西安电子科技大学

Java 程序设计 课程实验报告

实验名称 银行账户系统异常处理

计算机科学与技术 学院 2103051 班

姓名 张平 学号 21030540006

同作者_____

实验日期 2022 年 05 月 09 日

成 绩

指导教师评语： 指导教师： _____年____月____日
实验报告内容基本要求及参考格式 一、实验目的 二、实验内容 三、实验过程 四、实验结果分析 五、实验小结（实验过程感受和建议）

一、实验目的

1. 掌握面向对象的类、继承、多态概念，能运用 Java 完成基于面向对象的指定功能的程序制作。

2. 在第四章实验的基础上，熟悉异常的定义、异常的类型层次，以及异常处理的动机。

3. 掌握异常处理的两种方法：捕获并处理异常；将异常抛出。

4. 熟悉如何定义并使用自己的异常类。了解断言的作用。

二、实验内容

为方便阅览，这里列出第四章实验 3 的实验要求如下：

Design **BankSystem** class. You need to design more than one class. For example:

(1) You need to design a class BankAccount to model users' bank accounts. Probably different bank accounts (CashAccount, CreditAccount, …).

(2) The account should keep a user's name and balance, accurate to the nearest cent…

(3) The user should be able to make deposits and withdrawals on his/her account, as well as changing the account's name at any time.

(4) Also, the system needs to be able to find out how many BankAccounts have been created in total.

(5) For each account, only the last 6 Transactions should be able to store in ascending order and be printed.

这一章将对第四章实验内容 3 添加异常处理、完善其运行机制。要求如下：

(1) Add the CheckingAccount class of BankSystem , Designing and Using Classes to throw an IllegalArgumentException in any of the following circumstances:

a. when the account is constructed with a negative balance,

b. when a negative amount is deposited, or

c. when the account is overdrawn (when the amount withdrawn exceeds the current balance).

(2) An IllegalArgumentException is an unchecked exception that is thrown to indicate that a method has been passed an illegal or inappropriate argument.

Instructions:

Add (modify if you've already finished) the CheckingAccount class to handle errors and write a test program as indicated above.

三、实验过程

1. 实验环境

操作系统：Windows 11

集成开发环境：Eclipse IDE for Enterprise Java and Web Developers (includes Incubating components) 2022-03 (4.23.0)

2. 题目分析

这次实验比较简单，对其分析发现，本次实验要求我们给此前的银行账户系统添加异常处理，对这三种情况——账户对象以负数余额构造、存储负数金额进入账户、账户过度取款，要求抛出一个非法参数异常。为此，题目要求在银行系统中添加一个名为 CheckingAccount 的类。

而我们将在这个类中定义三个静态方法，用于分别检查这三种情况并抛出异常。之所以选择将其定义为静态方法，是因为我们不需要个性化的 CheckingAccount 类对象，我们只需要调用相关方法、完成相应检查操作即可。事实上，在第四章实验 3 中，我就已经完成了简单的异常处理，本次实验只用稍作修改：

```
// 存款
public void deposit(double money) {
    if (money < 0) throw new IllegalArgumentException("negative deposit");
    this.balance += money;
    addTransaction("Deposit", "+" + String.valueOf(money)); // 增加存款事务
}
// 取款
public void withdraw(double money) {
    if (money < 0 || money > balance)
        throw new IllegalArgumentException("illegal withdrawal");
    this.balance -= money;
    addTransaction("Withdraw", "-" + String.valueOf(money));
}
```

```java
@Override
public void withdraw(double change) { // Credit账户可以透支额度,因此需要重写父类的withdraw方法
    if (change > getBalance() + remainder)
        throw new IllegalArgumentException("illegal withdrawal");
    if (change <= remainder) remainder -= change;
    else {
        setBalance(getBalance() - (change - remainder));
        remainder = 0; // 用完额度
    }
    addTransaction("Withdraw", "-" + String.valueOf(change));
}
```

此外，这里不使用 Erlang/Elixir 的编程哲学 Let it crash，而是在抛出异常后使用 try/catch 语句进行捕捉、打印错误提示、并继续执行程序。

