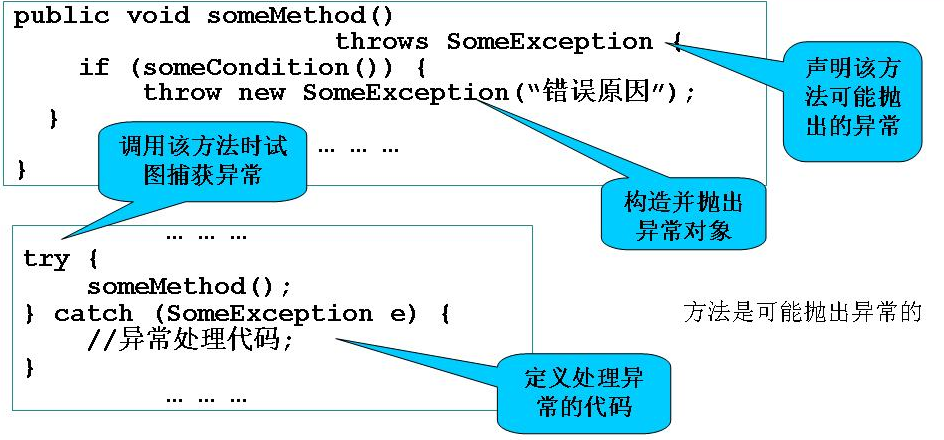
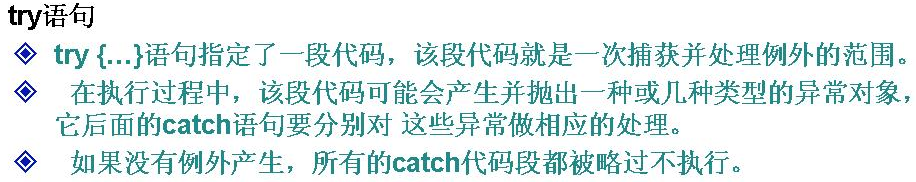
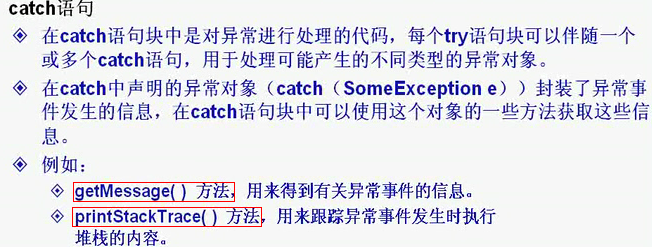
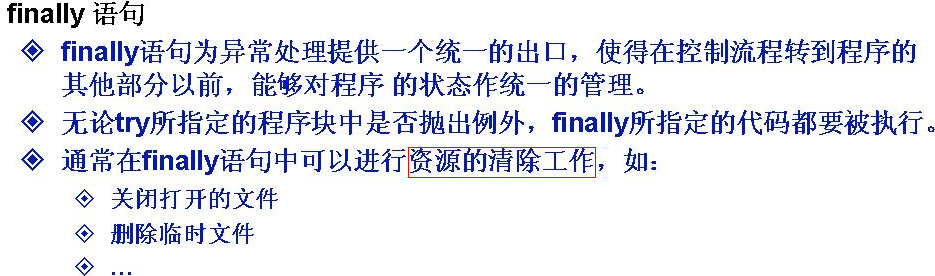
1. 异常：**运行期间**出现的错误(观察错误的名字和行号最重要，要敢于调试)。
   1. Java异常是java提供的用于处理程序中错误的一种机制。
   2. 所谓错误是指在程序运行的过程中发生的一些异常事件(如：除0溢出、数组下标越界、所要读取的文件不存在等)。
   3. 设计良好的程序应该在异常发生时提供处理这些错误的方法，使得程序不会因为异常的发生而阻断或产生不可预见的结果。
   4. Java程序的执行过程中如出现异常事件，可以生成一个异常类对象，该异常对象封装了异常事件的信息并将其提交给java运行时系统，这个过程称为抛出(throw)异常。
   5. 当java运行时系统接收到异常对象时，会寻找能处理这一异常的代码并把当前异常对象交给其处理，这一过程成为捕获(catch)异常。



