

第13章

Introduction to Java Programming, 8E

异常处理

学习目标

- 了解异常和异常处理的概况 (13.2节)。
- 探究使用异常处理的优点 (13.3节)。
- 区别异常的类型: **Error** (致命的) 和 **Exception** (非致命的) 以及必检和免检异常 (13.4节)。
- 在方法头中声明异常 (13.5.1节)。
- 在方法中抛出异常 (13.5.2节)。
- 编写 **try-catch** 块处理异常 (13.5.3节)。
- 解释异常是如何传播的 (13.5.3节)。
- 在 **try-catch** 块中使用 **finally** 子句 (13.6节)。
- 只为非预期错误使用异常 (13.7节)。
- 在 **catch** 块中重新抛出异常 (13.8节)。
- 创建链式异常 (13.9节)。
- 定义自定义的异常类 (13.10节)。

13.1 引言

在程序运行过程中, 如果环境检测出一个不可能执行的操作, 就会出现运行时错误 (runtime error)。例如, 如果使用一个越界的下标访问数组, 程序就会产生一个 **ArrayIndexOutOfBoundsException** 的运行时错误。为了从文件中读取数据, 需要使用 `new Scanner(new File(filename))` 创建一个 **Scanner** 对象 (参见程序清单9-6)。如果该文件不存在, 程序将会出现一个 **FileNotFoundException** 的运行时错误。

在Java中, 异常会导致运行时错误。异常就是一个表示阻止执行正常进行的错误或者情况。如果异常没有被处理, 那么程序将会非正常终止。该如何处理这个异常, 以使程序可以继续运行或者平稳终止呢? 这就是本章要介绍的主题。

13.2 异常处理概述

为了演示异常处理, 包括一个异常是如何创建的以及如何抛出的, 我们从一个读取两个整数并显示它们的商的例子 (程序清单13-1) 开始。

程序清单13-1 **Quotient.java**

```
1 import java.util.Scanner;
2
3 public class Quotient {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         // Prompt the user to enter two integers
8         System.out.print("Enter two integers: ");
9         int number1 = input.nextInt();
10        int number2 = input.nextInt();
11
12        System.out.println(number1 + " / " + number2 + " is " +
13            (number1 / number2));
```




```
14 }
15 }
```

Enter two integers: 5 2 

5 / 2 is 2



Enter two integers: 3 0 


Exception in thread "main" java.lang.ArithmeticException: / by zero
at Quotient.main(Quotient.java:11)



如果为第二个数字输入的是0，那就会产生一个运行时错误，因为不能用0除一个整数（回顾一下，一个浮点数除以0是会产生异常的）。解决这个错误的一个简单方法就是添加一个if语句来测试第二个数字，如程序清单13-2所示。

程序清单13-2 QuotientWithIf.java

```
1 import java.util.Scanner;
2
3 public class QuotientWithIf {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         // Prompt the user to enter two integers
8         System.out.print("Enter two integers: ");
9         int number1 = input.nextInt();
10        int number2 = input.nextInt();
11
12        if (number2 != 0)
13            System.out.println(number1 + " / " + number2
14                + " is " + (number1 / number2));
15        else
16            System.out.println("Divisor cannot be zero ");
17    }
18 }
```

Enter two integers: 5 0 

Divisor cannot be zero



为了演示异常处理的概念，包括如何创建、抛出、捕获以及处理异常，我们改写程序清单13-2为如程序清单13-3所示。

程序清单13-3 QuotientWithException.java


```
1 import java.util.Scanner;
2
3 public class QuotientWithException {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         // Prompt the user to enter two integers
8         System.out.print("Enter two integers: ");
9         int number1 = input.nextInt();
10        int number2 = input.nextInt();
11
12        try {
13            if (number2 == 0)
```

新华书店
PDFG


```

14      throw new ArithmeticException("Divisor cannot be zero");
15
16      System.out.println(number1 + " / " + number2 + " is " +
17          (number1 / number2));
18  }
19  catch (ArithmeticException ex) {
20      System.out.println("Exception: an integer " +
21          "cannot be divided by zero ");
22  }
23
24  System.out.println("Execution continues ...");
25  }
26  }

```

Enter two integers: 5 3 
 5 / 3 is 1
 Execution continues ...



Enter two integers: 5 0 
 Exception: an integer cannot be divided by zero
 Execution continues ...



程序包含一个try块和一个catch块。try块（第12~18行）包含的是在正常情况下执行的代码。catch块（第19~22行）包含的是在number2为0时执行的代码。在这种情况下，程序会通过执行下面的语句来抛出一个异常：

```
throw new ArithmeticException("Divisor cannot be zero");
```

在这种情况下new ArithmeticException("Divisor cannot be zero")下抛出的值，可以称为一个异常（exception）。throw语句的执行称为抛出一个异常（throwing an exception）。异常就是一个从异常类创建的对象。在这种情况下，异常类就是java.lang.ArithmeticException。

当异常被抛出时，正常的执行流程就被中断。就像它的名字所提示的，“抛出异常”就是将异常从一个地方传递到另一个地方。异常被catch块捕获。执行catch块中的代码以处理这个异常（handle the exception）。然后，执行catch块后的语句（第24行）。

throw语句类似于方法的调用，但不同于调用方法的是，它调用的是catch块。从某种意义上讲，catch块就像带参数的方法定义，这些参数匹配抛出的值的类型。但是，它不像方法，它在执行完catch块之后，程序控制不返回到throw语句；而是执行catch块后的下一条语句。

catch块的头部的标识符ex

```
catch (ArithmeticException ex)
```

的作用很像是方法中的参数。所以，这个参数称为catch块的参数。ex之前的类型（例如，ArithmeticException）指定了catch块可以捕获的异常类型。一旦捕获该异常，就能从catch块体中的参数访问这个抛出的值。

总之，一个try-throw-catch块的模板可能会如下所示：

```

try {
    Code to try;
    Throw an exception with a throw statement or
    from method if necessary;
    More code to try;
}
catch (type ex) {
    Code to process the exception;
}

```

可以使用try块中的throw语句直接抛出一个异常，或者调用一个可能会抛出异常的方法。

13.3 异常处理的优势


从程序清单13-3中已经看出了一个异常是如何创建、抛出、捕获和处理的。你可能会奇怪这样做有什么好处。为了能看出这些优势，我们改写程序清单13-3，使用方法来计算商，如程序清单13-4所示。

程序清单13-4 QuotientWithMethod.java


```

1 import java.util.Scanner;
2
3 public class QuotientWithMethod {
4     public static int quotient(int number1, int number2) {
5         if (number2 == 0)
6             throw new ArithmeticException("Divisor cannot be zero");
7
8         return number1 / number2;
9     }
10
11     public static void main(String[] args) {
12         Scanner input = new Scanner(System.in);
13
14         // Prompt the user to enter two integers
15         System.out.print("Enter two integers: ");
16         int number1 = input.nextInt();
17         int number2 = input.nextInt();
18
19         try {
20             int result = quotient(number1, number2);
21             System.out.println(number1 + " / " + number2 + " is "
22                 + result);
23         }
24         catch (ArithmeticException ex) {
25             System.out.println("Exception: an integer " +
26                 "cannot be divided by zero ");
27         }
28
29         System.out.println("Execution continues ...");
30     }
31 }

```

Enter two integers: 5 3 
 5 / 3 is 1
 Execution continues ...



