实验内容	第 08 周实验　异常处理			成 绩	
姓　名	王秋锋	学号	2015111948	班 级	计算机-03 班
专　业	计算机科学与技术			日 期	2017 年　10 月　25 日

【实验目的】--异常处理

- ◆ 掌握异常处理机制
- ◆ 掌握自定义异常类
- ◆ 深入掌握异常处理机制的应用

【实验内容】

1、编辑、编译、运行下面 java 程序

```
public class TestException
{
    public static void main(String[] args) {
        int[] intArray = new int[3];
        try {
            for (int i = 0; i <= intArray.length; i++) {
                intArray[i] = i;
                System.out.println("intArray[" + i + "] = " + intArray[i]);
                System.out.println("intArray[" + i + "]模  " + (i - 2) + "的值:    "
                        + intArray[i] % (i - 2));
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("intArray 数组下标越界异常。");
        } catch (ArithmeticException e) {
            System.out.println("除数为 0 异常。");
        }
        System.out.println("程序正常结束。");
    }
}
```

要求：

（1）分析该程序，写出运行结果

【实验结果与分析】

运行结果：

```
intArray[0] = 0
intArray[0]模-2的值:  0
intArray[1] = 1
intArray[1]模-1的值:  0
intArray[2] = 2
除数为0异常。
程序正常结束。
```

程序分析：

此程序有 try，catch 的异常捕获机制，在 i=2 时，（i-2）=0,此时模数为 0，因而抛出了算术异常输出除数为 0 异常的提示

2、编辑、编译、运行下面 java 程序,理解异常

```
import java.util.*;
public class TestException
{
    static int quotient(int x, int y) throws MyException    // 定义方法抛出异常
    {
        if (y < 0) { // 判断参数是否小于 0
            throw new MyException("除数不能是负数"); // 异常信息
        }
        return x/y; // 返回值
    }

    public static void main(String args[]) { // 主方法
        int    a =12, b, result;
    Scanner reader=new Scanner(System.in);
        System.out.println("请一个整数,作为除数：");

        b=reader.nextInt();
        System.out.printf("a=%d    b=%d\n",a,b);

        try { // try 语句包含可能发生异常的语句
                result = quotient(a, b); // 调用方法 quotient()
                System.out.printf("a=%d    b=%d    result=%d",a,b,result);
        } catch (MyException e) { // 处理自定义异常
                System.out.println(e.getMessage()); // 输出异常信息
        } catch (ArithmeticException e) { // 处理 ArithmeticException 异常
                System.out.println("除数不能为 0"); // 输出提示信息
        } catch (Exception e) { // 处理其他异常
                System.out.println("程序发生了其他的异常"); // 输出提示信息
        }
    }

}
```

```
class MyException extends Exception { // 创建自定义异常类
    String message; // 定义 String 类型变量
    public MyException(String ErrorMessagr) { // 父类方法
        message = ErrorMessagr;
    }

    public String getMessage() { // 覆盖 getMessage()方法
        return message;
    }
}
```

要求：

（1）分析该程序，写出运行结果

提示：分别输入，0，和非 0 整数

【实验结果与分析】

根据输入的不同，该程序有四种不同的输出

1) 输入 5，符合输入要求规范，此时 try 中的程序没有任何异常抛出，三个 catch 语句都不执行，直接输出 a,b,result 的结果。

```
请一个整数,作为除数:
5
a=12  b=5
a=12  b=5  result=2
```

2) 输入为 0，然而除数不能为 0，系统抛出 ArithmeticException e 异常，该异常被第二个 catch (ArithmeticException e)捕获，输出除数不能为 0 的提示信息

```
请一个整数,作为除数:
0
a=12  b=0
除数不能为0
```

3、编辑并运行下面程序，理解异常处理机制

```
import java.io.*;
import java.util.*;
class InsufficientFundsException extends Exception
{
    private double amount;
    public InsufficientFundsException(double amount)
    {
        this.amount = amount;
    }
    public double getAmount()
    {
        return amount;
    }
```

```
}

// File Name CheckingAccount.java

class CheckingAccount
{
    private double balance;
    private int number;
    public CheckingAccount(int number)
    {
        this.number = number;
    }

    public void deposit(double amount)
    {
        balance += amount;
    }
    public void withdraw(double amount) throws
                                        InsufficientFundsException
    {
        if(amount <= balance)
        {
            balance -= amount;
        }
        else
        {
            double needs = amount - balance;
            throw new InsufficientFundsException(needs);
        }
    }
    public double getBalance()
    {
        return balance;
    }
    public int getNumber()
    {
        return number;
    }
}


// File Name BankDemo.java
public class TestException
{
    public static void main(String [] args)
    {
        CheckingAccount c = new CheckingAccount(101);
    Scanner reader=new Scanner(System.in);
        System.out.println("请输入存款数：");
        c.deposit(reader.nextDouble());

        try
        {
            System.out.println("Withdrawing $100...");
            c.withdraw(100.00);
            System.out.println("Withdrawing $600...");
            c.withdraw(600.00);
        }catch(InsufficientFundsException e)
        {
            System.out.println("Sorry, but you are short $" + e.getAmount());
```

