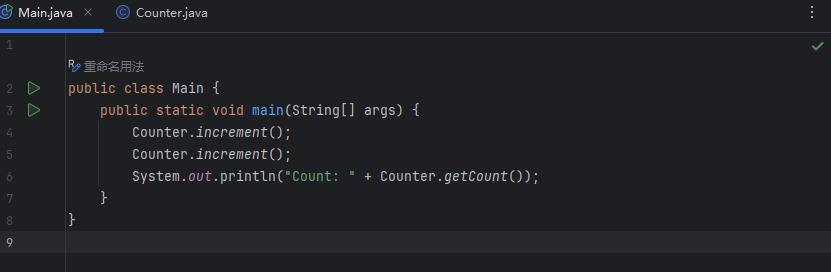
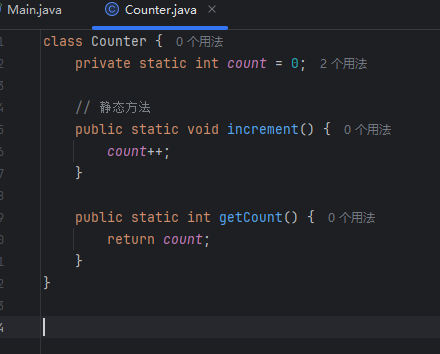
运行结果：



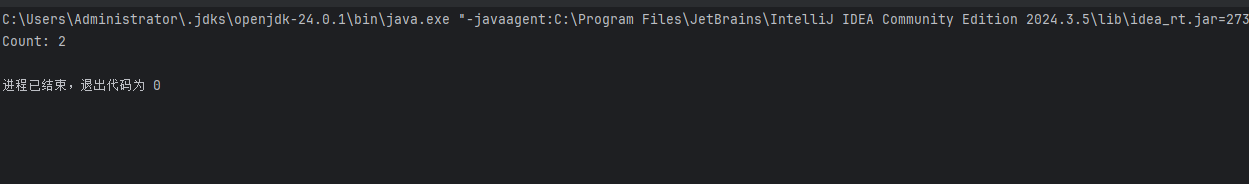
**题目3：构造方法与对象初始化**

定义一个 Car 类，包含以下属性：brand（String 类型）maxSpeed（int 类型）编写一个构造方法，要求：如果传入的 maxSpeed 超过 300，则自动设置为 300（防止不合理速度）。如果 brand 为 null 或空字符串，则设置为 "未知品牌"。在 main 方法中测试以下情况：Car car1 = new Car("奔驰", 350);Car car2 = new Car("", 200);Car car3 = new Car(null, 180);输出各 Car 对象的 brand 和 maxSpeed。

源代码



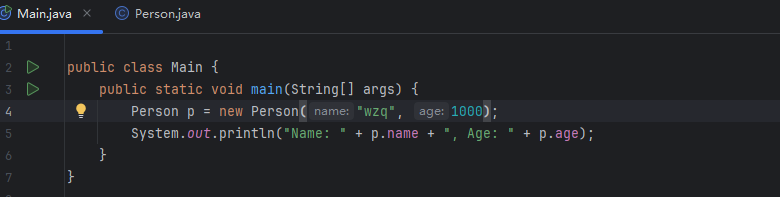
运行结果：



#### ****题目4：访问修饰符的作用范围****

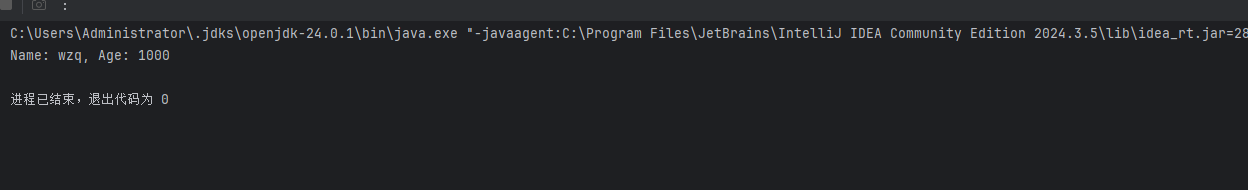
定义一个 Person 类，包含以下成员：public String name;private int age;protected String gender;String address;（默认 default 访问权限）在 Person 类中编写一个 displayInfo() 方法，打印所有属性。创建另一个类 TestPerson，在 main 方法中：实例化 Person 对象，尝试直接访问 name、age、gender、address，观察哪些可以访问，哪些会报错。调用 displayInfo() 方法，验证是否能正确输出所有信息。

