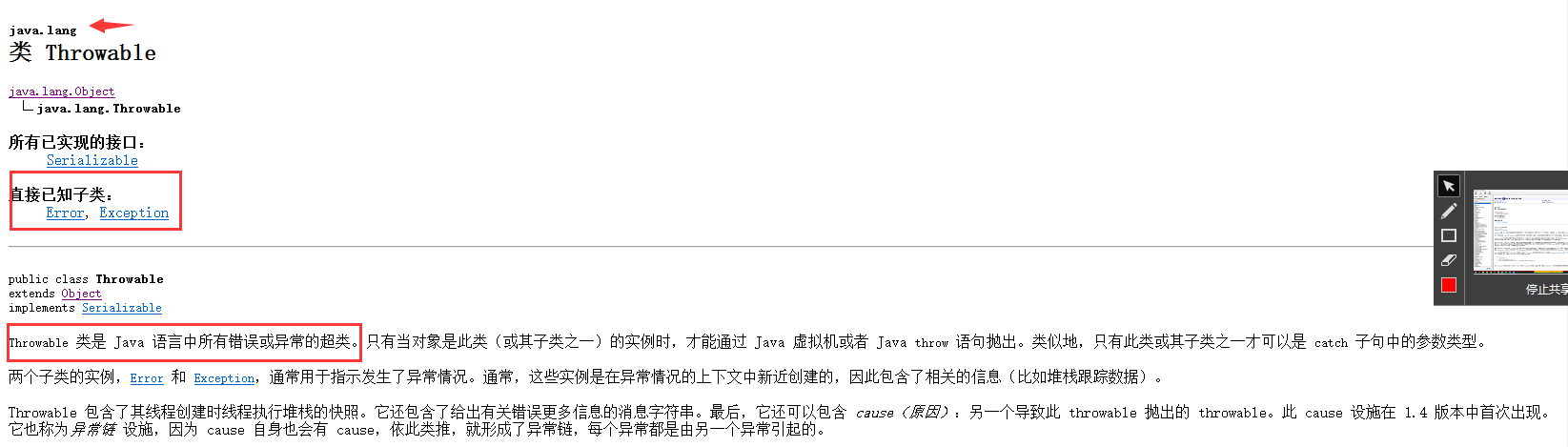
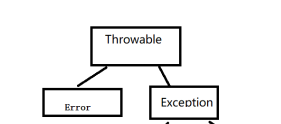
# 回顾

1. Object:toString()/equals()/clone():浅克隆.深克隆:克隆 实现克隆接口的对象本身,也会把克隆对象中关联的对象一起克隆
2. Finalize() :每个对象都有;是垃圾回收器调用;当垃圾回收器检测到一个对象很久都没有引用的时候,会去调用该对象的Finalize()进行回收
3. 日期时间对象:Date(掌握获取当前系统时间) Calendar:api里面获取/设置时间的方法
4. Date与Calendar互转:getTime() setTime()->Date转Calendar
5. SimpleDateFormat辅助类:(创建对象时,有参构造放的是格式:yyyy年MM月dd日 HH时mm分ss秒)将日期格式化输出 format()->Date转字符串 parse()->字符串转Date

