第5章 异常处理

本章学习目标

- 了解异常的概念以及异常的分类。
- 掌握捕获和处理异常的方法。
- 掌握自定义异常和抛出异常的方法。

本章首先介绍了异常处理的意义以及异常的分类，然后介绍了如何使用异常处理机制处理异常，最后介绍了自定义异常和抛出异常对象的方法。

5.1 异常处理简介

异常是程序在运行过程中发生的由于硬件设备问题、软件设计错误等导致的程序异常事件，对于面向对象的程序设计语言 Java 中来说异常本身是一个对象，产生异常就是产生了一个异常对象。

5.1.1 异常处理的意义

对于计算机程序而言，没有人能保证自己写的程序永远不会出错。就算程序没有错误，也无法保证用户总是按照你的意愿来输入，就算用户都是非常“聪明而且配合”的，也无法保证运行该程序的操作系统、机器硬件、网络链接等不发生意外情况。而对于一个程序设计人员来说，需要尽可能地预知所有可能发生的意外情况，尽可能地保证程序在所有可能的情况下都可以运行。针对这种情况，Java 语言引入了异常，以异常类的形式对这些非正常情况进行封装，通过异常处理机制对程序运行时发生的各种异常情况进行处理。接下来通过一个案例来认识一下什么是异常。

例 5.1 认识异常。

```
public class ExceptionExam {
    public static void main(String[] args) {
        int i = 0;
            int a[] = {1,2,3};
            while (i < 4) {
                System.out.println(a[i]);
                i++;
            }
```

```
        System.out.println(" ********** 程序运行结束 ********** ");
    }
}
```

程序运行结果如图 5.1 所示。

```
1
2
3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at ExceptionExam.main(ExceptionExam.java:7)
```

图 5.1　例 5.1 程序运行结果

在以上程序中，因为引用数组元素出现了下标越界的情况，即抛出了 ArrayIndexOfBoundsException 异常。从运行结果可以发现，如果不对异常进行处理，则一旦出现了异常，程序就立刻退出，所以后面的语句并没有执行。

例 5.2　对例 5.1 中的异常进行处理。

```
public class ExceptionExam {
    public static void main(String[] args) {
        int i = 0;
            int a[] = {1,2,3};
            while (i < 4) {
                System.out.println(a[i]);
                i++;
            }
        System.out.println(" ********** 程序运行结束 ********** ");
    }
}
```

程序运行结果如图 5.2 所示。

```
1
2
3
出现异常了! java.lang.ArrayIndexOutOfBoundsException: 3
**********程序运行结束**********
```

图 5.2　例 5.2 程序运行结果

在本例程序中进行了异常处理，虽然也出现了 ArrayIndexOfBoundsException 异常，但程序完整地执行了。那么为什么需要异常处理呢？通过本例我们可以了解进行异常处理的目的：

- 当程序运行时出现了异常时，能够改变程序的执行流程，给程序以正确的执行出口，让程序能够完整地运行。
- 为程序员标识出异常代码的位置（通常在异常处理中利用输出语句）。

Java 语言是面向对象的程序设计语言，因此也使用面向对象的方法来处理异常。在程序运行过程中，一旦发生了异常，这个方法就生成代表该异常的一个对象，并把该对象交付给运行时的系统，运行时系统找到相应的代码来处理这一异常，这一过程称为异常的捕获

(catch)。使用 Java 异常处理机制有以下优点：

(1) 将异常处理代码从常规代码中分离出来，增强了程序的可读性。

(2) 将出现的异常按类型和差别进行分组，增加程序可读性的同时，分清了责任。

(3) 可以对无法预测的异常进行捕获和处理。

(4) 克服了传统方法受 if 语句限制、错误信息有限的问题。

(5) 可以把异常向上传送到调用它的方法中。

Java 异常处理通常分为两步：一是抛出(throw)异常。在方法的运行过程中，如果发生了异常，则该方法生成一个代表该异常的对象并把它交给运行时系统，运行时系统便寻找相应的代码来处理这一异常；二是捕获(catch)异常，运行时系统在方法的调用栈中查找，从生成异常的方法开始进行回溯，直到找到包含相应异常处理的方法为止。

5.1.2　异常的分类

ArrayIndexOfBoundsException 异常只是 Java 异常类中的一种，在 Java 中还提供了大量的异常类，这些类都继承自 java. lang. Throwable 类。接下来通过一张图来展示 Throwable 类的继承体系，如图 5.3 所示。

