public class Test {

public static void main(String[] args) {

System.out.println(decision());

}

static boolean decision() {

try {

return true;

} finally {

return false;

}

}

}

你可能会认为这个程序是不合法的。毕竟，decision方法不能同时返回true和false。如果你尝试一下，就会发现它编译时没有任何错误，并且它所打印的是false。为什么呢？

原因就是在一个try-finally语句中，finally语句块总是在控制权离开try语句块时执行的。无论try语句块是正常结束的，还是意外结束的，情况都是如此。一条语句或一个语句块在它抛出了一个异常，或者对某个封闭型语句执行了一个break或continue，或是象这个程序一样在方法中执行了一个return时，将发生意外结束。它们之所以被称为意外结束，是因为它们阻止程序去按顺序执行下面的语句。

当try语句块和finally语句块都意外结束时，在try语句块中引发意外结束的原因将被丢弃，而整个try-finally语句意外结束的原因将于finally语句块意外结束的原因相同。在这个程序中，在try语句块中的return语句所引发的意外结束将被丢弃，而try-finally语句意外结束是由finally语句块中的return造成的。简单地讲，程序尝试着（try）返回（return）true，但是它最终（finally）返回（return）的是false。

丢弃意外结束的原因几乎永远都不是你想要的行为，因为意外结束的最初原因可能对程序的行为来说会显得更重要。对于那些在try语句块中执行break、continue或return语句，只是为了使其行为被finally语句块所否决掉的程序，要理解其行为是特别困难的。

总之，每一个finally语句块都应该正常结束，除非抛出的是不受检查的异常。千万不要用一个return、break、continue或throw来退出一个finally语句块，并且千万不要允许将一个受检查的异常传播到一个finally语句块之外去。

try/catch/finally语句下，finally子句是肯定会执行的。但是很多人做不同的测试，却得出了不同的结论。

具体的原理最好是去看《深入java虚拟机》，里面对jsr、ret等几个指令做了详细的说明。这里不深入分析，而仅仅是从表现形式上看一下finally的特征。

代码：

/\*

\* author: Zang XT

\*/

public class TestFinal {

public static void main(String[] args) {

System.out.println("test1:"+testFinal1());

System.out.println("test2:"+testFinal2());

System.out.println("test3:"+testFinal3());

System.out.println("test4:"+testFinal4());

}

static int testFinal1(){

int i = 1;

try{

return i;

}

finally{

System.out.println("in testFinal1():finally 肯定会被执行的！");

i = 48;

}

}

static String testFinal2(){

String str = "try";

try{

return str;

}

finally{

System.out.println("in testFinal2():finally 肯定会被执行的！");

str = "finally";

}

}

static StringBuilder testFinal3(){

StringBuilder build = new StringBuilder("try ");

try{

return build;

}

finally{

System.out.println("in testFinal3():finally 肯定会被执行的！");

build.append("finally");

build = new StringBuilder("你猜我是谁！");

}

}

static String testFinal4(){

try{

return "return in try";

}

finally{

System.out.println("in testFinal4():finally 肯定会被执行的！");

return "return in finally";

}

}

}

输出是：

in testFinal1():finally 肯定会被执行的！

test1:1

in testFinal2():finally 肯定会被执行的！

test2:try

in testFinal3():finally 肯定会被执行的！

test3:try finally

in testFinal4():finally 肯定会被执行的！

test4:return in finally

结论很明显，finally的语句确实执行了，而且肯定是在方法return之前执行的，而且，如果finally中有return语句的话，方法直接结束。这里需要注意的只有一点：在try中的return语句会将返回结果值压栈，然后转入到finally子过程，等到finally子过程执行完毕之后（没有return），再返回。

下面具体看4个例子：

在testFinal1()中，return i;会将结果i的值，也就是1压入栈。即使在finally中将i修改了（i=48），也不回对已经压入栈里的1造成任何影响。

在testFinal2()中，return str;将str的内容压入栈，比如我们假设str的内容为0x108(只是一个地址值),通过这个地址值我们能找到"try"，那栈里的内容就是0x108。执行str = "finally"，这时候str这个变量的内容可能变为0x237了，这是串"finally"的地址。方法调用结束后，返回的是什么？return时压入栈里的0x108。所以在打印结果时，我们打印的是通过0x108找到的字符串"try"。

在testFinal3()中，return 压栈的是build这个变量的值，比如是0x3579，通过这个值我们可以找到StringBuilder对象。finally语句块中对这个对象的内容进行了修改。build = new StringBuilder("你猜我是谁！");让build变量指向了一个新的对象，这时候build的值可能是0x4579了。但是，别忘了，原来的StringBuilder对象仍然在0x3579处，而我们压栈的正是0x3579啊！方法返回后，我们得到的返回值0x3579，通过这个引用值找到相应的StringBuilder对象，所以打印的结果是test3:try finally。

在testFinal4()中，finally有return语句，直接返回，方法结束。

为什么不同的人有不同的结论？关键是没有正确理解压栈的是什么东西。其实初学java的时候，如果理解了变量是什么，并区分引用和对象本身就不会得到错误的结论了。再有，如果理解java中，方法调用都是采用传值模式的话，这里也就类似的可以明白了。