生存指南

拿到题目如何动手

在答疑过程中发现很多同学的做题模式是这样的：

→ 拿到文档，看一眼背景，看一眼要什么
→ "我已经花了10分钟完全了解了这个项目"
→ 打开项目准备编码，卧槽这是什么？卧槽那是什么？
→ "让我看看测试用例长什么样"
→ 做了一定尝试之后发现和自己想的不太一样，"助教写的什么垃圾玩意(ノ‘□’)ノ ┻┻"
→ 遇到了问题，打开QQ群，开始匿名聊天

当然啦这只是一部分同学的行为，但我相信确实也会有很多人会觉得拿到题目很难开始动手。先说读文档，这里的文档指的是题目/项目的指导文档。**读文档一两遍是绝对不够的**，从亲身经历讲，一个200行左右的项目我大概整个过程中会查阅20-50遍。

Step1 读文档

在最开始读的时候先尝试回答这样几个问题：这个题目要做一个什么东西，它用了什么方法去做。在这个过程中遇到一些名词看不懂可以先略过。
有了大概的了解之后，详细阅读。以第一个作业为例，很多人遇到了弄不清"用户变量"是什么的问题(当然这部分也有助教的锅，这个问题后续再讲)。
文档中如果出现了很难理解的名词，有这么几个名词：1)助教觉得你应该会，所以没有进一步解释 2)文档结构不够好导致了名词的定义不显眼，容易被忽略 3)弱智助教又手抖了

相比于直接面向测试用例编程，一个不错的做法是搜索关键字。一个名词如果在文档中出现了多次，那么前几次中一般会出现定义。如果只出现一次，那么很有可能是某个约定俗成的概念，不妨打开搜索引擎。

Step2 做标注

在详细阅读中，我推荐所有人都应该做标注。
标注，可以是高亮，也可以是一些注释，甚至是给自己提一些问题，然后尝试在之后的阅读中去回答这些问题。

可以使用不同的颜色来标注不同的信息。例如：红色标注题目要求，黄色标注不明确的需求，绿色标注推荐的实现方法...
每一次读文档的时候都可以添加一些信息，当完成了一些东西就可以删除一些信息。
通过对标注的管理就可以知道自己目前还需要做什么。

目前的文档主要分为两种，一种是PDF文件，另一种是网页。
PDF上的标注推荐直接用Adobe或者Foxit。
网页上的标注推荐chrome插件diigo，官方有500条免费的标注量，做题目用用是完全够的(有别的插件也欢迎安利，会更新在文档里)
有了这个插件之后就可以在官方文档上自由做笔记了，不用每次打开都全文阅读一遍。

Step3 设计

接着，我们终于可以开始做题目了。
首先!
想一想怎么做，好好做一些设计。

提供一个个人的解题思路，仅供参考。
第一步，将问题做一个分解，也就是结构化编程的思想。
于是你会有了一些模糊的模块的概念。
第二步，设计一些对象，每个对象负责做点什么，负责参与到哪些模块中。
第三步，思考一下如何进行代码复用，如何让自己减少出错的可能。

当然，也许目前读到文档的你还没有一个很好的解决问题的能力，不用担心，在之后的学习中你们会不断的巩固这些流程并找到最适合自己的做法的。

Step4 编码

对于初学者来说，最常见的几个恐惧：
我是谁我在哪我该用哪个API?
为什么又又又又报错了?
这个框架代码能不能动?
IDE看起来好高级啊?

这些问题让你寸步难行，对吧。
关于debug，会在之后专门一部分讲解，这里先回答其他几个问题。

1. API的使用：一般助教会将可能使用到的第三方API放进文档中进行说明，对于这一类API不知道就是真的不知道。第三方库也就是通常maven里面的依赖。而标准库里有什么，建议从一些基础教学材料开始看起。个人是不太推荐一个系列视频刷到底或者整整一本书全部通读之后再开始编程这种做法的。可以找一本参考书，然后需要使用到某些功能的时候就对着目录找一下。当你对各种库都比较熟悉的时候也可以根据自己的需求输入关键字去一些论坛上找找有没有对应的提示。
2. 框架代码能不能动：如果是文档中的规定的接口，不能动，因为这样会让测试用例暴毙。此外的内容，作为助教，负责的说，在这门课里我**非常鼓励你去做自己的设计**。尽管我会提供一些推荐的实现，但也许这不符合你的思路。我们的代码设计也是有非常大的改进空间的，如果你觉得你的思路更好，欢迎来讨论~被采用了之后我们会更新框架和文档。
   也许你会怕自己写的会让整个项目挂掉。不需要担心这个问题，IDE里面都会提供本地历史这种功能让你回滚，实在害怕你也可以在clone下来之后备用一份源码。意思就是说，请尽可能地折腾你的代码，这是让你进步最快的方式。
3. IDE看起来好高级：遇到IDEA的问题你总会第一时间打开百度，看了一圈发现——看了很多CSDN、博客园但依然配不好一个简单的项目。很正常，因为打开方式不对。其实IDEA里面很多的问题都是来源于maven、gradle，直接去官方学习虽然会耽误一些时间但它是非常值得的。