源代码：





运行结果：

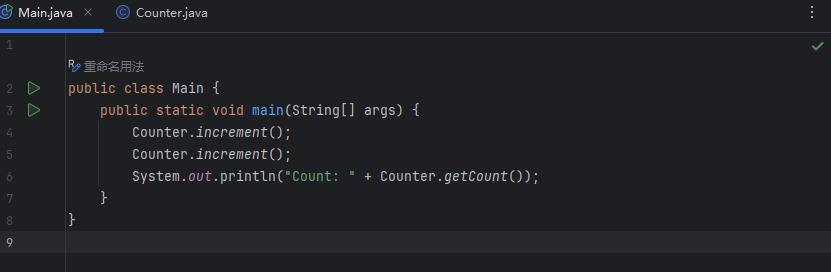
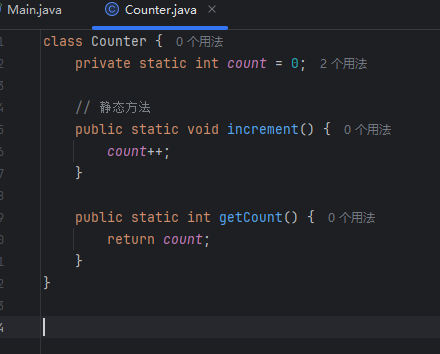


**题目5：static 修饰符的应用**

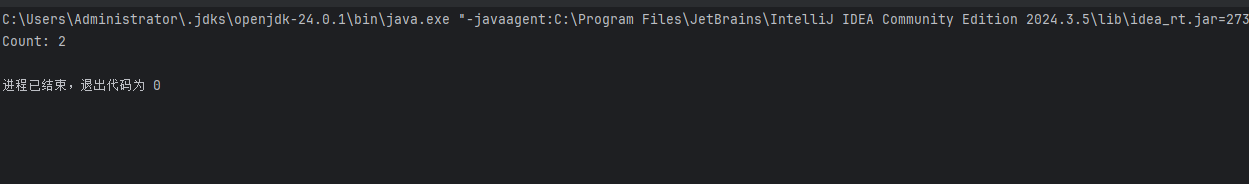
定义一个 Counter 类，包含：private static int count = 0;（静态变量，记录对象数量）public Counter()（构造方法，每次创建对象时 count++）public static int getCount()（静态方法，返回当前 count 值）。

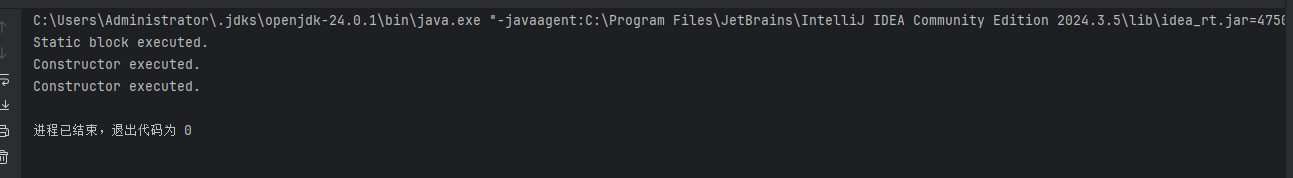
在 main 方法中：创建 Counter 对象 c1、c2、c3。调用 Counter.getCount() 并输出结果，观察 static 变量的共享特性。

源代码



运行结果：



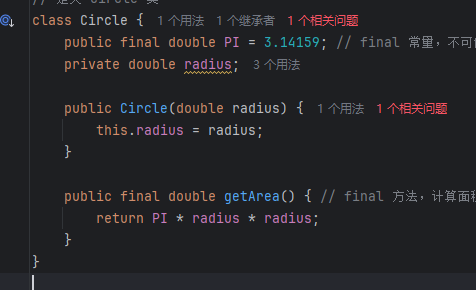
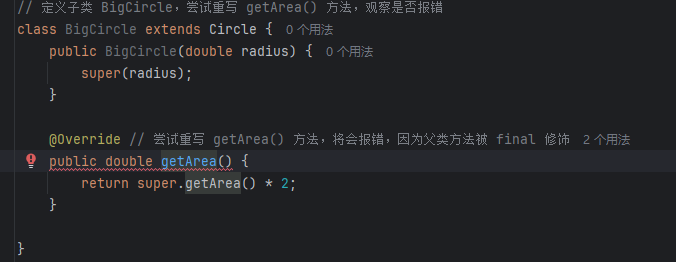


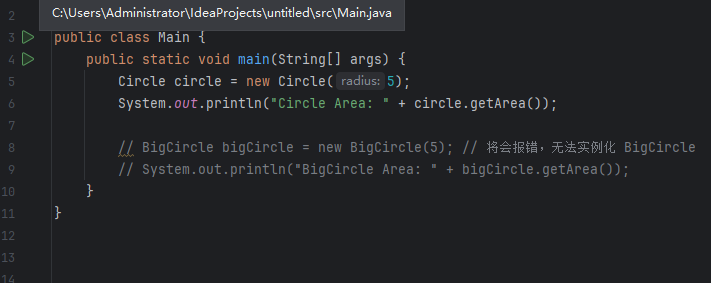
#### ****题目6：final 修饰符的作用****

定义一个 Circle 类，包含：public final double PI = 3.14159;（final 常量，不可修改）private double radius;构造方法，初始化 radius。public final double getArea()（final 方法，计算面积，禁止子类重写）。