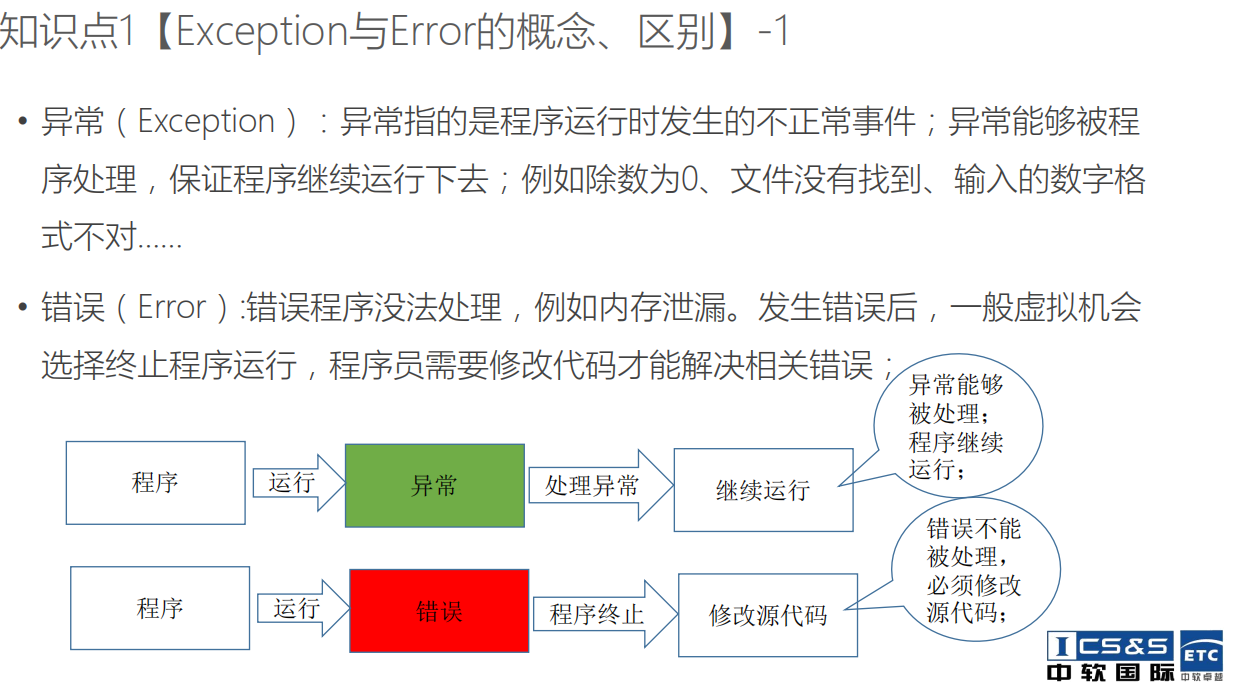
# 作业

# Throwable





# 异常Exception



问题引入:

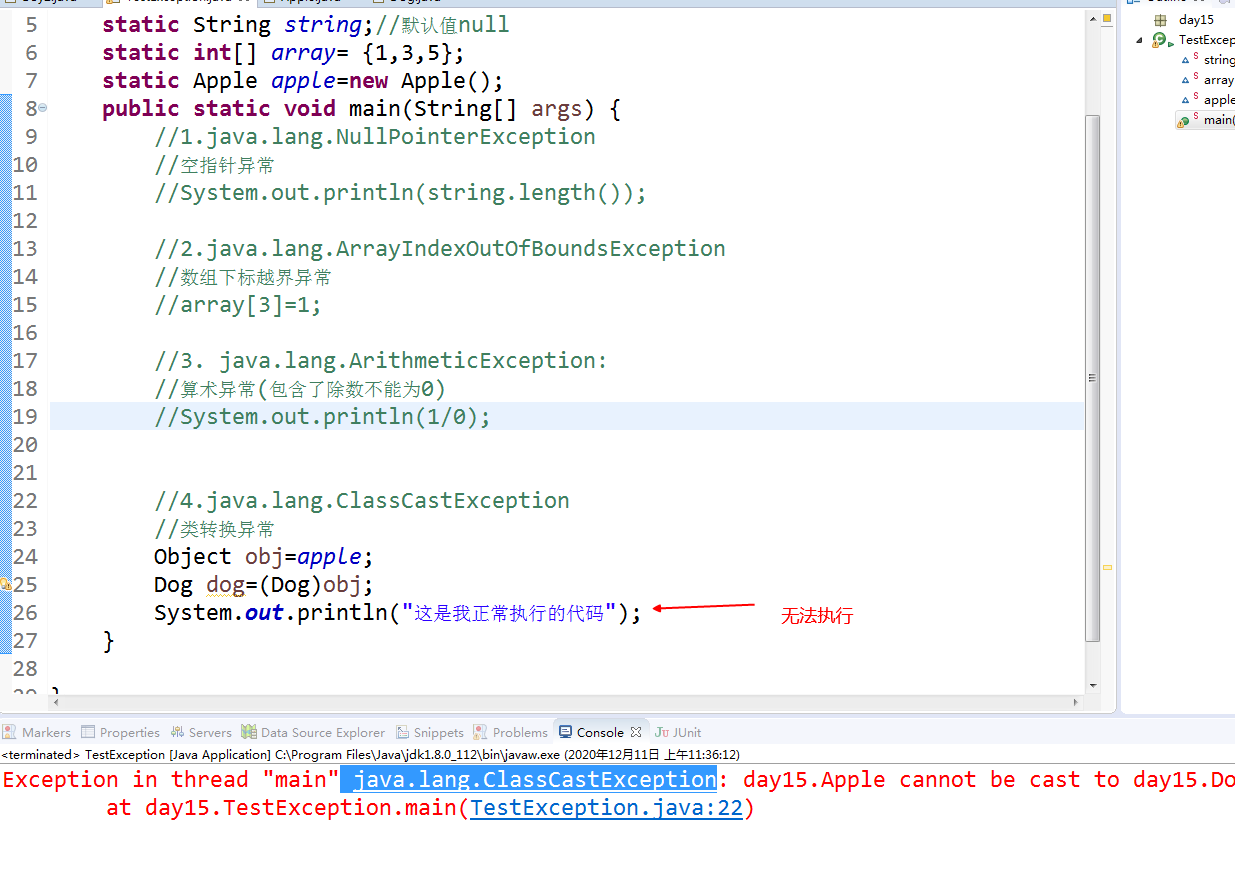
## 常见的异常:

1. java.lang.NullPointerException 空指针异常
2. ArrayIndexOutOfBoundsException 数组越界异常
3. java.lang.ArithmeticException: / by zero 算术异常(包含了除数不能为0的异常)
4. java.lang.ClassCastException 类型转换异常

出现了上面的问题带来的结果是什么?

