**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **Java语言程序设计** |
| **学生姓名：** | **葛子午** |
| **学生学号：** | **202330550381** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2024-2025学年第一学期** |

**软件学院**

**2024年10月**

### 实验三 接口实现与异常处理

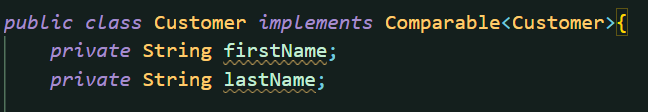
#### 一、实验目的

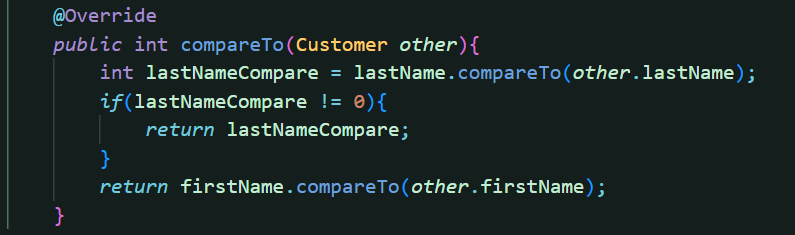
1. 熟悉在Eclipse IDE（或IDEA等其他）集成开发环境中开发与调试Java项目；
2. 掌握接口的定义与接口的实现，区分接口与抽象类；
3. 掌握异常的概念及异常处理的机制；
4. 掌握try-catch-finally异常处理语句的使用；
5. 熟悉用户自定义异常及处理用户自定义异常的方法。

#### 实验内容及结果

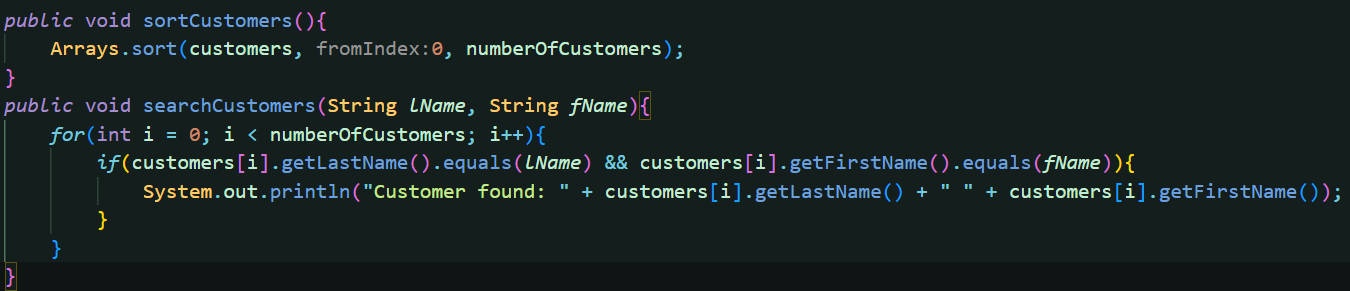
1. Step6： 在系统中添加新的功能，支持银行客户按姓名排序和搜索；

首先修改Customer类实现Comparable 接口；

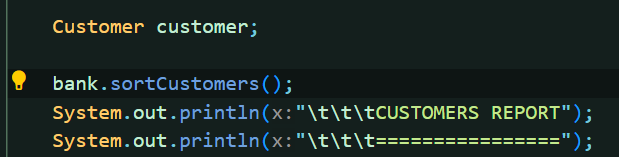




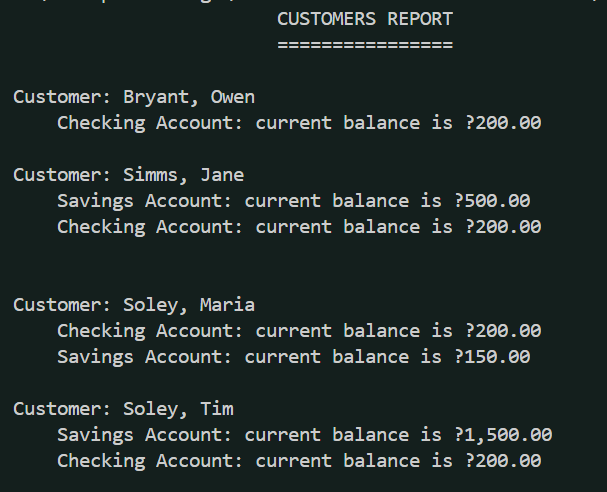
然后修改Bank类，增加sortCustomers 和searchCustomers 方法；



