List<Apple> result = *filterApples*(inventory,

(Apple apple) -> "red".equals(apple.getColor()));

可以在函数式接口上使用Lambda表达式。在上面的代码中，你可以把Lambda表达式作为第二个参数传给filter方法，因为它这里需要Predicate<T>，而这是一个函数式接口。

#### 1、函数式接口

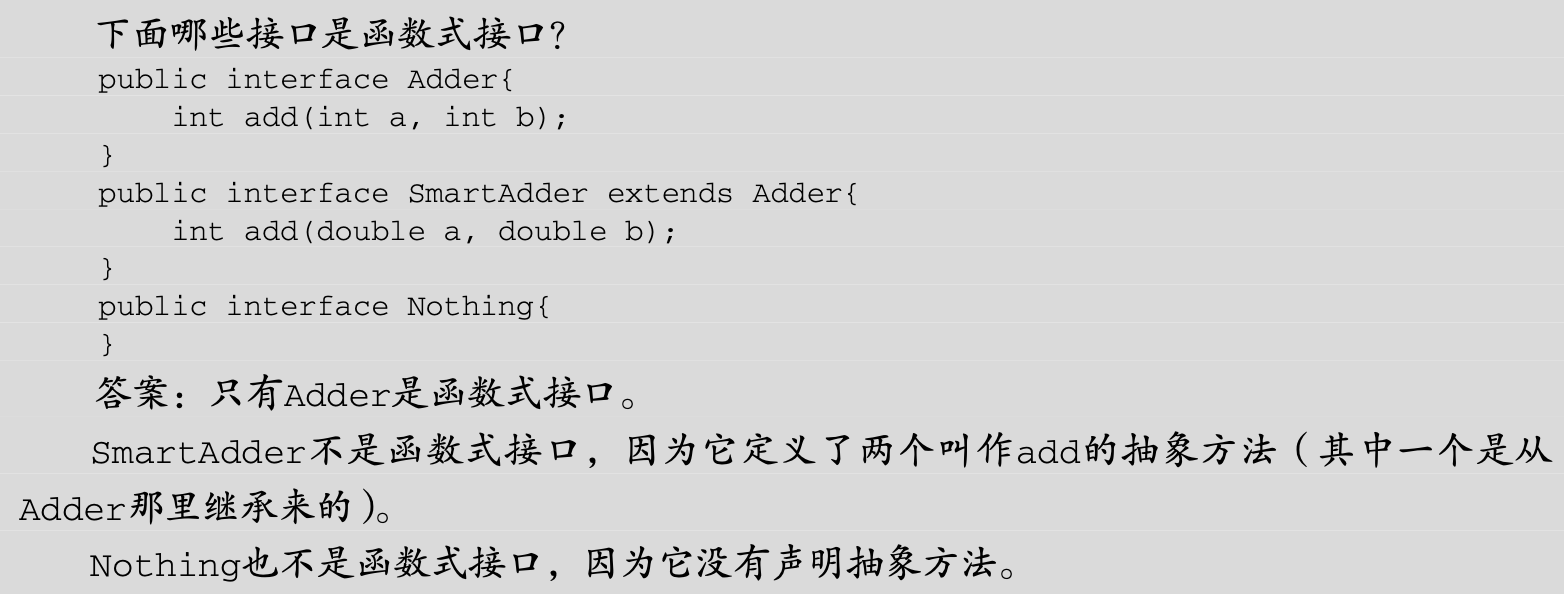
之前，为了参数化filter方法的行为而创建的Predicate<T>接口，这就是一个函数式接口。Predicate仅仅只定义了一个抽象方法：

boolean test(T t);