3. 代码实现

CheckingAccount 类的实现如下所示，除了完成题目要求的情况外，还做了进一步完善。checkConstructor 方法检查实例化账户对象时、是否以负数余额构造，checkNegativeDeposit 方法检查是否存储负数金额进入账户，checkOverdrawn 方法检查（储蓄/信用）账户是否过度取款，checkNegativeWithdraw 方法检查取款金额是否为负数。此外，**代码中在判断浮点数是否为负数时，没有直接让浮点数与 0 比较**。

```java
package bank;

public class CheckingAccount {
    private static final double eps = 1e-8;
    // 小于函数,用于浮点数比较
    private static boolean less(double a, double b) {
        return (a - b) < (-eps); // 只有小于b-eps的数a才能判断为小于b
    }

    // 大于函数,用于浮点数比较
    private static boolean greater(double a, double b) {
        return (a - b) > eps; // 只有大于b+eps的数a才能判断为大于b
    }

    // 检查是否存储负数金额进入账户
    public static void checkNegativeDeposit(double amount) {
        if (less(amount, 0.0)) {
            throw new IllegalArgumentException("A negative amount is deposited");
        }
    }

    // 检查实例化账户对象时、是否以负数余额构造
    public static void checkConstructor(double amount) {
        if (less(amount, 0.0)) {
            throw new IllegalArgumentException("The account is constructed with a negative amount");
        }
    }
```

```
    // 检查（储蓄/信用）账户是否过度取款
    public static void checkOverdrawn(double change, double amount) {
        if (greater(change, amount)) {
            throw new IllegalArgumentException("The account is overdrawn");
        }
    }

    // 检查取款金额是否为负数
    public static void checkNegativeWithdraw(double amount) {
        if (less(amount, 0.0)) {
            throw new IllegalArgumentException("A negative amount is withdrawn");
        }
    }
}
```

原先的检查语句，现在换成对 CheckingAccount 相应静态方法的调用，如果这些静态方法正常完成，则对应的存款、取款、初始化等操作继续执行；否则停止执行、并继续向上层调用抛出异常：

```
// 存款
public void deposit(double money) {
    CheckingAccount.checkNegativeDeposit(money);
    this.balance += money;
    addTransaction("Deposit", "+" + String.valueOf(money)); // 增加存款事务
}
// 取款
public void withdraw(double money) {
    CheckingAccount.checkOverdrawn(money, balance);
    CheckingAccount.checkNegativeWithdraw(money);

    this.balance -= money;
    addTransaction("Withdraw", "-" + String.valueOf(money));
}

@Override
public void withdraw(double change) { // Credit账户可以透支额度,因此需要重写父类的withdraw方法
    CheckingAccount.checkOverdrawn(change, getBalance() + remainder); // 检查是否过度透支
    CheckingAccount.checkNegativeWithdraw(change); // 检查是否取负数金额

    if (change <= remainder) remainder -= change;
    else {
        setBalance(getBalance() - (change - remainder));
        remainder = 0; // 用完额度
    }
    addTransaction("Withdraw", "-" + String.valueOf(change));
}

public BankAccount(int id, String name, String password, double balance) {
    CheckingAccount.checkConstructor(balance);
    this.id = id;
    this.acName = name;
    this.password = password;
    this.balance = balance;
    transactions = new String[6];
}
public CreditAccount(int id, String name, String password, double balance, double ceiling) {
    super(id, name, password, balance);
    CheckingAccount.checkConstructor(ceiling);
    this.ceiling = ceiling;
    this.remainder = ceiling;
}
```