Enter two integers: 5 0 
 Exception: an integer cannot be divided by zero
 Execution continues ...



方法quotient（第4~9行）返回两个整数的商。如果number2为0，它不会返回值。所以，在第6行会抛出一个异常。

main方法调用quotient（第20行）。如果求商方法正常执行，它会返回一个值给调用者。如果quotient方法遇到一个异常，它会抛出一个异常给它的调用者。这个调用者的catch块处理该异常。

现在，你看到了使用异常处理的优点。它能使方法抛出一个异常给它的调用者。这个调用者可以处理该异常。如果没有这个能力，那么被调用的方法就必须自己处理异常或者终止该程序。被调用的方法通常不知道在出错的情况下该做些什么。这是库方法的一般情况。库方法可以检测出错误，但是只有调用者才知道出现错误时需要做些什么。异常处理最根本的优势就是将检测错误（由调用的方法完成）从

处理错误（由调用方法完成）中分离出来。

很多库方法都会抛出异常（throw exception）。程序清单13-5给出一个处理调用Scanner(File file)构造方法时出现异常FileNotFoundException的例子。

程序清单13-5 FileNotFoundExceptionDemo.java

```

1 import java.util.Scanner;
2 import java.io.*;
3
4 public class FileNotFoundExceptionDemo {
5     public static void main(String[] args) {
6         Scanner inputFromConsole = new Scanner(System.in);
7         // Prompt the user to enter a file name
8         System.out.print("Enter a file name: ");
9         String filename = inputFromConsole.nextLine();
10
11         try {
12             Scanner inputFromFile = new Scanner(new File(filename));
13             System.out.println("File " + filename + " exists ");
14             // Processing file ...
15         }
16         catch (FileNotFoundException ex) {
17             System.out.println("Exception: " + filename + " not found");
18         }
19     }
20 }

```

Enter a file name: c:\book\Welcome.java Enter
File c:\book\Welcome.java exists



Enter a file name: c:\book\Test10.java Enter
Exception: c:\book\Test10.java not found



该程序为文件创建一个Scanner（第12行）。如果这个文件不存在，那么它的构造方法就会抛出一个异常FileNotFoundException，该异常在catch块中被捕获。

程序清单13-6给出一个处理异常InputMismatchException的例子。

程序清单13-6 InputMismatchExceptionDemo.java

```

1 import java.util.*;
2
3 public class InputMismatchExceptionDemo {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         boolean continueInput = true;
7
8         do {
9             try {
10                 System.out.print("Enter an integer: ");
11                 int number = input.nextInt();
12                 // Display the result
13                 System.out.println(
14                     "The number entered is " + number);
15                 continueInput = false;
16             }
17             catch (InputMismatchException ex) {
18                 System.out.println("Try again. (" +
19                     "Incorrect input: an integer is required)");
20                 input.nextLine(); // Discard input
21             }
22         }
23     }

```

新华书店
PDG


```

24     } while (continueInput);
25 }
26 }

```

```

Enter an integer: 3.5
Try again. (Incorrect input: an integer is required)
Enter an integer: 4
The number entered is 4

```



当执行() (第11行) 时, 如果键入的输入不是一个整数, 就会出现一个InputMismatchException异常。假设输入的是3.5, 就会出现一个InputMismatchException异常, 而且控制被转移到catch块。现在, 执行catch块中的语句。第22行的语句()丢弃当前的输入行, 所以, 用户就可以键入一个新行。变量continueInput来控制循环。它的初始值为true (第6行), 当接收到的是一个合法值时, 该值就变成false (第17行)。

13.4 异常类型

13.3节使用了三个异常ArithmeticException、FileNotFoundException和InputMismatchException。是否还有可以使用的其他类型的异常? 回答是肯定的。在Java API中有很多预定义的异常类。图13-1给出它们中的一部分。

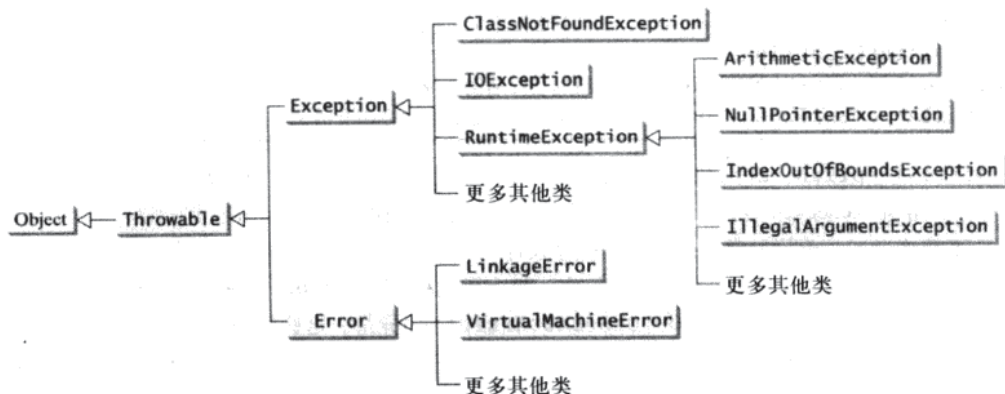


图13-1 抛出的异常都是这个图中给出的类的实例, 或者是这些类的子类的实例

注意 类名Error、Exception和RuntimeException有时候容易引起混淆。这三种类都是异常, 这里讨论的错误都发生在运行时。

Throwable类是所有异常类的根。所有的Java异常类都直接或者间接地继承自Throwable。可以通过扩展Exception或者Exception的子类来创建自己的异常类。

这些异常类可以分为三种主要类型: 系统错误、异常和运行时异常。

1) 系统错误 (system error) 是由Java虚拟机抛出的, 用Error类表示。Error类描述的是内部系统错误。这样的错误很少发生。如果发生, 除了通知用户以及尽量稳妥地终止程序外, 几乎什么也不能做。表13-1列出Error子类的一些例子。

