```
3           //下面的代码定义了一个try...catch...finally语句用于捕获异常
4             try {
5                 int result = divide (4, 0) ;       //调用divide () 方法
6                 System.out.println (result) ;
7             } catch (Exception e) {                 //对捕获到的异常进行处理
8                 System.out.println ("捕获的异常信息为: " + e.getMessage () ) ;
9               return;                             //用于结束当前语句
10            } finally {
11                System.out.println ("进入finally代码块") ;
12            }
13          System.out.println ("程序继续向下执行…") ;
14      }
15      //下面的方法实现了两个整数相除
16      public static int divide (int x, int y) {
17          int result = x / y;             //定义一个变量result记录两个数相除的结果
18          return result;                  //将结果返回
19      } .
20  }
```

文件 4–26 的运行结果如图 4–27 所示。

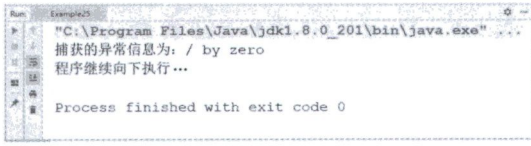

图4–26　文件4–25的运行结果

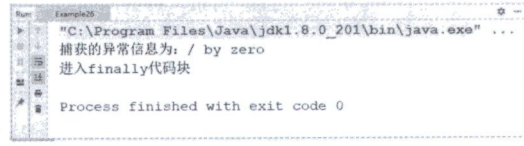

图4–27　文件4–26的运行结果

在文件 4–26 中，第 9 行代码是在 catch 代码块中增加了一个 return 语句，用于结束当前方法，这样第 13 行代码就不会执行了，而 finally 代码块中的代码仍会执行，不受 return 语句的影响。也就是说，不论程序是发生异常，还是使用 return 语句结束，finally 中的语句都会执行。因此，在程序设计时，通常会使用 finally 代码块处理完成必须做的事情，例如释放系统资源。

需要注意的是，finally 中的代码块在一种情况下是不会执行的，那就是在 try...catch 中执行了 System.exit（0）语句。System.exit（0）表示退出当前的 Java 虚拟机，Java 虚拟机停止了，任何代码都不能再执行了。

4.7.3　throws 关键字

在文件 4–26 中，由于调用的是自己编写的 divide（）方法，因此很清楚该方法可能发生的异常。但是在实际开发中，大部分情况下会调用别人编写的方法，并不知道别人编写的方法是否会发生异常。针对这种情况，Java 允许在方法的后面使用 throws 关键字对外声明该方法有可能发生的异常，这样调用者在调用方法时，就明确地知道该方法是否有异常，并且必须在程序中对异常进行处理，否则编译无法通过。

throws 关键字声明抛出异常的语法格式如下：

```
修饰符 返回值类型 方法名（参数1，参数2……）throws 异常类1，异常类2……{
    //方法体……
}
```

从上述语法格式中可以看出，throws 关键字需要写在方法声明的后面，throws 后面需要声明方法中发生异常的类型。下面修改文件 4–26，在 divide（）方法中声明可能出现的异常类型，如文件 4–27 所示。

文件 4-27　Example27.java

```
1   public class Example27 {
2       public static void main (String[] args) {
3           int result = divide (4, 2) ;    //调用divide () 方法
4           System.out.println (result) ;
5       }
6       //下面的方法实现了两个整数相除，并使用throws关键字声明抛出异常
7       public static int divide (int x, int y) throws Exception {
8           int result = x / y;             //定义一个变量result记录两个数相除的结果
9           return result;                  //将结果返回
10      }
11  }
```

编译文件 4–27，编译器报错，如图 4–28 所示。