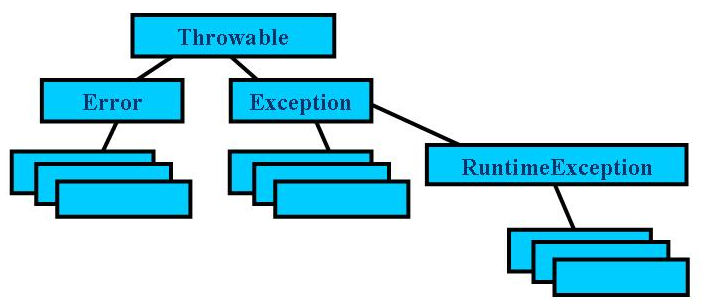






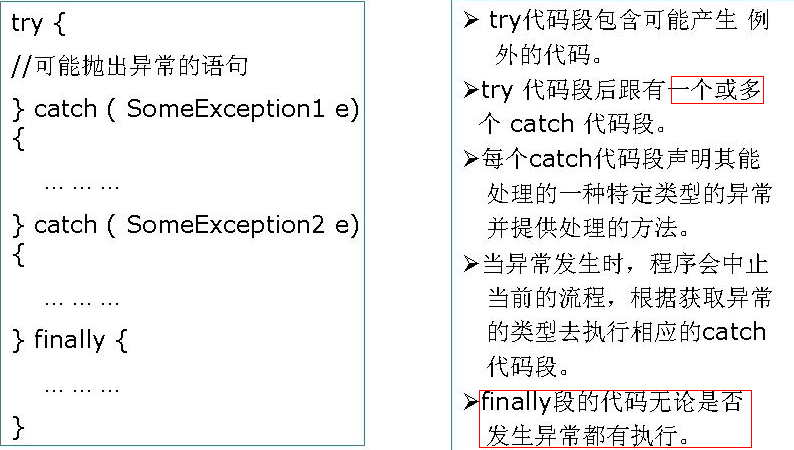
如果try或者catch块**有return指令，finally还是会执行**！流程会跳到finally然后再回到return指令。