PS：一些国内的论坛比如CSDN，在那上面的回答往往良莠不齐，很少有帖子非常系统的讲解原理是什么，遇到问题该怎么解决。大部分则是"今天我项目里遇到了一个坑，我尝试了ABCD，最后D方法work了，好，那我去写个帖子"。然后论坛上又出现了一篇看起来很正确的教你如何配置xxx的文章。但这未必真的就能够解决你的问题，也许你能解决一些，但你总会遇到解决不了的问题的。下面就会详细讲这个问题。

如何查找资料

非常非常非常建议：使用英文关键词+某科学上网工具
一个经常使用的论坛：StackOverFlow

另一个问题：“我的英语不怎么样怎么办？”
我的回答：My English is not good two, but I still use it to learn JVAV!

看官方文档其实需要静下心看，偶尔遇到不认识的词去查一下翻译就能基本畅通无阻了，毕竟这些文档的作用是把一件事说清楚，教会他的用户如何去使用就可以了，是不太会有复杂的语法和高级词汇的。(最不济也可以全文google翻译看个大概再一句句对着找需要的信息

如何debug

在项目里出bug是再平常不过的事情。
每一个经历过项目折磨的人都会面对过一种恐惧——大面积报红。
(最绝望的时候甚至想买点绿色装饰品来冲冲喜√)
报错——不知道怎么debug——感到挫败——赶不上ddl了找大佬求代码——自己水平得不到提高
这显然是一个恶性循环。

debug需要方法，以及耐心

先说方法，这里列举一些常用的(星数代表个人推荐程度)：

1. 良好的设计 ☆☆☆
   首先，一份好的代码不应该是通过debug得到的，而应该是建立在合理的设计和一些必要的验证之上的。在编码之前就应该知道某些写法一定不会出错。好的设计包括使用了复制粘贴的地方尽可能去转换为代码复用、将每个模块的功能分隔开等等原则，你们将在后续的课程里会有更深入的了解
2. 防御式编程☆☆☆
   这是一个比较复杂的概念，简单来说就是把所有的输入都当成“邪恶”的，它们会用最极端的值、调用方式来让你的程序崩溃，为了防御这些攻击，你需要做的是“宣告某些事绝对不能发生”，例如入参为null
3. 注释代码定位☆
   注释掉一些代码，反复执行测试用例试图缩小注释范围然后找到出错的具体位置。这个方法需要注意的是在系统行为比较复杂的时候也许并不好用，因为很多报错是因为一些方法的副作用造成的。
4. 使用工具 ☆☆
   由于Eclipse用的比较少，这里使用IDEA中的debug相关机制作为说明。参见官方文档，**推荐优先查看breakpoint章节，读完用可以一个小demo练练手**

再就是耐心的问题：(一杯茶，一支烟，一个bug改一天.jpg
如果一个bug困扰你几个小时甚至几天，真的都是小场面。
请相信一件事：**机器永远是对的**，一定是自己有什么不知道的地方写错了。
每当你遇到bug的时候不妨耐心的读一读自己代码思考是不是真的考虑到了所有的情况，然后利用搜索工具去查阅你需要的知识。在这个过程中，也许你会因为一个知识点牵扯出很多知识点，你会越搜索越迷茫。没错，对于一个新的领域，就是这样的。

但是，很多概念其实并不需要特别熟练掌握，很多时候debug只需要大概能定位到问题出现在哪里或者知道抛出的异常是什么含义就能解决了。哪怕你真的对一个bug找错了方向，走了很多的弯路。这些经验都是很宝贵的，不知不觉你的debug能力就会提高很多。

请坚持，会有回报的。

其他

1. 关于为什么助教有时候看起来这么不近人情?

首先，助教会有一个判断的原则—什么可以帮，什么应该你们自己解决。

你们真的想要一个非常贴心随叫随到有问必答的助教吗?
我知道看到这里你内心os"没—错—我—需—要"。

作为助教，我们的职责是尽可能提供详细的文档，回答一些有讨论价值的问题，辅助大家的学习。
但绝对不是详细解读文档中的每个要求该怎么实现，也不是没有感情的debug机器。
一些事情你应该自己思考或者交给搜索引擎。
这个作业最大的意义，在我们看来是帮助你提高自学能力，其次才是JVM的知识点。

如果真的只是想舒服的趟过，那有一万种办法可以做到，不是吗？

2. 如何帮助改进教学?

我们不需要的反馈 ❌
"文档里出现了xxx这个关键词，我在搜索时没能找到相关出处。"
"因为定义和使用隔得太远，建议助教以后注意在使用的时候强化一下概念，不然很令人困扰"

我们需要的反馈 ✅
"文档里xxx概念不清楚，xxx到底是什么"

简而言之，我们需要知道你们因为什么做出了怎么样的思考。即过程。
只提供一个结果却不试图共同解决问题的你们真的很像难搞的甲方(x

最后

作为这个作业的助教，我们真诚希望能帮助各位少走一些弯路。
不足的地方我们都在努力改进了，大家互相体谅吧。
祝实验愉快(降低难度是不可能的hhhh)