尝试在 main 方法中修改 PI 的值，观察是否报错。

定义一个子类 BigCircle，尝试重写 getArea() 方法，观察是否报错。







#### ****题目7：继承的基本用法与方法重写****

定义一个父类 Animal，包含：protected String name;public Animal(String name)（构造方法，初始化 name）public void makeSound()（默认输出 "动物发出声音"）。

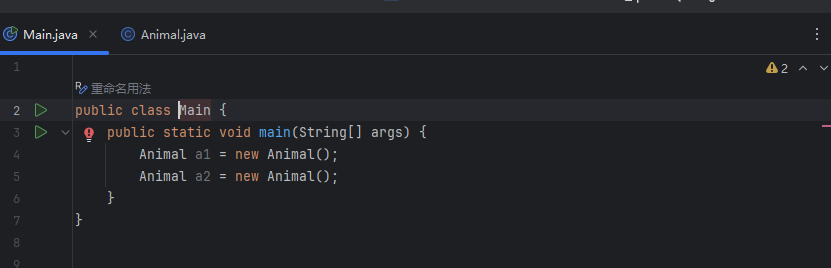
定义子类 Dog 和 Cat，继承 Animal，并重写 makeSound() 方法：

Dog 输出 "汪汪汪！"

 Cat 输出 "喵喵喵！"

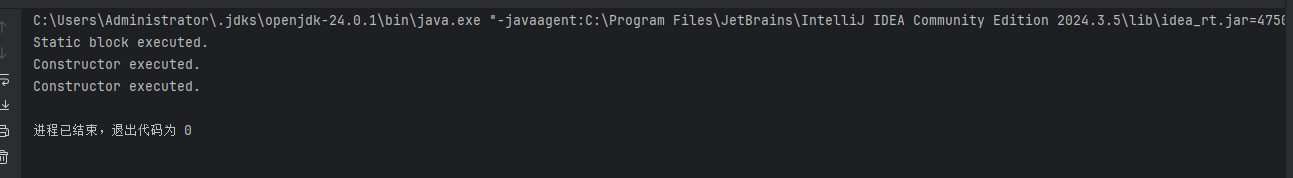
在 main 方法中创建 Dog 和 Cat 对象，调用 makeSound()，观察多态效果。

源代码：





运行结果：



****题目8：super 关键字的使用****

定义一个父类 **Vehicle**，包含：

**protected String brand;**

**public Vehicle(String brand)**（构造方法，初始化 **brand**）

**public void start()**（输出 **"车辆启动"**）

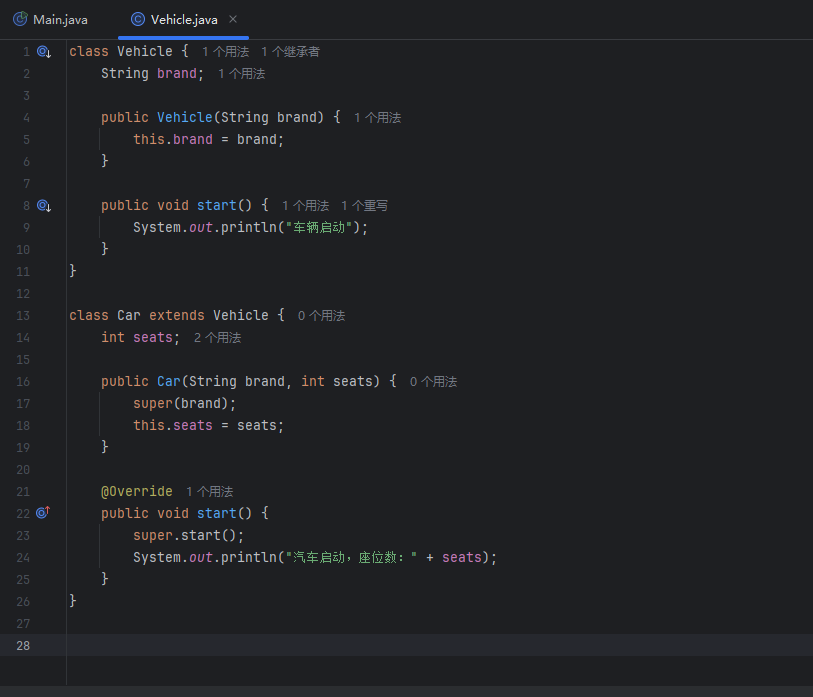
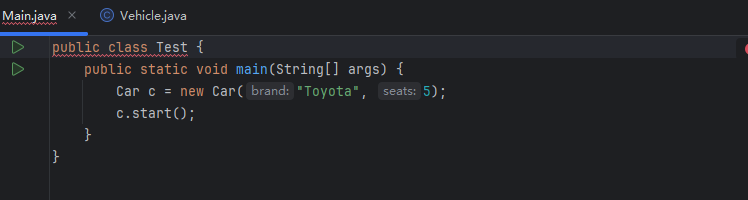
定义子类 **Car**，继承 **Vehicle**，并添加：

