异常处理

陈德创　　19030500217
西安电子科技大学

日期：2020 年 5 月 19 日

目录

1 小组名单

学号	姓名	工作
19030500217	陈德创	完成程序、联系例程、调试、小组讨论

2 题目

1. PPT 练习

   P20,P22,P25,P26,P30,P31,P32

2. 编写程序，实践自定义异常的使用，要求：

   1). 设计一个 MyNatrualNum 类，该类的构造方法定义为私有方法。该类表述数据范围在 0 到 100 的自然数

   2). MyNatrualNum 类中设计一个 getInstance 静态方法，该方法验证用户输入，满足 MyNatrualNum 类数据条件时调用构造方法，并返回对象实例，不满足时，抛出异常

   3). 设计一个自定义异常类 SelfException，该类表述 MyNatrualNum 类的创建异常，根据用户输入的情况，可能返回数据过大信息或数据过小信息或数据格式错误信息

   4). 构建测试类，该类实现与用户的交互，向用户提示操作信息，并接收用户的操作请求

3 练习

3.1 P20. 程序输出

输出为：

```
Hello World!
This is always printed
Hello!
This is always printed
Hello!
This is always printed
Re-setting Index Value
This is always printed
```

死循环，因为在 catch 块中我们对 i 进行了重新设置。无论如何都会执行 finally 块，这也是设计它的初衷吧。当然 $ArrayIndexOutOfBoundsException$ 是免检异常，我们更应该在编码的时候小心注意。

3.2 P22.writeList

我们可以依次列出多个 catch 块来捕捉可能出现的多个异常，不过要注意的是一个 catch 只能捕捉一种（即某类即其子类）的异常，每个 try-catch 每次只会捕捉一个异常，因为当 try 中

出现异常会中断代码运行，依次向下判断 catch 的异常类型。所以子类异常应当在父类异常之上（不恰当地，审查条件逐渐变得宽松）。

3.3 P25.TestListOfNumbersDeclared

打开一个文件并写入，如下：异常来自于 *FileWriter* 方法。异常很多一部分都是处理 IO（或

```
Value at: 0 = 0
Value at: 1 = 1
Value at: 2 = 2
Value at: 3 = 3
Value at: 4 = 4
Value at: 5 = 5
Value at: 6 = 6
Value at: 7 = 7
Value at: 8 = 8
Value at: 9 = 9

```

者不符合预期的人机交互）的吧。

3.4 P26. 异常处理过程

```
try{
  s1; s2; s3;
} catch(ExceptionType1 e){}
catch(ExceptionType2 e){}
finally{s4}
s5;
```

```
try{
  s1; s2; s3;
} catch(ExceptionType1 e){}
catch(ExceptionType2 e){}
s5;
```

异常的好处在于[1]：

1. 将错误处理代码与「常规」代码分离优化异常状况处理，如果采用返回值或者判断的方法，程序将会变得冗长不易理解。更何况没有返回值的情况以及没有合适的表示错误的返回值的情况。
2. 传播错误调用堆栈通过异常机制我们可以方便的追踪方法调用情况，更快速精准地定位到错误代码。
3. 分组和区分错误类型借助面向对象思想，异常可以被区分为各种类并形成继承树，更更贴近现实并且方便处理。进一步，我们可以方便的利用已有的或新建的类，并且统一方便地打印错误信息。

如上图，由于 s2 会抛出异常，s3 一定不会被执行（s2 会终止代码执行），s2 和 s1 一定会执行。finally 块中的 s4 一定会被执行。在 try-catch-finally 中没有返回语句、抛出无法捕捉的异常、break、continue 等的情况下，s5 会被执行。

[1]参考自：JAVA8 官网笔记教程——异常的优点

3.5 P30,P31,P32.ServerTimeOutException

自定义了一个异常类，并且定义了属性 reason 和 port 用于保存异常信息。在 connectMe 中抛出异常，在 findServer 中被捕捉到。如果我们打印异常可以得到如下的调用过程：

```
Server timed out, try another
chapter6.ServerTimeOutException
    at chapter6.ServerTimeOutException.connectMe(ServerTimeOutException.java:26)
    at chapter6.ServerTimeOutException.findServer(ServerTimeOutException.java:34)
    at chapter6.ServerTimeOutException.main(ServerTimeOutException.java:47)
No server avaliable
```