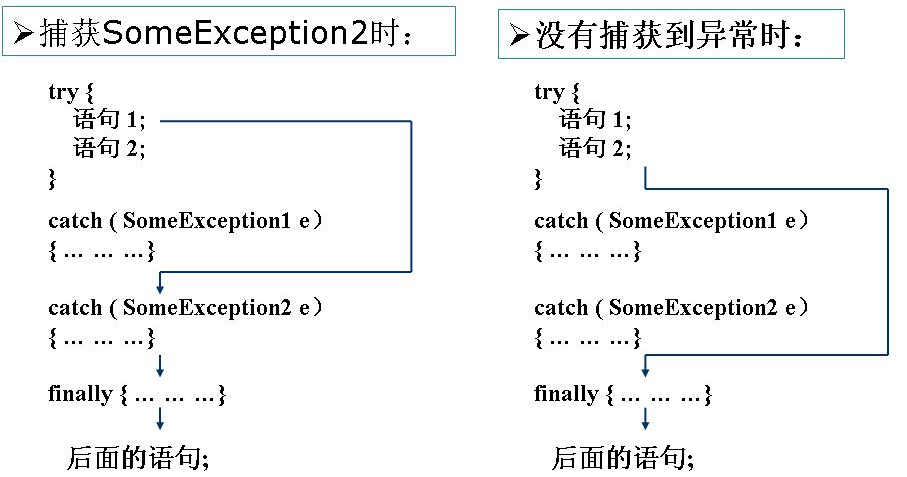
1. 异常的分类

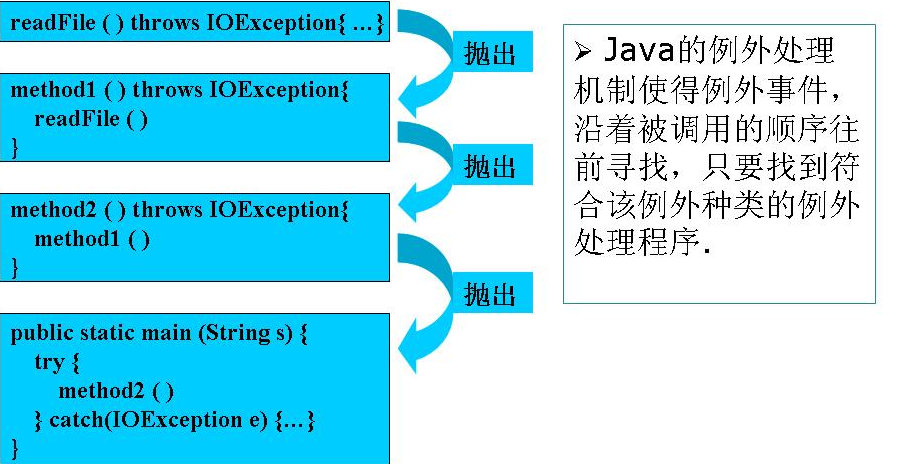


* 1. Error：称为错误，表示应用程序本身无法克服和恢复的一种严重问题，由java虚拟机生成并抛出，包括动态链接失败、虚拟机错误、内存溢出和线程死锁等，我们的程序对其不作处理。
  2. Exception：所有异常类的父类。其子类对应了各种各样可能出现的异常事件，一般需要用户显示的声明或捕获。
  3. RuntimeException：一类特殊的异常，如被0除，数组下标超范围等，其产生比较频繁，处理麻烦，如果显示的声明或捕获将会对程序可读性和运行效率影响很大，因此系统自动检测并将它们交给缺省的异常处理程序(用户可不必对其处理)。
  4. 必须catch的异常：凡是方法后面throws someException(非Runtime Exception)的Exception，必须catch或者再次抛出throws该异常。

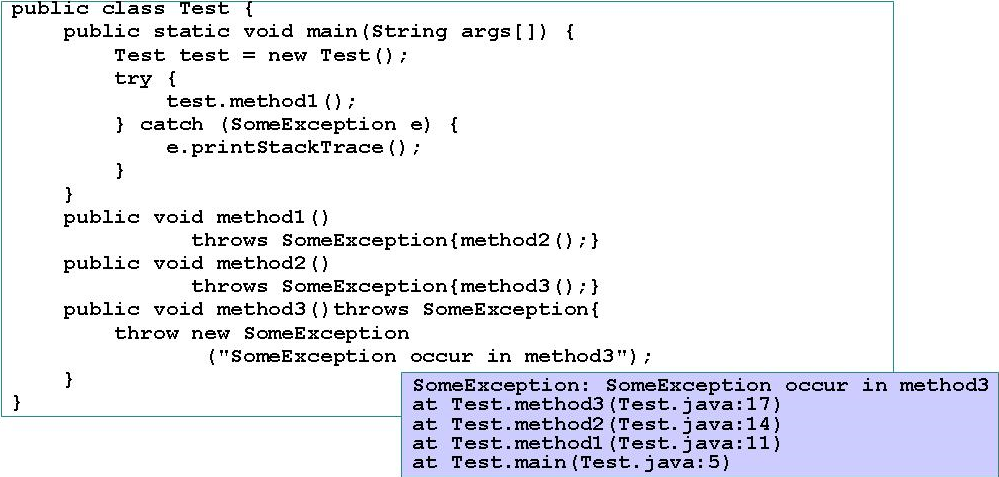
1. 异常的捕获和处理
   1. catch与finally不能没有try
   2. try与catch之间不能有程序
   3. try一定要有catch或finally
   4. 只带有finally的try必须要声明异常



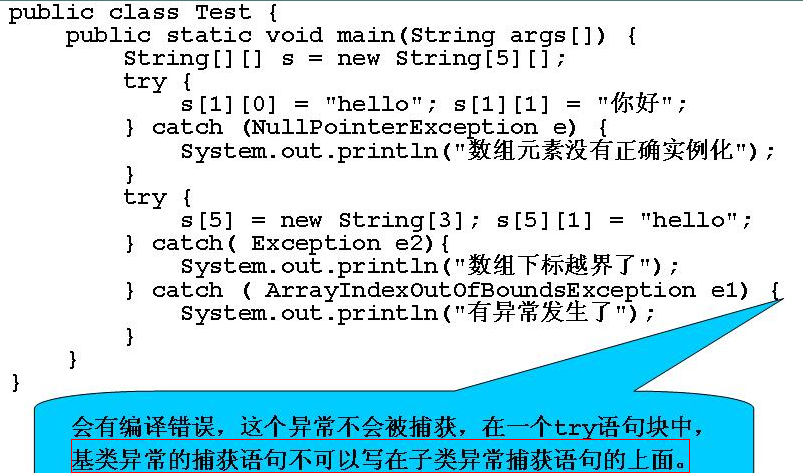




main()方法throws异常时(相当不好的编程习惯，应该catch并做出相应的处理)，会交给java运行时系统JVM，java运行时系统(JVM挂掉)会将其堆栈信息打印出来。



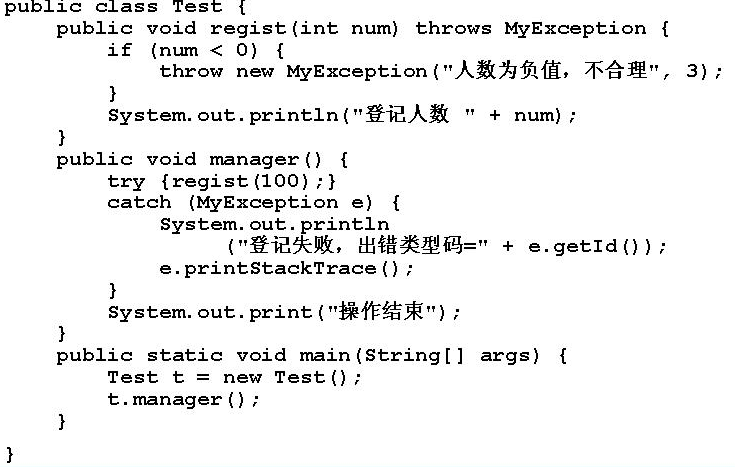
e.printStackTrace()，将堆栈信息打印出来(调试时改正最上面一个)。