**private int seats;**

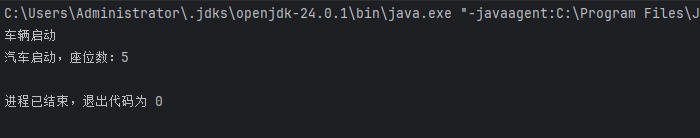
**public Car(String brand, int seats)**（使用 **super** 调用父类构造方法）

**@Override public void start()**（先调用父类 **start()**，再输出 **"汽车座椅数：" + seats**）

在 **main** 方法中创建 **Car** 对象，调用 **start()**，验证 **super** 的作用。



运行结果：



#### ****题目9：继承链与多层 super****

定义三个类形成继承链：

**Person**（父类）：

**protected String name;**

**public Person(String name)**

**public void introduce()**（输出 **"我是：" + name**）

**Student**（子类，继承 **Person**）：

**private String school;**

**public Student(String name, String school)**（使用 **super** 初始化 **name**）

**@Override public void introduce()**（先调用父类 **introduce()**，再输出 **"学校：" + school**）

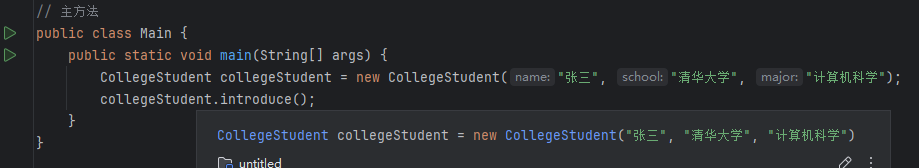
**CollegeStudent**（子类，继承 **Student**）：

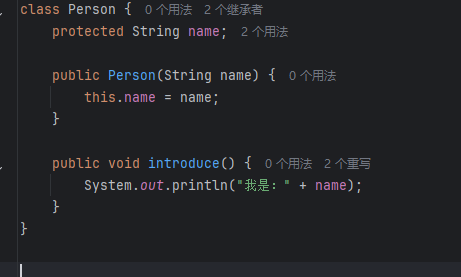
**private String major;**

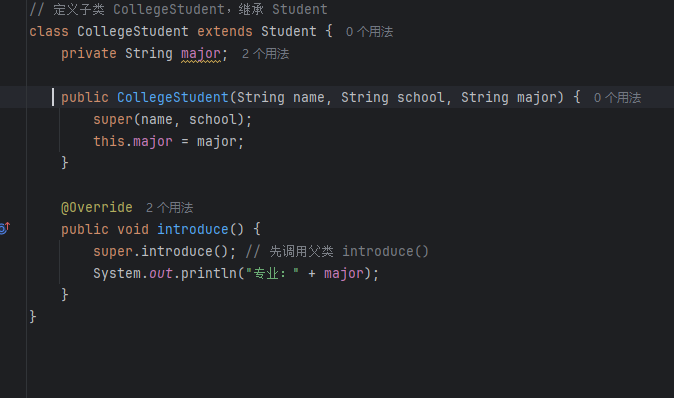
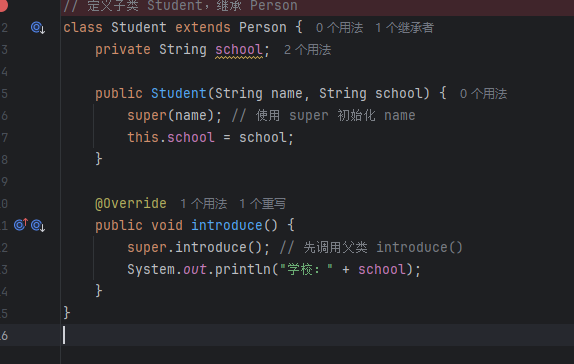
**public CollegeStudent(String name, String school, String major)**

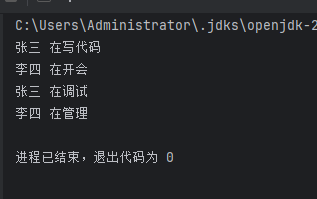
**@Override public void introduce()**（先调用 **super.introduce()**，再输出 **"专业：" + major**）