2) 异常 (exception) 是用Exception类表示的, 它描述的是由程序和外部环境所引起的错误, 这些错误能被程序捕获和处理。表13-2列出Exception类的子类的一些例子。

3) 运行时异常 (runtime exception) 是用RuntimeException类表示的, 它描述的是程序设计错误, 例如, 错误的类型转换、访问一个越界数组或数值错误。运行时异常通常是由Java虚拟机抛出的。表13-3列出RuntimeException的子类的一些例子。

表13-1 Error类的子类的例子

类	可能引起异常的原因
LinkageError	一个类对另一个类有某种依赖性, 编译前者后, 后者变得不兼容 Java虚拟机中断或者没有继续运行所必需的资源可用
VirtualMachineError	

表13-2 Exception类的子类的例子

类	可能引起异常的原因
ClassNotFoundException	企图使用一个不存在的类。例如, 如果试图使用命令java来运行一个不存在的类, 或者程序要调用三个类文件而只能找到两个, 都会发生这种异常
IOException	同输入/输出相关的操作, 例如, 无效的输入、读文件时超过文件尾、打开一个不存在的文件等。IOException的子类的例子有InterruptedException、EOFException (EOF是End Of File的缩写) 和FileNotFoundException

表13-3 RuntimeException类的子类的例子

类	可能引起异常的原因
ArithmeticException	一个整数除以0。注意, 浮点数的算术运算不抛出异常。参见附录E
NullPointerException	企图通过一个null引用变量访问一个对象
IndexOutOfBoundsException	数组的下标超出范围
IllegalArgumentException	传递给方法的参数非法或不合适

RuntimeException、Error以及它们的子类都称为免检异常 (unchecked exception)。所有其他异常都称为必检异常 (checked exception), 意思是指编译器会强制程序员检查并处理它们。

在大多数情况下, 免检异常都会反映出程序设计上不可恢复的逻辑错误。例如, 如果通过一个引用变量访问一个对象之前并未将一个对象赋值给它, 就会抛出NullPointerException异常; 如果访问一个数组的越界元素, 就会抛出IndexOutOfBoundsException异常。这些都是程序中必须纠正的逻辑错误。免检异常可能在程序的任何一个地方出现。为避免过多地使用try-catch块, Java语言不允许编写代码捕获或声明免检异常。

警告 目前, Java还没有抛出整数上溢或下溢异常。下面的语句给最大整数加1。

```
int number = Integer.MAX_VALUE + 1;
System.out.println(number);
```

它会显示-2147483648, 这在逻辑上是错误的。引起该问题的原因是上溢; 也就是说, 结果超出了int型值的最大范围。

Java今后的版本可能会通过抛出一个上溢异常来解决这个问题。

13.5 关于异常处理的更多知识

13.4节给出了异常处理的概况, 同时介绍了几个预定义的异常类型。本节对异常处理进行深入讨论。

Java的异常处理模型基于三种操作: 声明一个异常 (declaring an exception)、抛出一个异常 (throwing an exception) 和捕获一个异常 (catching an exception), 如图13-2所示。

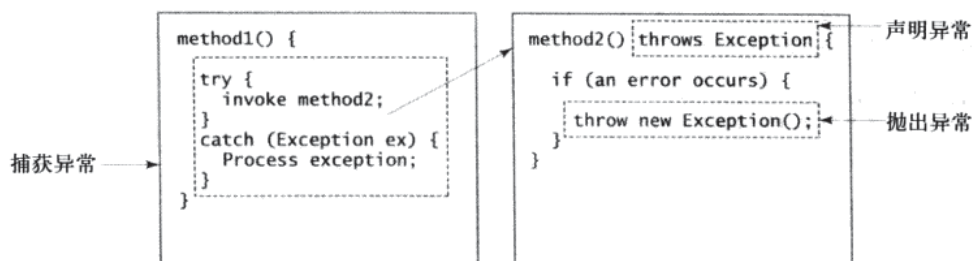


图13-2 Java中的异常处理包括声明异常、抛出异常以及捕获和处理异常

13.5.1 声明异常

在Java中，当前执行的语句必属于某个方法。Java解释器调用main方法开始执行一个程序。每个方法都必须声明它可能抛出的必检异常的类型。这称为声明异常（declaring exception）。因为任何代码都可能发生系统错误和运行时错误，因此，Java不要求在方法中显式声明Error和RuntimeException（免检异常）。但是，方法要抛出的其他异常都必须在方法头中显式声明，这样，方法的调用者会被告知有异常。

为了在方法中声明一个异常，就要在方法头中使用关键字throws，如下例所示：

```
public void myMethod() throws IOException
```

关键字throws表明myMethod方法可能会抛出异常IOException。如果方法可能会抛出多个异常，就可以在关键字throws后添加一个用逗号分隔的异常列表：

```
public void myMethod()
    throws Exception1, Exception2, ..., ExceptionN
```

注意 如果方法没有在父类中声明异常，那么就不能在子类中对其进行覆盖来声明异常。

13.5.2 抛出异常

检测一个错误的程序可以创建一个正确异常类型的实例并抛出它。这就称为抛出一个异常（throwing an exception）。这里有一个例子，假如程序发现传递给方法的参数与方法的合约不符（例如，方法中的参数必须是非负的，但是传入的是一个负参数），这个程序就可以创建IllegalArgumentException的一个实例并抛出它，如下所示：

```
IllegalArgumentException ex =
    new IllegalArgumentException("Wrong Argument");
throw ex;
```

或者，如果你愿意，也可以使用下面的语句：

```
throw new IllegalArgumentException("Wrong Argument");
```

注意 IllegalArgumentException是Java API中的一个异常类。通常，Java API中的每个异常类至少有两个构造方法：一个无参构造方法和一个带可描述这个异常的String参数的构造方法。该参数称为异常消息（exception message），它可以用getMessage()获取。

提示 声明异常的关键字是throws，抛出异常的关键字是throw。

13.5.3 捕获异常

现在知道了如何声明一个异常以及如何抛出一个异常。当抛出一个异常时，可以在try-catch块中捕获和处理它，如下所示：

```
try {
    statements; // Statements that may throw exceptions
}
```



```

catch (Exception1 exVar1) {
    handler for exception1;
}
catch (Exception2 exVar2) {
    handler for exception2;
}
...
catch (ExceptionN exVar3) {
    handler for exceptionN;
}

