

异常处理

一、请阅读并运行**AboutException.java**示例，然后通过后面的几页PPT了解**Java**中实现异常处理的基础知识

```
import javax.swing.*;

class AboutException {

    public static void main(String[] a)
    {

        int i=1, j=0, k;

        //double i=1, j=0, k;

        k=i/j;

        //System.out.println(k);

    try
    {

        k = i/j;    // Causes division-by-zero exception

        //throw new Exception("Hello.Exception!");

    }

    catch ( ArithmeticException e)

    {
```

```
System.out.println("被0除. " + e.getMessage());  
  
}  
  
catch (Exception e)  
  
{  
  
if (e instanceof ArithmeticException)  
  
System.out.println("被0除");  
  
else  
  
{  
  
System.out.println(e.getMessage());  
  
}  
  
}  
  
finally  
  
    {  
  
        JOptionPane.showConfirmDialog(null, "OK");  
  
    }  
  
}  
  
}
```

结果：



Java中实现异常处理的基础知识:

异常的对象有两个来源，一是Java运行时环境自动抛出系统生成的异常，而不管你是否愿意捕获和处理，它总要被抛出！比如除数为0的异常。二是程序员自己抛出的异常，这个异常可以是程序员自己定义的，也可以是Java语言中定义的，用throw 关键字抛出异常，这种异常常用来向调用者汇报异常的一些信息。

异常是针对方法来说的，抛出、声明抛出、捕获和处理异常都是在方法中进行的。

Java异常处理通过5个关键字try、catch、throw、throws、finally进行管理。基本过程是用try语句块包住要监视的语句，如果在try语句块内出现异常，则异常会被抛出，你的代码在catch语句块中可以捕获到这个异常并做处理；还有以部分系统生成的异常在Java运行时自动抛出。你也可以通过throws关键字在方法上声明该方法要抛出异常，然后在方法内部通过throw抛出异常对象。finally语句块会在方法执行return之前执行，一般结构如下：

```
try{
    程序代码
}catch(异常类型1 异常的变量名1){
    程序代码
}catch(异常类型2 异常的变量名2){
    程序代码
}finally{
    程序代码
}
```

二、阅读以下代码（CatchWho.java），写出程序运行结果

```
public class CatchWho {

    public static void main(String[] args) {
```

```
try {  
    try {  
        throw new ArrayIndexOutOfBoundsException();  
    }  
    catch(ArrayIndexOutOfBoundsException e) {  
        System.out.println( "ArrayIndexOutOfBoundsException" + "/内层try-catch");  
    }  
    throw new ArithmeticException();  
}  
catch(ArithmeticException e) { //算数  
    System.out.println("发生ArithmeticException");  
}  
catch(ArrayIndexOutOfBoundsException e) { //越界  
    System.out.println( "ArrayIndexOutOfBoundsException" + "/外层try-catch");  
}  
}
```

结果:



三、写出**CatchWho2.java**程序运行的结果

```
public class CatchWho2 {  
    public static void main(String[] args) {  
        try {  
            try {  
                throw new ArrayIndexOutOfBoundsException();  
            }  
            catch(ArithmeticException e) {  
                System.out.println( "ArrayIndexOutOfBoundsException" + "/内层try-catch");  
            }  
            throw new ArithmeticException();  
        }  
        catch(ArithmeticException e) {  
            System.out.println("发生ArithmeticException");  
        }  
    }  
}
```

```

    catch(ArrayIndexOutOfBoundsException e) {
        System.out.println( "ArrayIndexOutOfBoundsException" + "/外层try-catch");
    }
}
}
}

```

结果:



四、 请先阅读 **EmbedFinally.java** 示例，再运行它，观察其输出并进行总结

```

public class EmbeddedFinally {
    public static void main(String args[]) {
        int result;
        try {
            System.out.println("in Level 1");
            try {
                System.out.println("in Level 2");
                // result=100/0; //Level 2
            }
        }
    }
}