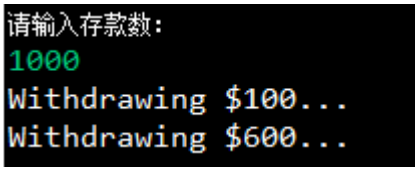
```
            e.printStackTrace();
        }
    }
}
```

要求：运行程序，给出正确的程序运行结果，理解异常处理机制，自定义异常类。

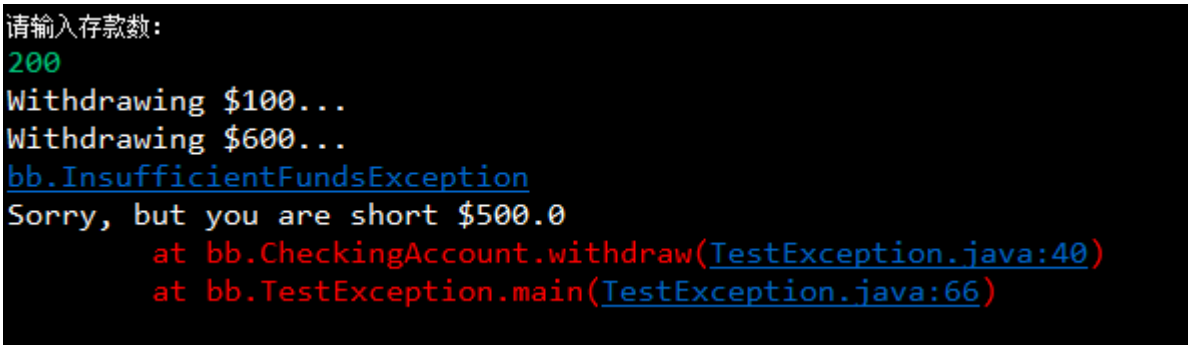
提示：分别输入，大于 800，和小于 800 的数验证，并理解程序

【实验结果与分析】

1) 输入为大于等于 800 时程序正常运行没有异常抛出。



2) 输入为小于 800 时，程序抛出异常，并显示不足的钱数。



4、编辑并运行下面程序，理解包装类的使用

```
public class TestBoxType {
    public static void main(String[] args)
    {
        int a = 1;
        String str = "123";
        Integer b = a;   //自动装箱
        int c = b;//自动拆箱
        Integer d = new Integer(a);   //手动装箱
        int e = d.intValue();//手动拆箱
        /*
            * 实验 int 类型与 String 类型之间的互转
        */
        //int 转换 String
        int i = 123;
        //转换开始
        String s1 = i+"";//第一种方法
        String s2 = String.valueOf(i);//第二种方法
        String s3 = Integer.toString(i);//第三种方法
        //下面用于输出转换结果
```

```
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        //String 转换 int
        String s = "456";
        //开始转换
        int i1 = Integer.valueOf(s);//第一种方法
        int i2 = Integer.parseInt(s);//第二种方法
        //下面输出转换结果
        System.out.println(i1);
        System.out.println(i2);
    }
}
```

【实验结果与分析】

实验分析：

该程序演示了实验 int 类型与 String 类型之间的互相转化的几种方法

1. int 类型转 String 类型的三种方法：

1) i+"";　　//i 为 int 类型

2) String.valueOf(int i)

3) Integer.toString(int i)

2. String 类型转 int 类型的两种方法：

1) Integer.valueOf(String s);

2) Integer.parseInt(String s);

运行结果：

```
123
123
123
456
456
```

5、根据下面的要求，编辑编译程序，并对所编写的出进行测试

假定银行的一个存取款系统有两类客户，一类是现金用户，一类是信用卡用户。银行对每个客户都要登记其姓名 name，并为之分配一个唯一的账户号码 id，现金用户还要记录其卡的类型(工资卡、借记卡、理财卡)，而信用卡用户则根据其信用级别有一定的透支限额 lineOfCredit（A 级 10000 元、B 级 5000 元、C 级 2000 元、D 级 1000 元）。每种客户都可以实现存 deposit、取 withdraw、和查询余额 getBalance，信用卡用户还可以查询透支情况 findOverdraw。对于现金用户，每次取款操作只能在账户实际额度 balance 内操作，允许现金用户改变自己的帐户类型。