```

如果在执行try块的过程中没有出现异常，则跳过catch子句。

如果try块中的某条语句抛出一个异常，Java就会跳过try块中剩余的语句，然后开始查找处理这个异常的代码的过程。处理这个异常的代码称为异常处理器（exception handler），可以从当前的方法开始，沿着方法调用链，按照异常的反向传播方向找到这个处理器。从第一个到最后一个逐个检查catch块，判断在catch块中的异常类实例是否是该异常对象的类型。如果是，就将该异常对象赋值给所声明的变量，然后执行catch块中的代码。如果没有发现异常处理器，Java会退出这个方法，把异常传递给调用这个方法的方法，继续同样的过程来查找处理器。如果在调用的方法链中找不到处理器，程序就会终止并且在控制台上打印出错信息。寻找处理器的过程称为捕获一个异常（catching an exception）。

假设main方法调用method1，method1调用method2，method2调用method3，method3抛出一个异常，如图13-3所示。考虑下面的情形：

- 1) 如果异常类型是Exception3，它就会被method2中处理异常ex3的catch块捕获。跳过statement5，然后执行statement6。
- 2) 如果异常类型是Exception2，则退出method2，控制被返回给method1，而这个异常就会被method1中处理异常ex2的catch块捕获。跳过statement3，然后执行statement4。
- 3) 如果异常类型是Exception1，则退出method1，控制被返回给main方法，而这个异常就会被main方法中处理异常ex1的catch块捕获。跳过statement1，然后执行statement2。
- 4) 如果异常类型没有在method2、method1和main方法中被捕获，程序就会终止。不执行statement1和statement2。

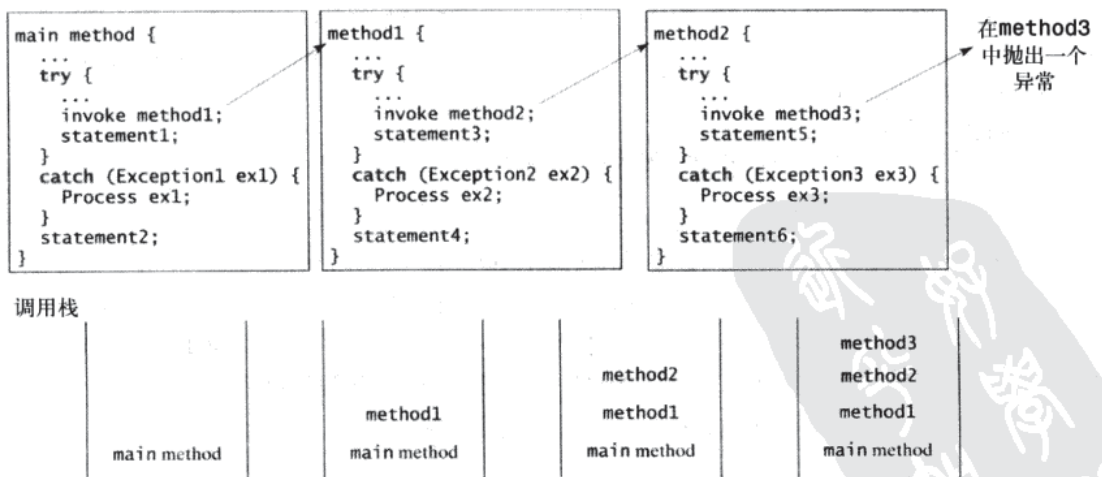
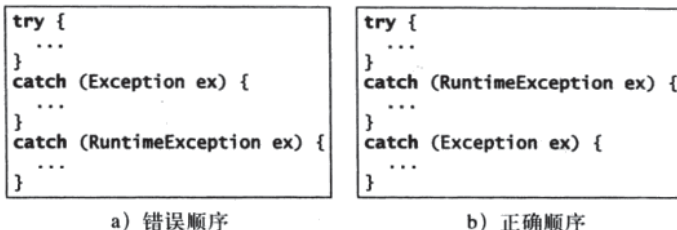


图13-3 如果异常没有在当前的方法中被捕获，就被传给该方法的调用者。

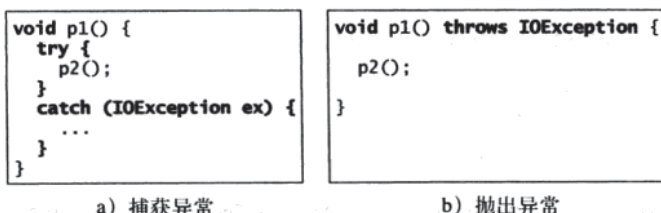
这个过程一直重复，直到异常被捕获或被传给main方法

注意 从一个通用父类可以派生出各种异常类。如果一个catch块可以捕获一个父类的异常对象，它就能捕获那个父类的所有子类的异常对象。

注意 在catch块中异常被指定的顺序是非常重要的。如果父类的catch块出现在子类的catch块之前,就会导致编译错误。例如,图a中的顺序是错误的,因为RuntimeException是Exception的一个子类。正确的顺序应该如图b中所示。



注意 Java强迫程序员处理必检异常。如果方法声明了一个必检异常(即Error或RuntimeException之外的异常),就必须在try-catch块中调用它,或者在调用方法中声明要抛出异常。例如,假定方法p1调用方法p2,而p2可能会抛出一个必检异常(例如,IOException),就必须如图a和图b所示编写代码。



13.5.4 从异常中获取信息

异常对象包含关于异常的有价值的信息。可以利用下面这些java.lang.Throwable类中的实例方法获取有关异常的信息,如图13-4所示。printStackTrace()方法在控制台上输出栈跟踪信息。getStackTrace()方法提供方案,来访问由printStackTrace()打印输出的栈跟踪信息。

java.lang.Throwable	
+getMessage(): String	返回这个消息对象的消息
+toString(): String	返回以下三个字符串的连接: (1) 异常类的全名; (2) ":" (冒号或空格); (3) getMessage()方法
+printStackTrace(): void	在控制台上打印Throwable对象以及它的调用栈的跟踪信息
+getStackTrace(): StackTraceElement[]	返回栈跟踪元素构成的数组来表示这个可抛出的栈跟踪信息

图13-4 Throwable是所有异常类的根类

程序清单13-7给出了一个例子,它使用Throwable中的方法来显示异常信息。第4行调用sum方法返回数组中所有元素的和。第23行有一个错误,该错误引起一个异常ArrayIndexOutOfBoundsException,它是IndexOutOfBoundsException的子类。该异常在try-catch块中被捕获。第7、8、9行使用printStackTrace()、getMessage()和toString()方法显示栈跟踪、异常信息、异常对象和信息,如图13-5所示。第10行将栈跟踪元素放入一个数组。每个元素表示一个方法调用。可以获得每个元素的方法(第12行)、类名(第13行)和异常行号(第14行)。

