****

****

**信息学院软件工程系**

**《JAVA程序设计》实验报告**

实验七

**姓名：任宇**

**学号：33920212204567**

**学院：信息学院**

**专业：软件工程**

**完成时间：2024-04-12**

1. **实验目的及要求**

* 熟悉异常处理
* 熟悉泛型方法和泛型类

1. **实验题目及实现过程**
2. 题目一：

完善LAB6，用异常处理语句处理输入类型不匹配、数据转换类型异常、创建对象时初始值不合法等问题。

1. 实验环境

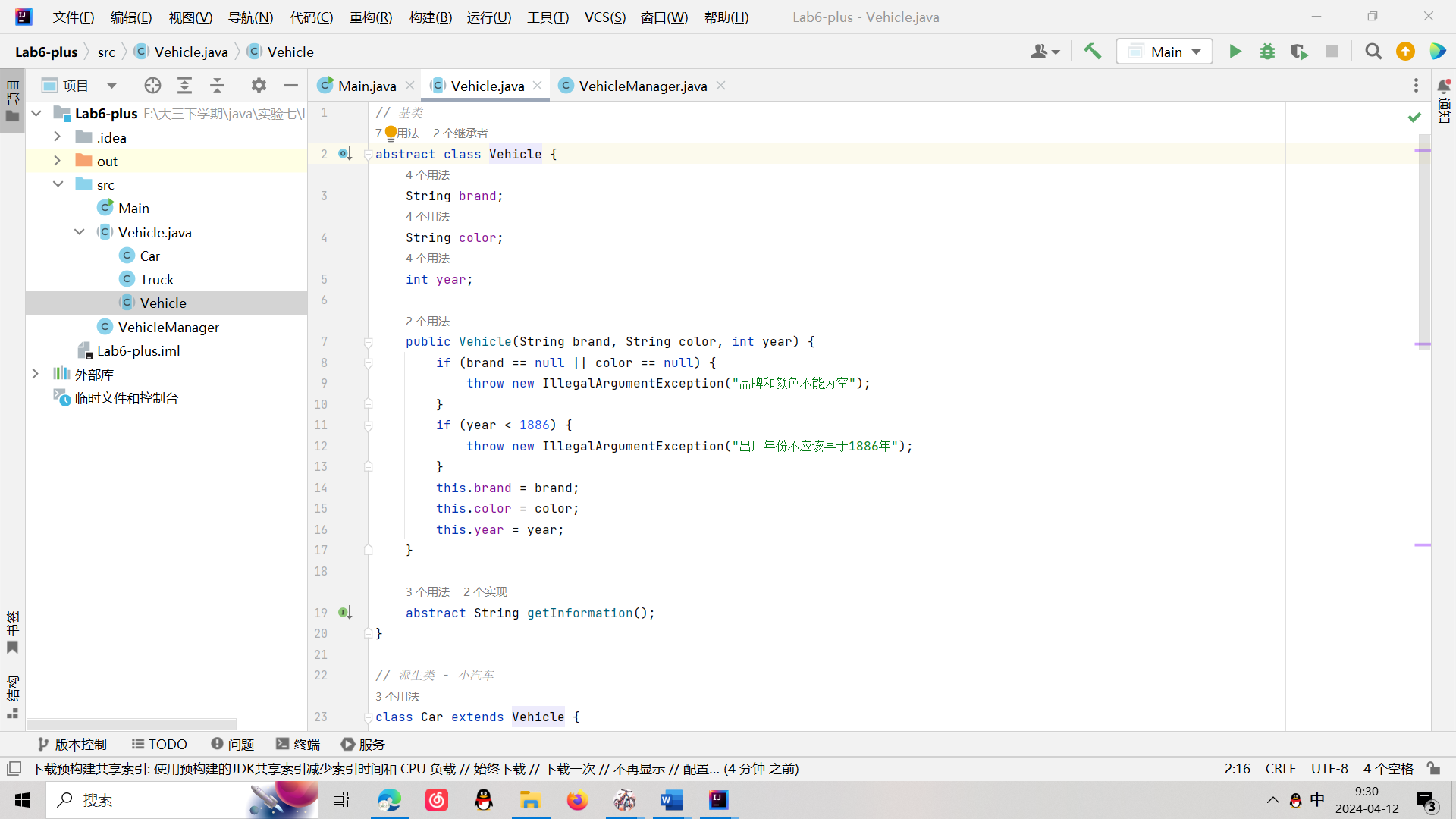
集成开发环境：IDEA Community Edition 2022.3.2

JDK版本：JDK17

1. 实现过程

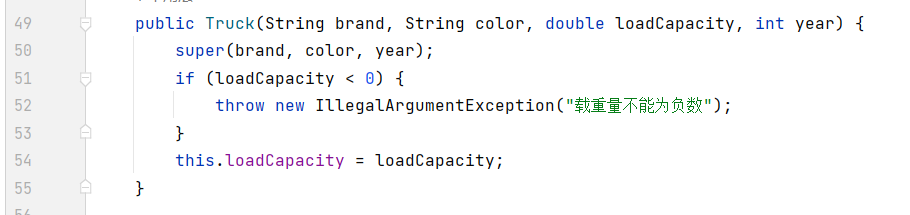
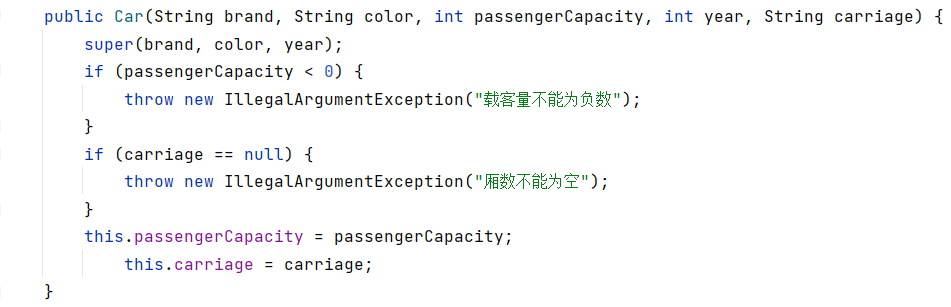
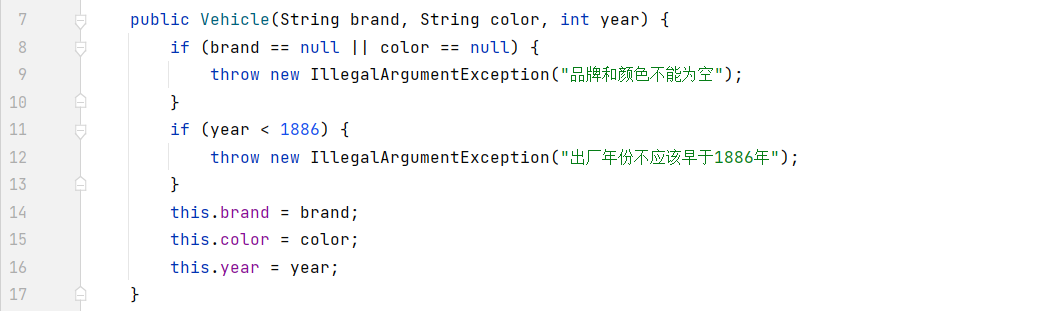
在实验六的基础上，对于Vehicle、Truck以及Car的构造函数进行了异常检查，如果Brand和Color属性为空，抛出异常；如果生厂年份早于1886，抛出异常；如果载客量和载重量为负数，抛出异常；如果厢数为空，抛出异常。接着对于Main类中的addVehicle方法进行了完善，有三种异常情况：输入格式不正确、数据转换类型异常以及交通工具种类不正确。在发现异常后会提示相关信息，并告诉用户继续输入。