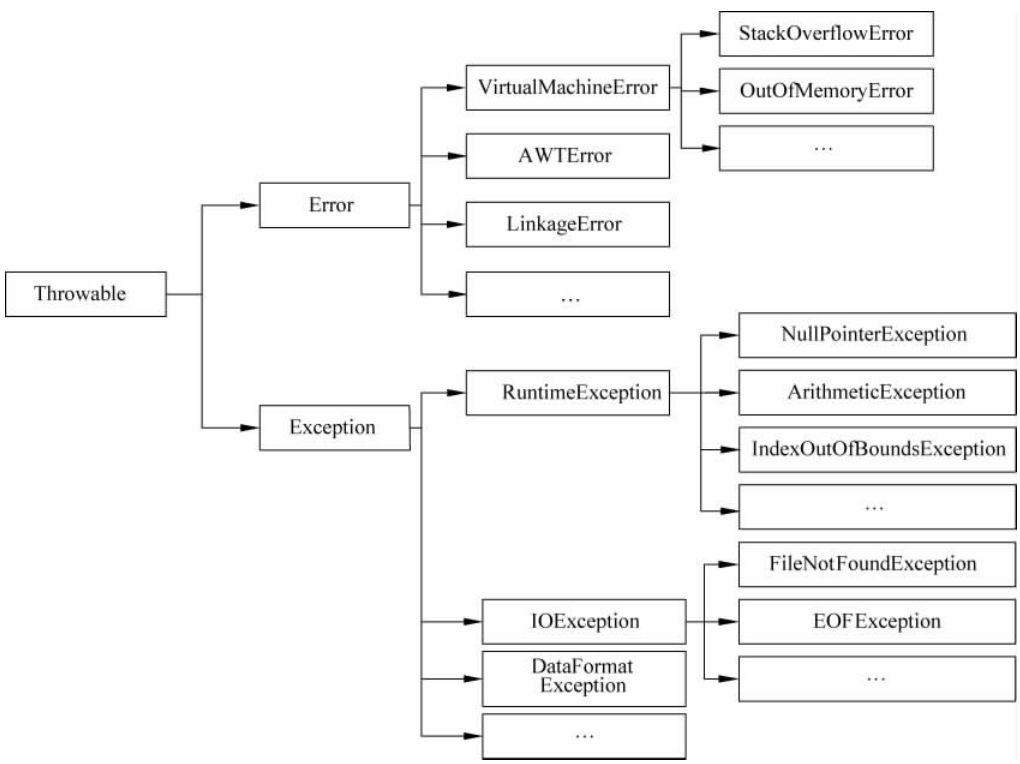

图 5.3　Throwable 类的继承体系

通过图 5.3 可以看出，Throwable 有两个直接子类 Error 和 Exception，其中 Error 代表程序中产生的错误，Exception 代表程序中产生的异常。

1. Error 类

Error 类也称为错误类，它表示 Java 运行时产生的系统内部错误或资源耗尽的错误，是比较严重的错误，仅靠修改程序本身是不能恢复执行的。这种错误很少发生，如果发生，除了通知用户以及尽量稳妥地结束程序外，几乎什么也不能做。例如，使用 Java 命令去运行一个不存在的类就会出现 Error 错误。

2. Exception 类

Exception 类称为异常类，它表示程序本身可以处理的错误，在开发 Java 程序中进行的异常处理，都是针对 Exception 类及其子类。在 Exception 类的众多子类中有一个特殊的 RuntimeException 类，该类及其子类用于表示运行时异常，除了此类，Exception 类下所有其他的子类都用于表示编译时异常，即系统定义的异常分为运行时异常和编译时异常（也称为非运行时异常）。

- 运行时异常：例如，数组下标越界、算术运算异常等，该类异常在语法上不强制程序员必须处理，即使不处理这样的异常也不会出现语法错误。并且这种异常可以通过适当的编程加以避免，例如，数组下标就不应该越界。由于这类异常可以避免，所以从算法角度来看，Java 不需要捕获这种异常，程序不处理这类异常也能通过编译，但有异常出现时程序会发生中断，程序不能完整地执行。
- 编译时异常（非运行时异常）：例如，I/O 异常，该类异常在语法上强制程序员必须进行处理，如果不进行处理则会出现语法错误，编译不能通过。

也就是说，异常分两种：一种是必须处理的，另一种是不要求处理的。对于必须处理的，则要么捕获（catch），要么声明抛出（throws），即所谓的“要么捕获，要么抛出”。

熟悉异常类的分类，将有助于后续语法中的处理，也使得在使用异常类时可以选择恰当的异常类类型。Java 预定义了一些常见异常，如图 5.3 所示，下面对几种最常见的系统异常加以介绍。

1）Exception

Exception 类是 Throwable 的一个子类，是其他异常的直接或间接父类，通常用以下两种构造方式：

- Exception()——构造详细消息为 null 的新异常。
- Exception(String message)——构造带指定详细消息的新异常。

Exception 类从父类 Throwable 那里继承了若干方法，其中常用的有以下几种：

- public String getMessage()——返回此 Exception 的详细消息字符串。
- public void printStackTrace()——在当前的标准输出（一般就是屏幕显示）将此 Exception 打印输出当前异常对象的堆栈使用痕迹，也就是程序先后调用并执行了哪些对象或类的哪些方法，使得运行过程中产生了这个异常对象。
- public String toString()——返回此 Exception 的简短描述。