在 **main** 方法中创建 **CollegeStudent** 对象，调用 **introduce()**，观察多层 **super** 的调用链









#### ****题目10：接口的基本定义与实现****

定义一个接口 **Swimmable**，包含：

**void swim();**（抽象方法）

定义一个接口 **Flyable**，包含：

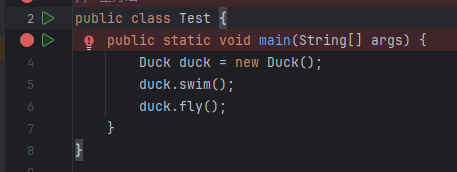
**void fly();**（抽象方法）

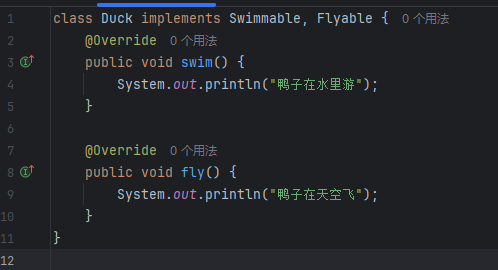
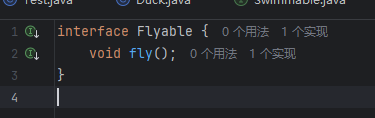
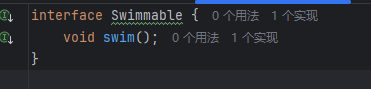
定义一个类 **Duck**，实现 **Swimmable** 和 **Flyable** 接口，并实现它们的方法：

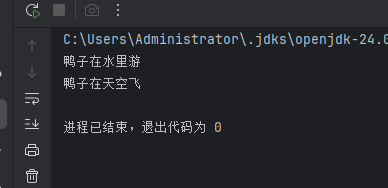
**swim()** 输出 **"鸭子在水里游"**

**fly()** 输出 **"鸭子在天空飞"**

在 **main** 方法中创建 **Duck** 对象，调用 **swim()** 和 **fly()**，验证接口的实现。







#### ****题目11：接口的默认方法与静态方法（Java 8+）****

定义一个接口 **Calculator**，包含：

**int add(int a, int b);**（抽象方法）

**default int subtract(int a, int b)**（默认方法，返回 **a - b**）

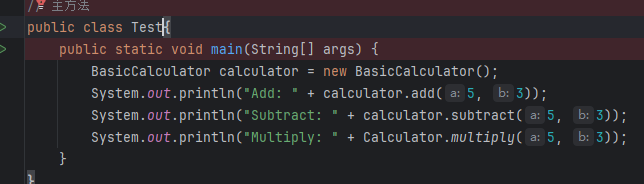
**static int multiply(int a, int b)**（静态方法，返回 **a \* b**）

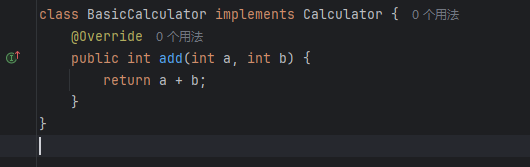
定义一个类 **BasicCalculator**，实现 **Calculator** 接口，仅需实现 **add()** 方法。

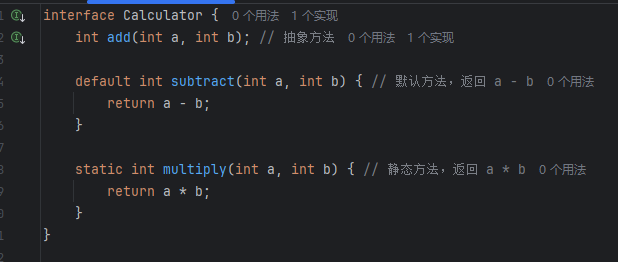
在 **main** 方法中：

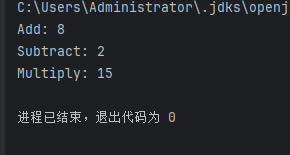
创建 **BasicCalculator** 对象，调用 **add()** 和 **subtract()**。

