**实 验 报 告**

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

**软件学院**

**2023年11月**

### 实验三 接口实现与异常处理

1. **实验目的**
2. 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；
3. 掌握接口的定义与接口的实现，区分接口与抽象类；
4. 掌握异常的概念及异常处理的机制；
5. 掌握try-catch-finally异常处理语句的使用；
6. 熟悉用户自定义异常及处理用户自定义异常的方法。

**二、实验内容及结果**

**1. step6**

**（1）**修改Customer类实现Comparable 接口

1. // 实现 Comparable 接口的 compareTo 方法
2. @Override
3. **public** **int** compareTo(Customer other) {
4. // 比较姓氏，如果姓氏相同，则按名字排序
5. **int** lastNameComparison = **this**.lastName.compareToIgnoreCase(other.lastName);
6. **if** (lastNameComparison != 0) {
7. **return** lastNameComparison;
8. }
9. **return** **this**.firstName.compareToIgnoreCase(other.firstName);
10. }

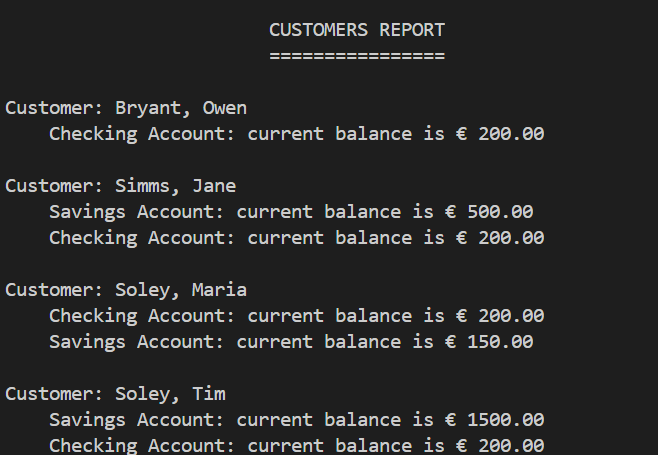
修改Bank类，增加sortCustomers 和searchCustomers 方法

1. **public** **void** sortCustomers() {
2. Collections.sort(customers);
3. }
5. // 搜索客户
6. **public** Customer searchCustomers(String firstName, String lastName) {
7. **for** (Customer customer : customers) {
8. **if** (customer.getFirstName().equalsIgnoreCase(firstName) && customer.getLastName().equalsIgnoreCase(lastName)) {
9. **return** customer;
10. }
11. }
12. **return** **null**; // 如果没有找到，返回 null
13. }

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

1. **public** **void** generateReport() {
2. Bank bank = Bank.getBank();  // 使用单例 Bank 对象
4. // 在生成报告之前，先排序客户
5. bank.sortCustomers();
7. System.out.println("\t\t\tCUSTOMERS REPORT");
8. System.out.println("\t\t\t================");
10. **for** (**int** cust\_idx = 0; cust\_idx < bank.getNumOfCustomers(); cust\_idx++) {
11. Customer customer = bank.getCustomer(cust\_idx);
13. System.out.println();
14. System.out.println("Customer: "
15. + customer.getLastName() + ", "
16. + customer.getFirstName());
18. **for** (**int** acct\_idx = 0; acct\_idx < customer.getNumOfAccounts(); acct\_idx++) {
19. Account account = customer.getAccount(acct\_idx);
20. String account\_type = "";
22. **if** (account **instanceof** SavingsAccount) {
23. account\_type = "Savings Account";
24. } **else** **if** (account **instanceof** CheckingAccount) {
25. account\_type = "Checking Account";
26. }
28. System.out.println("    " + account\_type + ": current balance is € "
29. + String.format("%.2f", account.getBalance()));
30. }
31. }
32. }

运行 TestBanking程序得到：

****

**2. step7**

添加发生账户透支错误时的异常处理功能；

2.1 定义新的异常类OverdraftException

1. **package** banking.domain;
3. **public** **class** OverdraftException **extends** Exception {
4. **private** **double** deficit;  // 亏空金额
6. // 构造函数，接受消息和亏空金额
7. **public** OverdraftException(String message, **double** deficit) {
8. **super**(message);  // 调用父类构造函数传递消息
9. **this**.deficit = deficit;
10. }
12. // 访问器方法，返回亏空金额
13. **public** **double** getDeficit() {
14. **return** deficit;
15. }
16. }

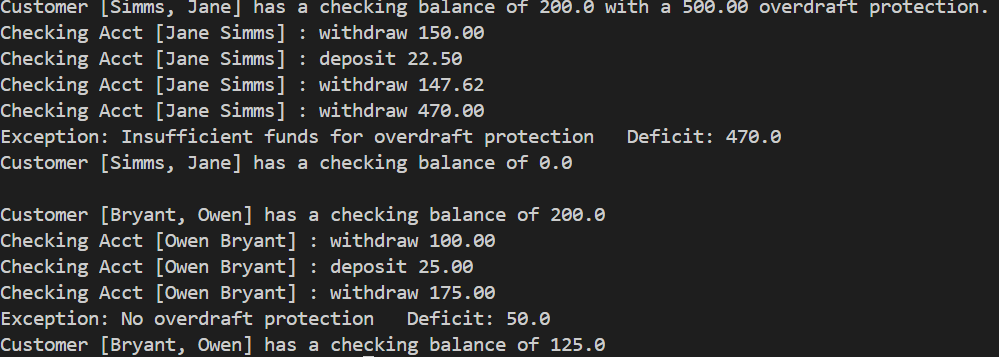
2.2 修改Account类和CheckingAccount 类声明和抛出OverdraftException异常