2）ArithmeticException

算数异常，当整数除法中除数为 0 时，则会产生该类异常，例如：

```
int i = 85/0;
```

3）NullPointerException

如果访问的对象还没有实例化，那么访问该对象将出现空指针异常 NullPointerException。例如：

```
Date d = null;
System.out.println(d.getTime());
```

此刻会产生 NullPointerException 异常。

4）NegativeArraySizeException

数组元素的个数应该大于等于零，如果创建数组时元素个数是负数，则会出现 NegativeArraySizeException 异常。

5）ArrayIndexOutOfBoundsException

数组用 length 常量来记录数组的大小。访问数组元素时，如果数组下标越界，则产生 ArrayIndexOutOfBoundsException 异常，如例 5.1。

6）ArrayStoreException

试图将错误类型的对象存储到一个对象数组时抛出的异常。例如，以下代码会生成一个 ArrayStoreException：

```
Object x[] = new String[3];
x[0] = new Integer(0);
```

7）FileNotFoundException

当试图打开一个不存在的文件时，抛出此异常。在不存在具有指定路径名的文件时，此异常将由 FileInputStream、FileOutputStream 或 RandomAccessFile 构造方法抛出。如果该文件存在，但是由于某些原因不可访问，比如试图打开一个只读文件进行写入，则此时这些构造方法仍然会抛出该异常。

8）IOException

当发生某种 I/O 异常时，抛出此异常。此类是失败或中断的 I/O 操作生成的异常的通用类。

5.1.3 捕获和处理异常

对于编译时异常，Java 强迫程序必须进行处理。异常处理包括以下 3 个步骤：

(1) 程序的执行过程中如果出现异常，会自动生成一个异常类对象，该异常对象将被提交给 Java 运行时系统，这个过程称为抛出(throw)异常。

(2) 当 Java 在运行时系统接收到异常对象，会寻找处理这种异常对象的代码，并把当前异常对象交给其处理，这一过程称为捕获(catch)异常。

(3) 如果 Java 运行时系统找不到可以捕获异常的方法，也就是说，出现异常但未捕获，则运行时系统将终止，Java 程序将退出。

1. 捕获异常的 try-catch-finally 语句

Java 中捕获异常会用到 try-catch-finally 语句。将可能抛出异常的代码写在 try 语句块中，用 catch 方法来捕获异常及相应的处理代码。具体语法格式如下：

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

运行时，根据发生的异常类型，找到相应的 catch 方法，然后执行其后的语句序列。finally 语句块的作用通常是用于释放资源，finally 不是必需的，如果有 finally 部分，那么无论是否捕获到异常，总要执行 finally 后面的语句块。

例 5.3 运行时异常处理。

```
public class ExceptionIndexOutOf{
    public static void main(String[ ] args)     {
        String student[ ] = {"李强","张海波","刘兴军"};
        try {
            for(int i = 0;i < 5;i++) {
                System.out.println(student[i]);
            }
        } catch(ArrayIndexOutOfBoundsException e)  {
            System.out.println("出错,数组下标越界!!");
        }finally{
            System.out.println("这里是 finally 执行的部分");
        }
        System.out.println("\n 程序结束!!");
        }
}
```

本例中，程序执行循环部分，当 i 等于 4 时，执行“System.out.println(student[i]);”语句会产生抛出一个 ArrayIndexOutOfBoundsException（数组下标越界）异常，该异常会被 catch 捕获，捕获后进行了简单处理，输出了产生异常的提示信息。之后执行了 finally 后面的语句输出“这里是 finally 执行的部分”。之后程序继续向后执行，程序得以完整地执行，程序运行结果如图 5.4 所示。

```
李强
张海波
刘兴军
出错，数组下标越界！！
这里是Finally执行的部分

程序结束！！
```

图 5.4 程序运行结果

在本程序中发生了 ArrayIndexOutOfBoundsException(数组下标越界)异常,该异常是运行时异常,如果不进行异常处理,程序是能够通过编译并执行的,只是执行循环部分,当 i 等于 4 时出现了数组下标越界异常,程序发生中断,不能完整地执行。

例 5.4 编译时异常处理。

```
import java.io.*;
public class IOExceptionTest {
 public static void main (String[] args) {
 char c;
 System.out.println("请输入一个字符:");
 try{
     c = (char)System.in.read();     //read()方法中有未处理的编译时异常,即 IOException
     System.out.println("输入的字符是" + c);
      }catch(IOException e){
      e.getMessage();   }
 System.out.println("程序运行结束!");
   }
}
```

本例中的 read()方法中有未处理的编译时异常,即 IOException,对于编译时异常必须进行异常处理,否则会产生编译错误,因此这里采用了 try-catch-finally 语句进行了异常处理,使得程序能够通过编译运行。