1. 控制台(界面),上会显示异常的信息,给用户带来不来的体验
2. 发生异常的时候,程序立即结束,后续代码也无法执行

基于以上2点,我们要做异常的处理,同时还要知道有哪些常见的异常类型,这样才能针对性的处理



## 异常处理

如何对异常进行处理?

### 方式1:try-catch-finally

**try{  
   //监视作用  
  
}catch (异常){  
  //捕获并处理  
  
}finally{  
  //不管是否有异常发生，finally语句块中的语句始终保证被执行  
  //finally语句块主要用于解决资源泄露问题  
}**

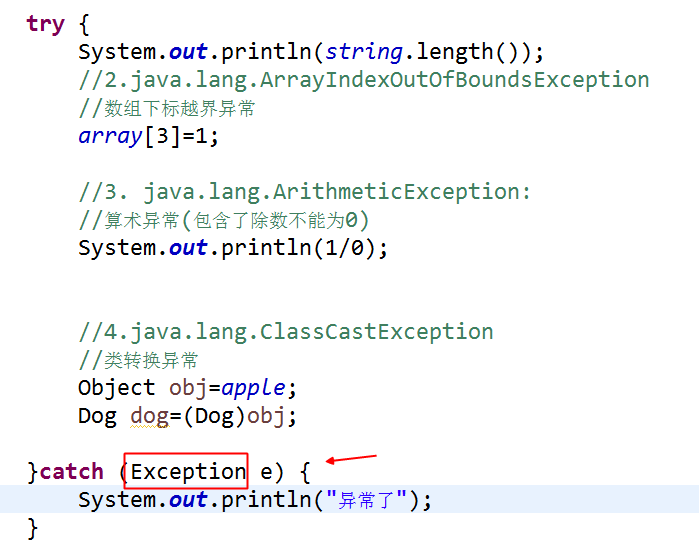
可能发生异常的代码块,选中,右键 找到surround with ->try catch

