JAVA 的面向对象特性

陈德创　　19030500217

西安电子科技大学

日期：2020 年 4 月 26 日

目录

1 小组名单 **2**

2 题目 **2**

3 练习 **2**

3.1 P7.StaticInitialization 2

3.2 P21.InheritStaticInit 2

3.3 P31.BlankFinal 3

3.4 P53.Interface 3

3.5 P64.FindDups 4

3.6 P70.UseArrayList 4

3.7 P74.QueueDemo 4

3.8 P79.Freq 4

3.9 P81.UseHashMap 4

3.10 P83.TestIterator 4

3.11 P94.CoinTest 4

3.12 P99.AutoBoxingTest 5

4 题目分析 **5**

5 程序实现 **6**

5.1 具体思路 6

5.2 人机交互 7

5.3 问题解决 7

6 执行结果 **7**

7 个人总结 **8**

1 小组名单

学号	姓名	工作
19030500217	陈德创	完成程序、联系例程、调试、小组讨论

2 题目

1. PPT 练习

   P7,21,31,53,64,70,74,79,81,83,94,99

2. 程序设计，程序实现图书管理功能，要求：

   1). 程序提供图书馆馆藏书籍管理功能：

      a. 书籍信息包括书名、作者 (可能多人，使用 ArrayList 实现)、出版社以及馆藏数量

      b. 书籍可能存在书名相同，作者、出版社不同的情况

      c. 提供针对书名、作者、出版社的查询功能，支持模糊查询，可返回多条结果

      d. 可以进行书籍的添加、删除 (删除时不能存在借阅情况)

   2). 程序提供图书馆借阅管理功能 (选作)

      a. 每位同学可同时借阅三本书籍，程序可查询某位同学的借阅情况

      b. 当同学尝试借阅更多书籍或者书籍库存不足时，借阅失败

      c. 当同学归还书籍时，修正同学借阅数据及馆藏书籍数据

      d. 可针对学号查询某位同学的借阅情况

   3). 初始数据实现

      a. 初始数据可固化在代码中

      b. 初始数据可通过文件读入 (选作)

3 练习

3.1 P7.StaticInitialization

静态变量，当系统加载其所在类的时候分配空间并初始化。静态变量至多初始化一次。当创建该类对象，该类静态变量，静态方法的时候会初始化。老师给的例子故意没有把 *static* 变量放在类开头，但是这不重要，就算放在构造方法之后第一次调用该类时还是会先初始化静态变量。

3.2 P21.InheritStaticInit

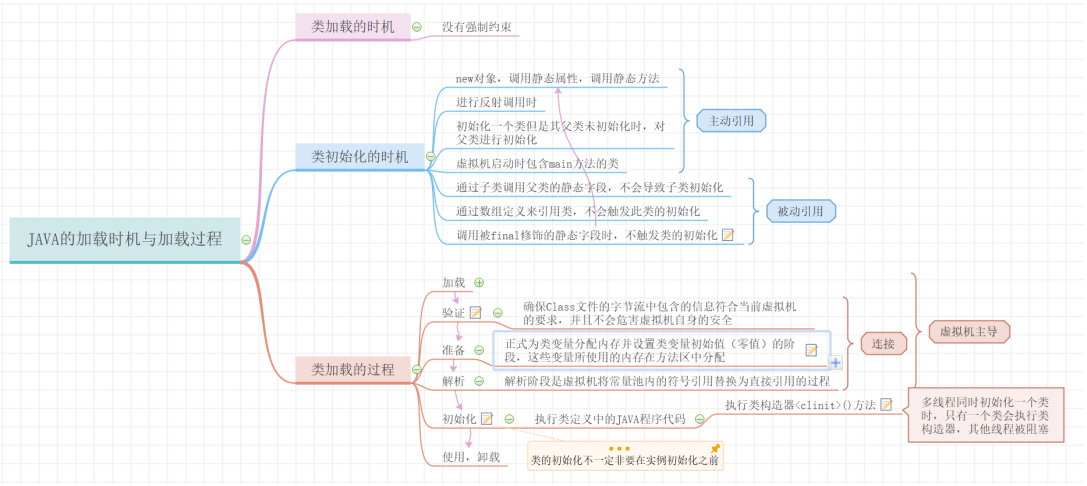
初始化一个对象会先初始化它的类，初始化一个类会先初始化它的父类。

我一开始认为继承状态下类的初始化是一个压栈弹栈的过程，我们先初始化 $T2$ 类，压入栈内为其分配内存空间并对静态变量默认初始化后发现其有父类 $T1$ 类，于是将 $T1$ 入栈，注意此时 $T2$ 只进行了默认初始化。这时我们 $T1$ 中调用了 $T2$ 的静态变量，$T2$ 已经初始化过了 (在栈中)，所以不会对其进行初始化。而 $T2$ 的静态变量只被初始化为了默认值，于是输出了 $s2$ 为 0。