2. throws 声明异常

在定义一个方法时可以使用 throws 关键字声明,使用 throws 声明的方法表示此方法不处理异常,而交给方法的调用处进行处理,即谁调用此方法谁处理此异常。throws 使用格式如下:

```
public 返回值类型  方法名称(参数列表) throws 异常类列表{
      方法体
}
```

其中,异常类列表中的多个异常类之间用逗号间隔。

例 5.5 使用 throws 关键字。

```
class Math{
    public int div(int x,int y) throws Exception{//方法不处理异常,谁调用谁处理
        int result = x/y;
        return result;
    }
}
public class Test{
    public static void main(String args[]){
        Math o = new Math();
        try{
    //Math 类的 div()方法使用 throws 声明了异常,不管是否会产生异常,都必须处理
            System.out.println("除法操作:" + o.div(10, 0));
        }catch(Exception e){
```

```
            e.printStackTrace();
        }
        System.out.println(" ***** 程序运行结束 ***** ");
    }
}
```

在本例 Math 类中定义的 div()方法时，由于考虑到运算可能会产生异常，但在这里不想进行处理，因此使用了 throws 关键字，表示不管是否会有异常，在调用此方法处都必须进行异常处理。

例 5.6 对编译时异常采用 throws 关键字进行声明。

```
import java.io.*;
public class IOExceptionTest {
 public static void main(String[] args) throws IOException {
    char c;
    System.out.println("请输入一个字符:");
    c = (char)System.in.read();
    System.out.println("输入的字符是" + c);
   }
}
```

本例的 read()方法中有未处理的编译时异常，即 IOException，对于编译时异常必须进行异常处理，否则产生编译错误，因此这里在定义 main()主方法时，使用关键字 throws 声明了此异常，表示此时主方法不处理异常。

主方法为程序的起点，所以此时主方法再向上声明抛出异常，则只能抛给 Java 虚拟机进行处理了，程序发生异常时会导致程序发生中断。

3. 多异常处理

捕获异常时，catch 方法可以有一个或多个，而且至少要有一个 catch 方法或 finally 语句。每个 catch 方法通常会用同种方式来处理它所接收的所有异常，但常常在一个 try 语句块中可能产生多种不同的异常，也就是说，有多个异常需要捕获时，就需要定义多个 catch 方法来实现，每个 catch 方法用来接收和处理一种特定异常。

当 try 语句抛出一个异常时，程序流程首先转向第一个 catch 方法，并检查当前异常对象是否可以被这个 catch 方法所接收。可接收是指异常对象可以与 catch 方法的参数类型相匹配。这可以是 3 种情况：

- 异常对象与参数属于相同的类；
- 异常对象是参数类的子类；
- 异常对象实现了参数类接口。

如果异常对象被第一个 catch 方法所接收，则程序的流程将直接跳转到这个 catch 语句块中，语句执行完成后，跳出该方法。如果 try 语句抛出的异常与第一个 catch 方法参数不匹配，就转向第二个，如果第二个不匹配就转向第三个……直到找到匹配的参数类型。如果存在 finally 语句，catch 方法执行完成后则执行 finally 语句，之后继续执行后面的代码；否则直接执行后面的代码。

如果此过程中所有的 catch 方法的参数类型都不能与当前的异常对象相匹配，则说明

当前方法不能处理这个异常，程序流程将返回到调用该方法的上层方法。如果这个上层方法中定义了与所产生的异常对象相匹配的 catch 方法，流程则跳转到这个 catch 方法中，否则继续回溯到更上层的方法。如果所有的方法中都找不到合适的 catch 方法进行异常处理，则由 Java 运行时系统来处理这个异常对象，通常会中止程序的运行，退出虚拟机返回到操作系统，并在标准输出上输出相关的异常信息。

在多异常处理过程中，处理异常类型的顺序很重要，在类层次中，一般的异常类型要放在后面，特殊的放在前面。因此在设计 catch 方法处理不同的异常时，要注意如下的问题：

- catch 语句块中的语句应根据异常的不同而执行不同的操作，这样当有异常发生时有助于程序员快速地定位产生异常的代码，比较通用的操作是打印异常和错误的相关信息，包括异常名称、产生异常的方法名等。
- 由于异常对象与 catch 语句的匹配是按照 catch 语句的先后排列顺序进行的，所以在处理多异常时应注意认真设计各 catch 语句的先后顺序。一般地，将处理较具体和较常见的异常的 catch 语句放在前面，而可以与多种异常相匹配的 catch 语句放在较后的位置。若将子类异常的 catch 处理语句放在父类的后面，则编译不能通过。

