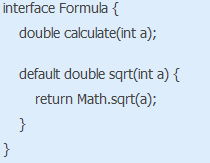
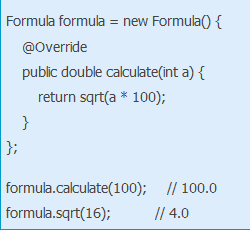
java1.8新特性

1. **接口的默认方法**

java 8允许我们给接口添加一个非抽象的方法实现，只需要使用 default关键字即可，这个特征又叫做扩展方法，示例如下：

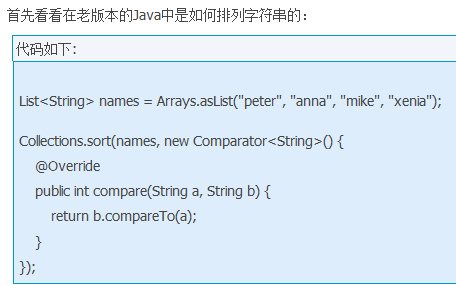


Formula接口在拥有calculate方法之外同时还定义了sqrt方法，实现了Formula接口的子类只需要实现一个calculate方法，默认方法sqrt将在子类上可以直接使用。



尽管默认方法非常强大，但是在使用默认方法时我们需要小心注意一个地方：在声明一个默认方法前，请仔细思考是不是真的有必要使用默认方法，因为默认方法会带给程序歧义，并且在复杂的继承体系中容易产生编译错误。

1. **Lambda 表达式**



只需要给静态方法 Collections.sort 传入一个List对象以及一个比较器来按指定顺序排列。通常做法都是创建一个匿名的比较器对象然后将其传递给sort方法。

在Java 8 中你就没必要使用这种传统的匿名对象的方式了，Java 8提供了更简洁的语法，lambda表达式：

Collections.sort(names, (String a, String b) -> {

return b.compareTo(a);

});

看到了吧，代码变得更段且更具有可读性，但是实际上还可以写得更短：

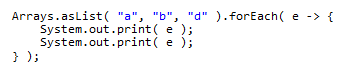
Collections.sort(names, (String a, String b) -> b.compareTo(a));

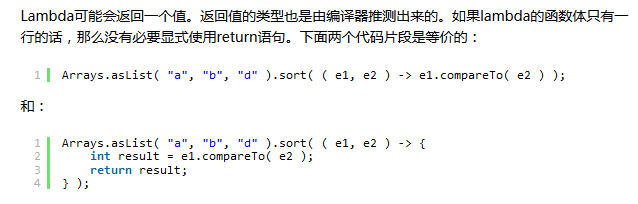
对于函数体只有一行代码的，你可以去掉大括号{}以及return关键字，但是你还可以写得更短点：

Collections.sort(names, (a, b) -> b.compareTo(a));

Java编译器可以自动推导出参数类型，所以你可以不用再写一次类型。接下来我们看看lambda表达式还能作出什么更方便的东西来.

在某些情况下lambda的函数体会更加复杂，这时可以把函数体放到在一对花括号中，就像在Java中定义普通函数一样。例如：





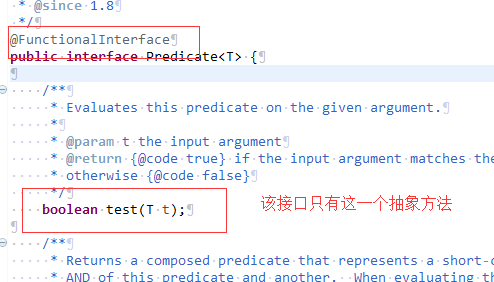
**注**：通常都会把lambda表达式内部变量的名字起得短一些。这样能使代码更简短，放在同一行。所以，在上述代码中，变量名选用a、b或者x、y会比even、odd要好。

1. **函数式接口**

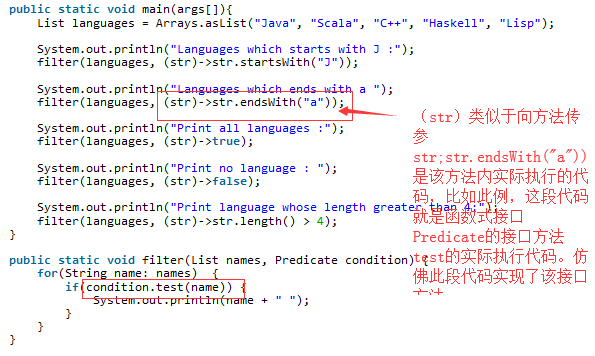
函数式接口就是只有一个抽象方法的普通接口。像这样的接口，可以被隐式转换为lambda表达式。java.lang.Runnable与java.util.concurrent.Callable是函数式接口最典型的两个。