### try-catch

### try-catch-catch....

try里面可以放很多可能发生异常的代码,但是一次只能捕捉一个来处理

不那么细分

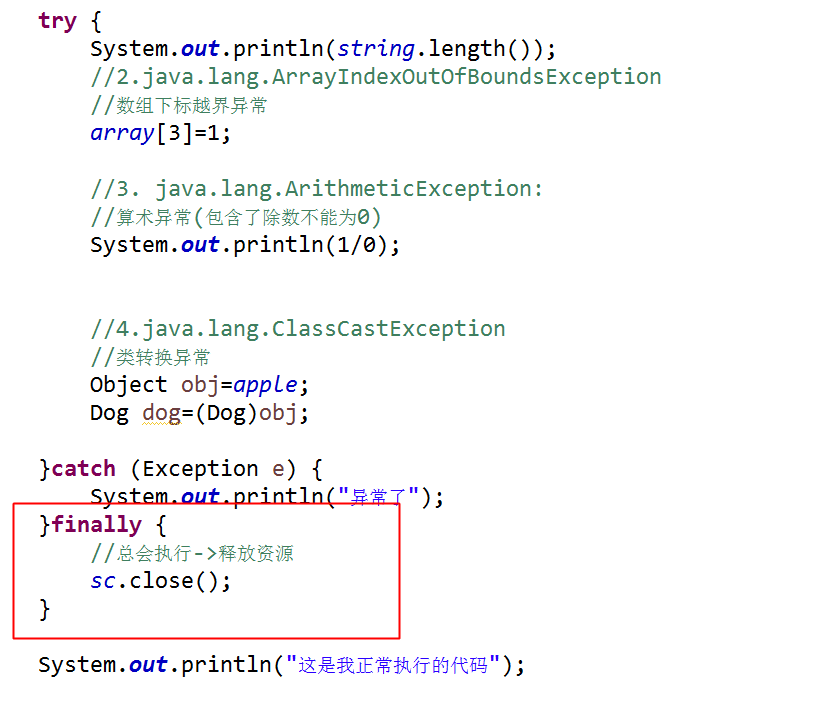


还有一种: 注意：try的后面可以有多重catch,但是注意层级关系：越高越往下写。



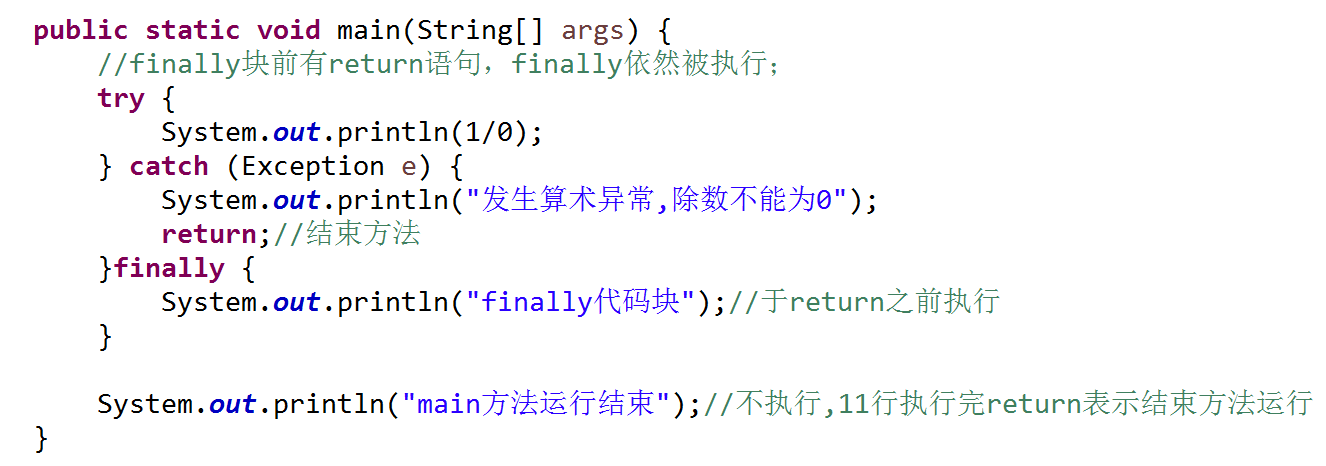
### finally->总会执行

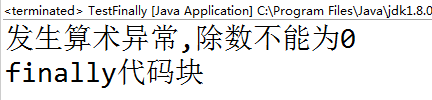
①try catch   
②try 多个catch   
③try..finally  
④try catch finally