（1）分析有哪些属性和方法可以作为两个子类的共同属性和方法，写出抽象类 Account 定义。

```
abstractclass Account {
```

```
    private String name;
    public String id;
    privatedoublebalance;

    publicvoid setBalance(double balance) {
        this.balance = balance;
    }

    publicdouble getBalance() {
        returnbalance;
    }

    public Account(String name, String id, double balance) {
        super();
        this.name = name;
        this.id = id;
        this.balance = balance;
    }

    public Account(String name, String id) {
        super();
        this.name = name;
        this.id = id;
    }
    publicvoid deposit(double amount){
        this.balance+=amount;
    }

abstractvoid withdraw(double amount);
}
```

（2）分析 CashAccount 有那些新增的属性和方法，定义一个继承于 Account 的子类 CashAccount。

```
publicclass CashAccount extends Account {
    public String cashsort;
    public String getCashsort() {
        returncashsort;
    }
    publicvoid setCashsort(String cashsort) {
        this.cashsort = cashsort;
    }
    public CashAccount(String name, String id, double balance, String cashsort) {
        super(name, id, balance);
        this.cashsort = cashsort;
    }
```

```
    publicvoid  withdraw(double amount){
        if(this.getBalance()>=amount){
this.setBalance(this.getBalance()-amount);

        }
        else{
            System.out.println("错误");
        }

    }

}
```

（3）分析 CreditAccount 有那些新增的属性和方法，然后定义一个继承于 Account 的子类 CreditAccount，添加增加的属性和方法。

```
publicclass CreditAccount extends Account {
    doublestaticoverdraw;
    doubleoverdraw;

    public CreditAccount(String name, String id, double balance,
            double staticoverdraw, double overdraw) {
        super(name, id, balance);

        this.staticoverdraw = staticoverdraw;
        this.overdraw = overdraw;
    }
    publicvoid  findOverdraw(){
        if(this.getBalance()<0){
            System.out.print("透支");
            overdraw=-(this.getBalance());
        }
        else {
            System.out.println("未透支");

        }
    }
    publicvoid  withdraw(double amount){
        if(this.getBalance()+staticoverdraw>=amount){
this.setBalance(this.getBalance()-amount);

        }
        else{
            System.out.println("错误");
        }
```

```
    }
}
```

（4） 请按照要求编写一个程序 **Test**，用你所定义的类完成下列业务操作（**并采用异常处理机制处理取款，对于现金用户，如果取钱数超过余额，抛出异常**）。

A、用 Account 作为类型定义两个变量 credit 和 debit，分别引用 CreditAccount 和 CashAccount 的对象，并完成存款 500 元的操作。

B、每个对象完成取款 200 元的操作后再次取款 400 元，请输出各自的余额。

C、可以通过 credit 查看引用对象的透支额吗，如果不能，怎样修改可以查看？

```
publicclass Test {

    publicstaticvoid main(String[] args){
    Account credit=new CreditAccount("zqq","20094070149",0,500,0);
  Account debit=new CashAccount("zsq","20094070101",0,"借记卡");


  credit.deposit(500);
  debit.deposit(500);

  credit.withdraw(200);
  debit.withdraw(200);
  credit.withdraw(400);
  debit.withdraw(400);
  System.out.println("credit balance  "+credit.getBalance());
  System.out.println("cash balance "+debit.getBalance());

}}
```

要求：编辑上面的程序，给出运行结果，并对程序进行分析，理解掌握类、抽象类和继承的概念与应用。同学们，也可以基于自己的理解，对上述程序进行扩展，模拟银行业务，编写相应的应用程序

【实验结果与分析】

1) 两个子类的共同属性和方法

共同属性：

private String name; //姓名

public String id; //账户号码

private double balance;//账户实际额度

共同方法：

public void setBalance(double balance)//设置余额

public double getBalance()//查询余额

public Account(String name, String id, double balance) //构造函数

public Account(String name, String id) //构造函数

public void deposit(double amount) //存款

abstract void withdraw(double amount); //取款

2）CashAccount 新增属性和方法
新增属性：
public String cashsort;//账户类型
新增方法：

public String getCashsort() //获取账户类型

public void setCashsort(String cashsort) //设置账户类型

public CashAccount(String name, String id, double balance, String cashsort)// 构造函数

3）CreditAccount 新增属性和方法
新增属性：

double staticoverdraw;//透支限额

double overdraw;//透支金额

新增方法：

public CreditAccount(String name, String id, double balance,double staticoverdraw, double overdraw) //