直接调用 **Calculator.multiply()**，验证静态方法的使用。









#### ****题目12：接口的多继承与 instanceof 检查****

定义两个接口：

**Runnable**（包含 **void run();**）

**Jumpable**（包含 **void jump();**）

定义一个抽象类 **Animal**，包含 **abstract void eat();**。

定义一个类 **Rabbit**，继承 **Animal** 并实现 **Runnable** 和 **Jumpable**：

**run()** 输出 **"兔子奔跑"**

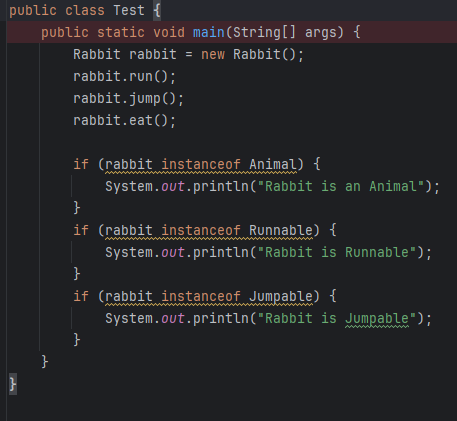
**jump()** 输出 **"兔子跳跃"**

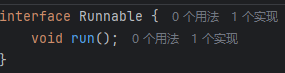
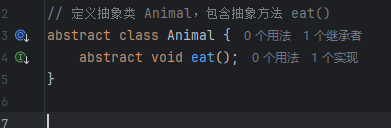
**eat()** 输出 **"兔子吃胡萝卜"**

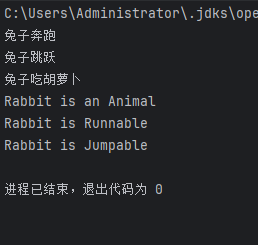
在 **main** 方法中：

创建 **Rabbit** 对象，调用所有方法。

使用 **instanceof** 检查该对象是否属于 **Animal**、**Runnable**、**Jumpable** 类型。







#### ****题目13：方法重写与动态绑定****

定义一个父类 **Shape**，包含：

**public void draw()**（默认输出 **"绘制图形"**）

定义两个子类 **Circle** 和 **Rectangle**，分别重写 **draw()** 方法：

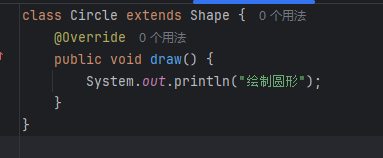
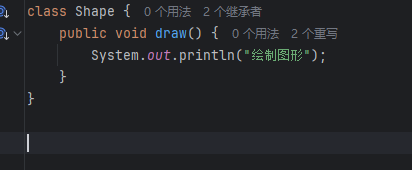
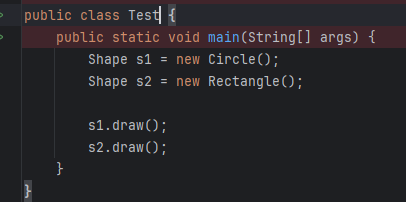
**Circle** 输出 **"绘制圆形"**

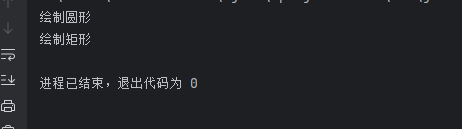
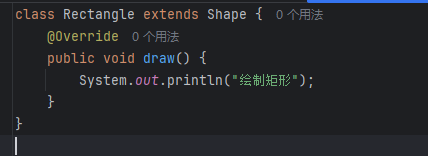
**Rectangle** 输出 **"绘制矩形"**

在 **main** 方法中：

创建 **Shape s1 = new Circle();** 和 **Shape s2 = new Rectangle();**（向上转型）

调用 **s1.draw()** 和 **s2.draw()**，观察输出结果。





#### ****题目14：多态在方法参数中的应用****

定义一个父类 **Animal**，包含：

**public void makeSound()**（默认输出 **"动物发出声音"**）

定义两个子类 **Dog** 和 **Cat**，分别重写 **makeSound()** 方法：

**Dog** → **"汪汪汪！"**

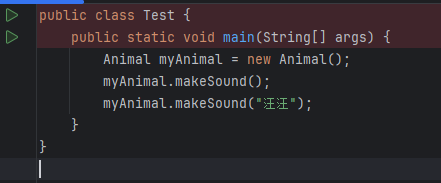
**Cat** → **"喵喵喵！"**

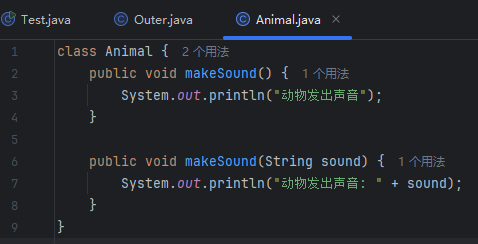
定义一个工具类 **AnimalUtils**，包含静态方法：

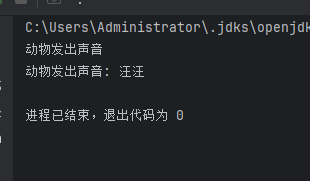
**public static void playWithAnimal(Animal animal)**（调用 **animal.makeSound()**）