1. 过程截图
2. 全屏截图：

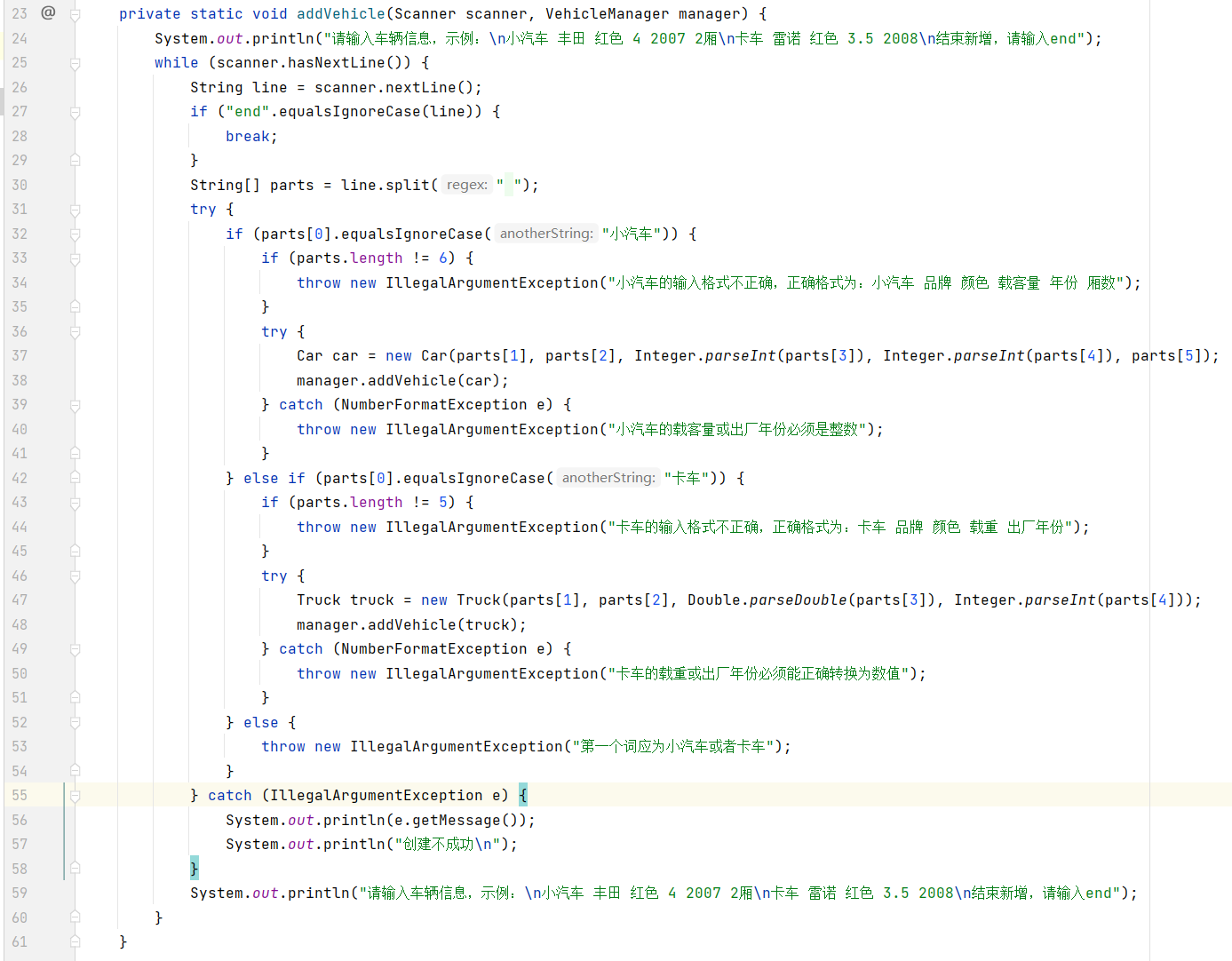


1. 主要代码：

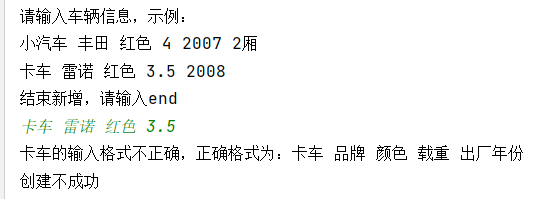
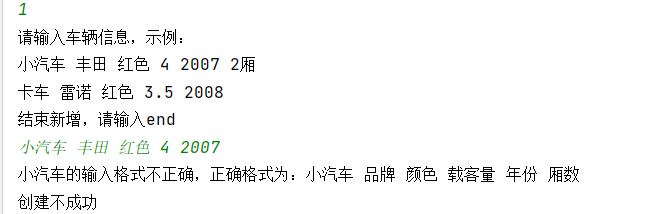
对于构造函数的检查：

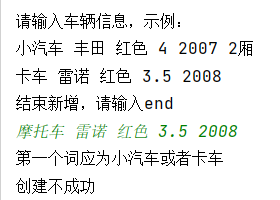
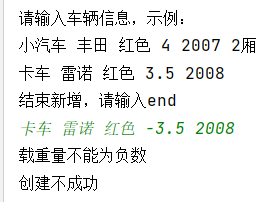
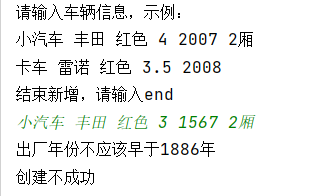
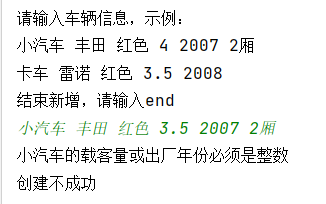


addVehicle方法的完善：



1. 运行结果：





1. 题目二：

编写一个程序，给出一个构造方法，它将关于构造方法失败的信息传递给一个异常处理器。定义一个 SomeClass 类，它在构造方法中抛出异常。程序应创建一个 SomeClass 型的对象，并捕获由这个构造方法抛出的异常。

1. 实验环境

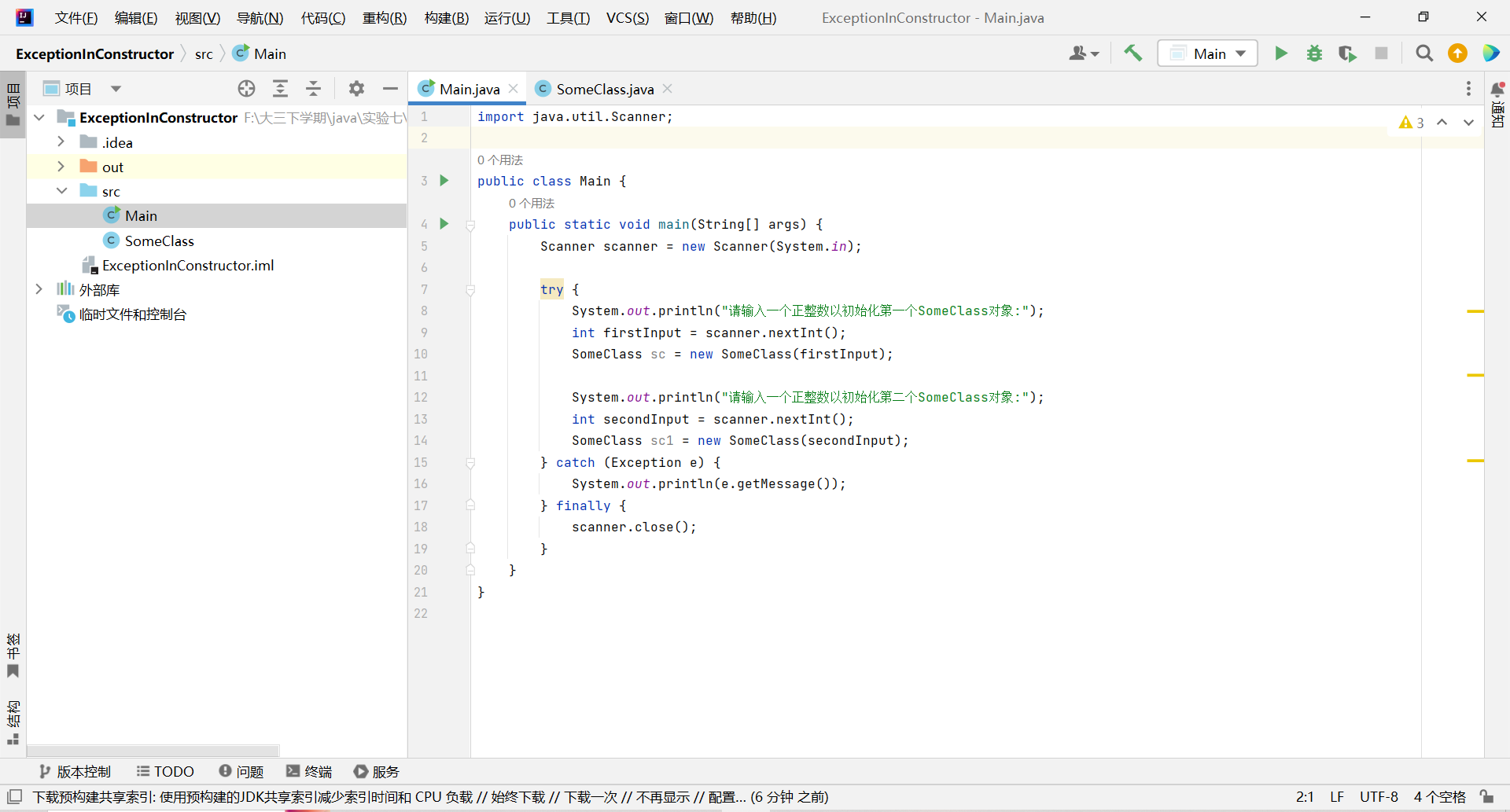
集成开发环境：IDEA Community Edition 2022.3.2

JDK版本：JDK17

1. 实现过程

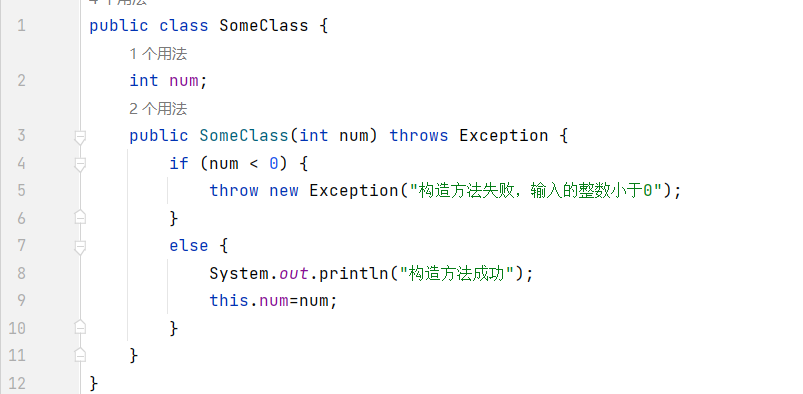
在这个程序中，我设计了两个类：SomeClass和Main。SomeClass类是为了展示如何在构造方法中抛出异常并处理异常。它尝试根据输入的整数来初始化对象，并在输入值不符合预期（即小于0）时抛出异常。在Main类的main方法中，首先通过Scanner类读取用户输入。然后，使用这些输入值尝试创建 SomeClass类的实例。如果输入值不合法（如小于0），SomeClass的构造方法将抛出异常，这些异常会在 Main 类中的try-catch结构被捕获和处理。最后，无论是否发生异常，finally块确保Scanner对象被关闭，以释放资源。