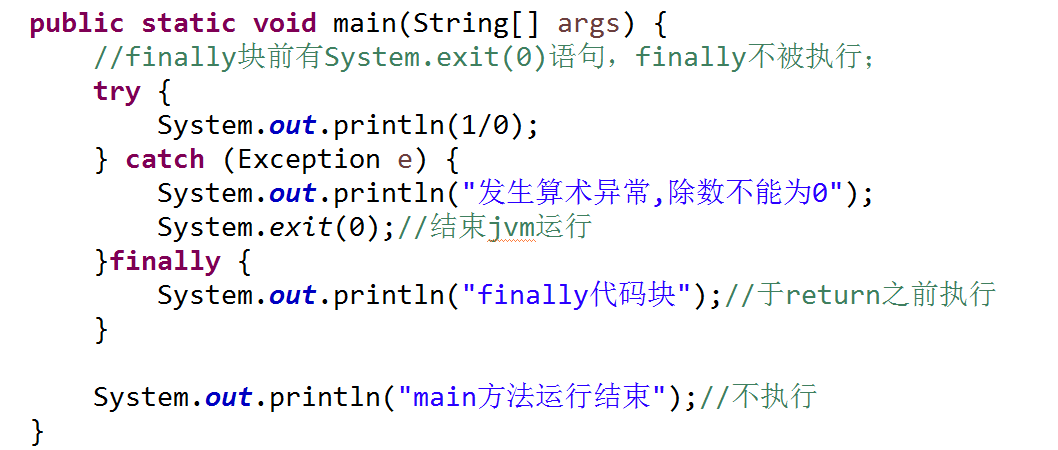
### finally注意事项

#### finally块前有return语句，finally依然被执行；

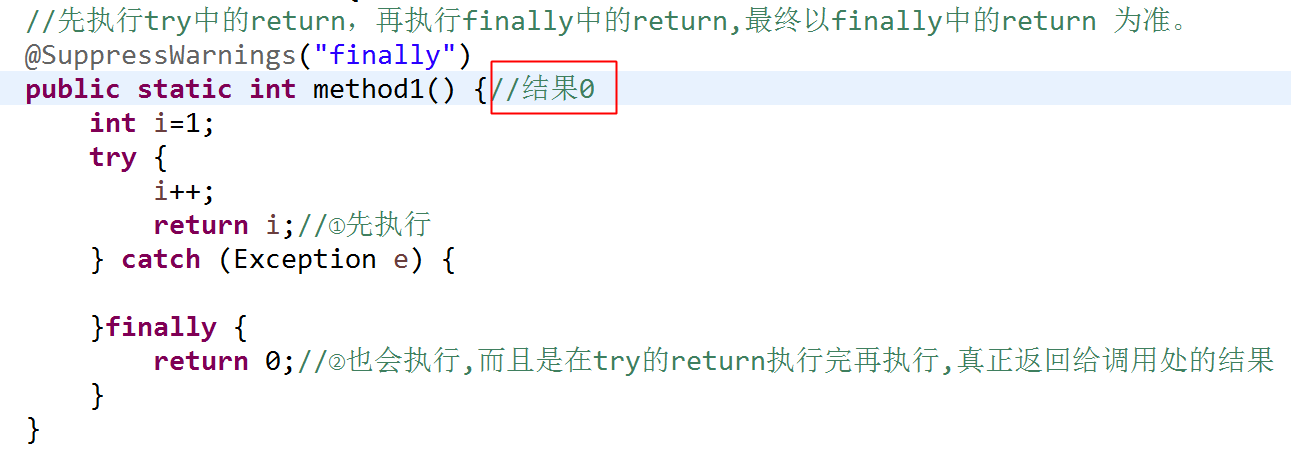




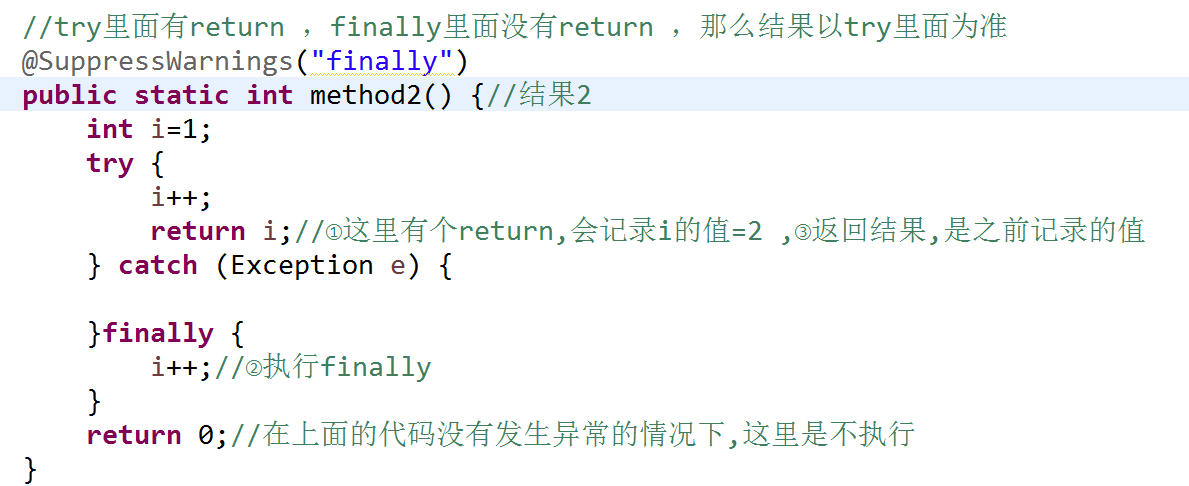
#### 2.finally块前有System.exit(0)语句，finally不被执行；





**两个经典题：**1.没有异常的情况下,测试return语句在try和finally中执行顺序  
先执行try中的return，再执行finally中的return,最终以finally中的return 为准。  


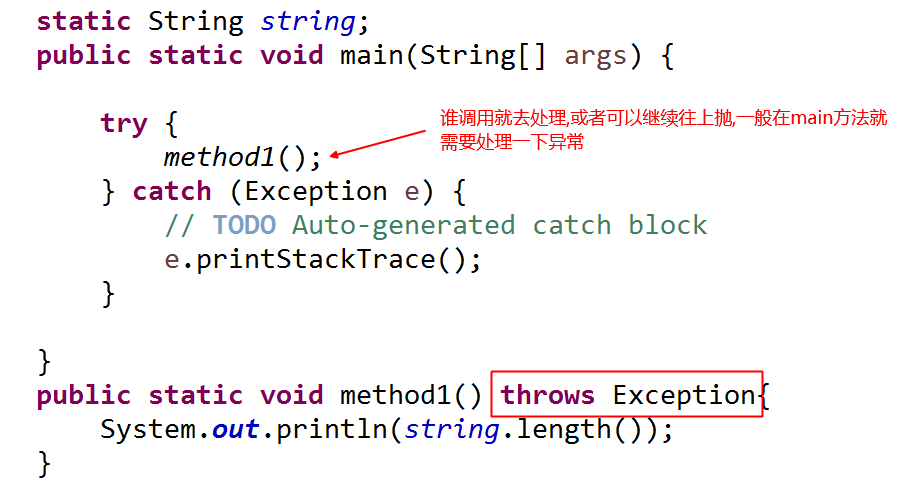
2. try里面有return ，finally里面没有return ，那么结果以try里面为准