在 **main** 方法中：

创建 **Dog** 和 **Cat** 对象，分别传递给 **playWithAnimal()** 方法，观察输出。







#### ****题目15：多态与类型转换****

定义一个父类 **Employee**，包含：

**protected String name;**

**public Employee(String name)**

**public void work()**（默认输出 **name + "在工作"**）

定义两个子类：

**Programmer**：重写 **work()**，输出 **name + "在写代码"**，新增方法 **debug()**。

**Manager**：重写 **work()**，输出 **name + "在开会"**，新增方法 **manage()**。

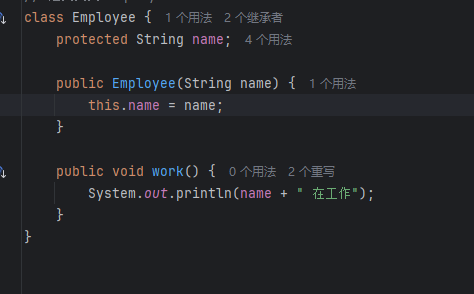
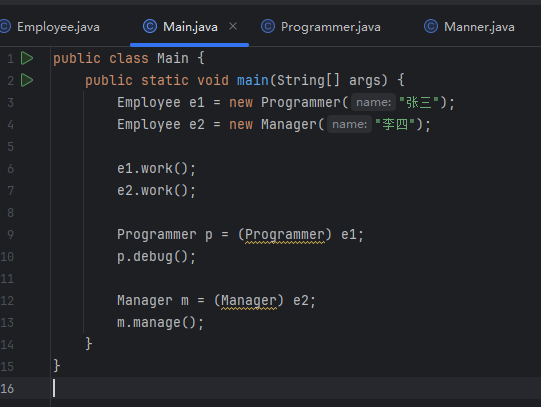
在 **main** 方法中：

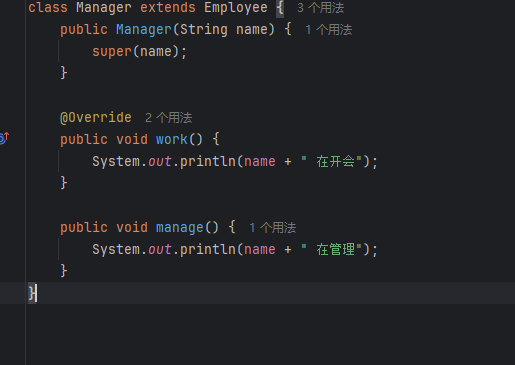
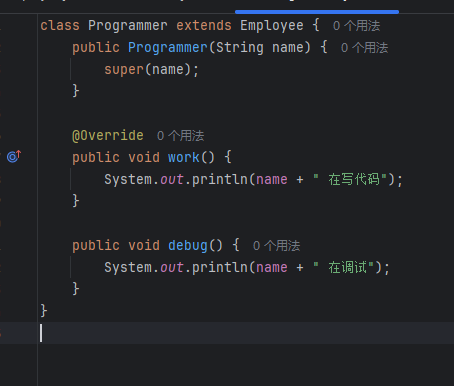
创建 **Employee e1 = new Programmer("张三");** 和 **Employee e2 = new Manager("李四");**

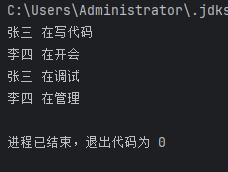
调用 **e1.work()** 和 **e2.work()**（多态）。

尝试将 **e1** 向下转型为 **Programmer**，调用 **debug()**；将 **e2** 转型为 **Manager**，调用 **manage()**。

错误示范：尝试将 **e1** 转型为 **Manager**，捕获 **ClassCastException**。







#### ****题目16：成员内部类（Member Inner Class）****

定义一个外部类 **Outer**，包含：

私有字段 **private String outerField = "外部类字段"**

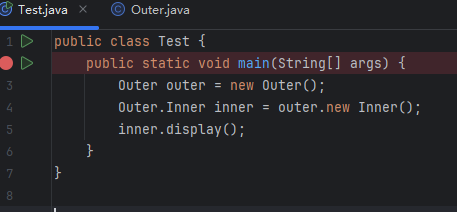
定义一个**成员内部类** **Inner**，包含方法 **void print()**，打印 **outerField**

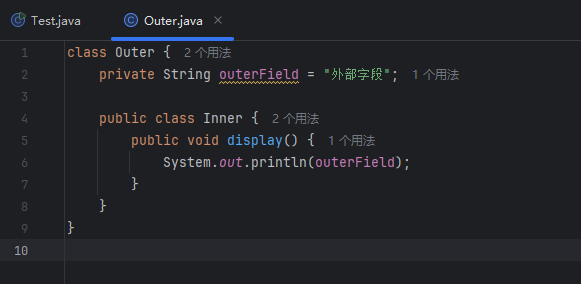
在 **Outer** 中编写方法 **void useInner()**，创建 **Inner** 对象并调用 **print()**