1. 过程截图
2. 全屏截图：



1. 主要代码：

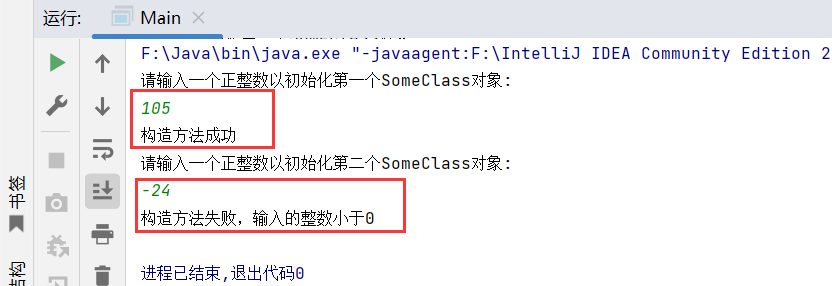
SomeClass类实现：



Main类实现：



1. 运行结果：



1. 题目三：

编写一个演示重抛异常的程序。定义两个方法 someMethod 和 someMethod2, someMethod2 方法的功能就是抛出一个异常。someMethod 方法调用 someMethod2，捕获一个异常并重抛它。用 main 方法调用 someMethod 方法，并捕获被重抛的异常。输出这个异常的栈踪迹。

1. 实验环境

集成开发环境：IDEA Community Edition 2022.3.2

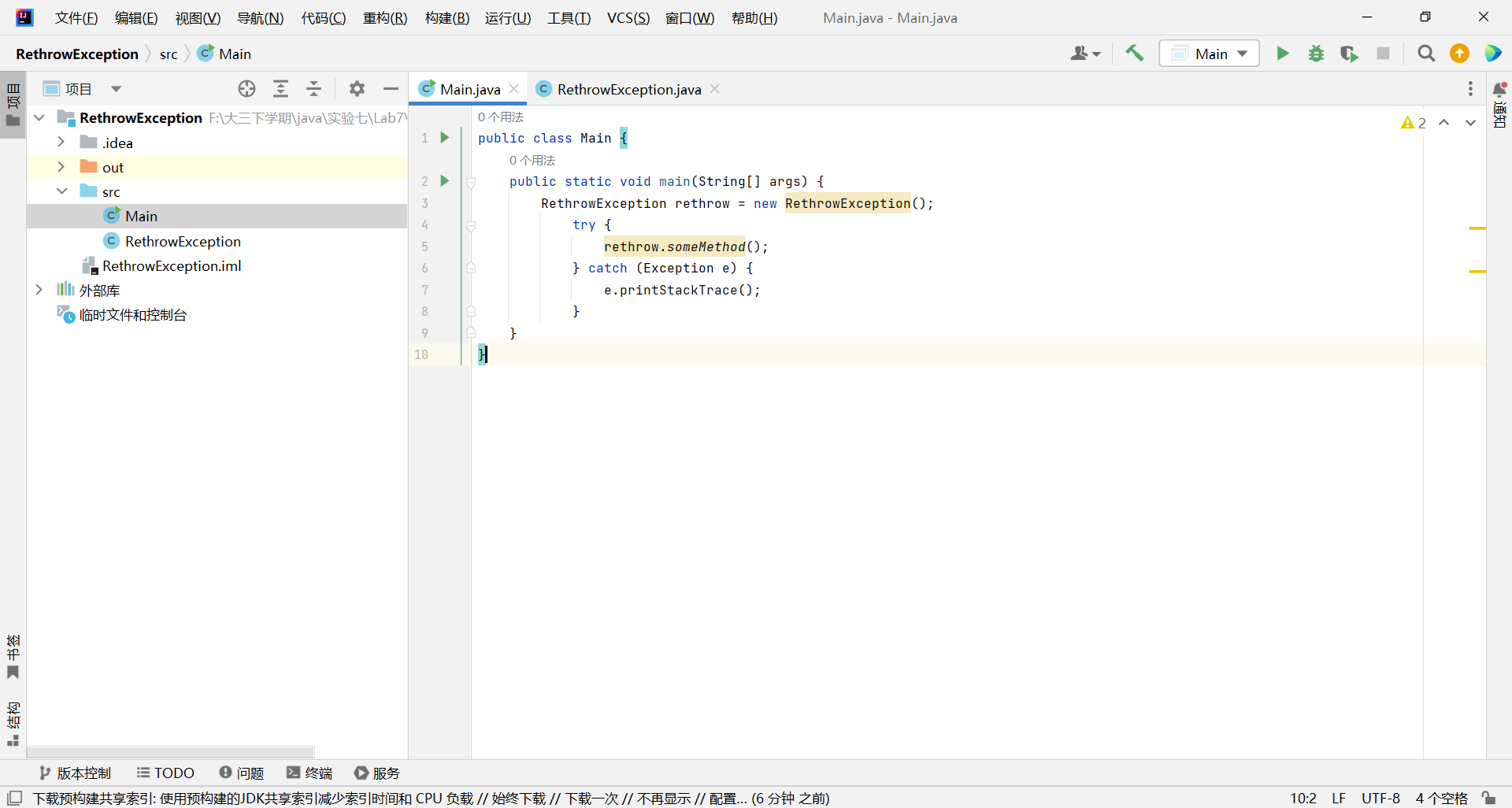
JDK版本：JDK17

1. 实现过程

在这个程序中，我设计了两个类：RethrowException和Main。RethrowException 类用来实现异常的重抛操作。它包含了两个静态方法，其中someMethod2直接抛出异常，而someMethod则调用someMethod2，捕获并重抛一个新的异常。对于someMethod()方法，它方法调用someMethod2，并用try-catch结构来捕获从someMethod2抛出的异常。在catch块中，它创建并抛出一个新的异常对象，并将原始异常作为原因传递给新异常，这是通过在新异常的构造函数中传递捕获的异常来实现的。

对于Main类的main方法，它创建一个RethrowException对象，然后尝试调用someMethod。当someMethod抛出异常时，main方法将捕获这个异常并调用e.printStackTrace()来打印异常的堆栈跟踪。这不仅显示异常消息，还显示异常的来源，包括原始异常信息。

1. 过程截图
2. 全屏截图：

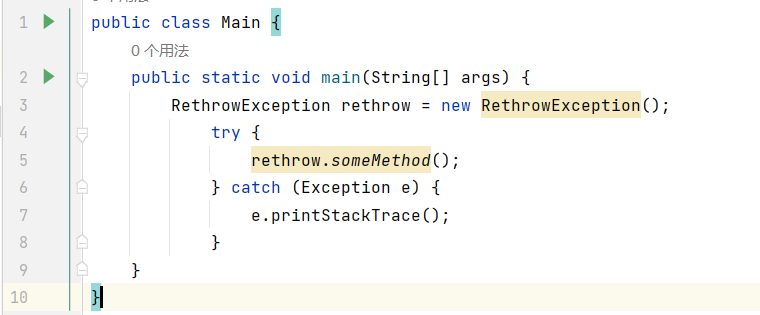


1. 主要代码：

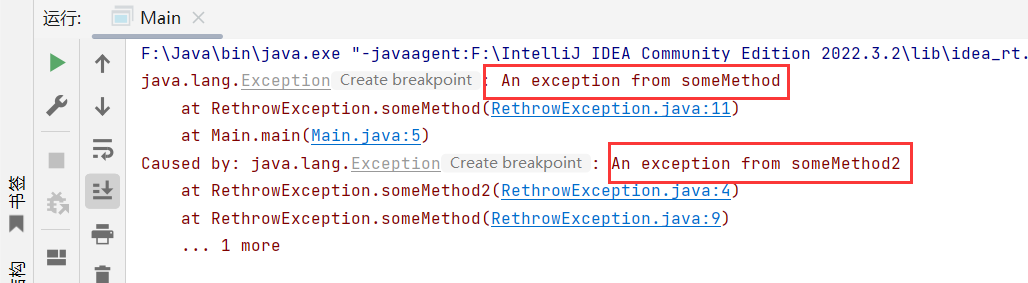
RethrowException类：



Main类：



1. 运行结果：



1. 题目四：

1) 自定义两个异常类：非法姓名异常IllegaNameException和非法地址异常IllegalAddressException。

(2) 定义Student类包含name和address属性，和setName、setAddress方法，当姓名长度小于1或者大于5时抛出IllegaNameException，当地址中不含有“省”或者“市”关键字时抛出IllegalAddressException。