例 5.7 多异常处理。

```
class Demo{
    int div(int a,int b) throws ArithmeticException,ArrayIndexOutOfBoundsException{
        //在功能上通过 throws 的关键字声明该功能可能出现除零异常或数组下标越界
          int []arr = new int [a];
          System.out.println(arr[4]);     //制造的第一处异常,a 长度小于 5 则下标越界
          return a/b;                     //制造的第二处异常,除零异常
    }
}
public class MutliExceptionDemo{
    public static void main(String[] args)  {
        Demo d = new Demo();
          try  {
               int x = d.div(4,0);
                //程序运行的 3 组示例,分别对应此处的 3 行代码
                //int x = d.div(5,0);
                //int x = d.div(4,1);
                System.out.println("x = " + x);
        }
        catch (ArithmeticException e)    {
            System.out.println(e.toString());
        }
        catch (ArrayIndexOutOfBoundsException e)   {
            System.out.println(e.toString());
        }
        catch (Exception e) {
            //父类写在此处是为了捕捉其他没预料到的异常,只能写在子类异常的代码后面
            System.out.println(e.toString());
        }
        System.out.println("程序结束 ");
    }
}
```

当主方法代码执行“int x = d.div(4,0);”时产生如图 5.5 所示的异常，即下标越界异常，除零异常没有发生。这说明第一个异常被捕获以后就终止了 try 块内的程序运行，转入到 catch 块中执行。

```
java.lang.ArrayIndexOutOfBoundsException: 4
程序结束
```

图 5.5 数组下标越界异常

当执行“int x = d.div(5,0);”时产生除零算数异常，如图 5.6 所示。

```
0
java.lang.ArithmeticException: / by zero
程序结束
```

图 5.6 除零异常

当执行“int x = d.div(4,1);”时产生数组下标越界异常，如图 5.7 所示。这与图 5.5 显示的效果一样。

```
java.lang.ArrayIndexOutOfBoundsException: 4
程序结束
```

图 5.7 数组下标越界异常

本例中定义方法 int div(int a,int b)时，throws 关键字声明抛出了 ArithmeticException 和 ArrayIndexOutOfBoundsException 两种异常，这是一种处理异常的方法。异常的处理一般有两种方法：

- 一是使用 try-catch-finally 语句，捕获所发生的异常，并进行相应的处理。
- 二是指不在当前方法内处理，而是把异常声明抛出到调用方法中，由调用它的方法进行异常的处理。

如本例所示，定义方法时可以通过 throws 关键字声明多个异常，此时谁调用此方法，就由谁通过 try-catch-finally 语句处理 throws 声明的异常，也可以继续使用 throws 继续向上声明抛出给调用它的方法，直到最顶层的主方法。如果是所有方法都用 throws 声明抛出了异常，则最后由 JVM(Java 虚拟机)来最终捕获处理该异常，通常是输出相关的错误信息，并终止程序的运行。将例 5.5 程序修改如下：

例 5.8 在主方法定义中使用 throws 关键字。

```
class Math{
    public int div(int x,int y) throws Exception{          //方法不处理异常,谁调用谁处理
        int result = x/y;
        return result;
    }
}
public class Test{                                          //主方法不处理异常,交由 JVM 处理
    public static void main(String args[]) throws Exception{
```

```
        Math o = new Math();
        System.out.println("除法操作:" + o.div(10, 0));   //程序中断,下面的语句不执行
        System.out.println(" ***** 程序运行结束 ***** ");
    }
}
```

多个异常需要捕获时,异常类型的顺序很重要,在类型层次中,一般的异常类型放在后面,特殊的放在前面。将例 5.7 程序修改如下。

例 5.9 多异常处理中异常类型的顺序。

```
class Demo{
    int div(int a,int b) throws ArithmeticException,ArrayIndexOutOfBoundsException{
        //在功能上通过 throws 的关键字声明该功能可能出现除零异常或数组下标越界
          int []arr = new int [a];
          System.out.println(arr[4]);       //制造的第一处异常,a 长度小于 5 则下标越界
          return a/b;                       //制造的第二处异常,除零异常
    }
}
public class ExceptionDemo{
     public static void main(String[] args)  {
        Demo d = new Demo();
          try  {
                  int x = d.div(4,0);
                 System.out.println("x = " + x);
        }
        catch (Exception e) {
                  System.out.println(e.toString());
               }
        catch (ArithmeticException e)  {
            System.out.println(e.toString());
        }
        catch (ArrayIndexOutOfBoundsException e)   {
            System.out.println(e.toString());
        }
        System.out.println("程序结束 ");
    }
}
```

此时编译器对 catch (Exception e)正常通过,但当编译到 catch(ArithmeticException e)和 catch(ArrayIndexOutOfBoundsException e)时报错,显示该两类异常无法获取。

5.2 自定义异常类与抛出异常对象

5.2.1 声明自己的异常类

对于常见的可预见的运行错误可使用系统定义的异常,但对于某个应用所特有的运行错误,则需要程序员根据程序逻辑,在用户程序中自行创建用户自己的异常类,我们称之为自定义异常类和异常对象。这种用户自定义的异常类通常用来处理用户程序中特定的可预