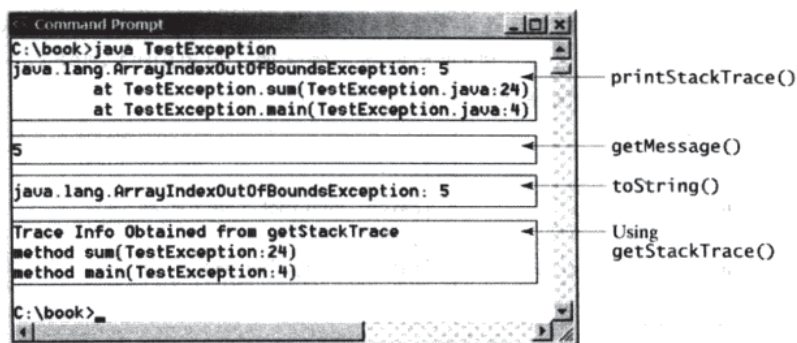


图13-5 可以使用printStackTrace()、getMessage()、toString()和getStackTrace()方法从异常对象获取信息

程序清单13-7 TestException.java

```

1 public class TestException {
2     public static void main(String[] args) {
3         try {
4             System.out.println(sum(new int[] {1, 2, 3, 4, 5}));
5         }
6         catch (Exception ex) {
7             ex.printStackTrace();
8             System.out.println("\n" + ex.getMessage());
9             System.out.println("\n" + ex.toString());
10
11             System.out.println("\nTrace Info Obtained from getStackTrace");
12             StackTraceElement[] traceElements = ex.getStackTrace();
13             for (int i = 0; i < traceElements.length; i++) {
14                 System.out.print("method " + traceElements[i].getMethodName());
15                 System.out.print("(" + traceElements[i].getClassName() + " ");
16                 System.out.println(traceElements[i].getLineNumber() + ")");
17             }
18         }
19     }
20
21     private static int sum(int[] list) {
22         int result = 0;
23         for (int i = 0; i <= list.length; i++)
24             result += list[i];
25         return result;
26     }
27 }

```

13.5.5 举例：声明、抛出和捕获异常

本例改写程序清单8-9中Circle类的setRadius方法来演示如何声明、抛出和捕获异常。如果半径是负数，那么新的setRadius方法会抛出一个异常。

将程序清单13-8中给出的circle类重命名为CircleWithException，除了setRadius(double newRadius)方法在参数newRadius为负时会抛出一个IllegalArgumentException异常之外，它与Circle3类是一样的。

程序清单13-8 CircleWithException.java

```

1 public class CircleWithException {
2     /** The radius of the circle */
3     private double radius;
4
5     /** The number of the objects created */
6     private static int numberOfObjects = 0;

```

```

7
8  /** Construct a circle with radius 1 */
9  public CircleWithException() {
10     this(1.0);
11 }
12
13 /** Construct a circle with a specified radius */
14 public CircleWithException(double newRadius) {
15     setRadius(newRadius);
16     numberOfObjects++;
17 }
18
19 /** Return radius */
20 public double getRadius() {
21     return radius;
22 }
23
24 /** Set a new radius */
25 public void setRadius(double newRadius)
26     throws IllegalArgumentException {
27     if (newRadius >= 0)
28         radius = newRadius;
29     else
30         throw new IllegalArgumentException(
31             "Radius cannot be negative");
32 }
33
34 /** Return numberOfObjects */
35 public static int getNumberOfObjects() {
36     return numberOfObjects;
37 }
38
39 /** Return the area of this circle */
40 public double findArea() {
41     return radius * radius * 3.14159;
42 }
43 }

```

程序清单13-9给出使用新Circle类的测试程序。

程序清单13-9 TestCircleWithException.java

```

1 public class TestCircleWithException {
2     public static void main(String[] args) {
3         try {
4             CircleWithException c1 = new CircleWithException(5);
5             CircleWithException c2 = new CircleWithException(-5);
6             CircleWithException c3 = new CircleWithException(0);
7         }
8         catch (IllegalArgumentException ex) {
9             System.out.println(ex);
10        }
11
12        System.out.println("Number of objects created: " +
13            CircleWithException.getNumberOfObjects());
14    }
15 }

```

```

java.lang.IllegalArgumentException: Radius cannot be negative
Number of objects created: 1

```



原始的Circle类除了下面三点之外其他保持不变：将类名改为CircleWithException，加入一个新的构造方法CircleWithException(newRadius)以及如果半径为负则setRadius方法声明了一个异

常并抛出它。

`setRadius`方法在方法头中声明为抛出`IllegalArgumentException`异常(`CircleWithException.java`中的第25~32行)。即使在方法声明中删除`throws IllegalArgumentException`子句, `CircleWithException`类也仍然会编译, 因为该异常是`RuntimeException`的子类, 而且不管是否在方法头中声明, 每个方法都能抛出`RuntimeException`异常(免检异常)。

测试程序创建三个`CircleWithException`对象: `c1`、`c2`和`c3`, 测试如何处理异常。调用`new CircleWithException(-5)`(程序清单13-9中的第5行)会导致对`setRadius`方法的调用, 因为半径为负, 所以`setRadius`方法会抛出`IllegalArgumentException`异常。在`catch`块中, 对象`ex`的类型是`IllegalArgumentException`, 它与`setRadius`方法抛出的异常对象相匹配, 因此, 这个异常被`catch`块捕获。

异常处理器使用`System.out.println(ex)`打印一个有关异常的短信息`ex.toString()`(第9行)。

注意 在异常事件中, 执行仍然会继续。如果处理器没有捕获到这个异常, 程序就会突然中断。

由于这个方法抛出`RuntimeException`(免检异常)子类`IllegalArgumentException`的一个实例, 所以, 如果不使用`try`语句, 这个测试程序也能编译。如果方法抛出`RuntimeException`和`Error`之外的异常, 那么此方法就必须在`try-catch`块内调用。

13.6 finally子句

有时候, 不论异常是否出现或者是否被捕获, 都希望执行某些代码。Java有一个`finally`子句, 可以用来达到这个目的。`finally`子句的语法如下所示:

```
try {
    statements;
}
catch (TheException ex) {
    handling ex;
}
finally {
    finalStatements;
}
```

在任何情况下, `finally`块中的代码都会执行, 不论`try`块中是否出现异常或者是否被捕获。考虑下面三种可能出现的情况:

- 1) 如果`try`块中没有出现异常, 执行`finalStatements`, 然后执行`try`语句的下一条语句。
- 2) 如果`try`块中有一条语句会引起异常, 并被`catch`块捕获, 然后跳过`try`块的其他语句, 执行`catch`块和`finally`子句。执行`try`语句之后的下一条语句。
- 3) 如果`try`块中有一条语句引起异常, 但是没有被任何`catch`块捕获, 就会跳过`try`块中的其他语句, 执行`finally`子句, 并且将异常传递给这个方法的调用者。

即使在到达`finally`块之前有一个`return`语句, `finally`块还是会执行。

注意 使用`finally`子句时可能会忽略`catch`块。

`finally`子句的一个常见用法是用于I/O程序设计中。为确保一个文件在所有情况下均被关闭, 可能要在`finally`块中放置一条文件关闭语句, 如程序清单13-10所示。

程序清单13-10 `FinallyDemo.java`

```
1 public class FinallyDemo {
2     public static void main(String[] args) {
3         java.io.PrintWriter output = null;
4
5         try {
6             // Create a file
7             output = new java.io.PrintWriter("text.txt");
```

```
8
9    // Write formatted output to the file
10   output.println("Welcome to Java");
11   }
12   catch (java.io.IOException ex) {
13       ex.printStackTrace();
14   }
15   finally {
16       // Close the file
17       if (output != null) output.close();
18   }
19
20   System.out.println("End of program");
21 }
22 }
```

第7行和第10行的语句可能会抛出异常`IOException`，因此，将它们放在`try`块中。在`finally`块中，`output.close()`语句关闭`PrintWriter`对象输出。不管`try`块中是否出现异常或者是否捕获异常，该语句都会执行。

13.7 何时使用异常

`try`块包含正常情况下执行的代码。`catch`块包含异常情况下执行的代码。异常处理将错误处理代码从正常的程序设计任务中分离出来，这样，可以使程序更易读和更易修改。但是，应该注意，由于异常处理需要初始化新的异常对象，需要从调用栈返回，而且还需要沿着方法调用链来传播异常以便找到它的异常处理器，所以，异常处理通常需要更多的时间和资源。

异常出现在方法中。如果想让该方法的调用者处理异常，应该创建一个异常对象并将其抛出。如果能在发生异常的方法中处理异常，那么就不需要抛出或使用异常。

一般来说，一个项目中多个类都会发生的共同异常应该考虑作为一种异常类。对于发生在个别方法中的简单错误最好进行局部处理，无须抛出异常。

在代码中，什么时候应该使用`try-catch`块呢？当必须处理不可预料的错误状况时应该使用它。不要用`try-catch`块处理简单的、可预料的情况。例如，下面的代码：

```
try {
    System.out.println(refVar.toString());
}
catch (NullPointerException ex) {
    System.out.println("refVar is null");
}
```

最好用以下代码代替：

```
if (refVar != null)
    System.out.println(refVar.toString());
else
    System.out.println("refVar is null");
```

哪些情况是异常的，哪些情况是可预料的，有时是很难判断的。但有一点要把握住，不要把异常处理用作简单的逻辑测试。

13.8 重新抛出异常

如果异常处理器没有处理某异常，或者处理器只是希望它的调用者注意到该异常，Java就允许异常处理器重新抛出该异常。这个语法如下所示：

```
try {
    statements;
}
catch (TheException ex) {
```

```

    perform operations before exits;
    throw ex;
}

```

语句 `throw ex` 重新抛出异常给调用者，以便调用者的其他处理器获得处理异常 `ex` 的机会。

13.9 链式异常

在前一节中，`catch` 块重新抛出原始的异常。有时候，可能需要同原始异常一起抛出一个新异常（带有附加信息）。这称为链式异常（chained exception）。程序清单13-11解释了如何产生和抛出链式异常。

程序清单13-11 **ChainedExceptionDemo.java**

```

1 public class ChainedExceptionDemo {
2     public static void main(String[] args) {
3         try {
4             method1();
5         }
6         catch (Exception ex) {
7             ex.printStackTrace();
8         }
9     }
10
11    public static void method1() throws Exception {
12        try {
13            method2();
14        }
15        catch (Exception ex) {
16            throw new Exception("New info from method1", ex);
17        }
18    }
19
20    public static void method2() throws Exception {
21        throw new Exception("New info from method2");
22    }
23 }

```

```

java.lang.Exception: New info from method1
    at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:16)
    at ChainedExceptionDemo.main(ChainedExceptionDemo.java:4)
Caused by: java.lang.Exception: New info from method2
    at ChainedExceptionDemo.method2(ChainedExceptionDemo.java:21)
    at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:13)
    ... 1 more

```



`main` 方法调用 `method1`（第4行），`method1` 调用 `method2`（第13行），`method2` 抛出一个异常（第21行）。该异常被 `method1` 的 `catch` 块所捕获，并在第16行被包装成一个新异常。该新异常被抛出，并在 `main` 方法中的第4行的 `catch` 块中被捕获。示例输出在第7行中 `printStackTrace()` 方法的结果。首先，显示从 `method1` 中抛出的新异常，然后显示从 `method2` 中抛出的原始异常。

13.10 创建自定义异常类

Java 提供相当多的异常类，尽量使用它们而不要创建自己的异常类。然而，如果遇到一个不能用预定义异常类恰当描述的问题，那就可以通过派生 `Exception` 类或其子类（例如，`IOException`）来创建自己的异常类。

在程序清单13-8中，当半径为负时，`setRadius` 方法会抛出一个异常。假设希望把这个半径传递给处理器。在这种情况下，就必须创建自定义的异常类，如程序清单13-12所示。

程序清单13-12 InvalidRadiusException.java

```

1 public class InvalidRadiusException extends Exception {
2     private double radius;
3
4     /** Construct an exception */
5     public InvalidRadiusException(double radius) {
6         super("Invalid radius " + radius);
7         this.radius = radius;
8     }
9
10    /** Return the radius */
11    public double getRadius() {
12        return radius;
13    }
14 }

```

这个自定义异常类扩展 `java.lang.Exception` (第1行)。而 `Exception` 类扩展 `java.lang.Throwable`。 `Exception` 类中的所有方法 (例如, `getMessage()`、`toString()` 和 `printStackTrace()`) 都是从 `Throwable` 继承而来的。 `Exception` 类包括四个构造方法, 其中经常使用的是下面两个构造方法:

java.lang.Exception	
+Exception() +Exception(message: String)	构造一个无消息的异常 构造一个带有特定消息的异常