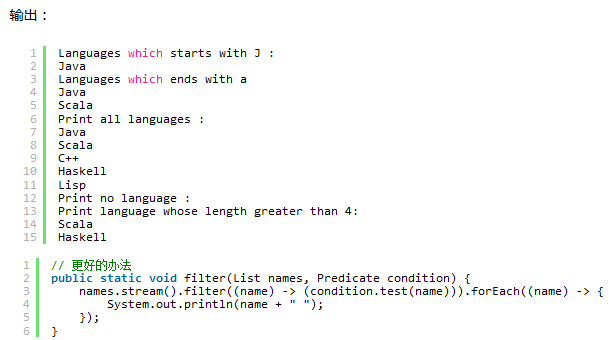
Lambda表达式是如何在java的类型系统中表示的呢？每一个lambda表达式都对应一个类型，通常是接口类型。而“函数式接口”是指仅仅只包含一个抽象方法的接口，每一个该类型的lambda表达式都会被匹配到这个抽象方法。因为 默认方法 不算抽象方法，所以你也可以给你的函数式接口添加默认方法。

我们可以将lambda表达式当作任意只包含一个抽象方法的接口类型，确保你的接口一定达到这个要求，你只需要给你的接口添加 @FunctionalInterface 注解，编译器如果发现你标注了这个注解的接口有多于一个抽象方法的时候会报错的。



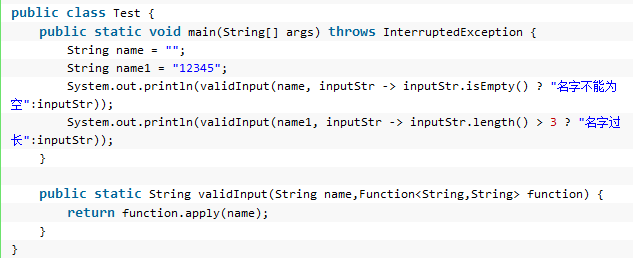
除了在语言层面支持函数式编程风格，Java 8也添加了一个包，叫做 java.util.function。它包含了很多类，用来支持Java的函数式编程。其中一个便是Predicate，使用 java.util.function.Predicate 函数式接口以及lambda表达式，可以向API方法添加逻辑，用更少的代码支持更多的动态行为。





**Function接口：**

接受一个参数，返回一个参数。



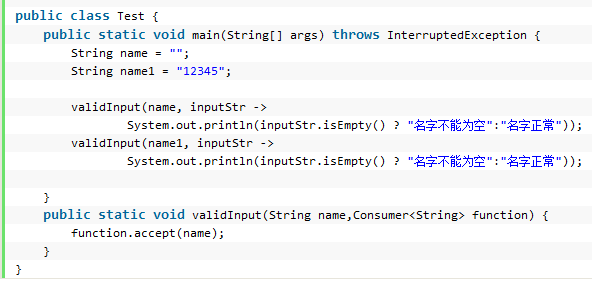
解释：

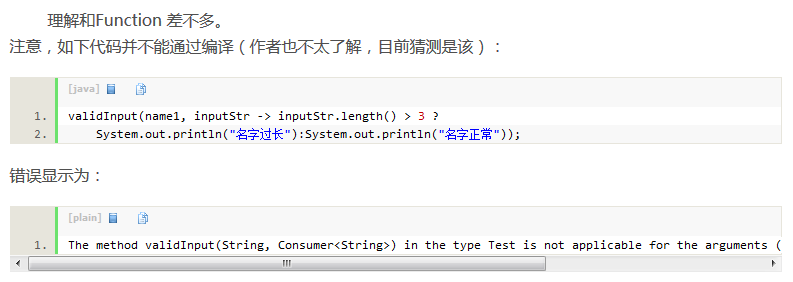
1. 定义 validInput 方法，传入 function 接口，然后在该方法中定义 function.apply(name)，也就是说，传入一个 name 参数，应用某些规则，返回一个结果，至于是什么规则，先不定义。