(3) 编写程序抛出这两种异常，在main方法中进行捕获并合理地处理。

1. 实验环境

集成开发环境：IDEA Community Edition 2022.3.2

JDK版本：JDK17

1. 实现过程

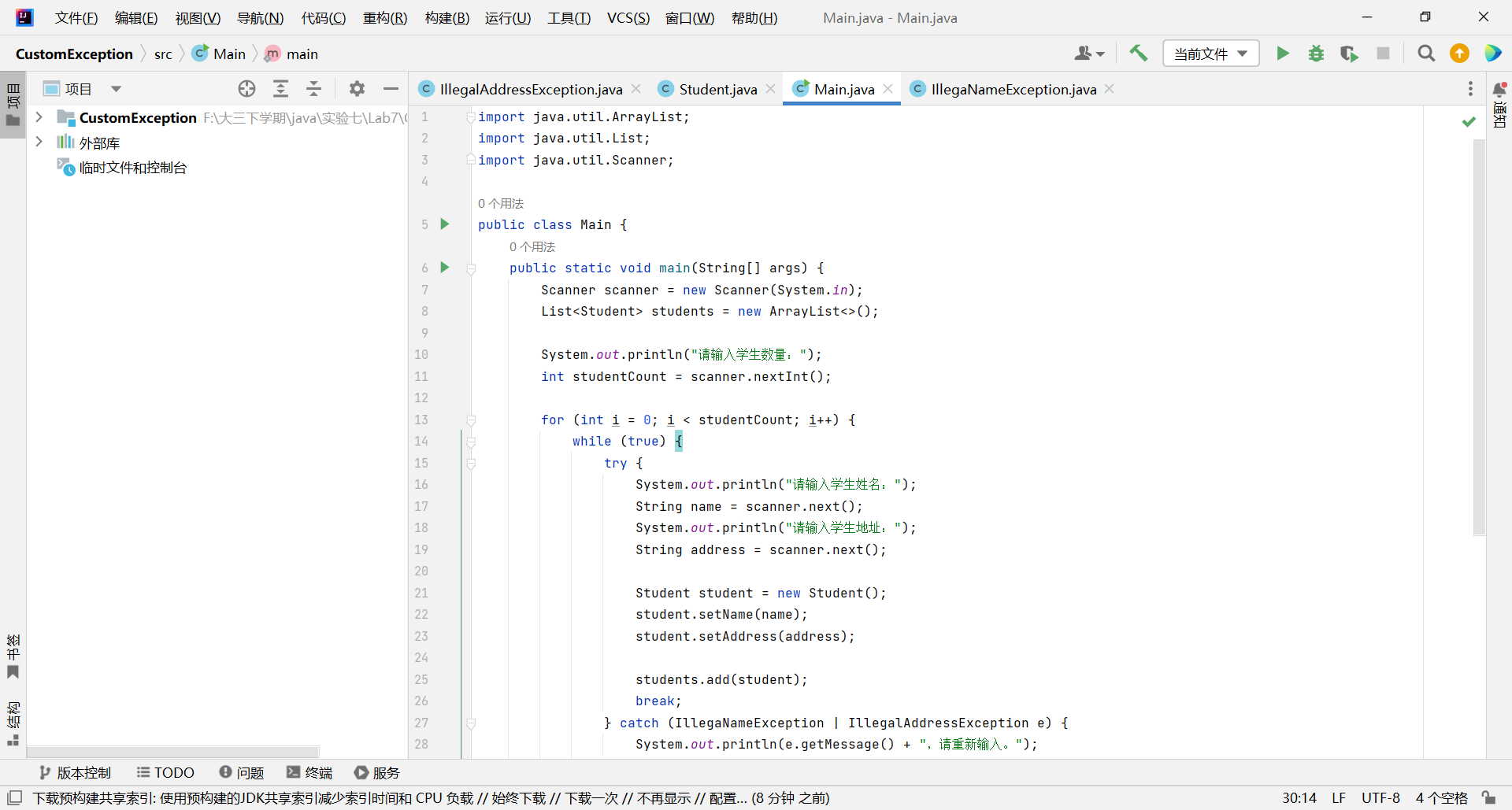
在这个Java程序中，我定义了两个自定义异常类和一个学生类，并在主方法中实现了对这些类的使用和异常处理。

对于IllegalAddressException类和IllegaNameException类，它们分别用于处理特定的数据验证错误：IllegalAddressException处理地址验证失败，而IllegaNameException处理姓名验证失败。这两个类都有一个接受错误消息字符串 message 作为参数的构造方法。这个字符串用来在异常被捕获时提供具体的错误信息。

对于Student类，setName方法设置学生的姓名，如果姓名为 null 或长度小于2或大于5，将抛出IllegaNameException。setAddress方法用于设置学生的地址，如果地址不包含“省”或“市”中的任何一个，则抛出IllegalAddressException。

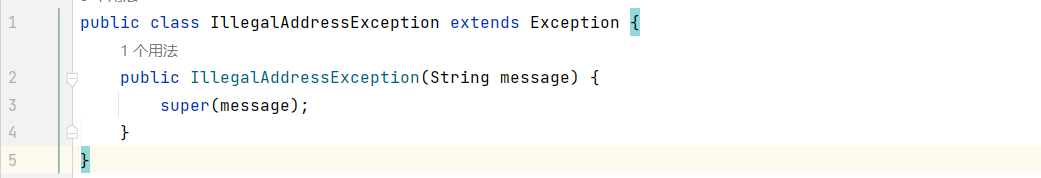
Main类中的main方法，是程序的入口点。该类负责用户交互、创建学生对象、设置学生的属性，并捕获及处理异常。在 main 方法中，通过 Scanner 获取用户输入后，尝试创建和配置 Student 对象。如果 setName 或 setAddress方法抛出异常，相应的异常被捕获并处理，用户需要重新输入直至成功。验证成功的学生信息会被存储在ArrayList中并最终被遍历打印出来。

1. 过程截图
2. 全屏截图：

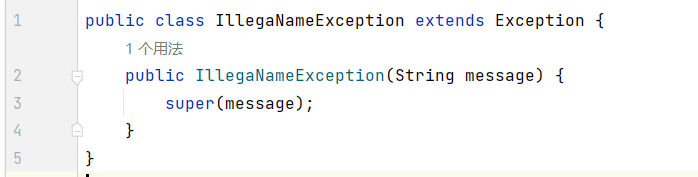


1. 主要代码：

IllegalAddressException类实现：



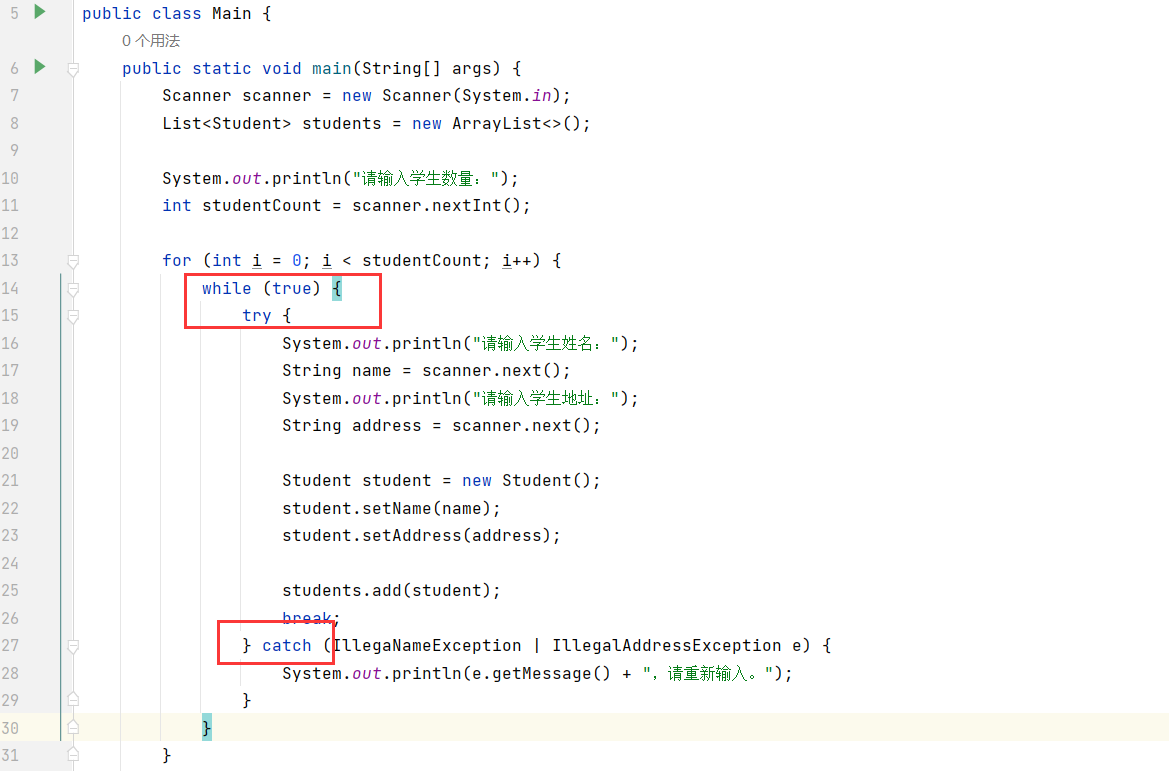
IllegaNameException类实现：



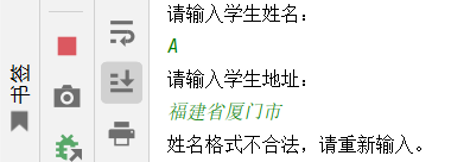
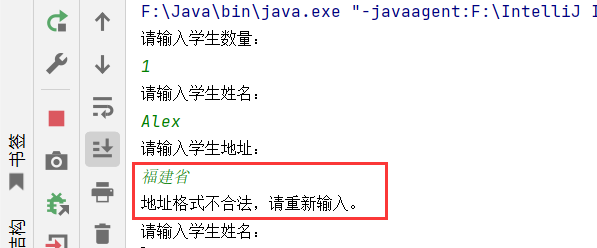
Student类实现：

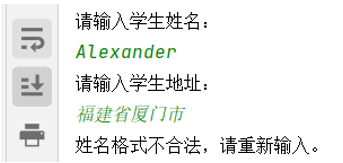
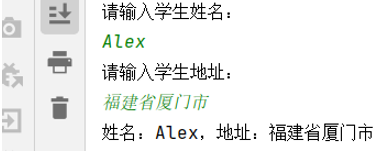


Main类实现：



1. 运行结果：



1. 题目五：

编写 isEqualTo 方法的一个简单泛型版本。它用 equals 方法比较两个实参相等时返回 true，否则返回 false。利用这个泛型方法，在程序中调用 isEqualTo 处理各种内置的类型，例如 Object 或 Integer。运行程序时，传递给 isEqualTo 方法的对象会根据它们的内容或者所引用的对象进行比较吗？