遵循题目要求，我们在主类 BankSystem 的 openAccount 方法（即创建储蓄账户和信用账户的方法）和 login 方法（即输入账户姓名和密码来登录账户，进行各种实际操作的方法）中，修改相关的代码。如果对账户对象的操作抛出异常，则用 try/catch 语句进行捕捉，并继续循环、直到输入正确数据（**不打印异常抛出的信息，将其与用户隔离开**）；若无异常，则 break 退出循环。部分代码如下所示：

```
while (true) {
    try {
        System.out.print("输入初始储蓄金额：");
        balance = sc.nextDouble();
        bankAccountList.add(new CashAccount(id, name, password, balance));
        System.out.println("您的账户已成功创建！账户ID为" + id + "！");
        ++acCount;
        break;
    } catch (Exception e) {
        System.out.println("输入了错误的初始储蓄金额！需要重新输入！");
    }
}
```

```
while (true) {
    System.out.print("输入初始储蓄金额：");
    balance = sc.nextDouble();
    System.out.print("输入信用额度：");
    ceiling = sc.nextDouble();
    try {
        bankAccountList.add(new CreditAccount(id, name, password, balance, ceiling));
        System.out.println("您的账户已成功创建！账户ID为" + id + "！");
        ++acCount;
        break;
    } catch (Exception e) {
        System.out.println("输入了错误的初始储蓄金额或信用额度！需要重新输入！");
    }
}
```

```
case 1:
    while (true) {
        try {
            System.out.print("输入存款金额：");
            change = sc.nextDouble();
            yourBA.deposit(change);
            break;
        } catch (Exception e) {
            System.out.println("输入了错误的存款金额（负数）！需要重新输入！");
        }
    }
    break;
case 2:
    while (true) {
        try {
            System.out.print("输入取款金额：");
            change = sc.nextDouble();
            yourBA.withdraw(change);
            break;
        } catch (Exception e) {
            System.out.println("输入了错误的取款金额（负数或超出限额）！需要重新输入！");
        }
    }
    break;
```

四、实验结果测试

在 Eclipse 中对实现的功能进行测试，结果如下：

```
选择要使用的功能：
1. 开户
2. 登录账户
3. 退出系统
1
选择账户类型(1: CashAccount, 2: CreditAccount)：1
输入现金账户名称：张平
输入账户密码：wdcamm123
输入初始储蓄金额：-3
输入了错误的初始储蓄金额！需要重新输入！
输入初始储蓄金额：100
您的账户已成功创建！账户ID为0！
选择要使用的功能：
1. 开户
2. 登录账户
3. 退出系统
1
选择账户类型(1: CashAccount, 2: CreditAccount)：2
输入信用账户名称：张月
输入账户密码：tdcamm123
输入初始储蓄金额：-123
输入信用额度：123
输入了错误的初始储蓄金额或信用额度！需要重新输入！
输入初始储蓄金额：2999
输入信用额度：1299
您的账户已成功创建！账户ID为1！
选择要使用的功能：
1. 开户
2. 登录账户
3. 退出系统
2
输入账户名称：张平
输入账户密码：wdcamm123
成功登录！
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
3
账户余额：100.0
```

```
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
3
账户余额：100.0
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
1
输入存款金额：-123
输入了错误的存款金额（负数）！需要重新输入！
输入存款金额：120
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
3
账户余额：220.0
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
2
输入取款金额：3300
输入了错误的取款金额（负数或超出限额）！需要重新输入！
输入取款金额：115.5
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
3
账户余额：104.5
```

```
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
5
最近的6个事务如下所示：
事务0: 事务类型: Deposit,余额变化: +120.0
事务1: 事务类型: Withdraw,余额变化: -115.5
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
6

选择要使用的功能：
1. 开户
2. 登录账户
3. 退出系统
1
选择账户类型(1: CashAccount, 2: CreditAccount)：2
输入信用账户名称：张宏泽
输入账户密码：zhzmm123
输入初始储蓄金额：1200
输入信用额度：1000
您的账户已成功创建！账户ID为2!
选择要使用的功能：
1. 开户
2. 登录账户
3. 退出系统
2
输入账户名称：张宏泽
输入账户密码：zhzmm123
成功登录!
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
2
输入取款金额：2500
输入了错误的取款金额（负数或超出限额）！需要重新输入!
输入取款金额：-12344
输入了错误的取款金额（负数或超出限额）！需要重新输入!
输入取款金额：1500
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
3
账户余额：700.0
信用额度：1000.0，剩余额度：0.0
```

```
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
5
最近的6个事务如下所示：
事务0: 事务类型: Withdraw,余额变化: -1500.0
选择要使用的功能：
1. 存款
2. 取款
3. 检查账户余额
4. 改变账户名称
5. 打印最近6个事务
6. 登出账户
6
选择要使用的功能：
1. 开户
2. 登录账户
3. 退出系统
3
```

如上所示，实验结果正确。

五、实验小结

本次实验中，实现了银行账户系统的异常处理，包括抛出异常、捕获异常并处理。对异常的理解和使用，是了解现代编程思想十分重要的一环。最近我在看《深入理解计算机系统》一书，书中的第八章异常控制流，则站在更加宽泛、更加底层的角度剖析异常处理机制，使我收获颇多，只是在编程时还需多加磨炼。