Account中：修改了withdraw

1. **public** **void** withdraw(**double** amount) **throws** OverdraftException {
2. **if** (amount > 0 && amount <= balance) {
3. balance -= amount;
4. } **else** {
5. // 如果余额不足，抛出 OverdraftException
6. **throw** **new** OverdraftException("Insufficient funds", amount - balance);
7. }
8. }

在CheckingAccount中，重写 withdraw 方法，抛出 OverdraftException 异常

1. @Override
2. **public** **void** withdraw(**double** amount) **throws** OverdraftException {
3. **if** (balance >= amount) {
4. balance -= amount;
5. } **else** **if** (overdraftProtection >= (amount - balance)) {
6. overdraftProtection -= (amount - balance);
7. balance = 0;
8. } **else** {
9. // 根据情况抛出不同的异常
10. **if** (overdraftProtection == 0) {
11. **throw** **new** OverdraftException("No overdraft protection", amount - balance);
12. } **else** {
13. **throw** **new** OverdraftException("Insufficient funds for overdraft protection", amount - balance);
14. }
15. }
16. }

得到输出

**3. step7.1**

运行给出的代码并回答问题

**（1）该程序编译后生成几个class文件？**

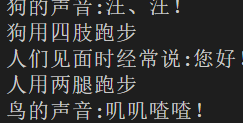
在 Java 中，一个.java源文件编译后会为每个内部定义的类（包括接口、普通类等）生成一个独立的.class文件。

上述代码中定义了 5 个类型，分别是Speakable接口、Runner接口、Dog类、Person类以及TestInterface主类，所以编译后会生成 5 个.class文件，分别是Speakable.class、Runner.class、Dog.class、Person.class、TestInterface.class。

**（2）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。**

1. **interface** Speakable{
2. **public** **void** speak();
3. }
4. **interface** Runner{
5. **public** **void** run();
6. }
7. **class** Dog **implements** Speakable,Runner{
8. **public** **void** speak(){
9. System.out.println("狗的声音:汪、汪！");
10. }
11. **public** **void** run(){
12. System.out.println("狗用四肢跑步");
13. }
14. }
15. **class** Person **implements** Speakable,Runner{
16. **public** **void** speak(){
17. System.out.println("人们见面时经常说:您好！");
18. }
19. **public** **void** run(){
20. System.out.println("人用两腿跑步");
21. }
22. }
23. **class** Bird **implements** Speakable{
24. **public** **void** speak(){
25. System.out.println("鸟的声音:叽叽喳喳！");
26. }
27. }
29. **public** **class** TestInterface{
30. **public** **static** **void** main(String[] args) {
31. Dog d = **new** Dog();
32. d.speak();
33. d.run();
35. Person p = **new** Person();
36. p.speak();
37. p.run();
39. Bird b = **new** Bird();
40. b.speak();
41. }
42. }

得到运行结果：

**（3）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**

1. **abstract** **class** Speakable {
2. **public** **abstract** **void** speak();  }
3. **abstract** **class** Runner {
4. **public** **abstract** **void** run();
5. }
6. **class** Dog **extends** Runner **implements** Speakable {
7. **public** **void** speak() {
8. System.out.println("狗的声音:汪、汪！");
9. }
11. **public** **void** run() {
12. System.out.println("狗用四肢跑步");
13. }
14. }  **class** Person **extends** Runner **implements** Speakable {
15. **public** **void** speak() {
16. System.out.println("人们见面时经常说:您好！");
17. }
19. **public** **void** run() {
20. System.out.println("人用两腿跑步");
21. }
22. }  **class** Bird **extends** Speakable {
23. **public** **void** speak() {
24. System.out.println("鸟的声音:叽叽喳喳！");
25. }
26. }
27. **public** **class** TestAbstract {
28. **public** **static** **void** main(String[] args) {
29. Dog d = **new** Dog();
30. d.speak();
31. d.run();
33. Person p = **new** Person();
34. p.speak();
35. p.run();
36. Bird b = **new** Bird();
37. b.speak();
38. }
39. }

**接口和抽象类的不同**：

* **设计目的**：
  + **接口**：主要用于定义一组相关的行为规范（方法签名），侧重于描述 “能做什么”，实现类必须实现接口中定义的所有方法，它更像是一种契约、标准，一个类可以实现多个接口，从而具备多种行为能力。
  + **抽象类**：可以包含抽象方法（没有方法体的方法），也可以包含普通方法以及成员变量等，它侧重于对一类事物的部分实现和通用属性、行为的抽取，是一种对事物进行抽象概括的模板，一个类只能继承一个抽象类（Java 单继承限制）。
* **方法实现**：
  + **接口**：接口中的方法默认都是public、abstract的（即使不写修饰符也是如此），不能有方法体，实现类必须全部实现接口的方法。
  + **抽象类**：抽象类中可以有抽象方法（用abstract修饰，无方法体），也可以有已经实现了的普通方法，子类继承抽象类时，需要实现抽象类中的抽象方法，但对于普通方法可以直接继承使用。
* **变量使用**：
  + **接口**：接口中的变量默认都是public、static、final的（常量），一旦赋值后不能修改。
  + **抽象类**：抽象类中可以定义各种访问修饰符的成员变量，并且可以根据需要修改其值（非final的情况下）。
* **使用场景**：
  + **接口**：适合定义一些通用的行为规范，让不同的类去实现这些规范来达到多态的效果，例如不同的动物都可以实现 “发声”“移动” 等接口；或者用于实现多重继承的效果，让类具备多种不同维度的能力。
  + **抽象类**：常用于在一组相关的类中有一些通用的逻辑、属性可以抽取，同时又有一些需要子类根据自身情况去具体实现的方法的场景，例如各种图形类可以继承抽象的图形类，抽象图形类中可以有计算面积的抽象方法由具体图形子类去实现。