见的逻辑运行错误。使得这种逻辑错误能够被系统识别并处理，而不是放置不管，任其扩散。从而使得程序更加健壮，增强了程序的容错性，使得整个系统更加健全。

创建一个用户自定义异常需要完成以下几方面的工作：

(1) 定义一个新的异常类，使之成为 Exception 类或其他已经存在的某个系统异常类或用户异常类的子类。

(2) 新的异常类中定义的属性和方法，或重载父类的属性和方法，能够体现该类所对应的错误信息。

当用户自定义异常类后，就可以和系统定义的异常类一样的使用。因此，对于一个完善的应用系统来说，定义足够多的异常类是使其稳定运行的重要基础之一。

通常计算两个整数之和的方法不应当有任何异常产生，但是，对某些特殊应用程序，可能不允许同号的整数做求和运算，比如当一个整数代表收入，一个整数代表支出时，这两个整数就不能是同号。在银行业务中，收入(入账资金)必须是正数，支出必须是负数。因此定义银行类 Bank 有一个 income(int in,int out)方法，对象调用该方法时，必须向参数 in 传递正整数、向参数 out 传递负数，并且 int＋out 必须大于等于 0，否则该方法就抛出异常 BankException。因此，Bank 类在声明 income(int in,int out)方法时，使用 throws 关键字声明要产生的异常。下面通过实现自定义异常类 BankException，掌握异常类的定义和使用。

例 5.10 自定义异常类 BankException。

```
public class BankException extends Exception{
    String mes;
    public BankException(int in,int out) {
        mes = "收入资金" + in + "是负数或支出" + out + "是正数,不符合系统要求.";
            }
    public String warnMess(){
        return mes;
            }
    }
public class Bank {
    private int money;
    public int getMoney() {
        return money;
      }
public void income(int in,int out) throws BankException{
    int newIncome = in + out;
    if (in < 0 || out > = 0 || newIncome < 0){
         throw new BankException(in,out);
     }
    System.out.println("本次收入: " + newIncome + "元");
    money = money + newIncome;
     }
public static void main(String[] args) {
    Bank b = new Bank();
    try{
    b.income(200, -100);
    b.income(400, -100);
```

```
        b.income(600, -100);
        System.out.println("银行现有存款: " + b.getMoney() + "元");
        b.income(200, 100);                    //此处支付为正数,会产生异常
        b.income(400, -100);
        }catch(BankException e){
        System.out.println("计算收益过程中出现问题: " + e.warnMess());
        }
        System.out.println("银行现有存款: " + b.getMoney() + "元");
        }
}
```

5.2.2 抛出异常对象

1. throw 异常对象

Java 程序在运行时,如果引发一个可识别的错误,就会产生一个与错误相对应的异常类对象,这个过程称为异常的抛出。根据异常类的不同,抛出异常的方法也不相同,一般有两种:

(1) 系统自动抛出异常对象。所有系统定义的异常都可以由系统自动抛出异常对象,当产生异常时,就会抛出指定异常对象,这个对象可以被捕获。

(2) 语句抛出异常。一般情况下,系统是无法自动抛出用户自定义异常的,必须借助 throw 语句来定义何种情况下会产生了这种异常,并抛出该异常类的新对象,其语法格式为:

```
throw 异常对象;
```

在例 5.10 中下面的语句控制抛出 BankException 对象。

```
if (in<0 || out>=0 || newIncome<0){
          throw new BankException(in,out);
     }
```

当 in<0 或 out>=0 再或者 newIncome<0 时,就会抛出 BankException 对象。

一般情况下,这种抛出异常的语句应该被定义在满足一定条件时执行,就像上面的语句,把 throw 语句放在 if 语句之中,只有当满足一定条件,即用户定义的逻辑错误发生时,才执行。

2. throws 与 throw 的区别

(1) throws 关键字通常被应用在声明方法时,用来指定可能抛出的异常,多个异常可以使用逗号隔开。当调用该方法时,如果发生异常,就会抛出指定的异常对象。

例 5.11 定义方法并使用 throws 声明抛出异常。

```
public class ExceptionThrowsTest {
    static void pop() throws NegativeArraySizeException {
        //定义方法并抛出 NegativeArraySizeException 异常
        int[] arr = new int[-4];                    //创建数组
    }
    public static void main(String[] args) {       //主方法
```

```
        try {
            pop();                                          //调用 pop()方法
        } catch (NegativeArraySizeException e) {
            System.out.println("pop()方法抛出的异常"); //输出异常信息
        }
    }
}
```

(2) throw 关键字通常用在方法体中,并且抛出一个异常对象,此异常必须得到处理,否则产生编译错误。throw 抛出的异常通常有两种处理方式:

- 如果要捕获 throw 抛出的异常,则必须使用 try-catch-finally 语句。throw 语句用在 try 语句块中,程序在执行到 throw 语句时立即停止,即 try 语句块中 throw 语句后面的语句都不执行。
- 通过 throw 抛出异常后,如果想在上一级方法中来捕获并处理异常,则需要在抛出异常的方法中使用 throws 关键字,在方法声明中指明 throw 抛出的异常。

例 5.12 使用关键字 throw 抛出异常的处理。

```
class MyException extends Exception {                       //创建自定义异常类
    String message;                                         //定义 String 类型变量
    public MyException(String ErrorMessagr) {               //构造方法
        message = ErrorMessagr;
    }
    public String getMessage() {                            //覆盖父类的 getMessage()方法
        return message;
    }
}
public class ExceptionThrowTest {
    static int quotient(int x, int y) throws MyException {  //定义方法抛出异常
        if (y < 0) {                                        //判断参数是否小于 0
            throw new MyException("除数不能是负数");      //异常信息
        }
        return x / y;                                       //返回值
    }
    public static void main(String args[]) {                //主方法
        try {                                               //try 语句包含可能发生异常的语句
            int result = quotient(3, -1);                   //调用方法 quotient()
        } catch (MyException e) {                           //处理自定义异常
            System.out.println(e.getMessage());             //输出异常信息
        } catch (ArithmeticException e) {
            //处理 ArithmeticException 异常
            System.out.println("除数不能为 0");             //输出提示信息
        } catch (Exception e) {                             //处理其他异常
            System.out.println("程序发生了其他的异常");
            //输出提示信息
        }
    }
}
```

5.3 使用 assert 断言

在程序运行过程中使用异常是为了避免程序运行时出现的不可控行为。而在程序编制过程中,还需要保证程序写得正确,也就是需要确认程序在某一时刻必须产生一个合理的结果。Java 中提供了这样一种机制,就是断言(assertion)。

编写代码时,我们总是会做出一些假设,断言就是用于在代码中捕捉这些假设。断言表示为一些布尔表达式,程序员相信在程序中的某个特定点该表达式值为真,可以在任何时候启用和禁用断言验证,在调试阶段让断言发挥作用,这样就可以发现一些致命错误。当程序正式部署运行时就可以关闭断言语句,但仍把断言语句保留在源代码中,如果以后应用程序又有需要,可以重新启动断言。

使用断言可以创建更稳定、品质更好且不易于出错的代码。若需要在一个值为 false 时中断当前操作的话,可以使用断言。单元测试必须使用断言(JUnit/JUnitX)。

Java 中使用 assert 声明断言,断言通常有两种格式:

```
assert 逻辑表达式;
assert 逻辑表达式:描述信息;
```

其中逻辑表达式是一个布尔值,描述信息是断言失败时输出的失败消息的字符串。

例如,对于如下断言语句:

```
assert money>=100;
```

如果表达式 money>=100 的值为 true,程序继续执行,否则程序立即结束。

例 5.13 在程序中使用断言 assert。

```
public class Assertion {
    public static void main(String[] args) {
        int[] score = {99,89,102,60, -3};
        int sum = 0;
       for (int i = 0;i < score.length;i++){
          assert score[i]>=0&&score[i]<=100:"成绩必须大于或等于零,小于或等于一百";
          sum = sum + score[i];
       }
   System.out.println("总成绩 = " + sum);
    }
}
```

正常运行该程序运行结果为

```
总成绩 = 347
```

程序并没有出现中断,这是因为默认情况下使用 Java 解释器运行应用程序时是关闭断言语句的,在程序调试时可以使用-ea 参数来启动断言,即

```
Java -ea Assertion
```

如果在 Eclipse 中使用断言,则可以如下操作:

运行打开“运行 配置”对话框，在“自变量”选项卡的“VM 自变量”文本框中加上断言开启的标志-enableassertions 或者-ea 就可以了，如图 5.8 所示。