2.在main 方法中调用 validInput(name,inputStr ...)，这里我们定义规则，利用lambda 表达式， 规则是：传入一个 inputStr 字符串，如果为空，返回 xx；否则 返回 xx。

**Consumer接口：**

接受一个参数，不返回参数。

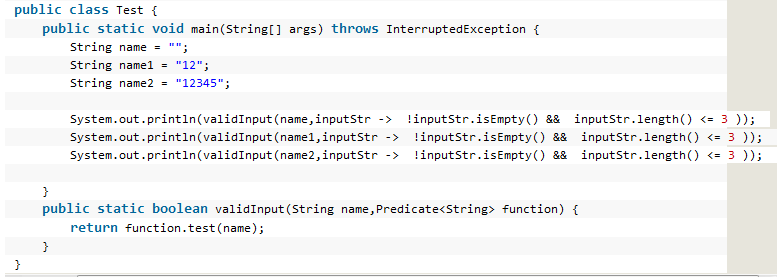




**Predicate接口：**

用于测试是否符合条件。

Predicate 方法 表示 判断 输入的对象是否 符合某个条件。

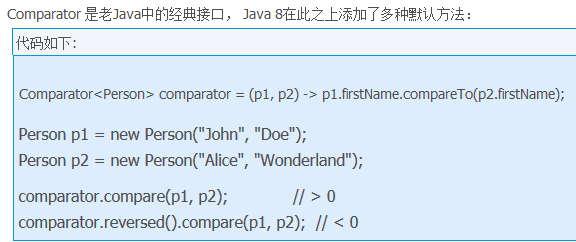


**Supplier接口：**

Supplier 接口返回一个任意范型的值，和Function接口不同的是该接口没有任何参数。



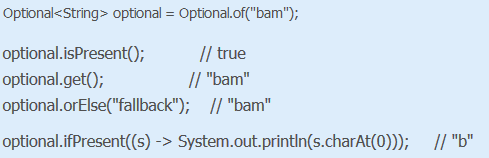
**Comparator接口：**



**Optional 接口：**

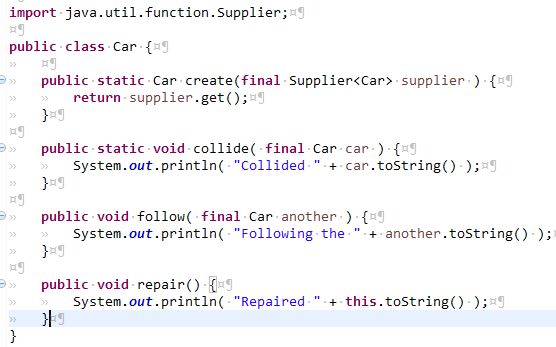
Optional 不是函数是接口，这是个用来防止NullPointerException异常的辅助类型，这是下一届中将要用到的重要概念，现在先简单的看看这个接口能干什么：

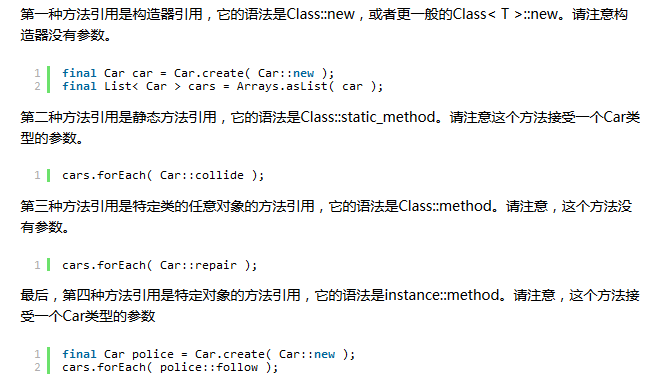
Optional 被定义为一个简单的容器，其值可能是null或者不是null。在Java 8之前一般某个函数应该返回非空对象但是偶尔却可能返回了null，而在Java 8中，不推荐你返回null而是返回Optional。



