行为参数化是一个很有用的模式，它能够轻松地适应不断变化的需求。这种模式可以把一个行为封装起来，并通过传递和使用创建的行为（例如对Apple的不同谓词）将方法的行为参数化。这种做法类似于策略设计模式。Java API中的很多方法都可以用不同的行为来参数化。这些方法往往与匿名类一起使用。

#### 1、用Comparator来排序

对集合进行排序是一个常见的编程任务。比如，想要根据苹果的重量对库存进行排序，或者他可能改了主意，希望你根据颜色对苹果进行排序。你需要一种方法来表示和使用不同的排序行为，来轻松地适应变化的需求。

在Java 8中，List自带了一个sort方法（你也可以使用Collections.sort）。sort的行为可以用java.util.Comparator对象来参数化，它的接口如下：

public interface Comparator<T> {

int compare(T o1, T o2);

}

因此你可以随时创建Comparator的实现，用sort方法表现出不同的行为。比如，你可以使用匿名类，按照重量升序对库存排序：

inventory.sort(new Comparator<Apple>() {

@Override  
 public int compare(Apple a1, Apple a2) {  
 return a1.getWeight().compareTo(a2.getWeight());  
 }  
});

可以随时创建一个Comparator来满足需求，并把它传递给sort方法。而如何进行排序这一内部细节都被抽象了。用Lambda表达式的话看起来就是这样的：

inventory.sort((Apple a1, Apple a2) -> a1.getWeight().compareTo(a2.getWeight()));

#### 2、用Runnable执行代码块

线程就像是轻量级的进程：它们自己执行一个代码块。但是怎么才能告诉线程要执行哪块代码呢？多个线程可能会运行不同的代码。需要一种方式来代表稍候执行的一段代码。在Java里，可以是Runnable接口表示一个要执行的代码块。

public interface Runnable {

public abstract void run();  
}

可以使用这个接口创建不同的线程：

Thread t = new Thread(new Runnable() {

@Override  
 public void run() {  
 System.out.println("Hello world");  
 }  
});

用Lambda表达式：

Thread t2 = new Thread(() -> System.out.println("Hello world!"));

#### 3、GUL事件处理

GUI编程的一个典型模式就是执行一个操作来响应特定事件，如鼠标单击或在文字上悬停。

例如，如果用户单击“发送”按钮，你可能显示一个弹出式窗口，或把行为记录在一个文件中。你还是需要一种方法来应对变化；你应该能够做出任意形式的响应。在JavaFX中，你可以使用EventHandler，把它传给setOnAction来表示对事件的响应：

Button button = new Button("Send");

button.setOnAction(new EventHandler<ActionEvent>() {  
 public void handle(ActionEvent event) {  
 label.setText("Sent!!");  
 }  
});

这里，setOnAction方法的行为就用EventHandler参数化了。用Lambda表达式的话，看起来就是这样：

button.setOnAction((ActionEvent event) -> label.setText("Sent!!"));