第五章 异常处理

涉及到课本章节:

- 第5章 异常处理
- 第8章 输入输出流和文件操作

异常处理

5.2 Java异常的处理机制

生活中的异常处理

- **校园生活**：学习中的困难先自己尝试搜索研究，实在没办法了找老师协助
- **日常生活**：发现火情先用学到的灭火知识扑救，实在控制不了拨打119呼叫消防队
- **工作经历**：遇到问题先极尽所能想办法解决，实在解决不了的问题向领导汇报

5.2 checked Exception的处理机制

1 捕获处理

2 向上传递处理

校园卡的异常处理

八、温馨提示

1.消费时切记不要过快刷卡！否则容易造成刷卡失败、卡片冻结等情况。

2.避免冻结的有效办法是保证账户的金额大于20元左右。

3.充值后先找到POS机刷一下，将过渡余额提取到校园卡上。

4.如果出现卡片冻结、无法刷卡、无法转账、余额不对等情况，请持卡到校园卡服务大厅咨询、解决。

5.除充值外所有到服务大厅办理的业务，如解冻、校正、挂失、解挂等，须本人携带有效证件前来办理，他人不可代办。

九、服务信息

1.校园卡网上服务平台：https://card2.sdu.edu.cn:8757/

2.校园卡服务大厅电话：（0631）5688803

监督电话：5688190 5688757

5.2 checked Exception的处理机制

Java 对checked Exception提供

捕获处理或者抛出(传递) 异常的机制。

1 捕获处理

如果一个方法产生了异常(对象)，在程序中自己可以直接捕获处理

使用Java语言描述：

---捕获：

- 使用try-catch-finally语句捕获处理；

捕获异常并进行处理：

```
try
{
    正常的代码；
    调用产生异常对象的方法、语句;
    其他正常的Java语句；
}
catch（异常类名  异常对象名）
{异常处理；}
catch（异常类名 异常对象名）
{异常处理；}
   ......
[finally                          //可以省略finally子句
   {最终处理；}
]
```

异常对象的产生途径:

- **(1) JVM自动生成异常对象**
- **(2) 使用throw语句手动生成异常对象**

举例: ---(1)

JVM自动产生异常

```
public class AutoEx{
   public static void main(String[] args){
            System.out.println(5/0);
   }
}
```

举例：----(2)

throw手动产生异常对象，在本方法内直接捕获异常

```
public void testTry(int age){
    if(age<0)
        throw new Exception("Age must be larger than 0");
        //创建一个异常类对象，并抛出
        ……..
}
```

Catch捕获异常原则(顺序):

- ---抛出异常对象与catch子句参数类型相同
- ---抛出异常对象为catch子句参数类的子类
- ---按照先后顺序捕获抛出异常对象,只捕获一次.

举例：

```
public void testTry(int age){
    try{
        if(age<0)
          throw new SQLException("Age must be larger than 0");
        //创建一个异常类对象，并抛出
        ……..
       }catch(Exception ex){
            System.out.println("父类Exception");
      }catch(SQLException ex){
            System.out.println("子类SQLException");
     }

}
```

try catch 语句中的return逻辑

```
try
 {
    正常的代码；
    return xxx;                          //try中的return
 }
catch（异常类名  异常对象名）
 {异常处理；return xxx; }                //catch中的return
catch（异常类名 异常对象名）
 {异常处理；}
    ......
[finally
   {最终处理；return xxx; }              // finally中的return
]
return xxx;                              // 末尾的return
```

try catch 语句中的return逻辑 let's have a try

- 遇到return
- 立即返回？

在try中执行到return语句时，不会真正的return，而是计算return中的表达式（本例为执行a+b）结果保存到一个临时栈中，继续执行finally中的语句，最后才会从临时栈中取出之前的结果返回。