1. 实验环境

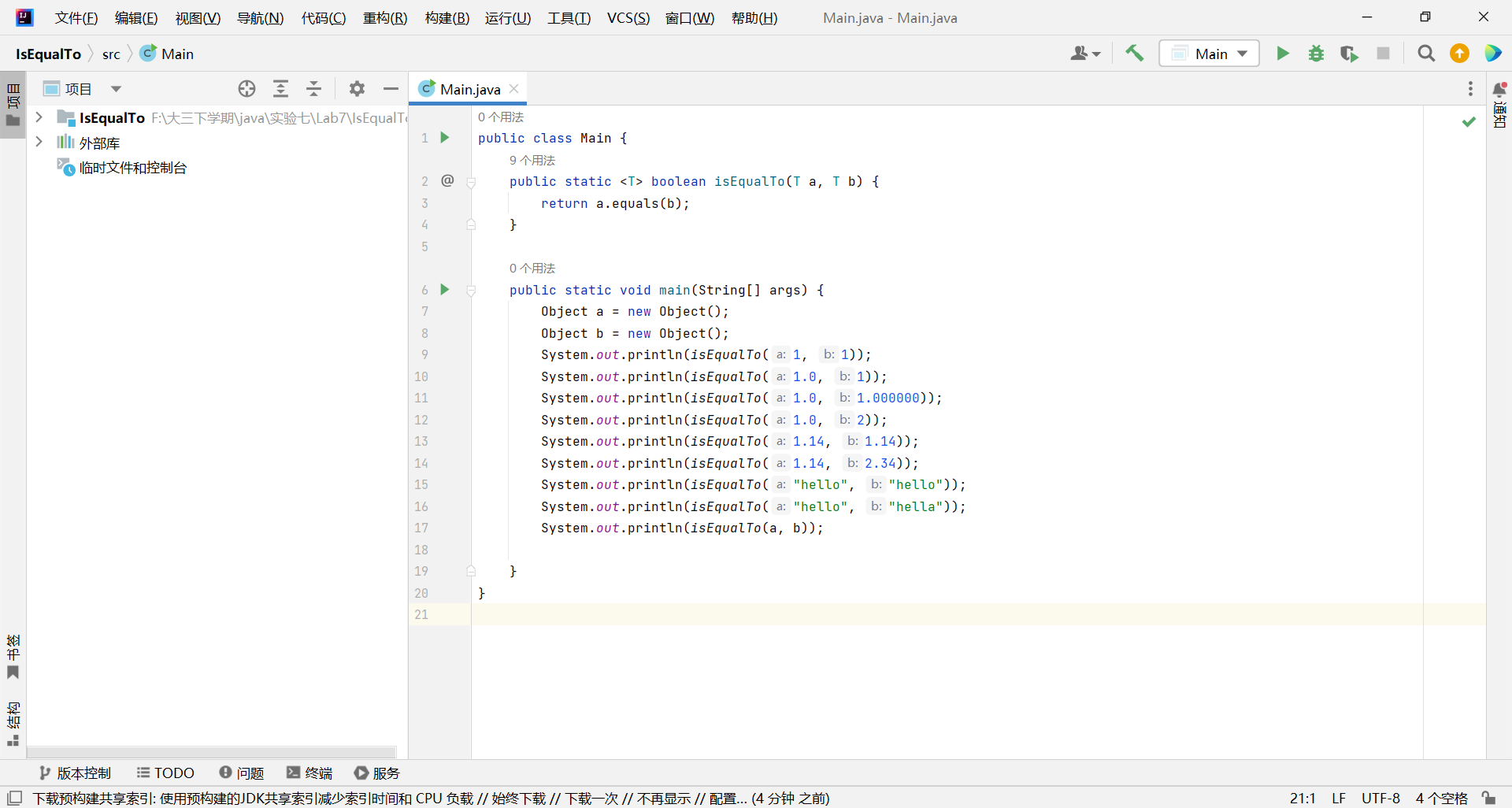
集成开发环境：IDEA Community Edition 2022.3.2

JDK版本：JDK17

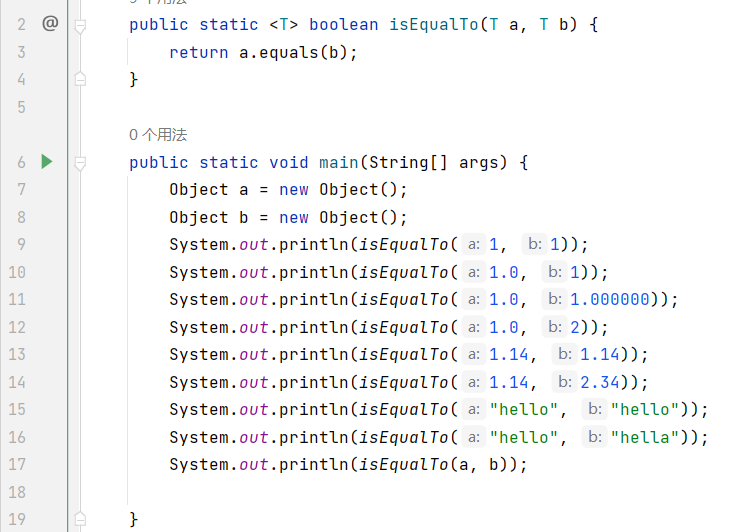
1. 实现过程

在程序中，我定义了一个isEqualTo方法的简单泛型版本，定义为<T> boolean isEqualTo(T a, T b)。它接受两个类型为T的参数a和b，并使用equals方法来比较这两个参数。如果它们相等，则返回true，否则返回false。当T是一个原生数据类型的包装类（如Integer、Double等）时，equals方法比较的是对象的值而非引用。当T是Object类型时，如果没有重写equals方法，默认的实现是比较对象的引用。

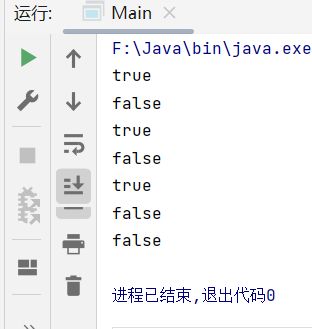
1. 过程截图
2. 全屏截图：



1. 主要代码：



1. 运行结果：



结果正确，分析一下为什么1.0和1比较返回false。在这个调用中，1.0是一个double类型的字面量，而1是一个int类型的字面量。由于isEqualTo方法是泛型的，Java编译器会尝试找到这两个参数的一个共同基类型，以便泛型T可以被正确推断。在这种情况下，double和int都可以被装箱为它们的包装类Double和Integer，但是由于它们属于不同的类型，Java编译器无法直接将这两种类型视为相同的泛型T类型。因此，1.0被装箱成一个Double对象，1 被装箱成一个Integer对象。isEqualTo方法中使用的equals方法定义在Object类中。调用equals方法时，equals方法通常首先检查两个对象是否属于相同的类。既然Double和Integer是不同的类，a.equals(b)就会返回 false。

1. 题目六：

编写一个泛型类 Pair，它有两个类型参数 F 和 S，分别代表一对值中第一个元素第二个元素的类型。为第一个元素和第二个元素添加 get 方法和 set 方法。

1. 实验环境

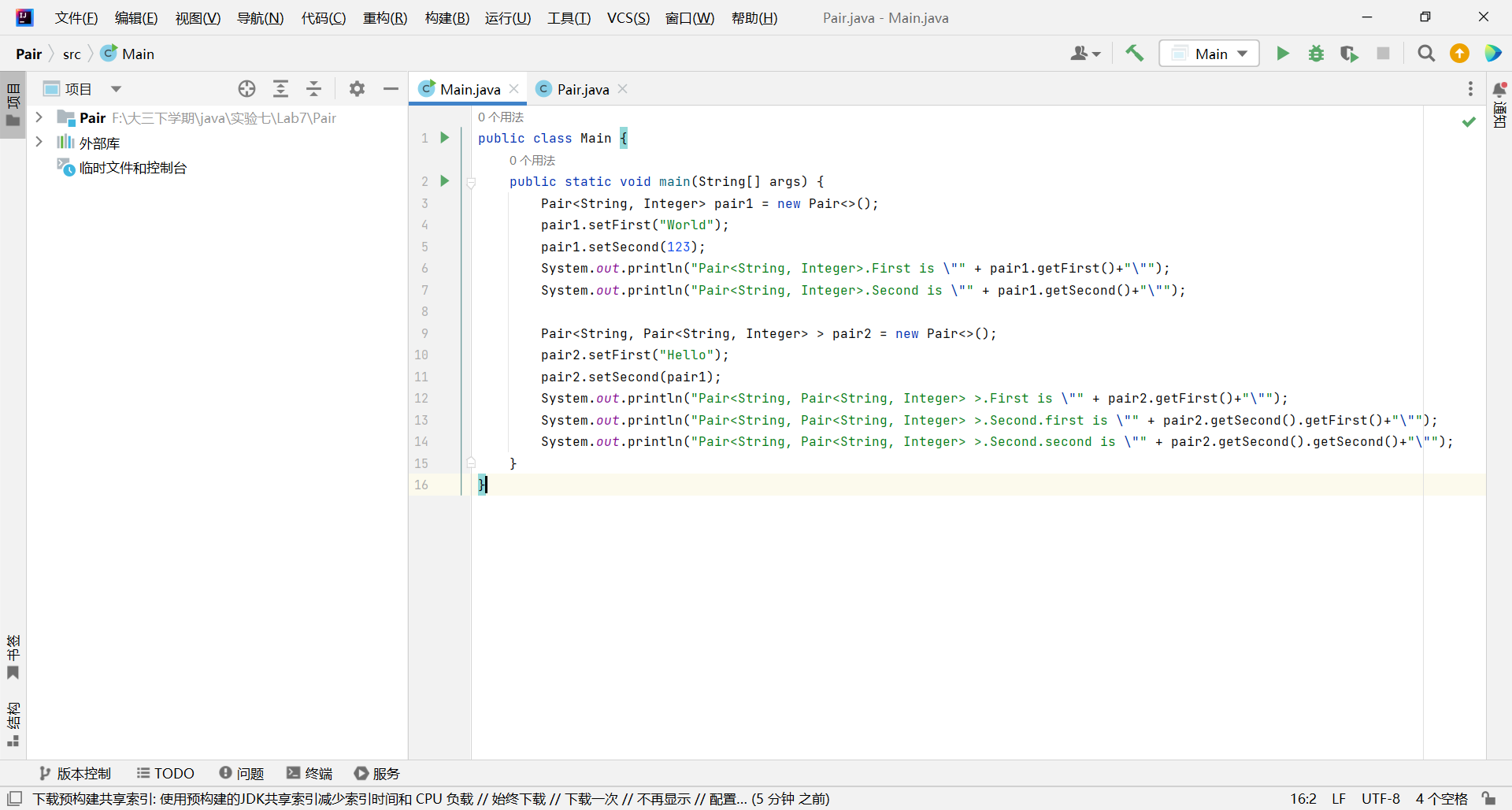
集成开发环境：IDEA Community Edition 2022.3.2

JDK版本：JDK17

1. 实现过程

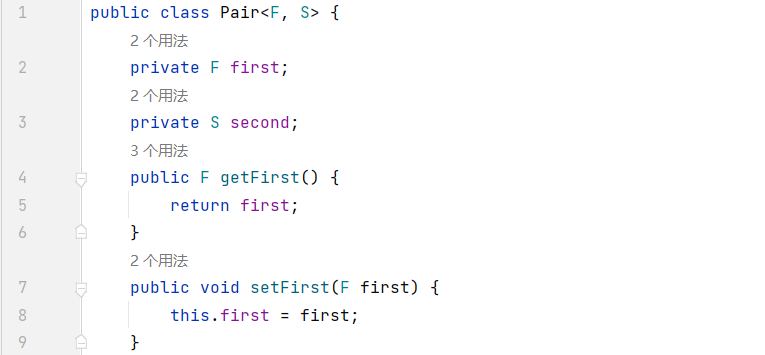
按照题目要求，定义泛型类Pair<F, S>，它用来存储和管理一对相关联的数据项。这对数据项可以是任意类型，由泛型参数F和S表示，分别代表一对中的第一个元素和第二个元素。在Main类的main方法中，我创建了两个 Pair 类的实例。pair1存储了一个String类型和一个Integer类型的对象。pair2存储了一个String类型的对象和另一个Pair<String, Integer>类型的对象作为第二个元素。首先展示为pair1设置第一和第二元素的值。接着演示如何将另一个Pair对象作为元素存储在Pair对象中，这体现了泛型的嵌套使用。