构造函数

4）

实验代码：

```
import java.io.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
class Fund_Exception extends Exception{
    private double amount;
    public Fund_Exception(double amount){
        this.amount = amount;
    }
    public double getAmount(){
        return amount;
        }
    }
```

```
abstract class Account{
    private String name;
    public String id;
    private double balance;
    public void setBalance(double balance){
        this.balance = balance;
    }
    public double getBalance(){
        return balance;
    }
    public Account(String name, String id, double balance){
        super();
        this.name = name;
        this.id = id;
        this.balance = balance;
    }
    public Account(String name, String id) {
        super();
        this.name = name;
        this.id = id;
    }
    public void deposit(double amount){
        this.balance+=amount;
    }
    abstract void withdraw(double amount)throws Fund_Exception;
    abstract void findOverdraw();
}
class CashAccount extends Account {
    public String cashsort;
    public String getCashsort() {
        return cashsort;
    }
    public void setCashsort(String cashsort) {
        this.cashsort = cashsort;
    }
    public CashAccount(String name, String id, double balance, String
cashsort) {
        super(name, id, balance);
        this.cashsort = cashsort;
    }
    public void  withdraw(double amount)throws Fund_Exception {
          if(this.getBalance()>=amount){
             this.setBalance(this.getBalance()-amount);
          }
          else{
             double needs = amount - this.getBalance();
```

```
                throw new Fund_Exception(needs);
            }
        }
        void findOverdraw(){}
    }

    class CreditAccount extends Account {
        double staticoverdraw;
        double overdraw;
        public CreditAccount(String name, String id, double balance,double
staticoverdraw, double overdraw) {
            super(name, id, balance);
            this.staticoverdraw = staticoverdraw;
            this.overdraw = overdraw;
        }
        public void  findOverdraw(){
            if(this.getBalance()<0){
                System.out.println("透支");
                overdraw=-(this.getBalance());
            }
            else {
                System.out.println("未透支");

            }
        }
        public void  withdraw(double amount){
            if(this.getBalance()+staticoverdraw>=amount){
                this.setBalance(this.getBalance()-amount);
            }
            else{
                System.out.println("错误");
            }
        }
    }

    public class Test {
        public static void main(String[] args){
            Account credit=new CreditAccount("zqq","20094070149",0,500,0);
            Account debit=new CashAccount("zsq","20094070101",0,"借记卡");
            credit.deposit(500);
            debit.deposit(500);
            try{
                credit.withdraw(200);
                debit.withdraw(200);
                credit.withdraw(400);
                debit.withdraw(400);
```

```
            }
            catch(Fund_Exception e){
                System.out.println("Your CashAccount overdraw $" + e.getAmount());
            }
            finally{
                System.out.println("credit balance  "+credit.getBalance());
                System.out.println("cash balance "+debit.getBalance());
                credit.findOverdraw();
            }
        }
    }
```

运行结果：

```
Your CashAccount overdraw $100.0
credit balance  -100.0
cash balance 300.0
透支
```

实验分析：

原程序中由于父类中没有 findOverdraw()抽象类,所以不能通过 credit 查看引用对象的透支额,修改的话需要在父类中加一个 abstract void findOverdraw()

6、设计一个程序实现 0～100 中整数的加、减、乘、除运算。如果操作数不在 0～100 范围内，抛出一个自定义异常 .用 JOptionPane 显示输入、输出

要求：按照要求编写程序，给出程序运行结果。

【实验结果与分析】

实验代码：

```
import java.io.*;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
class Number_Exception extends Exception{//创建异常类
    String message;
    public Number_Exception(String ErrorMessagr){   //父类方法
        this.message = ErrorMessagr;
    }
    public String getMessage(){     //覆盖getMessage()方法
        return message;
    }
}
```

```
public class Demo{
   static int JOptionPane(int x,int y)throws Number_Exception{  //定义方法抛出异常
      if(x>100||x<0||y>100||y<0){  //判断参数
         throw new Number_Exception("Number is exceed 100 or under 0 "); //抛出异常
      }
      return x/y;
   }
   public static void main(String args[]) {
      System.out.println("Input a and b");
      Scanner sc=new Scanner(System.in);
      try{
         int a=sc.nextInt(); int b=sc.nextInt();
         JOptionPane(a,b);
         System.out.println("a+b = "+(a+b));
         System.out.println("a-b = "+(a-b));
         System.out.println("a*b = "+(a*b));
         System.out.println("a/b = "+(a/b));
      }
      catch(Number_Exception e){   //处理自定义异常
         System.out.println(e.getMessage());
      }
      catch (ArithmeticException e) { // 处理ArithmeticException异常
          System.out.println("除数不能为0");
      }
      catch (Exception e) { // 处理其他异常
          System.out.println("程序发生了其他的异常");
      }
   }
}
```

运行结果：

```
Input a and b

34 2
a+b = 36
a-b = 32
a*b = 68
a/b = 17
```