在阅读了这篇博客之后，我明白了原来类的加载是有一套流程的。类在准备阶段将分配内存并附零值，然后进入类的初始化，从父类到子类依次初始化。

图 1: 博客中类加载的流程图

图 2: 根据博客内容整理的思维导图



3.3 P31.BlankFinal

final 成员变量如果声明的时候未被初始化，则必须在构造方法中初始化。这个我在写整数类的时候深有体会，因为我的整数类的成员属性的是 *final*，当时为了这个在构造方法上没少下功夫。

3.4 P53.Interface

(1) 错误定义了方法体，接口不能定义方法体
(2) 正确不过 *abstract* 的声明着实没有必要
(3) 错误还是定义了方法体
(4) 正确

3.5 P64.FindDups

Set 中不能拥有相同元素 (正如集合)，程序会把命令行参数中重复的字符串筛选出来并输出，最后输出 *Set*，即不重复的字符串。注意到输出顺序和输入顺序可能是不同的，因为 *HashSet* 是利用哈希值存储，不保证顺序 (效率更高)。

3.6 P70.UseArrayList

添加元素默认添加到末尾，*ArrayList* 仍是以 0 为索引开始。插入则由指定位置元素依次后移，然后插入到指定位置。程序输出为 86 99 98,77 98,[77, 98]。

3.7 P74.QueueDemo

结果为 8 1 1 1 5 14 3 1 0 1,*B r o n t o s a u r u s*，虽然用了随机数，但是多次试验都是这一个结果，充分说明了 *Random* 是伪随机，种子相同的情况下生成的序列相同。*Queue* 是一个 *FIFO*(先进先出) 的容器，没啥好说的。老师这里利用了 *queue.peek*() ! = *null* 来判断队列是否为空，可以用 *queue.size*() > 0，或者直接 !*queue.isEmpty*() 就可以了。

3.8 P79.Freq

词频统计，*Map* 定义了一种映射关系。遍里给定的单词表，获取词频，然后进行更改。这里可以看出 *put*，如果键不存在，则会新建键值对，如果存在，则会更改键所对应的值。输出结果为:8 distinct words detected:be=1, delegate=1, if=1, is=2, it=2, me=1, to=3, up=1。

3.9 P81.UseHashMap

常规的 *Map* 用法，增加、修改和删除。结果为: 按字符串输出：李二 =98, 海飞 =99, 张一 =86, 修改并删除之后, 按字符串输出：李二 =77, 海飞 =99

3.10 P83.TestIterator

迭代器的用法。用迭代器迭代遍历一般主要用到两个方法 *hashNext*() 和 *next*() 用于判断边界和获得元素并迭代。注意 *remove* 移除的是 *next* 产生的最后一个元素，所以如果在使用 *next* 之前就使用了 *remove*，程序会报错，如图:

```
Exception in thread "main" java.lang.IllegalStateException
    at java.base/java.util.ArrayList$Itr.remove(ArrayList.java:976)
    at chapter5.TestIterator.main(TestIterator.java:19)
```

3.11 P94.CoinTest

这里给每个枚举设置了属性 *value*，并且声明了构造方法。虽然这里构造方法是缺省权限的，但是编译时会自动声明为 *private* 权限，也就是说你在此类之外任何地方都无法创建新的枚举实例。这有点像单例设计模式，其实枚举类型就像是系统封装好的单例模式。结果如下：

```
 .m2\repository\org\jetbrains\annotations\17.0.0\annotations-17.0.0.jar chapter5.CoinTest
PENNY: 1, COPPER
NICKEL: 5, NICKEL
DIME: 10, SILVER
QUARTER: 25, SILVER

Process finished with exit code 0
```

3.12 P99.AutoBoxingTest

封装类，为了基本类型可以方便地使用类的特性，JAVA 内置了封装类。封装类在使用中和基本类型基本无异，可以直接使用 $+, -, =, /, *$ 等运算符号，但本质上并不是运算符重载，还是方法的调用。封装类可以直接赋值，而且形如 $Integer(int)$ 的构造方法已被弃用。

- 自动装箱 (autoboxing): 在应该使用对象的地方使用基本类型的数据时，编译器自动将该数据包装为对应的 Wrapper 类对象
- 自动拆箱 (autounboxing): 在应该使用基本类型数据的地方使用 Wrapper 类的对象时，编译器自动从 Wrapper 类对象中取出所包含的基本类型数据