1. 过程截图
2. 全屏截图：

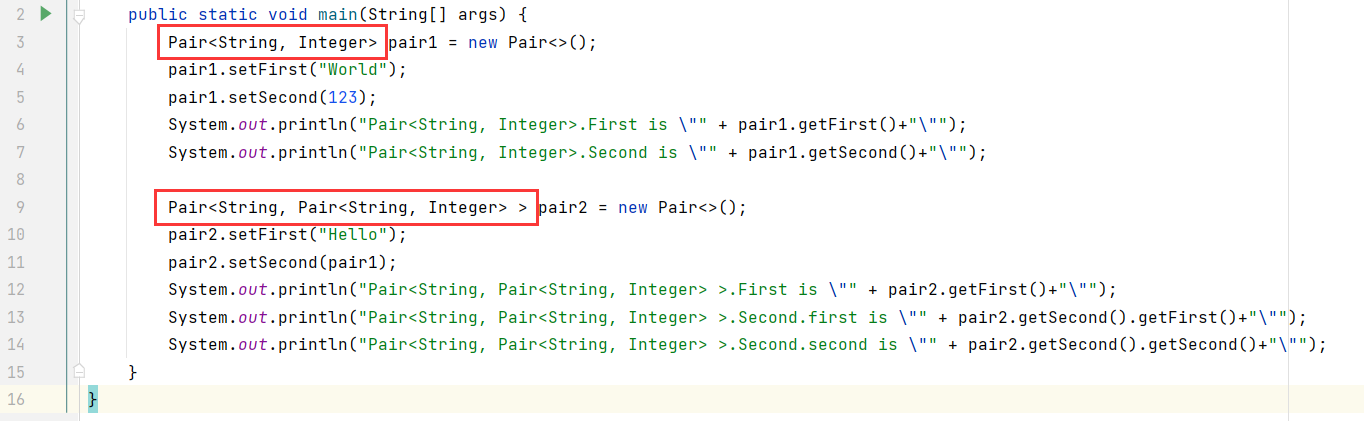


1. 主要代码：

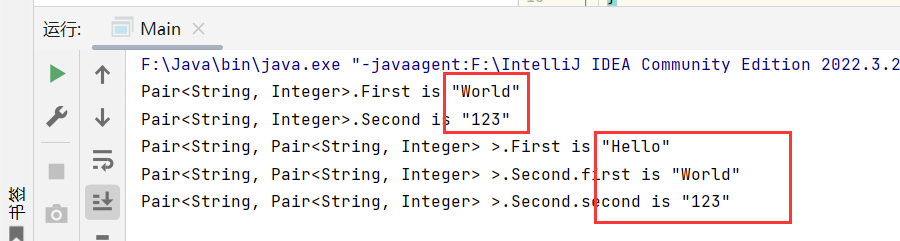
Pair类实现：



Main类主要代码：



1. 运行结果：



1. 题目七：

将CommissionEmploye—BaseCommissionEmployee 继承层次重新建模成一个 Employee 层次，使得每一种员工都具有不同 CompensationModel 对象。本练习中，要求重新实现 CompensationModel 类成为一个接口，提供的公共抽象方法 earnings 不包含参数，且返回一个 double 值。然后，实现了 CompensationModel 接口的如下几个类：

a) SalariedCompensationModel 类——对于周薪固定的员工，这个类需包含一个 weeklySalary 实例变量，且需要实现 earnings 方法，返回 weeklySalary 值。

b) HourlyCompensationModel 类——对于按时薪计酬（包括每周工作超过 40 小时的加班工资）的员工，这个类需包含 wage 和 hours 实例变量，且需根据工作的小时数实现 earnings 方法。

c) CommissionCompensationModel 类——对于按佣金付酬的员工，这个类需包含 grossSales 和 commissionRate 实例变量，还需要实现 earnings 方法，它返回 grossSales x commissionRate 的结果。

d) BasePlusCommissionCompensationMode 类——对按加金付酬的员工，这个类需包含 grossSales commissionRate 和 baseSalary 实例变量，还需要实现 earnings 方法，它返回 baseSalary + grossSales x commissionRate。

在测试程序中，需为上面描述的每一种 CompensationModel 创建一个 Employee 对象，然后显示每一类员工的收入。接着，动态地改变员工的 CompensationModel，重新显示他的收入。

1. 实验环境

集成开发环境：IDEA Community Edition 2022.3.2

JDK版本：JDK17

1. 实现过程

* CompensationModel接口
* 功能：定义一个公共的抽象方法 earnings()，该方法不带参数且返回一个 double 类型的值，表示员工的收入。
* CompensationModel的实现类
* SalariedCompensationModel

表示一个固定周薪的薪酬模型。包含一个weeklySalary实例变量。earnings()方法返回weeklySalary。

* HourlyCompensationModel

用于计算按小时工资的员工收入，包括超过40小时的加班工资。包含 wage和hours实例变量。earnings()方法根据工作时间和加班计算工资。

* CommissionCompensationModel

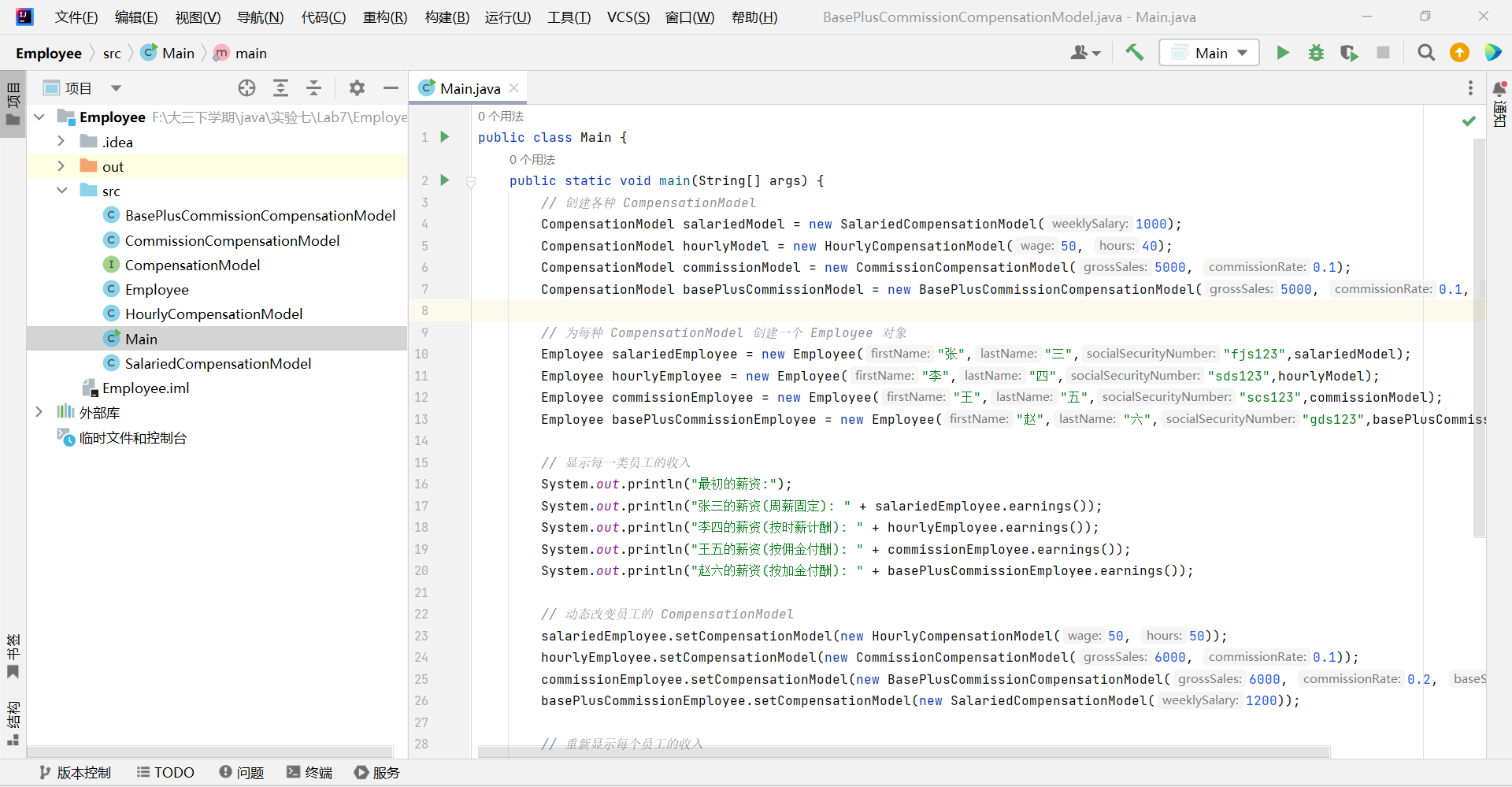
用于按销售佣金计算工资的模型。包含grossSales和commissionRate变量。earnings()方法返回基于销售额和佣金率计算的工资。