```

```
    System.out.println("in Level 3");  
  
    result=100/0; //Level 3  
  
}  
  
catch (Exception e) {  
  
    System.out.println("Level 3:" + e.getClass().toString());  
  
}  
  
finally {  
  
    System.out.println("In Level 3 finally");  
  
}  
  
// result=100/0; //Level 2  
  
}  
  
catch (Exception e) {  
  
    System.out.println("Level 2:" + e.getClass().toString());  
  
}  
  
    finally {  
  
        System.out.println("In Level 2 finally");  
  
    }  
  
    // result = 100 / 0; //level 1  
  
}
```

```
catch (Exception e) {  
    System.out.println("Level 1:" + e.getClass().toString());  
}  
  
finally {  
    System.out.println("In Level 1 finally");  
}  
  
}  
  
}
```

结果:



```
Console X  
<terminated> EmbededFinally [Java Application] C:\Program Files  
in Level 1  
in Level 2  
in Level 3  
Level 3: class java.lang.ArithmeticException  
In Level 3 finally  
In Level 2 finally  
In Level 1 finally
```

当有多层嵌套的**finally**时，异常在不同的层次抛出，在不同的位置抛出，可能会导致不同的**finally**语句块执行顺序。

finally语句块不一定会执行。

请看SystemExitAndFinally.java示例

```
public class SystemExitAndFinally {
```



```
public static void main(String[] args)
{
try{
System.out.println("in main");

throw new Exception("Exception is thrown in main");

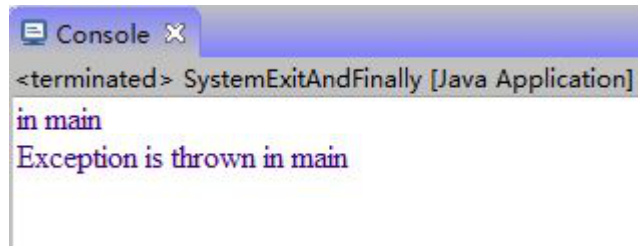
        //System.exit(0);
}

catch(Exception e)
{
System.out.println(e.getMessage());

System.exit(0);
}

finally
{
System.out.println("in finally");
}
}
}
```

结果:



当程序中出现异常时，JVM会依据方法调用顺序依次查找有关的错误处理程序。

可使用`printStackTrace` 和 `getMessage`方法了解异常发生的情况：

`printStackTrace`：打印方法调用堆栈。

每个`Throwable`类的对象都有一个`getMessage`方法，它返回一个字串，这个字串是在`Exception`构造函数中传入的，通常让这一字串包含特定异常的相关信息。

请通过 `PrintExpressionStack.java`示例掌握上述内容。依据对本讲多个示例程序的分析，请自行归纳总结出Java多层嵌套异常处理的基本流程。

```
// UsingExceptions.java
```

```
// Demonstrating the getMessage and printStackTrace
```

```
// methods inherited into all exception classes.
```

```
public class PrintExceptionStack {
```

```
    public static void main( String args[] )
```

```
{
```

```
    try {
```

```
        method1();
```

```
    }
```

```
    catch ( Exception e ) {
```