3.6 补充：try-catch-finally 执行顺序

首先，在一般情况下，无论有没有捕获到异常，finally 块都会执行到，除了以下情况[2]：

1). 如果在 try 或 catch 语句中执行了 System.exit(0)

2). 在 finally 之前死循环

3). 在 finally 之前 jvm 崩溃

4). 电源断电

以上情况可以说是相当极端了，需要注意的就是在 try 或者 catch 块中有 return 语句的情况，特别是返回值是什么。

```
1  public class TestReturn {
2
3    private static int test(){
4      int i = 0;
5      try{
6        throw new IOException();
7      } catch (IOException e) {
8        i = 2;
9        return i;
10     }finally {
11       i = 13;
12     }
13   }
14
15   public static void main(String[] args) {
16     int i = test();
17     System.out.println(i);
18   }
19 }
```

比如上面这个例子，返回值为 2，在执行到 *return i* 时，程序会直接去执行 finally 块，但是这时候返回值已经确定了，i 的改变并不会影响到返回值。

当然，如果我们采用引用对象，如下：

[2]Java 异常机制–try catch finally 执行顺序详解

```java
public class TestReturn {

  static class Inner{
    int i;
  }

  private static Inner test(){
    Inner x = new Inner();
    x.i = 0;
    try{
      throw new IOException();
    } catch (IOException e) {
      x.i = 2;
      return x;
    }finally {
      x.i = 13;
    }
  }

  public static void main(String[] args) {
    Inner x = test();
    System.out.println(x.i);
  }
}
```

显而易见的，此时输出为 13，但是如果我们在 finally 块中改变了引用对象，如下

```java
finally {
  x = new Inner();
  x.i = 13;
}
```

此时输出仍为 2，也算很好理解，抓住重点————在 return 语句执行时，返回值已经确定了，在 finally 块中对返回值的改变不会影响原来返回的值。对引用对象的修改实际上是对引用对象所指向的实例的修改，而不是对引用对象值的修改。（感觉 JAVA 中的引用对象就像是一个封装好的好用的指针）

4 题目分析

这次题目相对简单，主要是对异常类的基本应用，并且算法层面的内容很少。其实对于用于输入的处理在大数类的报告中就已经有了。当时我是利用返回值来做的，这次只要更换成异常就可以了。而且由于只用支持 0 100 范围的数字，难度大大降低，而且不必要支持四则运算。对于字符串到数字的解析直接应用 *Integer.praseInt*() 就可以了。

自定义一个异常类，甚至可以不用加任何东西。正如先前一位同学所说的“要的就是继承结构”。

5 程序实现

1). MyNatrualNum 类仅有一个私有属性 value 用于表示值。将构造方法设为私有并设置 *getInstance* 静态方法，在方法中调用 *Interger.praseInt* 方法，这个方法会抛出一个 *NumberFormatException*，是一个免检异常，但是我们这里对其进行捕获，代表输入为非数字（或者数字超过整形范围）的情况，如果捕获了异常，则我们抛出我们自己的 *SelfException*，并储存信息。随后对解析结果进行判断，如果不满足 0 100 的范围，我们仍抛出异常，并储存相应信息。

2). SelfException 类这个没啥好说的了，只需要一个继承结构（继承自 Exception，这个显然是必检异常），留一个接受字符串的构造方法就可以了。

3). Interact 类这个也简单多了，因为只有一种命令。死循环（直到接收到“q”），每次接收一行然后调用 *getInstance* 方法并捕获异常。因为随后调用异常的 *getMessage* 打印结果并作出文字提示就可以了。

6 执行结果

```
Input a STRING to create a number between 0 and 100.('q' to quit)
32
Number 32 created successfully.
123
123 is bigger than 100.
Please try again.
44234
44234 is bigger than 100.
Please try again.
-12312
-12312 is smaller than 0.
Please try again.
werji324
werji324 is not a legal number.
Please try again.
123.3123
123.3123 is not a legal number.
Please try again.
q
Thanks for your using!

Process finished with exit code 0
```

7 个人总结

这次报告算是相对简单多了，也算是张弛有度吧。异常我其实学的不是很好，看来还是不能松懈啊。

革命尚未成功，同志仍需努力。