* BasePlusCommissionCompensationModel

结合基本工资和销售佣金的薪酬模型。包含grossSales、commissionRate 和baseSalary变量。earnings()方法返回底薪加上基于销售额的佣金。

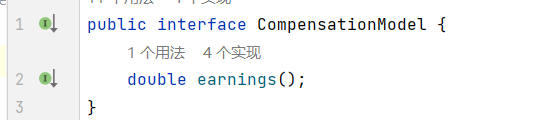
* Employee类
* 功能：表示一个员工，包含姓名、社保号以及一个CompensationModel 对象。
* earnings()方法：调用当前薪酬模型的earnings()方法来计算工资。
* setCompensationModel(CompensationModel model)方法：允许动态更改员工的薪酬模型。
* Main类
* main方法实现创建不同薪酬模型的员工并计算他们的收入，以及动态更改员工的薪酬模型，并展示这些更改如何影响员工的收入。
* 这个设计很好地利用了接口和类继承的特性来创建一个灵活且可扩展的工资计算系统。通过允许动态更改员工的薪酬模型，这个系统可以轻松适应不同的薪酬策略，而不需要修改现有的员工管理代码。这体现了面向对象设计原则中的开闭原则，即软件实体应对扩展开放，对修改封闭。

1. 过程截图
2. 全屏截图：



1. 主要代码：

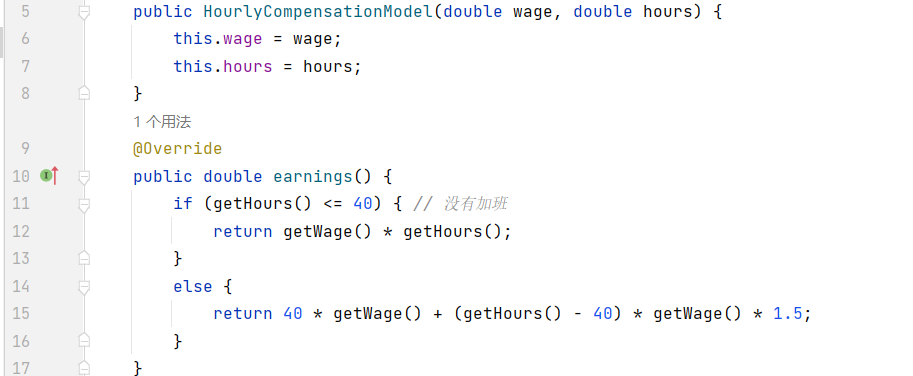
CompensationModel接口实现：



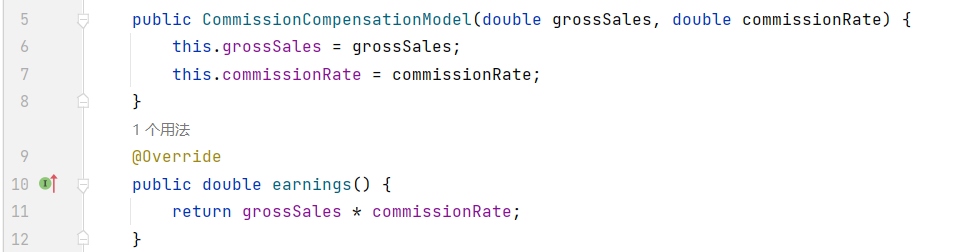
SalariedCompensationModel类主要代码：



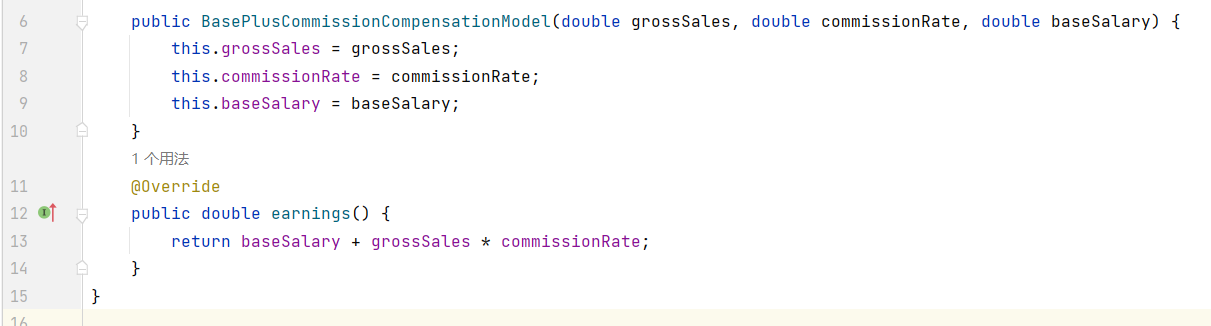
HourlyCompensationModel类主要代码：



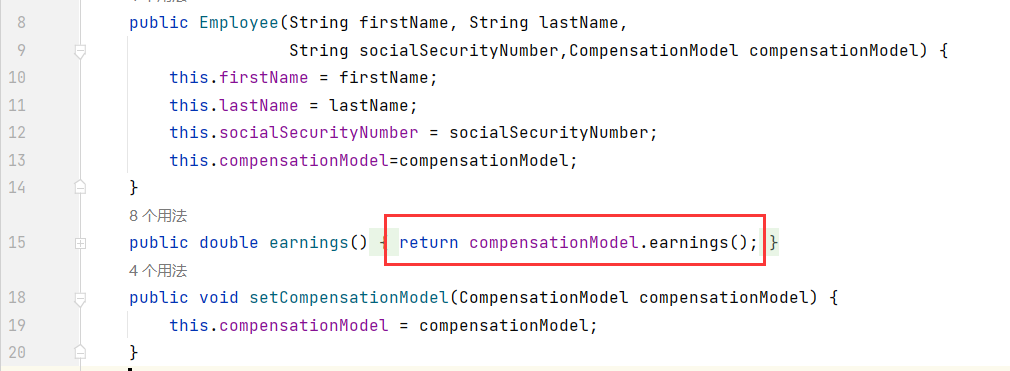
CommissionCompensationModel类主要代码：



BasePlusCommissionCompensationModel类主要代码：



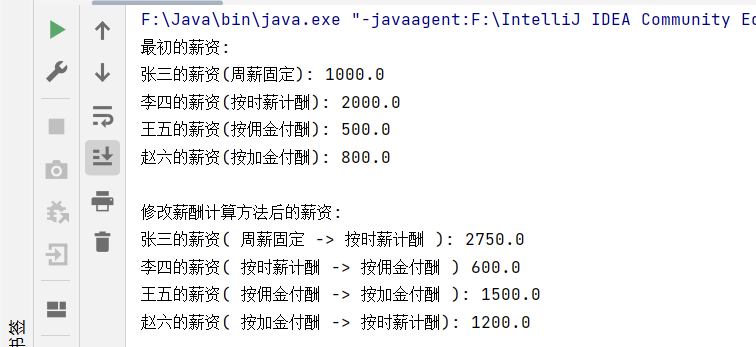
Employee类主要代码：



Main类主要代码：



1. 运行结果：



1. 题目八：

创建三个没有继承关系的小型类:Building、Car 和Bicycle类。为每个类定义一些独有的属性和行为。编写一个包含 getCarbonFootprint 方法的CarbonFootprint接口，让这三个类实现这个接口，利用 getCarbonFootprint 方法计算每一个类的大致碳足迹(需查看相关网站来了解如何计算碳足迹)。编写一个程序，创建每个类的对象，将这些对象的引用放入ArrayList<CarbonFootprint> ，然后迭代遍历这个 ArrayList，多态地调用每个对象的getCarbonFootprint 方法。对于每一个对调，需输出一些表明对象类别的信息及它的碳足迹信息。

1. 实验环境

集成开发环境：IDEA Community Edition 2022.3.2

JDK版本：JDK17

1. 实现过程

* CarbonFootprint接口
* 功能：定义一个getCarbonFootprint方法，返回一个double类型的值，代表对象的碳足迹（单位为吨CO2）。
* 实现CarbonFootprint接口的类
* Bicycle类

代表摩托车，包含名称、类型和里程。使用预定义的效率值（每公里的CO2排放量）计算碳足迹。getCarbonFootprint方法根据类型和行驶里程计算总碳排放量。