第6行调用父类的带有一条消息的构造方法。这条信息将会被设置在异常对象中, 并且可以通过在该对象上调用 `getMessage()` 获得。

提示 Java API中的大多数异常类都包含两个构造方法: 一个无参构造方法和一个带消息参数的构造方法。

要创建一个 `InvalidRadiusException` 类, 必须传递一个半径。所以, 程序清单13-8中的 `setRadius` 方法必须修改如下:

```

/** Set a new radius */
public void setRadius(double newRadius)
    throws InvalidRadiusException {
    if (newRadius >= 0)
        radius = newRadius;
    else
        throw new InvalidRadiusException(newRadius);
}

```

下面代码创建一个圆对象, 并且将它的半径设置为-5。

```

try {
    CircleWithException1 c = new CircleWithException1(4);
    c.setRadius(-5);
}
catch (InvalidRadiusException ex) {
    System.out.println("The invalid radius is " + ex.getRadius());
}

```

调用 `setRadius(-5)` 方法会抛出一个 `InvalidRadiusException` 异常, 它被处理器捕获。处理器在异常对象 `ex` 中显示半径。

提示 可以扩展 `RuntimeException` 声明一个自定义异常类吗? 可以, 但这不是一个好方法, 因为这会使自定义异常成为免检的。最好使自定义异常必检, 这样, 编译器就可以在程序中强制捕获这些异常。

关键术语

chained exception (链式异常)

checked exception (必检异常)

declare exception (声明异常)

exception (异常)

exception propagation (异常传播)

throw exception (抛出异常)

unchecked exception (免检异常)

本章小结

- 异常处理能够使一个方法给它的调用者抛出一个异常。
- Java异常是派生自`java.lang.Throwable`的类的实例。Java提供大量预定义的异常类，例如，`Error`、`Exception`、`RuntimeException`、`ClassNotFoundException`、`NullPointerException`和`ArithmeticException`。也可以通过扩展`Exception`类来定义自己的异常类。
- 异常发生在一个方法的执行过程中。`RuntimeException`和`Error`都是免检异常，其他所有的异常都是必检的。
- 当声明一个方法时，如果这个方法可能抛出一个必检异常，则必须声明为必检异常，告诉编译器可能会出现什么错误。
- 声明异常的关键字是`throws`，而抛出异常的关键字是`throw`。
- 如果调用声明了必检异常的方法，必须将该方法调用放在`try`语句中。在方法执行过程中出现异常时，`catch`块会捕获并处理异常。
- 如果一个异常没有被当前方法捕获，则该异常被传给调用者。这个过程不断重复直到异常被捕获或者传递给`main`方法。
- 可以从一个通用的父类派生出各种不同的异常类。如果一个`catch`块捕获到父类的异常对象，它也能捕捉这个父类的子类的所有异常对象。
- 在`catch`块中，异常被指定顺序是非常重要的。如果在一个类的父类的异常对象之前没有指定这个类的一个异常对象，就会导致一个编译错误。
- 当方法中发生异常时，如果异常没有被捕获，方法将会立刻退出。如果方法想在退出前执行一些任务，可以在方法中捕获这个异常，然后再重新抛给真正的处理器。
- 任何情况下都会执行`finally`块中的代码，不管`try`块中是否出现或者捕获了异常。
- 异常处理将错误处理代码从正常的程序设计任务中分离出来，这样，就会使得程序更易于阅读和修改。
- 不应该使用异常处理代替简单的测试。应该尽可能地测试简单异常，将异常处理保留为处理那些无法用`if`语句处理的异常。

复习题

13.1~13.3节

13.1 描述Java的`Throwable`类、子类以及异常的类型。如果有异常的话，下面的程序会抛出什么`RuntimeException`异常？

```
public class Test {
    public static void main(String[] args) {
        System.out.println(1 / 0);
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        int[] list = new int[5];
        System.out.println(list[5]);
    }
}
```

b)

```
public class Test {
    public static void main(String[] args) {
        String s = "abc";
        System.out.println(s.charAt(3));
    }
}
```

c)

```
public class Test {
    public static void main(String[] args) {
        Object o = new Object();
        String d = (String)o;
    }
}
```

d)

```
public class Test {
    public static void main(String[] args) {
        Object o = null;
        System.out.println(o.toString());
    }
}
```

e)

```
public class Test {
    public static void main(String[] args) {
        System.out.println(1.0 / 0);
    }
}
```

f)

13.2 给出下面代码的输出：

```
public class Test {
    public static void main(String[] args) {
        for (int i = 0; i < 2; i++) {
            System.out.print(i + " ");
            try {
                System.out.println(1 / 0);
            }
            catch (Exception ex) {
            }
        }
    }
}
```

a)

```
public class Test {
    public static void main(String[] args) {
        try {
            for (int i = 0; i < 2; i++) {
                System.out.print(i + " ");
                System.out.println(1 / 0);
            }
        }
        catch (Exception ex) {
        }
    }
}
```

b)

13.3 指出下面代码的问题。这些代码会抛出什么异常吗？

```
long value = Long.MAX_VALUE + 1;
System.out.println(value);
```

13.4 声明异常的目的是什么？怎样声明一个异常，在哪里声明？在一个方法头中可以声明多个异常吗？

13.5 什么是必检异常？什么是免检异常？

13.6 如何抛出一个异常？可以在一个throw语句中抛出多个异常吗？

13.7 关键字throw的作用是什么？关键字throws的作用是什么？

13.8 发生异常后，Java虚拟机会做什么？如何捕获一个异常？

13.9 下面代码的输出是什么？

```
public class Test {
    public static void main(String[] args) {
        try {
            int value = 30;
            if (value < 40)
                throw new Exception("value is too small");
        }
        catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        System.out.println("Continue after the catch block");
    }
}
```

如果将int value=30; 这一行替换为int value=50;，输出结果会是什么？

13.10 假设下面的try-catch块中的statement2引起一个异常：

```
try {
    statement1;
    statement2;
    statement3;
}
catch (Exception1 ex1) {
```

新华书店
PDG

```

}
catch (Exception2 ex2) {
}

```

statement4;

回答下列问题:

- 会执行statement3吗?
- 如果异常未被捕获, 会执行statement4吗?
- 如果在catch块中捕获了异常, 会执行statement4吗?
- 如果异常被传递给调用者, 会执行statement4吗?