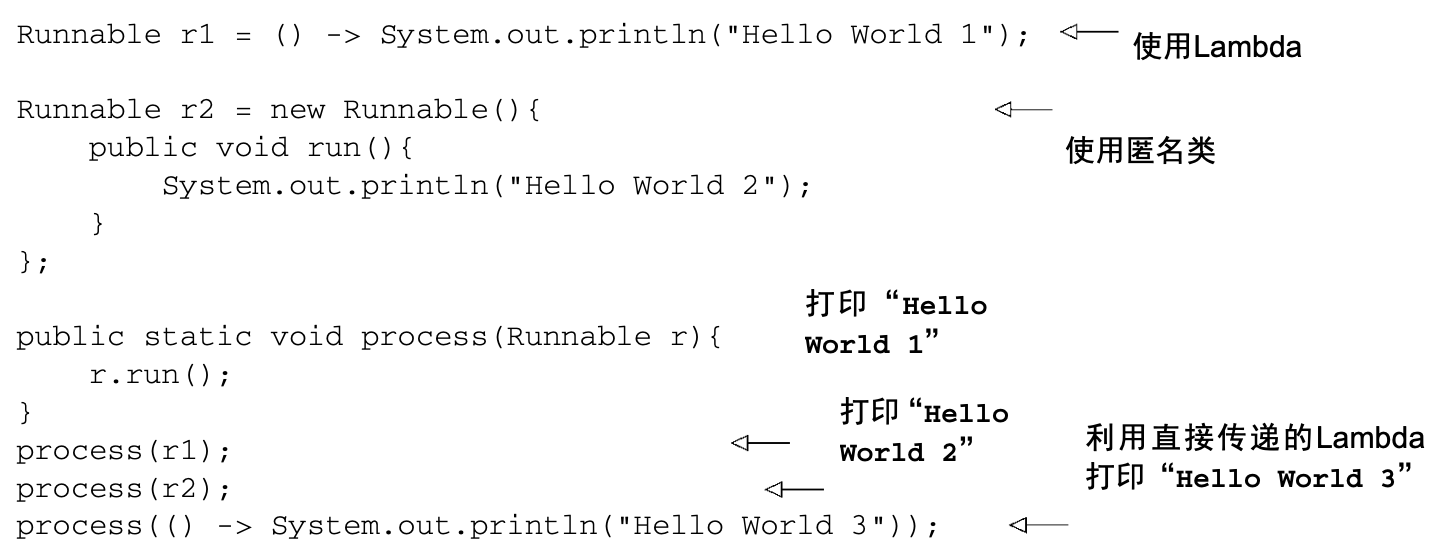
函数式接口就是只定义一个抽象方法，但是可以有多个非抽象方法的接口。

注意：接口现在还可以拥有默认方法（即在类没有对方法进行实现时，其主体

为方法提供默认实现的方法）。哪怕有很多默认方法，只要接口只定义了一个抽象方法，它就仍然是一个函数式接口。



用函数式接口可以干什么呢？Lambda表达式允许你直接以内联的形式为函数式接口的抽象方法提供实现，并把整个表达式作为函数式接口的实例（具体来说，是函数式接口一个具体实现的实例）。你用匿名内部类也可以完成同样的事情，只不过比较笨拙：需要提供一个实现，然后再直接内联将它实例化。下面的代码是有效的，因为Runnable是一个只定义了一个抽象方法run的函数式接口：



#### 2、函数描述符

函数式接口的抽象方法的前面基本上就是Lambda表达式的签名。我们将这种抽象方法叫作函数描述符。例如，Runnable接口可以看作一个什么也不接受什么也不返回（void）的函数的签名，因为它只有一个叫作run的抽象方法，这个方法什么也不接受，什么也不返回。

本章中使用一个特殊表示法来描述Lambda和函数式接口的签名。() -> void代表了参数列表的为空，且返回void的函数。这正是Runnable接口所代表的。举另一个例子，(Apple, Apple) -> int代表接受两个Apple作为参数且返回int的函数。我们会在3.4节和本章后面的 表3-2中提供关于函数描述符的更多信息。

你可能会想：Lambda表达式是怎么做类型检查的。下面会介绍编译器是如何检查Lambda在给定上下文中是否有效的。现在，只要知道Lambda表达式可以被赋给一个变量，或传递给一个接受函数式接口作为参数的方法就好了，当然这个Lambda表达式的签名要和函数式接口的抽象方法一样。

你可能会想：为什么只有在需要函数式接口的时候才可以传递Lambda呢？因为这种方式自然且能避免语言变得更复杂。此外，大多数Java程序员都已经熟悉了具有一个抽象方法的接口的理念（例如事件处理）。

@FunctionalInterface

函数式接口带有这个注解，用于表示该接口会设计成一个函数式接口。如果你用@FunctionalInterface定义一个接口，而它却不是函数式接口的话，编译器将会返回一个提示原因的错误。