### 方式2:throws(抛出异常)

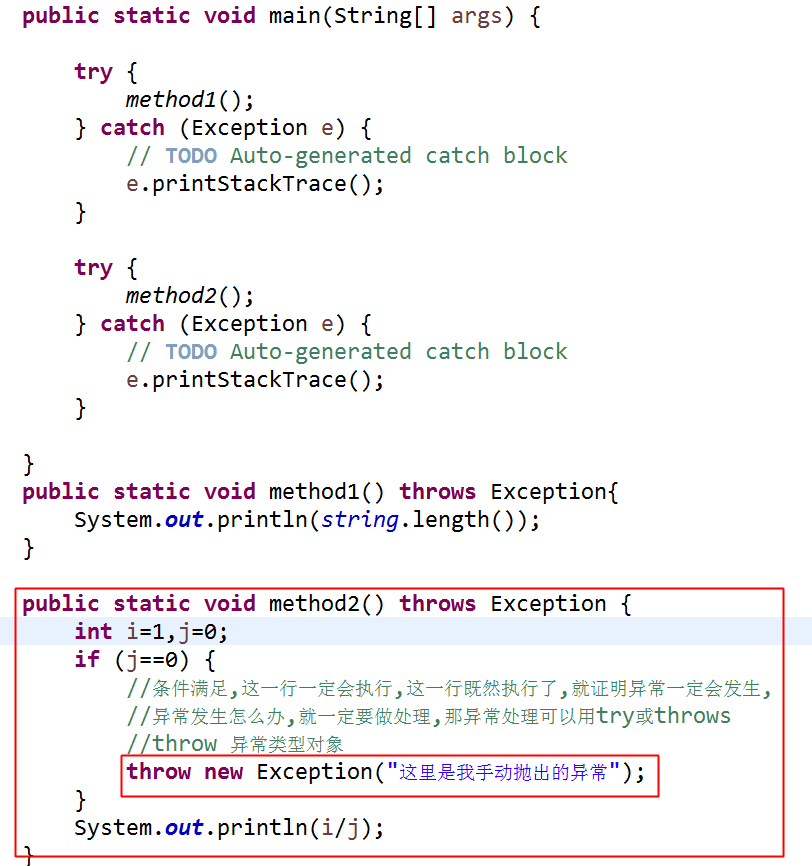
异常一直向上抛,最顶点就是JVM

可以称为声明了一个异常.可以理解为抛出异常，抛出给调用者，谁调用他就抛给谁;



### 方式3:throw :手动引发异常(了解)

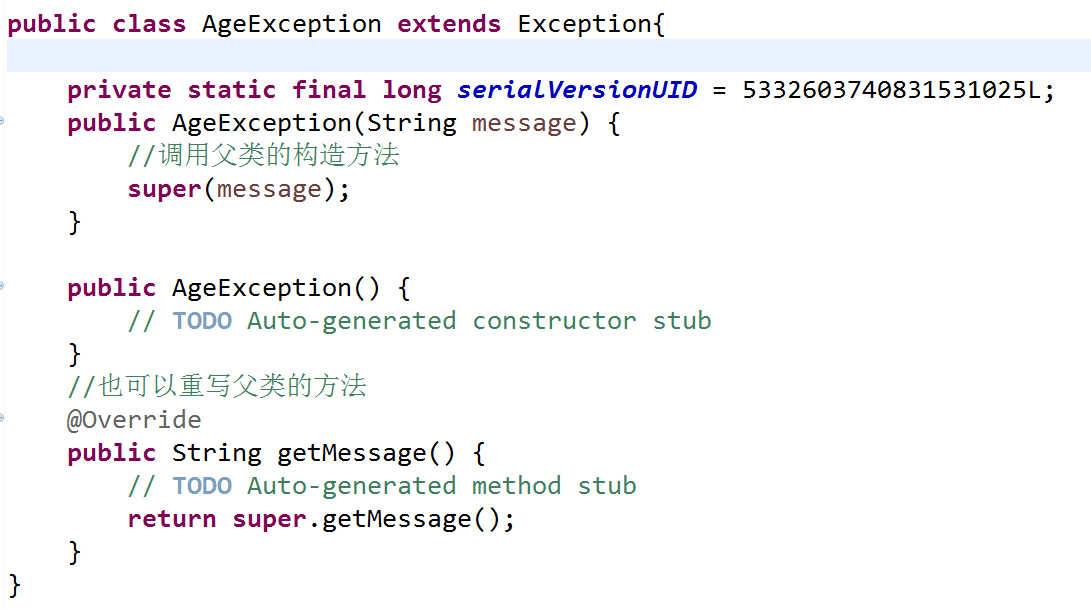
手动引发异常,发生在方法中(我认为满足特定的条件就是异常发生了)