13.11 运行下面程序时会显示什么?

```

public class Test {
    public static void main(String[] args) {
        try {
            int[] list = new int[10];
            System.out.println("list[10] is " + list[10]);
        }
        catch (ArithmeticException ex) {
            System.out.println("ArithmeticException");
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException");
        }
        catch (Exception ex) {
            System.out.println("Exception");
        }
    }
}

```

13.12 运行下面程序时会显示什么?

```

public class Test {
    public static void main(String[] args) {
        try {
            method();
            System.out.println("After the method call");
        }
        catch (ArithmeticException ex) {
            System.out.println("ArithmeticException");
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException");
        }
        catch (Exception e) {
            System.out.println("Exception");
        }
    }

    static void method() throws Exception {
        System.out.println(1 / 0);
    }
}

```

13.13 运行下面程序时会显示什么?

```

public class Test {
    public static void main(String[] args) {
        try {
            method();
            System.out.println("After the method call");
        }
        catch (RuntimeException ex) {
            System.out.println("RuntimeException in main");
        }
    }
}

```



```

    }
    catch (Exception ex) {
        System.out.println("Exception in main");
    }
}

static void method() throws Exception {
    try {
        String s = "abc";
        System.out.println(s.charAt(3));
    }
    catch (RuntimeException ex) {
        System.out.println("RuntimeException in method()");
    }
    catch (Exception ex) {
        System.out.println("Exception in method()");
    }
}
}

```

13.14 方法getMessage()可以做什么?

13.15 方法printStackTrace可以做什么?

13.16 没有异常发生时, try-catch块的存在会引起额外的系统开销吗?

13.17 修改下面代码中的编译错误:

```

public void m(int value) {
    if (value < 40)
        throw new Exception("value is too small");
}

```

13.4~13.10节

13.18 假设下面的语句中, statement2会引起一个异常:

```

try {
    statement1;
    statement2;
    statement3;
}
catch (Exception1 ex1) {
}
catch (Exception2 ex2) {
}
catch (Exception3 ex3) {
    throw ex3;
}
finally {
    statement4;
};
statement5;

```

回答以下问题:

- 如果异常没有被捕获, 会执行语句statement5吗?
- 如果异常类型是Exception3, 那么会执行statement4吗? 会执行statement5吗?

13.19 假定setRadius方法抛出13.7节中定义的RadiusException异常, 那么运行下面的程序时会显示什么?

```

public class Test {
    public static void main(String[] args) {
        try {
            method();
            System.out.println("After the method call");
        }
        catch (RuntimeException ex) {

```



```

        System.out.println("RuntimeException in main");
    }
    catch (Exception ex) {
        System.out.println("Exception in main");
    }
}

static void method() throws Exception {
    try {
        Circle c1 = new Circle(1);
        c1.setRadius(-1);
        System.out.println(c1.getRadius());
    }
    catch (RuntimeException ex) {
        System.out.println("RuntimeException in method()");
    }
    catch (Exception ex) {
        System.out.println("Exception in method()");
        throw ex;
    }
}
}
}

```

13.20 下面的方法检查一个字符串是否是数值字符串:

```

public static boolean isNumeric(String token) {
    try {
        Double.parseDouble(token);
        return true;
    }
    catch (java.lang.NumberFormatException ex) {
        return false;
    }
}

```

该方法是否正确? 不使用异常重写该方法。

编程练习题

13.2~13.10节

*13.1 (NumberFormatException异常) 程序清单9-5是一个简单的命令行计算器。注意, 如果某个操作数是非数值的, 程序就会中止。编写一个程序, 利用异常处理器来处理非数值操作数; 然后编写另一个不使用异常处理器的程序, 达到相同的目的。程序在退出之前应该显示一条信息, 通知用户发生了操作数类型错误 (参见图13-6)。

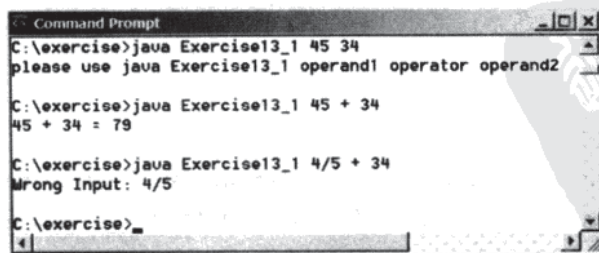


图13-6 程序完成算术运算并检查输入错误

*13.2 (NumberFormatException异常) 编写一个程序, 提示用户读取两个整数, 然后显示它们的和。程序应该在输入不正确时提示用户再次读取数字。

*13.3 (ArrayIndexOutOfBoundsException异常) 编写一个满足下面要求的程序:

- 创建一个由100个随机选取的整数构成的数组。
- 提示用户输入数组的下标，然后显示对应的元素值。如果指定的下标越界，就显示消息Out of Bounds。

*13.4 (IllegalArgumentException异常) 修改程序清单10-2中的Loan类，如果贷款总额、利率或年数小于或等于零，则抛出IllegalArgumentException异常。

*13.5 (IllegalTriangleException异常) 练习题11.1定义了带三条边的Triangle类。在三角形中，任意两边之和总大于第三边，三角形类Triangle必须遵从这一规则。创建一个IllegalTriangleException类，然后修改Triangle类的构造方法，如果创建的三角形的边违反了这一规则，抛出一个IllegalTriangleException对象，如下所示：

```
/** Construct a triangle with the specified sides */  
public Triangle(double side1, double side2, double side3)  
    throws IllegalTriangleException {  
    // Implement it  
}
```

*13.6 (NumberFormatException异常) 程序清单9-2实现了hexToDecimal(String hexString)方法，它将一个十六进制字符串转换为一个十进制数。实现这个hexToDecimal方法，在字符串不是一个十六进制字符串时抛出NumberFormatException异常。

*13.7 (NumberFormatException异常) 练习题9.8指出binaryToDecimal(String binaryString)方法是将一个二进制字符串转换为一个十进制数。实现binaryToDecimal方法，在字符串不是一个二进制字符串时抛出NumberFormatException异常。

*13.8 (HexFormatException异常) 练习题13.6实现hexToDecimal方法，在字符串不是一个十六进制字符串时抛出NumberFormatException异常。定义一个名为HexFormatException的自定义异常。实现hexToDecimal方法，在字符串不是一个十六进制字符串时抛出HexFormatException异常。

*13.9 (BinaryFormatException异常) 练习题13.7实现binaryToDecimal方法，在字符串不是一个二进制字符串时抛出BinaryFormatException异常。定义一个名为BinaryFormatException的自定义异常。实现binaryToDecimal方法，在字符串不是一个二进制字符串时抛出BinaryFormatException异常。

*13.10 (OutOfMemoryError错误) 编写一个程序，它能导致JVM抛出一个OutOfMemoryError，然后捕获和处理这个错误。

