API java.lang.reflect.Constructor 1.1

- Object newInstance(Object... params)
  将 params 传递到构造器，来构造这个构造器声明类的一个新实例。参见 5.9.7 节更多地了解如何提供参数。

API java.lang.Throwable 1.0

- void printStackTrace()
  将 Throwable 对象和栈轨迹打印到标准错误流。

5.9.2 声明异常入门

我们将在第 7 章中全面地介绍异常处理机制，但现在时常遇到一些可能抛出异常的方法。

当运行时发生错误时，程序就会“抛出一个异常”。抛出异常比终止程序要灵活得多，这是因为你可以提供一个处理器（handler）“捕获”这个异常并进行处理。

如果没有提供处理器，程序就会终止，并在控制台上打印出一个消息，给出异常的类型。你可能在前面已经看到过一些异常报告，例如，不小心使用了 null 引用或者数组越界时。

异常有两种类型：非检查型（unchecked）异常和检查型（checked）异常。对于检查型异常，编译器将会检查你（程序员）是否知道这个异常并做好准备来处理后果。不过，有很多常见的异常（例如，越界错误或者访问 null 引用）都属于非检查型异常。编译器并不期望你为这些异常提供处理器。毕竟，你应该集中精力避免这些错误的发生，而不是为它们编写处理器。

不是所有的错误都是可以避免的。如果竭尽全力还是可能发生异常，大多数 Java API 都会抛出一个检查型异常。Class.forName 方法就是一个例子。没有办法确保有指定名字的类一定存在。在第 7 章中，将会看到几种异常处理策略。现在，我们只介绍最简单的一个策略。

如果一个方法包含一条可能抛出检查型异常的语句，则在方法名上增加一个 throws 子句。

```
public static void doSomethingWithClass(String name)
      throws ReflectiveOperationException
{
   Class cl = Class.forName(name); // might throw exception
   do something with cl
}
```

调用这个方法的任何方法也都需要一个 throws 声明，这也包括 main 方法。如果一个异常确实出现，则 main 方法将终止并提供一个栈轨迹。（在第 7 章中，你将了解如何捕获异常而不是因异常终止程序。）

只需要为检查型异常提供一个 throws 子句。很容易找出哪些方法会抛出检查型异常——只要你调用了一个可能抛出检查型异常的方法而没有提供相应的异常处理器，编译器就会报错。