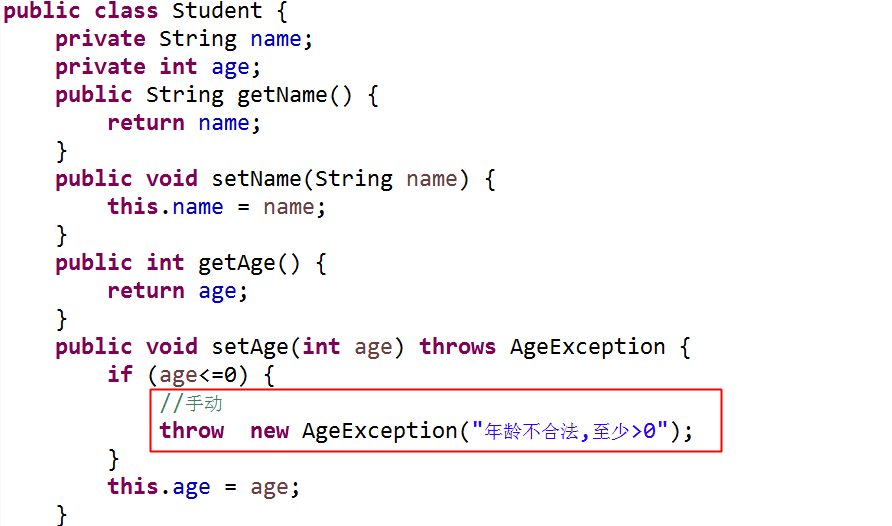


### 方式4:自定义异常(了解)

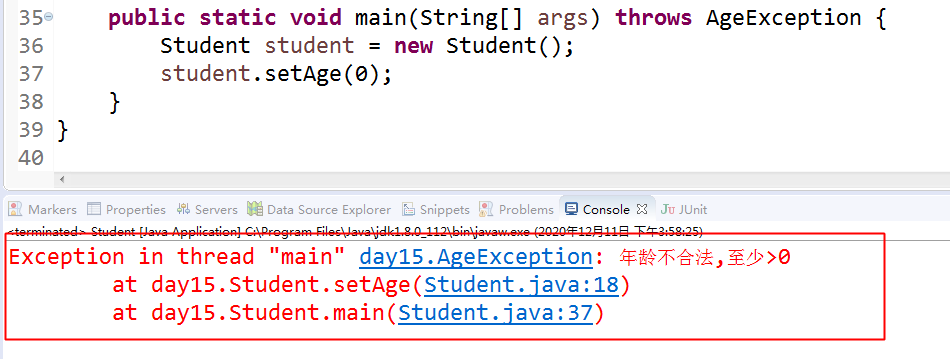
1. 创建一个自定义异常类:

①继承Exception(或其子类) ②添加一个带参数构造 ③重写getMessage()-可写可不写

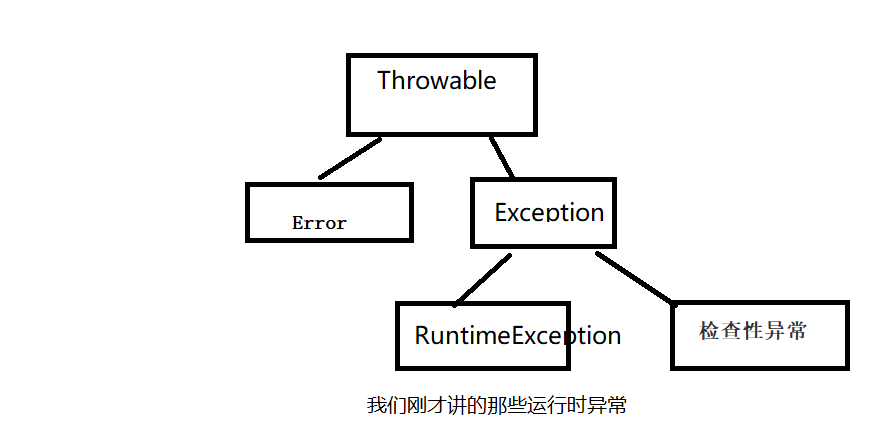




测试:



## 异常小结：



异常是程序中的一些错误，但并不是所有的错误都是异常，并且错误有时候是可以避免的。

•  **检查性异常：**最具代表的检查性异常是用户错误或问题引起的异常，这是程序员无法预见的。例如要打开一个不存在文件时，一个异常就发生了，这些异常**在编译时不能被简单地忽略**。

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

Class.*forName*("Apple");

}

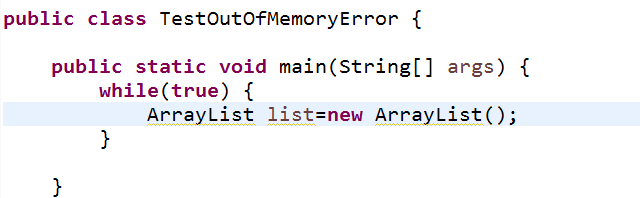
Exception, SQLException 非运行时异常.