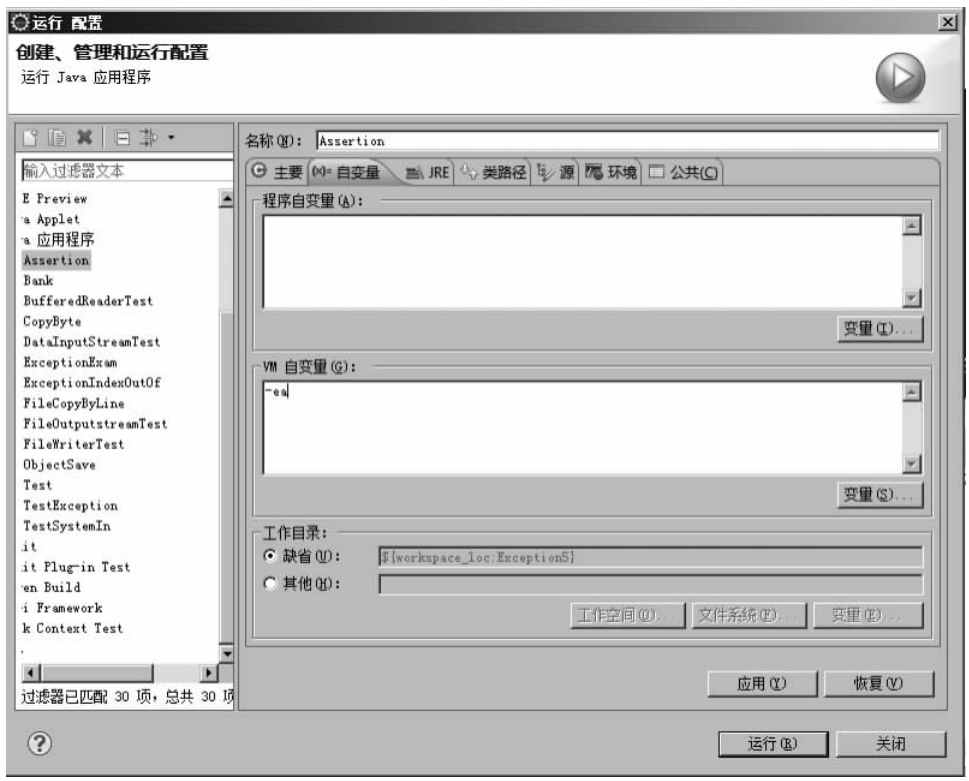

图 5.8 在 Eclipse 中启动断言运行

启动断言后，程序运行结果如图 5.9 所示，可见，启动断言后出现了错误提示。

```
Exception in thread "main" java.lang.AssertionError: 成绩必须大于等于零，小于等于一百
        at Assertion.main(Assertion.java:11)
```

图 5.9 启动断言后的运行结果

5.4 本章小结

(1) 异常处理的目的。

- 当程序运行时出现了异常，能够改变程序的执行流程，给程序以正确的执行出口，让程序能够完整地运行。
- 为程序员标识出异常代码的位置(通常在异常处理中利用输出语句)。

(2) 在 Java 中还提供了大量的异常类，这些类都继承自 java. lang. Throwable 类。Throwable 有两个直接子类 Error 和 Exception，其中 Error 类也称为错误类，它表示 Java 运行时产生的系统内部错误或资源耗尽的错误，是比较严重的错误，仅靠修改程序本身是不

能恢复执行的。Exception 类称为异常类，它表示程序本身可以处理的错误，在开发 Java 程序中进行的异常处理，都是针对 Exception 类及其子类。

(3) 在 Exception 类的众多子类中有一个特殊的 RuntimeException 类，该类及其子类用于表示运行时异常，除了此类，Exception 类下所有其他的子类都用于表示编译时异常，即系统定义的异常分为运行时异常和编译时异常(也称为非运行时异常)。

- 运行时异常：该类异常在语法上不强制程序员必须处理，即使不处理这样的异常也不会出现语法错误。Java 不需要捕获这种异常，程序不处理这类异常也能通过编译，但有异常出现时程序会发生中断，程序不能完整的执行。
- 编译时异常(非运行时异常)：该类异常在语法上强制程序员必须进行处理，如果不进行处理，则会出现语法错误，编译不能通过。

(4) 捕获和处理异常的方法：

- Java 中捕获异常会用到 try-catch-finally 语句。将可能抛出异常的代码写在 try 语句块中，用 catch 方法来捕获异常及相应的处理代码。
- 在定义一个方法时可以使用 throws 关键字声明，使用 throws 声明的方法表示此方法不处理异常，而交给方法的调用处进行处理，即谁调用此方法谁处理此异常。throws 使用格式如下：

```
public 返回值类型 方法名称(参数列表) throws 异常类列表{
    方法体
}
```

(5) 在多异常处理过程中，处理异常类型的顺序很重要，在类层次中，一般的异常类型要放在后面，特殊的放在前面。

(6) 创建一个用户自定义异常需要完成以下几方面的工作：

- 定义一个新的异常类，使之成为 Exception 类或其他已经存在的某个系统异常类或用户异常类的子类。
- 新的异常类中定义的属性和方法，或重载父类的属性和方法，能够体现该类所对应的错误信息。

(7) 根据异常类的不同，抛出异常的方法也不相同。一般有两种：

- 系统自动抛出异常对象。所有系统定义的异常都可以由系统自动抛出异常对象，当产生异常时，就会抛出指定异常对象，这个对象可以被捕获。
- 语句抛出异常。一般情况下，系统是无法自动抛出用户自定义异常的，必须借助于 throw 语句来定义何种情况下会产生了这种异常，并抛出该异常类的新对象，其语法格式为

```
throw 异常对象;
```

(8) Java 中使用 assert 声明断言，断言通常有两种格式：

```
assert 逻辑表达式;
assert 逻辑表达式:描述信息;
```

其中逻辑表达式是一个布尔值，描述信息是断言失败时输出的失败消息的字符串。