最后修改CustomerReport 类，调用sortCustomers进行输出。

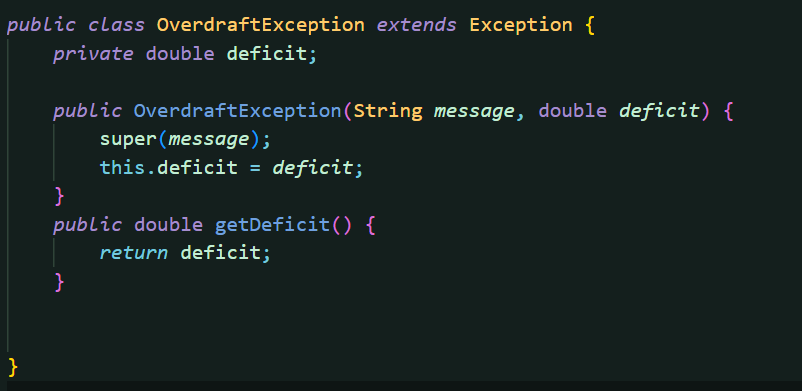


输出结果如下

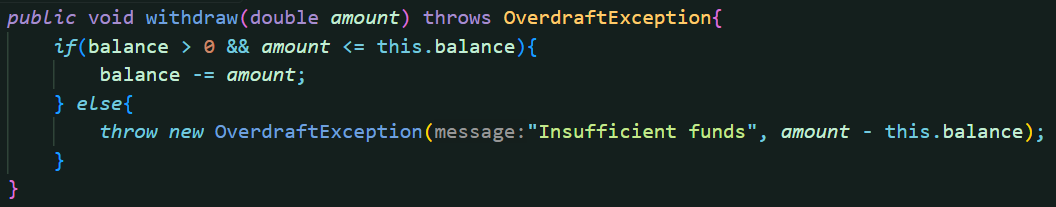


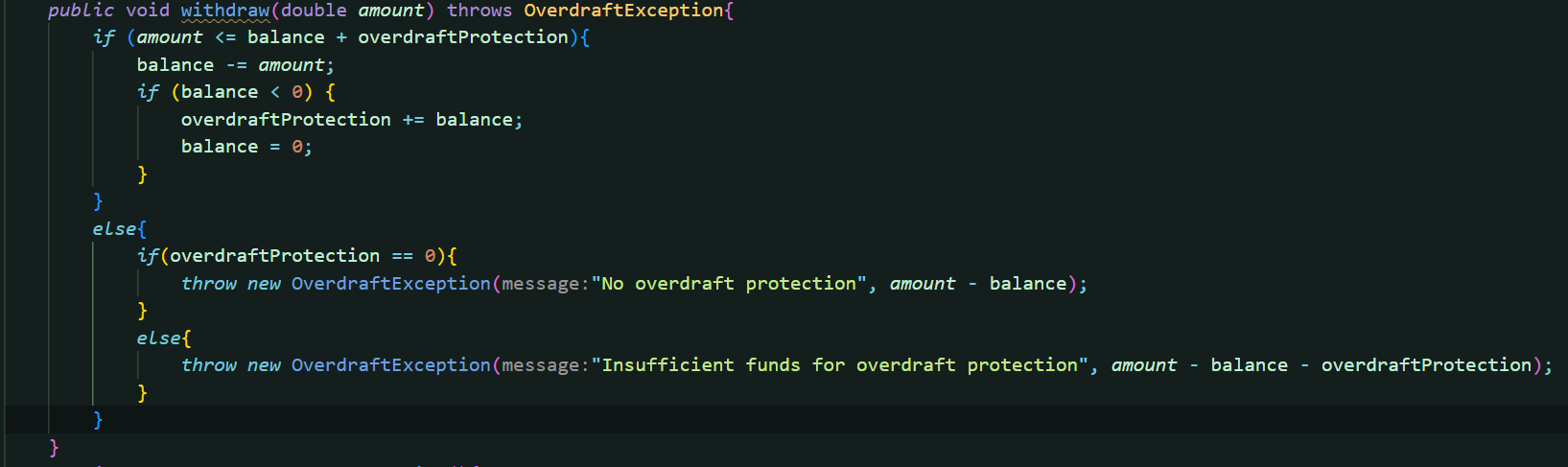
1. Step 7： 添加发生账户透支错误时的异常处理功能；

首先自定义新的异常类OverdraftException

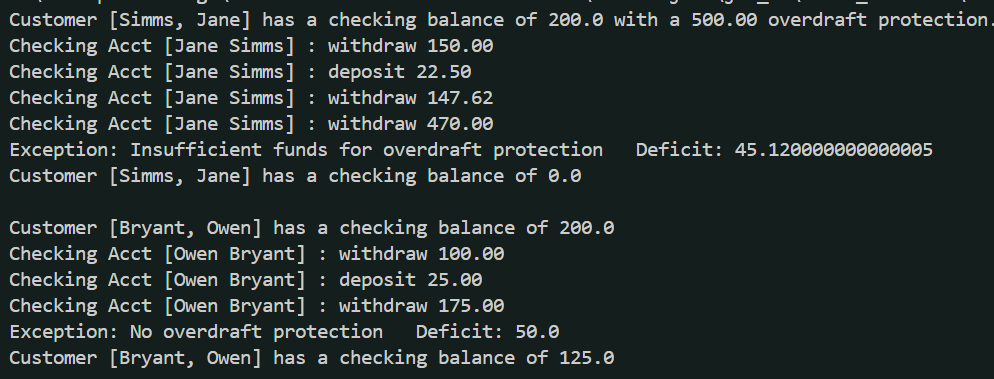


然后修改Account类和CheckingAccount 类声明和抛出OverdraftException异常。





运行结果如下



1. **新建一个新的Java项目，完成以下练习并回答问题：**

创建接口Speakable和Runner，然后创建两个类Dog和Person实现该接口。

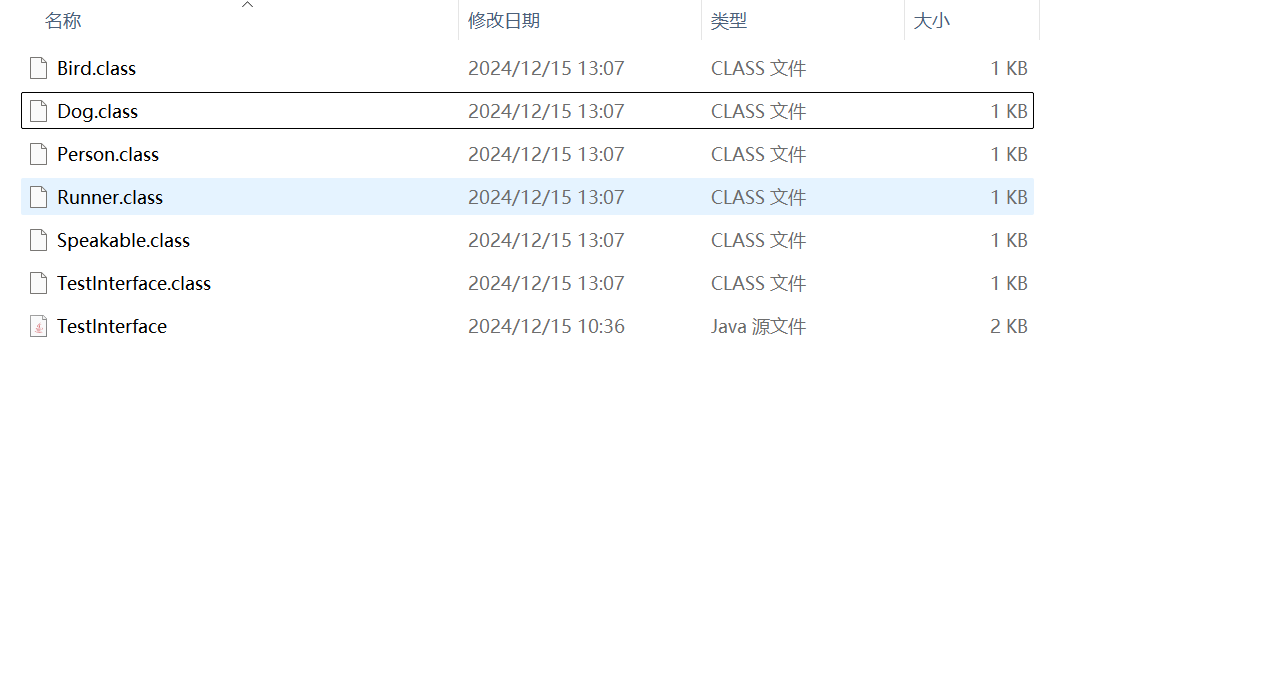
**程序代码：**

|  |
| --- |
| interface Speakable{  public void speak();  }  interface Runner{  public void run();  }  class Dog implements Speakable,Runner{  public void speak(){  System.out.println("狗的声音:汪、汪！");  }  public void run(){  System.out.println("狗用四肢跑步");  }  }  class Person implements Speakable,Runner{  public void speak(){  System.out.println("人们见面时经常说:您好！");  }  public void run(){  System.out.println("人用两腿跑步");  }  }  public class TestInterface{  public static void main(String[] args) {  Dog d=new Dog();  d.speak(); d.run();  Person p=new Person();  p.speak(); p.run();  }  } |

运行上面的程序，思考如下的问题：

1. 该程序编译后生成几个class文件？

生成五个class文件(不包括下题的Bird.class)