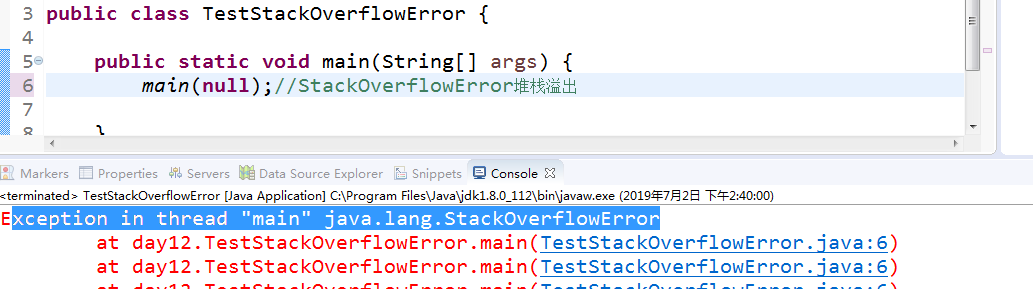
• **运行时异常：** 运行时异常是可能被程序员避免的异常。与检查性异常相反，运行时异常可以在编译时被忽略。

NullPointerException

• **错误：** 错误不是异常，而是脱离程序员控制的问题。错误在代码中通常被忽略。例如，当栈溢出时，一个错误就发生了，它们在**编译也检查不到**的。

OutOfMemoryError, 内存溢出、StackOverflowError堆栈溢出





所有的异常类是从 java.lang.Exception 类继承的子类:

1.分类：编译时异常、运行时异常

2.Exception异常：一般是逻辑或者数据内容的出错

Error错误：jvm底层的各种异常：内存溢出、堆栈溢出。

3.所有的编程语言都需要进行异常处理，针对是“运行期”。

4.处理异常作用：预先准备，避免程序因为异常而“终止”。

5.所有的异常类的父类的 Java.lang.Exception

6.Exception的四个直接子类

SQLException--数据库异常

ClassNotFountdException--找不到类

IOException--流异常

RuntimeException-运行期异常

7.RuntimeException-运行期异常的常见子类

ArithmeticException 算术异常，如：除数为0

IllegalArgumentException 方法接收到非法参数

ArrayIndexOutOfBoundsException 数组下标越界

NullPointerException 访问空引用

ClassNotFoundException 不能加载所需的类

NumberFormatException 字符串转换数字失败

8.异常处理

方式1：try{

//监视作用

}catch(){

//捕获作用 做处理

}finally{

//无论try中发生什么，finally都会执行，除非特殊情况

}

注意：try的后面可以有多个catch，但是注意层级关系：类层次越低的越上写，越高越往下写。

finally不是永远都会执行。有些特殊情况：finally语句中异常，退出程序...

方式2：throws语句用于抛出异常：不在方法中处理异常，向上抛。

方式3：throw语句用于手工抛出异常，很少用

## 常见面试题

①异常Exception和错误error的区别：

都是继承于Throwable 。

Exception异常：可以是可被控制(checked) ,是运行过程中的不正常事件，可以被异常机制处理，程序能够继续运行下去；

Error错误：jvm底层的各种问题：内存溢出,堆栈溢出。错误不能够被处理，发生错误后，程序就终止，程序员需要修改源代码才能解决错误；

② try catch fianlly关键字的作用

try：监视作用

catch：捕获异常,处理异常

finally:不管是否出现异常，都会执行finally中的代码块，finally块是可选的，可视具体情况决定是否添加； finally块必须和try块一起使用，不能单独存在。

③书写出几个常见的异常类型(5个以上)

④ final finalize fianlly 的区别

## 练习:

编码实现对一个公司的员工进行简单的访问: 增加员工(员工入职),查询员工(姓名 查询所有 根据员工编号 ,根据特定的条件来查询等);

分析: 每个员工都应该是一个对象(Employee) ->一个公司有多个员工(数组或者集合来处理) ->针对数组的操作: ->增加员工,查询员工(姓名 查询所有 根据员工编号等

以上代码发现数组的缺点:

1. 数组长度固定不变,不便于扩展
2. 数组的添加和查找方法都需要对数组进行遍历

将上面的案例使用集合去处理