1. **方法与构造函数引用**

方法引用提供了非常有用的语法，可以直接引用已有Java类或对象（实例）的方法或构造器。与lambda联合使用，方法引用可以使语言的构造更紧凑简洁，减少冗余代码。

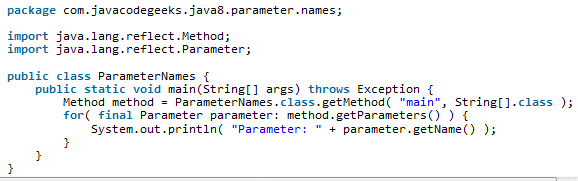


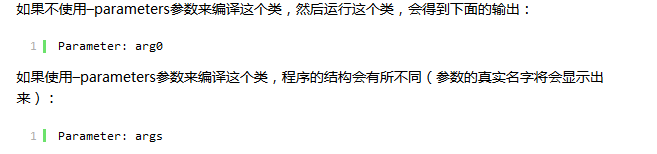


1. **Java编译器的新特性**

**参数名字:**

很长一段时间里，Java程序员一直在发明不同的方式使得[方法参数的名字能保留在Java字节码](http://www.javacodegeeks.com/2014/04/constructormethod-parameters-metadata-available-via-reflection-in-jdk-8.html)中，并且能够在运行时获取它们（比如，[Paranamer类库](https://github.com/paul-hammant/paranamer)）。最终，在Java 8中把这个强烈要求的功能添加到语言层面（通过反射API与Parameter.getName()方法）与字节码文件（通过新版的javac的–parameters选项）中。





对于有经验的**Maven**用户，通过maven-compiler-plugin的配置可以将-parameters参数添加到编译器中去。

<plugin>

    <groupId>org.apache.maven.plugins</groupId>

    <artifactId>maven-compiler-plugin</artifactId>

    <version>3.1</version>

    <configuration>

        <compilerArgument>-parameters</compilerArgument>

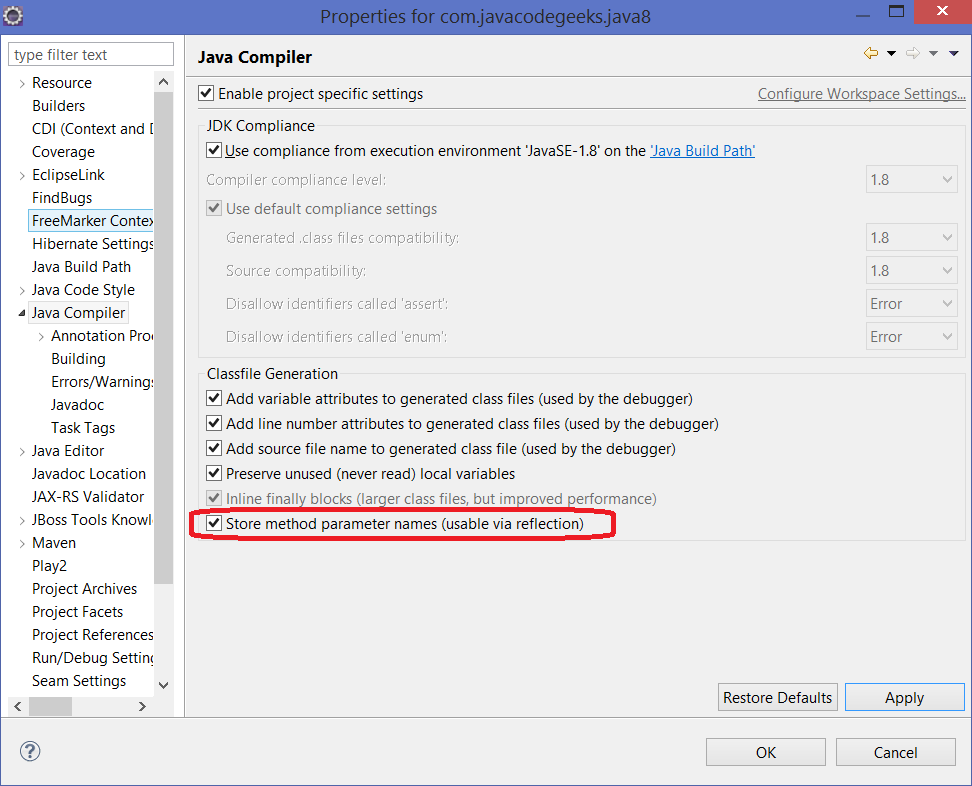
        <source>1.8</source>

        <target>1.8</target>

    </configuration>

</plugin>

针对Java 8最新发布的Eclipse Kepler SR2（请检查这里的下载说明）提供了非常实用的配置选项，可以通过下图的配置方式来控制编译器行为



此外，Parameter类有一个很方便的方法isNamePresent()来验证是否可以获取参数的名字。

1. **a**