结果为：x is greater than y,x + y = 37。

4 题目分析

题目要求提供图书馆书籍管理功能 即表示一本书、增加/删除一本书以及查询一本书。显然地，我们需要提供 *Book* 类用于表示一本书，并且应当包含书籍名称、作者、出版社等信息。并且重写 *equals* 和 *hashCode* 方法。

除此之外，我认为需要考虑一个 *Book* 实例，到底是表示一本书，还是一种书。我认为应当是表示一本书，也就是说，两本《core JAVA》，纵使它们作者相同、出版社相同，它们应当是可以区分的。可以有两个不同的实例分别表示这两本书。这样做的好处是可以方便地追踪某一本具体的书的流传过程。这样我们唯一确定一本具体的书的方法是，增加 id 属性，用于唯一指定一本书。我们再增加 kind 属性，用于区分书的种类。前面提到的两本《core JAVA》，应当 kind 相等但是 id 不等。

那我们 *equals* 相等的原则是什么呢？是判断 id 还是 kind 呢？我认为应当是 kind，因为一本具体的书我们只会产生一个实例（在给图书馆添加书籍时产生）。

程序提供图书馆借阅管理功能 即要求我们设计图书馆 *Library* 类和用户 *Borrower* 类。图书馆类应当提供对书籍的操作功能，比如书籍的查询，添加，借出，归还，删除。图书馆的实现涉及到 *Map* 的使用，我们用 *Map* 构造一个借阅关系，即 $Map < Integer, Borrower >$，表示一本书被一个借阅者借走了（Integer 表示书的 id，因为 equals 是以同种书为标准的，所以这里 *key* 不宜直接用 *Book*）。同样地，我们需要表示一类书的总量和借出的数量，即再构造两个 $Map < Book, Integer >$。图书管应当只有一个实例。

对于查询书籍的实现，遍历 *keySet* 逐个比较即可。虽然效率较低，但是应该可以接受。

借阅的实现 图书馆是管理借阅关系的。借阅和归还应当有两个参数，分别为书籍和借阅者。同时，借阅者应当实现借阅方法，用于借阅一本书并返回一个布尔值表示借阅成功与否。实质上借阅者的借阅方法也应该由图书馆调用，或者由一个 *Controller* 类来统一管理方法的调用。

查询某位同学的借阅情况 借阅者类应当包含列表属性借阅的书。在查询时直接调用就好了。借阅者应当还包括权限属性，用于表示其最大可借阅书的数量以及天数。可以新建权限类，可以使用枚举型。对于逾期不换的借阅者我们可以考虑适当减少其权限（程序未实现）。

5 程序实现

5.1 具体思路

1. Book 拥有属性 $bookName, author, press, id, kind$，以及实现了 *Serializable* 接口，（主要是用于写入文件以及后面安卓开发时由一个活动向另一个活动传递）。设置了一系列 $getter, setter$ 方法。并且设置了静态变量 $kinds, cnt$，分别表示种类数和本数，并与用产生 $kind, id$ 属性。为了可以更快地找到一本书和一类书，我们设置了 *idToBook* 和 *bookMap* 两个静态 *Map* 变量，并别提供了相应的方法。
2. Borrower 拥有属性 $name, studentId, authority, borrowedBook, bookNum$。提供方法借阅书籍、归还书籍以及一系列 $getter, setter$ 方法，没啥好说的。
3. Library 我觉得这次作业的重点就在于图书馆的设计。图书馆没有什么具体属性，只有三个映射关系，分别为书到书籍总数 *collectionNumber*、书到借出数量 *borrowingNumber* 以及书的 id 到借阅者 *bookToBorrower*。我们后面对书籍的增删查借出归还实际上都是对者这三个映射关系进行操作的。
    i. 增加书籍实现了 *addBook*(*Book*) 和 *addBooks*(*Book*[]) 两个方法，分别用于为图书馆新添一本书和一堆书。*addBooks* 方法实际是遍历数组分别调用 *addBook* 实现。方法返回一个布尔值表示书籍是否添加成功，只有当图书馆已经有被添加的书了（指同一本），添加失败。添加的方式很简单，判断 *collectionNumber* 中是否有这本书的键，有的话值加一，没有的话则新建键值并赋值数量，同时初始化 *borrowingNumber* 的同一键，并赋值 0。并且按照添加书的 *id* 为 *bookToBorrower* 新建键值对，值为 *null*，表示未被借出。（实际上 *bookToBorrower* 的 *keySet* 就表示了图书馆所有的藏书集合）
    ii. 删除书籍实现了 *deleteBook*(*Book*) 和 *deleteBook*(*Book*) 两个方法，分别为删除这一本书或者删除这一类书。注意参数都是 *Book* 型的而不是数组。删除时先判断图书馆是否拥有这本书、以及这本书是否被借出了（用 *bookToBorrower.get*(*id*) == *null*）来判断。如果都没有，则直接删除对应键值对并返回 *true* 表示删除成功。否则返回 *false* 表示失败。
    iii. 借出书籍同样地，先判断图书馆是否有这本书、这本书是否被借出去了，然后调用借阅者的 *borrowedBook* 方法，改方法会检查借阅者的借书总量来判断是否可以继续借书，如果可以则借阅这本书。如果返回值为 *true*，则更新图书馆的 *borrowingNumber*，