1. 创建一个类Bird（鸟），给出其声音特征，并在主类中创建一个Bird类的实例，输出其特征。

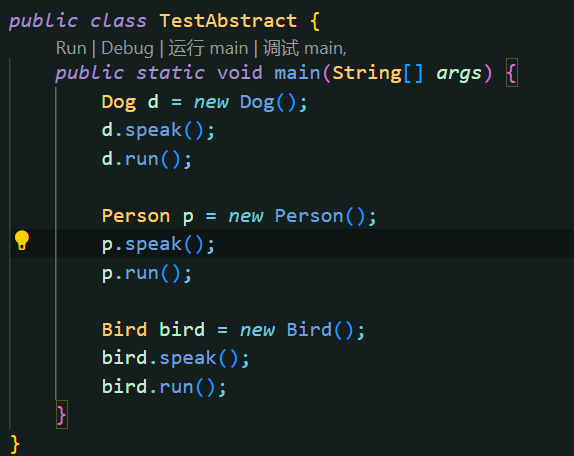
Bird类如下



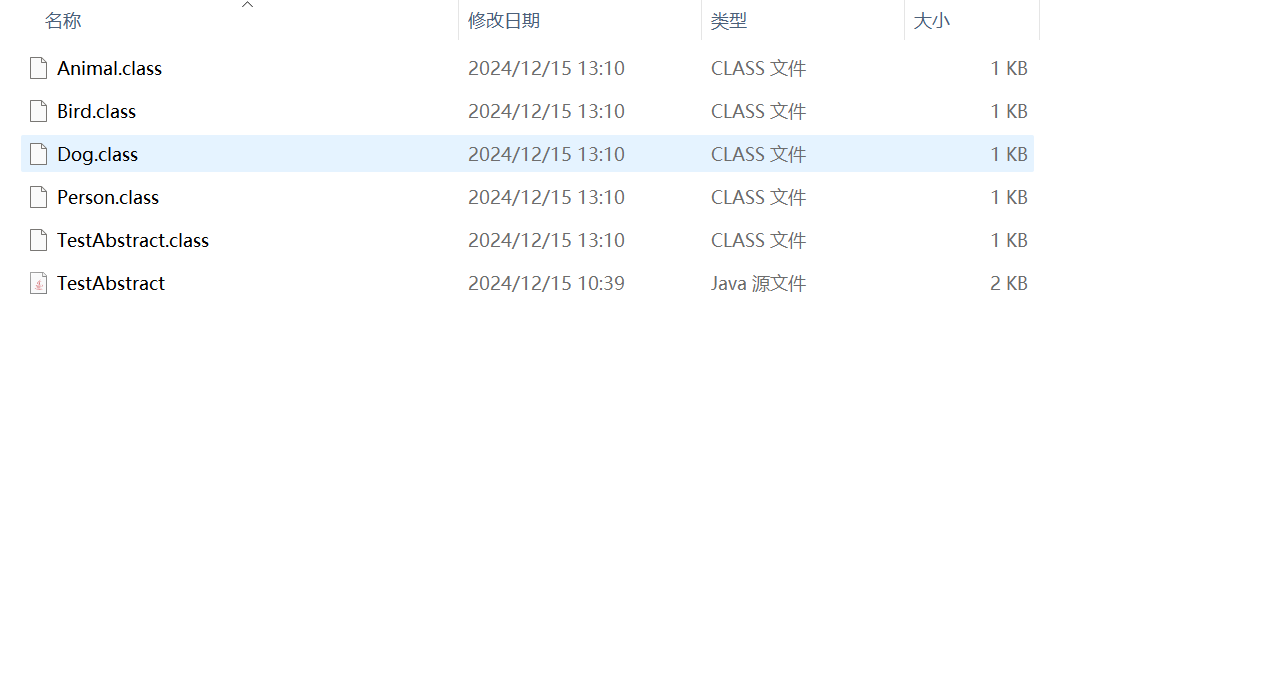
1. 如何编写抽象类代替程序中的接口，实现程序同样的功能。试比较它们的不同。

编写的抽象类如下





编译结果如下



接口和抽象类的比较

接口（Interface）：

接口中只能包含抽象方法和常量。

接口中的方法默认是public的，且必须被实现类实现。

一个类可以实现多个接口。

接口用于定义类的行为规范。

抽象类（Abstract Class）：

抽象类中可以包含抽象方法和非抽象方法。

抽象类中的方法可以有不同的访问修饰符。

一个类只能继承一个抽象类。

抽象类用于定义类的部分实现，为子类提供一个模板。

在这个例子中，使用接口和抽象类都可以实现相同的功能。但是，使用接口可以让代码更加清晰和模块化，因为接口只定义了行为规范，而不包含实现细节。而使用抽象类则可以提供一些默认的实现，减少子类的重复代码。