为参数传递给这个块。如你所见
原因（运行时系统还会将原因自动输出）。

在@catch 块中的最后一条语句执行后（这里只有一条语句），程序会立即继续执行之后的语句。在这个例子中会执行 NSLog 语句，验证能够继续执行并没有终止。

这是一个非常简单的例子，演示了如何在程序中捕获异常。无论@try 块中是否抛出异常@finally 块中的语句代码都会执行。

@throw 指令允许你抛出自己的异常。可以使用该指令抛出特定的异常，或者在@catch 块内抛出带你进入类似如下代码块的异常：

```
@throw;
```

自行处理异常后（例如，可能是在执行清除工作后），可能需要这么做。然后便可以让系统处理其余的工作。最后，可以使用多个@catch 块按顺序捕获并处理各种异常。

一般来说，你并不希望程序在运行时发生异常。这就需要考虑更好的编程实践，在错误发生之前做测试，而不是错误发生后捕获它。测试方法的错误并返回一些值作为错误的标识，而不是抛出异常。抛出异常通常会使用大量的系统资源，Apple 反对非必要的使用异常（例如，你不希望因为一个文件无法打开而抛出异常）。

9.7 练习

1. 如果在代码清单 9-1 中执行加法之后插入以下消息表达式，将发生什么情况？试一试并查看结果。

```
[compResult reduce];
```

2. 可以将代码清单 9-2 中定义的 id 变量 dataValue 分配给在第 8 章中定义的 rectangle 对象吗？就是说，表达式

```
dataValue = [[Rectangle alloc] init];
```

是否合法？为什么？

3. 给第 8 章中定义的 XYPoint 类中添加一个 print 方法。让它以格式（x，y）显示一个点。然后修改代码清单 9-2 来结合一个 XYPoint 对象，使修改后的程序创建一个 XYPoint 对象，设置其值，把这个值分配给 id 变量 datavalue，最后显示它的值。

4. 基于本章关于参数和返回类型的讨论，修改 Fraction 和 Complex 类的 add:方法来选取并返回 id 对象。然后编写一个程序并添加以下代码序列

```
result = [dataValue1 add: dataValue2];
[result print];
```

其中，result、dataValue1 和 dataValue2 都是 id 对象。确保在程序中适当地设置 dataValue1 和 dataValue2，并且在程序结束之前释放所有的对象。

注意

必须将方法名改为 add:的其他名称。这是因为系统的 NSObjectController 类也有一个 add:方法。如 9.4 节所述，如果在不同的类中有多个同名的方法，并且在编译时不知道接收器的类型，那么编译器就会执行一致性检查，确保参数和返回类型在名称类似的方法之间保持一致。

5. 根据本章中使用的 Fraction 和 Complex 类定义以及如下定义：

```
Fraction *fraction = [[Fraction alloc] init];
Complex  *complex = [[Complex alloc] init];
id    number = [[Complex alloc] init];
```

确定以下消息表达式的返回值。然后将它们输入一个程序，验证结果。

```
[fraction isMemberOfClass: [Complex class]];
[complex isMemberOfClass: [NSObject class]];
[complex isKindOfClass: [NSObject class]];
[fraction isKindOfClass: [Fraction class]];
[fraction respondsToSelector:@selector (print)];
[complex respondsToSelector:@selector (print)];
[Fraction instancesRespondToSelector:@selector (print)];
[number respondsToSelector: @selector(print)];
[number isKindOfClass: [Complex class]];
[[number class] respondsToSelector:@selector (alloc)];
```