先catch小的，再catch大的。

1. 自定义异常

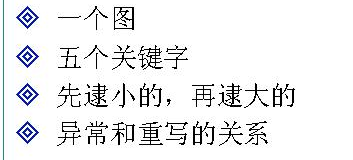




1. 继承中重写方法与异常的关系

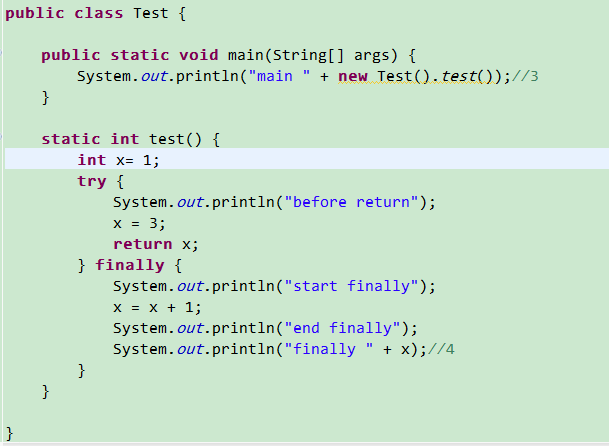


1. 异常总结

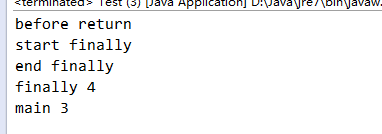


1. try{}里有一个return语句，那么紧跟在这个try后的finally{}里面的code会不会执行，什么时候执行，在return之前还是之后(<http://zhaohe162.blog.163.com/blog/static/3821679720127201300743/>)。

也许你的答案是在return之前，但往更细地说，我的答案是在return中间执行，请看下面程序代码的运行结果：



运行结果：



主函数调用子函数并得到结果的过程，好比主函数准备一个空罐子，当子函数要返回结果时，先把结果放在罐子里，然后再将程序逻辑返回到主函数。所谓返回，就是子函数说，我不运行了，你主函数继续运行吧，这没什么结果可言，结果是在说这话之前放进罐子里的。

try中的return语句调用的函数先于finally中调用的函数执行，也就是说return语句先执行，finally语句后执行。**return并不是让函数马上返回，而是return语句执行后，将把返回结果放置进函数栈中，此时函数并不是马上返回，它要执行finally语句后才真正开始返回**

在字节码中，return语句不是原子性操作，它会把getValue()中的return语句分解为几步操作：

１) 把value值存储到一个局部变量(这里命名为temp)中：

　　 iload\_0　　 // push local variable 0 - the 100

　　 istore\_2　 // pop an int (the 100), store into local varaible 2

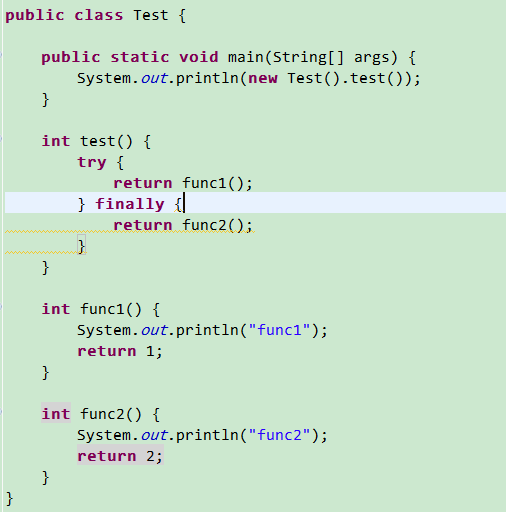
２) 调用finally微型子程序

３) 把局部变量(指temp)的值push到操作数栈顶，然后返回到调用方法

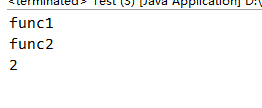
　 iload\_2　　// push local varaible 2 - the 100

　　ireturn // return int on top of the stack - the 100: return 100

由于return语句在返回之前会把返回值保存到一个临时的局部变量中，所以在finally子句内对value重新赋值不会影响返回值。



运行结果：



**finally中的代码比return和break语句后执行**