* Car类

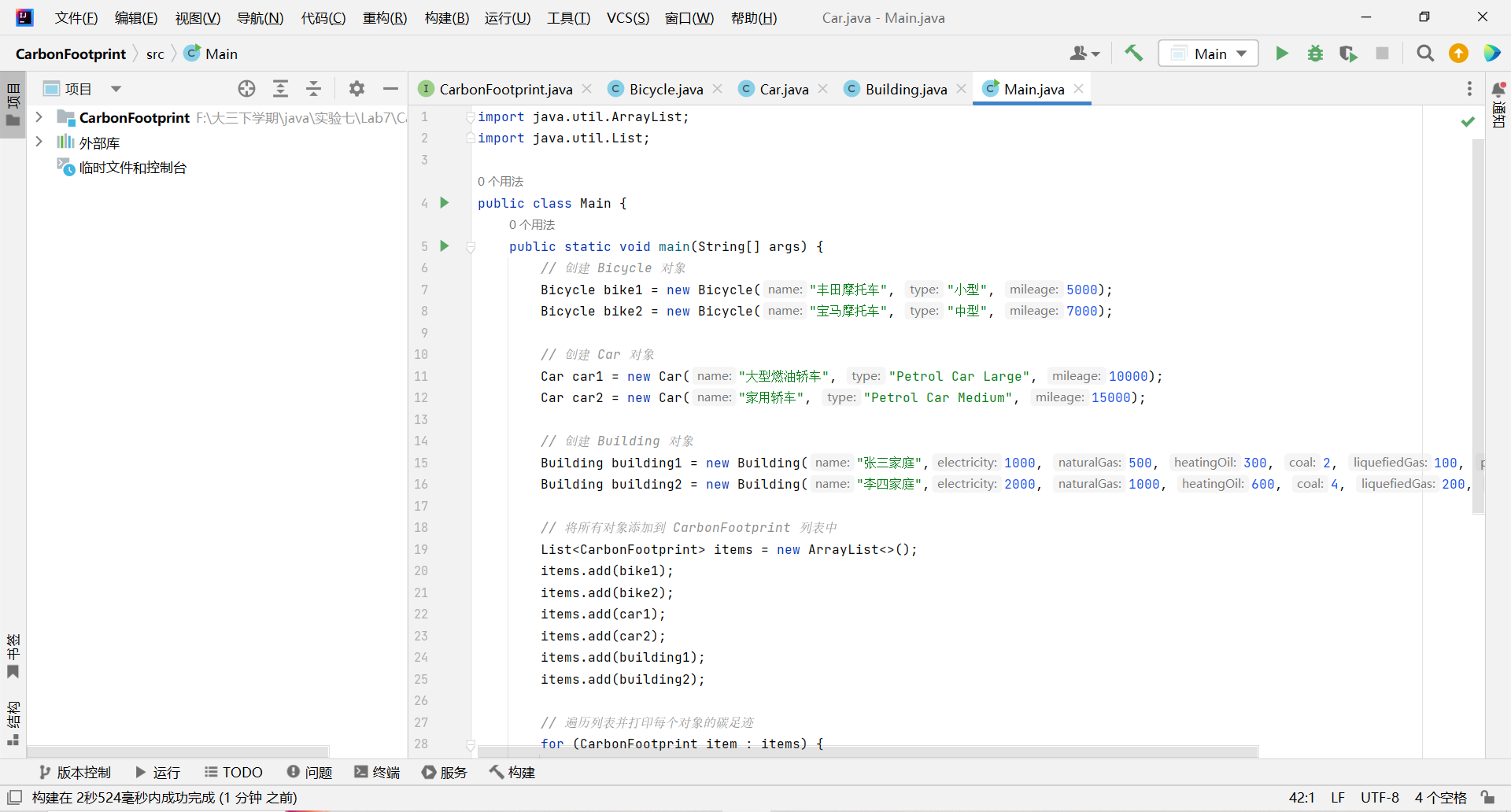
代表汽车，包含汽车的名称、类型和里程。根据不同的汽车类型，使用不同的CO2排放效率来计算碳足迹。getCarbonFootprint方法使用汽车类型和行驶里程来计算碳排放量。

* Building类

代表房屋，包含名称和各种能源的使用量（电、天然气、煤炭等）。使用每种能源的CO2排放系数来计算碳足迹。getCarbonFootprint方法计算所有能源的综合碳排放量。

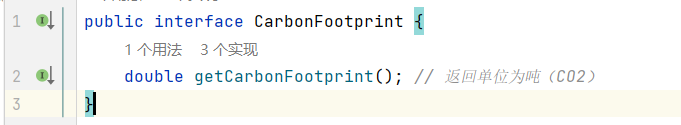
* Main类
* Main方法：创建具体对象，并设置其特有的属性。将这些对象存储在ArrayList<CarbonFootprint>中，演示了泛型集合的使用。遍历这个列表，多态地调用每个对象的getCarbonFootprint方法来计算并打印其碳足迹。通过输出每个对象的类型和名称来提供上下文信息，显示每种类型的对象其碳足迹。

1. 过程截图
2. 全屏截图：

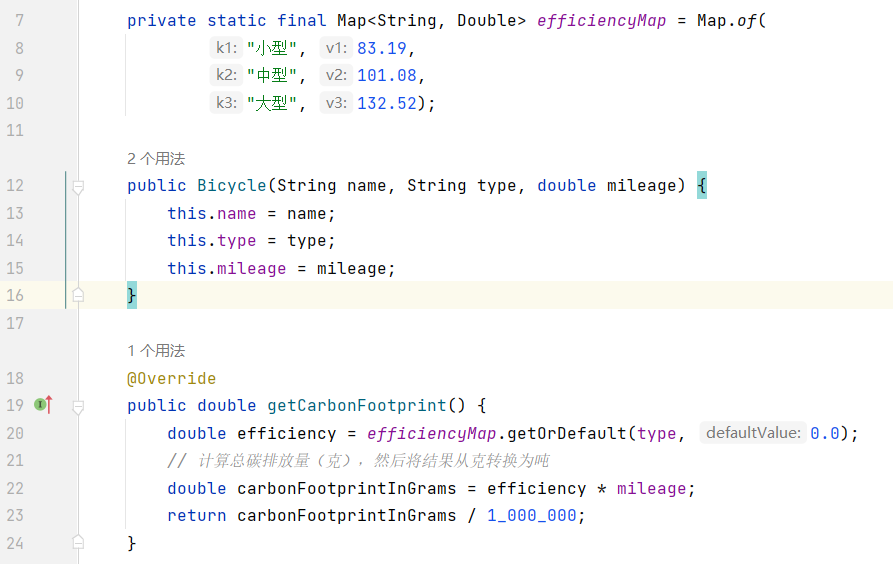


1. 主要代码：

CarbonFootprint接口：



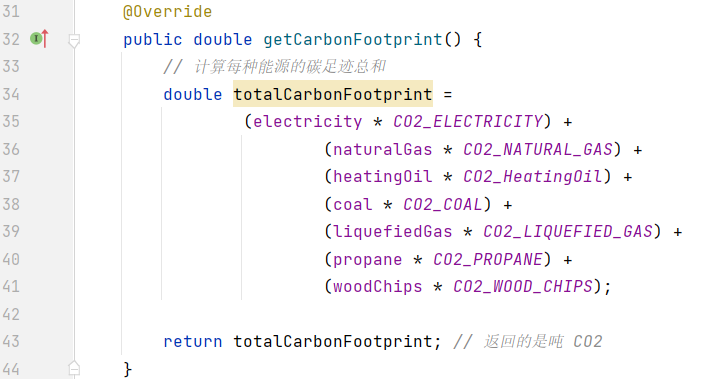
Bicycle类主要代码：



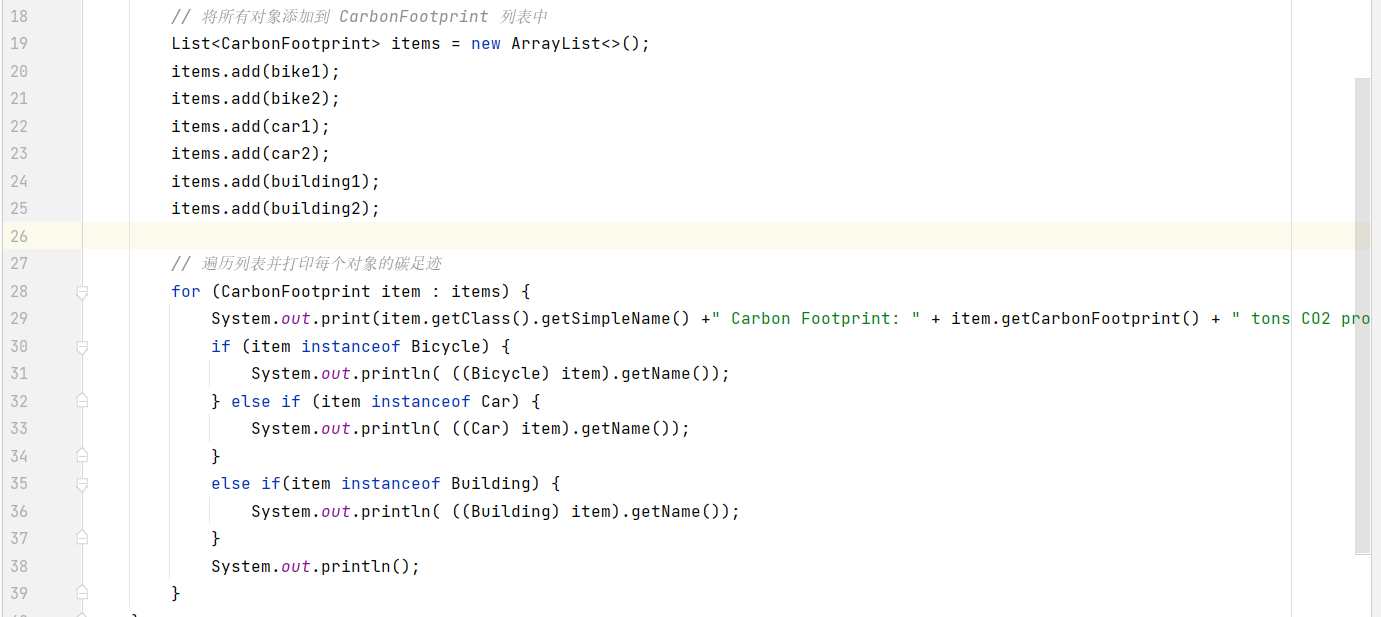
Car类主要代码：



Building类主要代码：



Main类主要代码：



c) 运行结果：



**三、实验总结与心得记录**

通过实验，我学习到了如何合理使用try-catch和throw语句来处理程序运行中可能遇到的各种异常情况，如输入类型不匹配、数据转换异常等。我还学习了如何自定义异常类来处理特定的错误情况，这使得代码更加模块化和易于维护。

泛型的学习让我能够编写更加灵活和通用的代码。在编写涉及CarbonFootprint接口的程序时，我实践了如何定义接口并让多个类实现这一接口。

在完成这系列的编程实验后，我深化了对Java异常处理、泛型程序设计、以及接口和多态性应用的理解。通过这些实践，我不仅提升了编程和问题解决技能，还学会了如何将理论知识有效地应用于实际编码任务中，特别是在自定义异常处理、泛型方法的实现，以及多态性的运用上。