在 **main** 方法中：

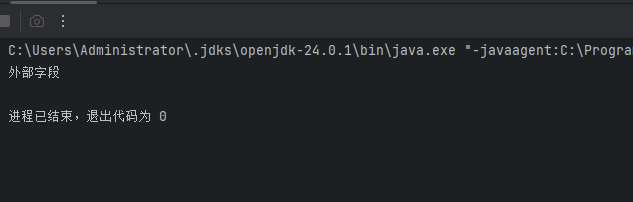
创建 **Outer** 对象，调用 **useInner()**

尝试直接创建 **Inner** 对象（**Outer.Inner inner = new Outer().new Inner()**），调用 **print()**





运行结果：



#### ****题目17：静态内部类（Static Nested Class）****

定义一个外部类 **Computer**，包含：

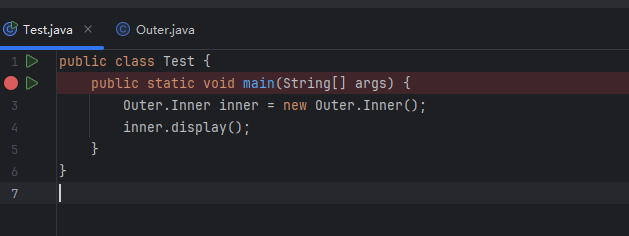
静态字段 **static String brand = "Apple"**

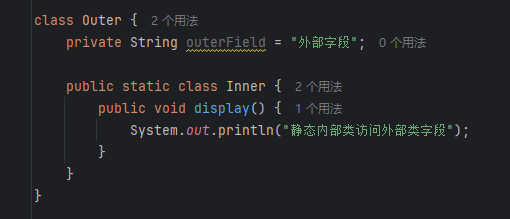
**静态内部类** **CPU**，包含方法 **void printBrand()**，打印 **brand**

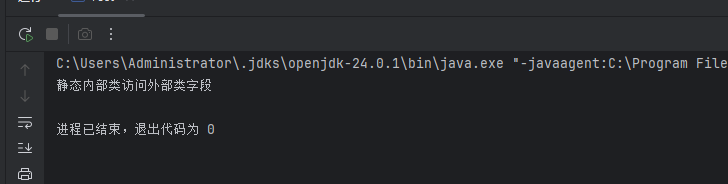
在 **main** 方法中：

直接创建 **Computer.CPU** 对象（无需外部类实例），调用 **printBrand()**

修改 **Computer.brand** 为 **"Intel"**，再次调用 **printBrand()**，观察变化

源代码：



运行结果：  


#### ****题目18：局部内部类（Local Inner Class）****

定义一个外部类 **Outer**，包含方法 **void createLocalClass()**，在该方法内：

定义一个**局部内部类** **LocalInner**，包含方法 **void print()**

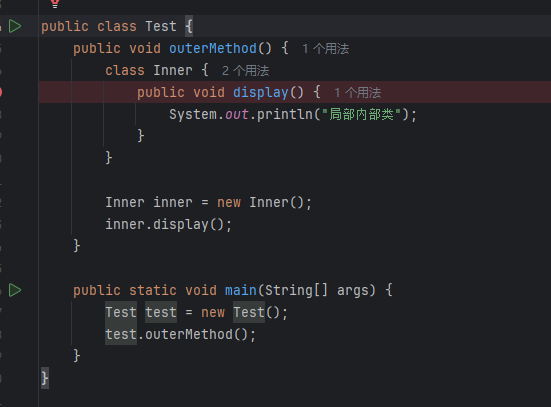
**print()** 方法打印外部类字段 **outerField** 和方法的局部变量 **localVar**（需为 **final** 或等效 **final**）

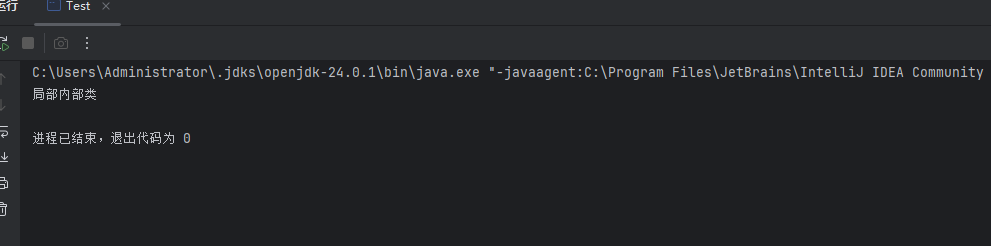
在 **createLocalClass()** 中创建 **LocalInner** 对象并调用 **print()**

外部类字段：**private String outerField = "外部类字段"**

在 **main** 方法中：

创建 **Outer** 对象，调用 **createLocalClass()**

源代码



运行结果：