- 运行结果
- 作何解释？

```
1  public class test {
2      public int add(int a,int b) {
3          try {
4              return a+b;
5          }catch(Exception e){
6              System.out.println("catch语句块");
7          }finally {
8              System.out.println("finally语句块");
9          }
10         return 0;
11     }
12     public static void main(String[] args) {
13         test t=new test();
14         System.out.println("和是"+t.add(9, 34));
15     }
16
17 }
```

try catch 语句中的return逻辑 let's have a test

- 针对猜测
- 做出调整

- 猜测是否
- 依然成立？

```java
public class test {
    public int add(int a,int b) {
        try {
            return a+b;
        }catch(Exception e){
            System.out.println("catch语句块");
        }finally {
            System.out.println("finally语句块");
            a=1;
        }
        return 0;
    }
    public static void main(String[] args) {
        test t=new test();
        System.out.println("和是"+t.add(9, 34));
    }

}
```

try catch 语句中的return逻辑 let's have an analysis

序号	return情况	return逻辑
1	try{}catch(){}finally{}return;	顺序执行
2	try{return;}catch(){}finally{}return;	即执行完try语句块，将return的值保存在临时栈中，再执行finally语句块，之后返回临时栈中的值。
3	try{}catch(){return;}finally{}return;	执行try，执行finally，再执行return;
4	try{}catch(){}finally{return;}	执行finally中的return语句
5	try{return;}catch(){return;}finally{};	根据有无异常执行情况二或情况三。
6	try{return;}catch(){}finally{return;}	执行完try语句块，将return的值保存在临时栈中，再执行finally语句块，因为finally中有return,所以返回finally中的return值
7	try{}catch(){return;}finally{return;}	执行完catch语句块，将return的值保存在临时栈中，再执行finally语句块，因为finally中有return,所以返回finally中的return值
8	try{return;}catch(){return;}finally{return;}	有异常：执行情况七。 无异常：执行情况六。

try catch 语句中的return逻辑

- return语句优先级：　　finally > catch > try

当try catch中的代码执行到return语句时，会先把该return的值存入临时栈中，继续执行finally，执行完finally语句后才返回临时栈中的值。

- 如果finally中有return那就把finally中的返回值当作方法体的返回值返回。
- 如果finally中没有return返回catch中的return值，如果catch中也没有就返回try中的值，如果都没有方法体继续向下执行。