bookToBorrower 并返回 *true*。

iv. 归还书籍同上，判断图书馆是否有这本书、这本书有没有被借出去然后调用借阅者的 *returnBook* 方法，改方法会检查借阅者是否确实借了这本书。最后还是更新 *borrowingNumber*，*bookToBorrower* 并返回 *true*。

v. 查询书籍有两种，一个是给定字符串查询满足字符串的书籍，另一个是给定一本书查询图书馆的总量和库存。前者遍历藏书然后各属性可以调用 *matches* 方法判断，后者直接根据 *borrowingNumber* 和 *collectionNumber* 返回就可以了。返回一个整形数组，表示总数和借出数量。

5.2 人机交互

人机交互我采用的安卓。因为写命令行各种判定太麻烦了，又看到群里大佬们的 GUI 很漂亮，所以就像做一个 GUI，中间去学了一段时间，最后实践的时候感觉运用很不自在，缺乏积累，于是暂时放弃了这个想法。后来考虑到寒假自己学过一段时间安卓，于是想到用安卓开发。不得不说安卓的界面和逻辑分开还是比较舒服的。

1). 登录界面登录界面用于用户验证，但是具体实现由于技术和时间问题，只用了一个账户。账号和密码都是 admin 。

2). 主界面主界面由一个搜索框和一个列表组成，搜索框用于根据图书内容进行搜索，列表用于对书籍的展示。

3). 书籍的添加书籍的添加在菜单里，会新建一个活动来完成信息的填写。

4). 书籍的借阅和删除 (未完成) 这一部分功能本应该放在在点即书籍之后进入另一个界面的。另一个界面显示了书籍的被借阅历程和更详细的介绍，并且在此界面可以对书籍进行借阅和删除（需要管理员权限）等。

5.3 问题解决

不得不说，写的时候 high 得飞起，然后运行的时候全是 bug。说多了都是泪啊！

1). equals 的重写这个是关于 *Book* 类的，一开始我只是简单地调用了各个属性的 *equals* 方法（都是字符串），万万没想到啊，作者出版社什么的可能是 *null*，就栽了。后来 *debug* 了好长时间.....

2). 查询信息无法返回即在程序中查询一个字符串更改列表后，点击返回无法返回原来的状态，这个因为安卓版本不兼容的问题，系统提供的控件没有提供返回原来状态的功能。emmmm 其实再给子控件添加监听就可以了，但是没时间了.....

3). 空指针异常在程序实践中有很多次空指针一样，提醒自己时刻注意对象是否初始化、对象是否可能为 *null*，对象为 *null* 后是否会调用实例方法。

6 执行结果

因为主要是以 APP 实现的，所以在这里就不放图片了，在作业文件夹里有一段我自己测试的视频，老师可以看一下。源代码在GitHub上。emmmm 因为我还不会打包 apk，所以没有负带

apk 文件，等这两天学会了再补过去。（更新：打包 apk 在同一文件夹，安装到手机上还是很惊喜的。）

7 个人总结

这次作业完成的不是很好，很多地方都没有具体实现，比如多用户，比如借阅追踪，比如借阅和删除功能，比如文件储存。

其实底层的除了人机交互的代码应该是在布置作业的那个星期就完成了，同样地练习也完成了。随后去学习 GUI 就先把作业放一边了，再加上数模选修结课要写两篇论文，一时没腾出时间来做。当然也不是找借口，没做好就是没做好。下次注意一下再努力吧。

这也算我第一个用安卓写出来的 APP 吧，虽然是个阉割版。（底层除了 GUI 部分是实现了的）这次程序设计有很多事物的地方，比如 *Book* 应该设计成一个 JAVA Bean，更多的功能应当新建一个类的。体会到开发一个软件确实不容易。

整体感觉自己还是比较欠缺这方面的经验的，不知道该去怎么整体地设计一个整体的架构，这或许也是这门课的难点所在吧。

进无止境。