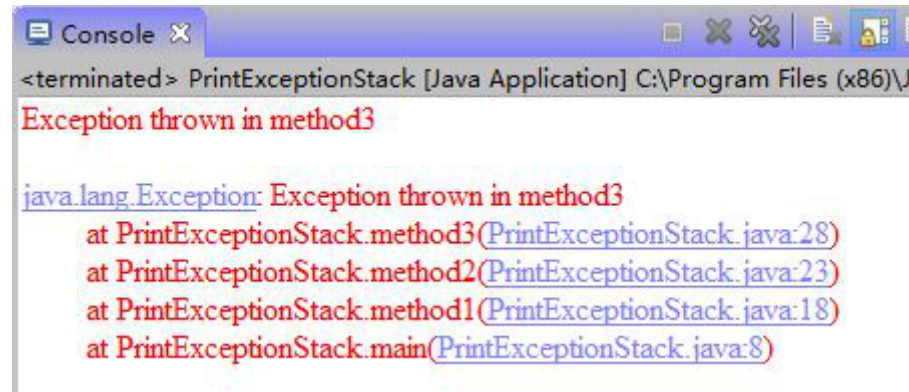
```
        System.err.println( e.getMessage() + "\n" );
        e.printStackTrace();
    }
}

public static void method1() throws Exception
{
    method2();
}

public static void method2() throws Exception
{
    method3();
}

public static void method3() throws Exception
{
    throw new Exception( "Exception thrown in method3" );
}
}
```

结果:



```
Console X
<terminated> PrintExceptionStack [Java Application] C:\Program Files (x86)\J
Exception thrown in method3

java.lang.Exception: Exception thrown in method3
    at PrintExceptionStack.method3(PrintExceptionStack.java:28)
    at PrintExceptionStack.method2(PrintExceptionStack.java:23)
    at PrintExceptionStack.method1(PrintExceptionStack.java:18)
    at PrintExceptionStack.main(PrintExceptionStack.java:8)
```

五、一个方法可以声明抛出多个异常

`int g(float h) throws OneException,TwoException`

`{ }`

ThrowMultiExceptionsDemo.java示例展示了相关特性。

注意一个Java异常处理中的一个比较独特的地方：

当一个方法声明抛出多个异常时，在此方法调用语句处只要catch其中任何一个异常，代码就可以顺利编译。

```
import java.io.*;
```

```
public class ThrowMultiExceptionsDemo {
```

```
    public static void main(String[] args)
```

```
    {
```

```
        try {
```

```
            throwsTest();
```

```

    }

    catch(IOException e) {

        System.out.println("捕捉异常");

    }

}

private static void throwsTest() throws ArithmeticException,IOException {

    System.out.println("这只是一个测试");

    // 程序处理过程假设发生异常

    throw new IOException();

    //throw new ArithmeticException();

}

}

```

结果:



一个子类的**throws**子句抛出的异常，不能是其基类同名方法抛出的异常对象的父类。

OverrideThrows.java示例展示了Java的这个语法特性。

```
import java.io.*;
```

```
public class OverrideThrows
{
    public void test()throws IOException
    {
        FileInputStream fis = new FileInputStream("a.txt");
    }
}

class Sub extends OverrideThrows
{
    //如果test方法声明抛出了比父类方法更大的异常,比如Exception
    //则代码将无法编译.....

    public void test() throws FileNotFoundException
    {
        //...
    }
}
```

六、编写一个程序，此程序在运行时要求用户输入一个整数，代表某门课的考试成绩，程序接着给出“不及格”、“及格”、“中”、“良”、“优”的结论

要求程序必须具备足够的健壮性，不管用户输入什么样的内容，都不会崩溃。

```
import java.util.Scanner;

public class Grade {

    public static void main(String[] args) throws NumberFormatException {

        double score;

        try{

            Scanner in=new Scanner(System.in);

            System.out.println("请输入成绩: ");

            score=in.nextDouble();

            if((score>=0)&&(score<60)){

                System.out.println("不及格");

            }

            else if((score>=60)&&(score<70)){

                System.out.println("及格");

            }

            else if((score>=70)&&(score<80)){

                System.out.println("中");

            }

            else if((score>=80)&&(score<90)){

                System.out.println("良");

            }

        }

    }

}
```

```
}  
  
else if((score>=90)&&(score<=100)){  
    System.out.println("优");  
}  
  
else if(score<0||score>100)  
    System.out.println("输入数字过大或过小");  
  
else  
    throw new Exception();  
}  
  
catch(Exception e){  
    System.out.println("输入的不是数字，请重新输入正确的数字");  
}  
}  
}
```

结果：