try catch 语句中的return逻辑

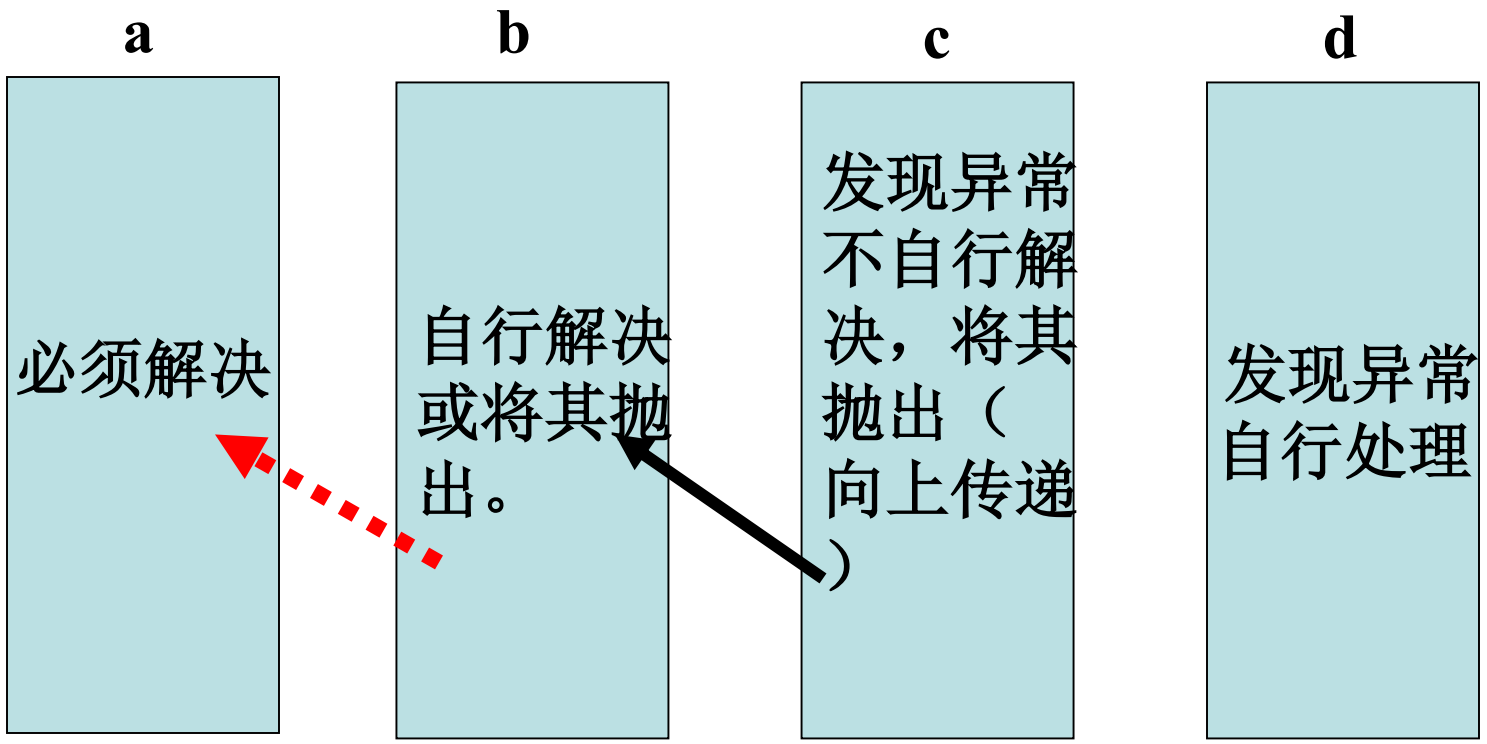
- finally中和try catch finally语句之后只能有一个return。
- 当try中有异常时，位于异常之后的代码（finally除外）都没有意义，此时return不会执行。

2 抛出(传递)

如果一个方法本身能产生异常，但是可以不提供处理，而是抛出传递给调用者;

调用者可以捕获异常使之得到处理，也可以回避异常，这时异常将在调用的堆栈中向上传递，直到被处理。

- 图例 (Java语言的异常处理机制)



Java语言描述：

传递异常：

- 通过**throws**子句在方法头中声明抛出（向上传递）异常

throws传递异常(向上传递异常)

- 当一个方法自己不处理可能产生的异常时，可以将异常传递给方法的调用者。
- 上传异常，在方法声明时添加throws说明。

Test2.java

```
static void read()
        throws IOException，anotherException

{
   FileReader  fin = new FileReader("Students.txt");
   BufferedReader  in = new BufferedReader(fin);
   String line = in.readLine();
   System.out.println(line);
   in.close();
   fin.close();
}
```

FileReader和in.readLine都可能产生IOException，但在read()方法中不处理，所以只能传递给read方法的调用者(使用者)。

```
public static void main(String[] args)
{
   try
   {
          对象名.read();
   }
   catch(IOException e)
   {
          System.out.println(e);
   }
}
```

main方法中调用了read()方法，可能产生IOException，所以必须处理。或者自己处理，或者继续上传。

传递异常举例：

- Test2.java
- 捕获文件读写异常：

```
try{   FileReader  fin = new FileReader("Students.txt");
       BufferedReader  in = new BufferedReader(fin);
       String line = in.readLine();
       System.out.println(line);
       in.close();
       fin.close();
}catch(FileNotFoundException e)
  {
       System.out.println(e);              // FileReader(“Students.txt”)产生
  }
catch(IOException e)              // in.readLine()产生
{      System.out.println(e);
}
```

