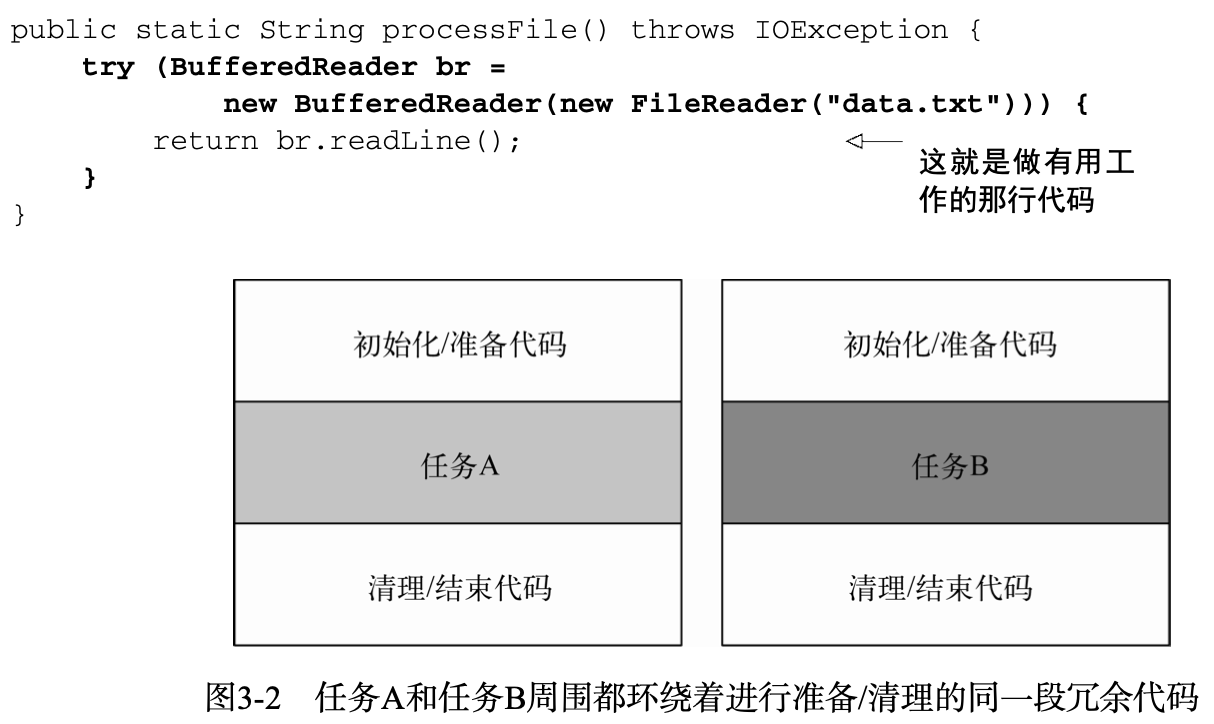
资源处理(例如处理文件或数据库)时一个常见的模式就是打开一个资源，做一些处理，然后关闭资源。这个设置和清理阶段总是很类似，并且会围绕着执行处理的那些重要代码。这就是所谓的环绕执行(execute around)模式，例如，在以下代码中，高亮显示的就是从一个文件中读取一行所需的模板代码(注意你使用了Java 7中的带资源的try语句，它已经简化了代码，因为你不需要显式地关闭资源了)：



#### 1、第1步：记得行为参数化

现在这段代码是有局限的。你只能读文件的第一行。如果你想要返回头两行，甚至是返回使用最频繁的词，该怎么办呢？在理想的情况下，你要重用执行设置和清理的代码，并告诉processFile方法对文件执行不同的操作。你需要把processFile的行为参数化。你需要一种方法把行为传递给processFile，以便它可以利用BufferedReader执行不同的行为。

传递行为正是Lambda的拿手好戏。那要是想一次读两行，这个新的processFile方法看起来又该是什么样的呢？基本上，你需要一个接收BufferedReader并返回String的Lambda。例如，下面就是从BufferedReader中打印两行的写法：

String result = processFile((BufferedReader br) -> br.readLine() + br.readLine());

#### 2、第2步：使用函数式接口来传递行为

Lambda仅可用于上下文是函数式接口的情况。你需要创建一个能匹配BufferedReader -> String，还可以抛出IOException异常的接口BufferedReaderProcessor：

@FunctionalInterface

public interface BufferedReaderProcessor {  
  
 String process(BufferedReader b) throws IOException;  
}

现在可以把这个接口作为新的processFile方法的参数了：

public static String processFile(BufferedReaderProcessor p) throws IOException {

……  
}

#### 3、第3步：执行一个行为

任何BufferedReader -> String形式的Lambda都可以作为参数来传递，因为它们符合函数式接口中定义的process方法的签名。现在需要一种方法在processFile主体内执行Lambda所代表的代码。Lambda表达式允许你直接内联，为函数式接口的抽象方法提供实现，并且将整个表达式作为函数式接口的一个实例。因此，你可以在processFile主体内，对得到的BufferedReaderProcessor对象调用process方法执行处理：

public static String processFile(BufferedReaderProcessor p)

throws IOException {  
 try (BufferedReader br = new BufferedReader(new FileReader("data.txt"))) {  
 return p.process(br); //处理BufferedReader对象  
 }  
}

#### 4、第4步：传递Lambda

通过不同的Lambda重用processFile方法，并以不同的方式处理文件了。

//处理一行

String oneLine = *processFile*((BufferedReader br) -> br.readLine());  
  
//处理两行  
String twoLine = *processFile*((BufferedReader br) -> br.readLine() + br.readLine());