- 注：因为catch能够捕获子类的异常，若将此文件中的两个子句进行调换如何？

关于异常处理说明：

- **checked Exception**只有
  捕获处理和抛出(传递)两种处理方式，
  二者必须选择 其一。

异常处理

5.1 Java异常的分类

5.2 Java异常的处理机制

5.3 自定义异常类

5.3 自定义异常类

- 创建自定义异常类：
  - 创建异常类只需从**Exception**或其子类派生一个子类即可。
  - 同样分**checked** 和**unchecked Exception**

```
Test.java
class SelectingException extends IOException
{       public SelectingException(){};
        public SelectingException(String msg)
        {       super(msg);
        }
}
```

```
public void select(int aCount) throws SelectingException
{   if(aCount>totalCount)
  { throw new SelectingException(“无法选出” +aCount+“名同学")
   }
}
```

自定义异常类举例：

（1）自定义异常类：

```
class NumberRangeException extends Exception(或者其他类)
  {
        NumberRangeException(String msg)
        {     super(msg);
        }
  }
```

举例：

```
public double Result( ) throws NumberRangeException
 {   double answer=0;

     if ((d1 < 0) || (d1 > 100) ||(d2 < 0) || (d2 > 100)){
      NumberRangeException ee = new
NumberRangeException
       (“输入的数字不在指定的范围！请重新输入。”);
    throw ee;                    }
   answer=d1/d2;
 return answer;
}
```

程序举例：

```
import java.awt.*;
import java.awt.event.*;
public class ExceptionDemo extends Frame implements ActionListener
 {    Label L1,L2;
      TextField tf1, tf2;
      String answerStr;
      double d1,d2;
     ExceptionDemo(String title){
      L1=new Label("请输入0到100之间的整数");
      Panel p=new Panel();
      p.add(L1,BorderLayout.NORTH);
      tf1 = new TextField(6);
      p.add(tf1,BorderLayout.CENTER);
      tf2 = new TextField(6);
      p.add(tf2,BorderLayout.SOUTH);
      L2=new Label(" 两数相除的结果：                    ");
      add(p,BorderLayout.NORTH);
      add(L2,BorderLayout.CENTER);
      Button b=new Button("计算");
      add(b,BorderLayout.SOUTH);
      setSize(400,400);
      setVisible(true);
```

```
public void actionPerformed(ActionEvent evt)
{      try       {
       d1=Double.valueOf(tf1.getText( )).doubleValue( );
       d2=Double.valueOf(tf2.getText( )).doubleValue( );
       double r=Result(d1,d2);

       L2.setText(String.valueOf(r));}
       catch(NumberFormatException e){
         answerStr="输入的必须是数字";
         L2.setText(answerStr);          }
       catch (NumberRangeException ee){
         answerStr = ee.getMessage( );
         L2.setText(answerStr);
       }

}
```

```
public double Result(double d1,double d2 ) throws
NumberRangeException
{
          double answer=0;
                   if ((d1 < 0) || (d1 > 100) ||(d2 < 0) || (d2 > 100))
            {
             NumberRangeException ee = new
                NumberRangeException
             ("输入的数字不在指定的范围！ 请重新输入.");
             throw  ee;
            }
           answer=d1/d2;
        return answer;
       }

  public static void main(String[] args)
   { ExceptionDemo ed=new ExceptionDemo("Exception");
   }
}
```

```
class NumberRangeException extends Exception
{
        NumberRangeException(String msg)
        {                       super(msg);
        }
}
```

补充：throws和throw

- 1.1　throws是方法可能抛出异常的声明。用在声明方法时，表示该方法可能要抛出异常。语法：

- [(修饰符)](返回值类型)(方法名)([参数列表])[throws(异常类)]{.....}
- public void doA(int a) throws Exception1,Exception3{.....}

- 1.2　throw是抛出一个异常，出现在函数体。

- 语法：throw (异常对象);
  throw e;

补充：throws和throw

- throws主要是声明这个方法会抛出这种类型的异常，使它的调用者知道要捕获这个异常。
  throw是具体向外抛异常的动作，所以它是抛出一个异常实例。

- throws说明有那个可能和倾向。
  throw是把那个倾向变成/真实的了。