OO第一次博客作业

1. **面向对象第一单元作业介绍**

面向对象课程第一单元的学习结束了，第一周的主要任务是使用面向对象思想进行了三次难度递增的表达式求导。下表说明了三次作业的要求情况。

|  |  |  |
| --- | --- | --- |
| 作业序号 | 简略的基本要求 | 示例 |
| 1 | 能够完成含有支持前导0的带符号整数系数、指数和幂函数的简单多项式求导 | 3 \* x ^ 1+12 \* x ^ 29 |
| 2 | 能够完成含有支持前导0的带符号整数系数、指数，幂函数和三角函数求导 | 3\*x^1\*cos(x)+12\*x^29\*sin(x) |
| 3 | 在（2）的基础上，进一步支持了因子的嵌套，引入了多项式因子。 | (cos((3\*x^1))+sin(x^12)^29)\*(x+12) |

下面我将通过自己的分析总结三次作业的不足和经验。

1. 对第一次作业的分析
2. 设计思路

第一次作业我的想法非常简单。首先，我们先通过正则表达式判断输入的正确性。对于错误的输入，直接输出“WRONG FORMAT”，执行结束；对于正确的输入，则进行下一步。在第二步，我设置了一个类StringProcessor，来处理字符串。其作用就是将字符串中的有用信息提取出来，并依次存入多项式类Polynomial中的HashMap中去，与此同时要控制多项式的同类项合并，方便优化。显然这次作业的有用信息只有次数和系数两个。然后在Polynomial类中单独实现Hashmap的合并、求导、输出当前存入的表达式等操作。可以看出整体上来看，这样的设计思路是简单并且清晰的。但是实际实现中会有不理想的情况发生，这部分会在稍后阐述。

由于第一周作业难度并不是很大，所以在作业前期我就开始考虑优化的方法。方法最后大概考虑如下：

① 正数项放在最前面；

② 0系数：该项不输出，除非是唯一的项；

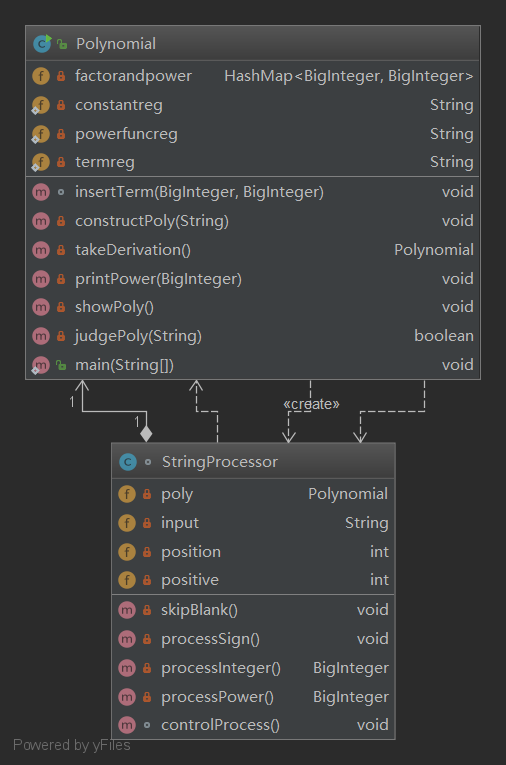
③ 1或-1系数、次数：系数不输出；

④ 0指数：将该幂函数化成1再来考虑。

可以看出以上的四种优化情况我们都可以通过调整输出来实现。所以对我们的架构没有本质上的影响。

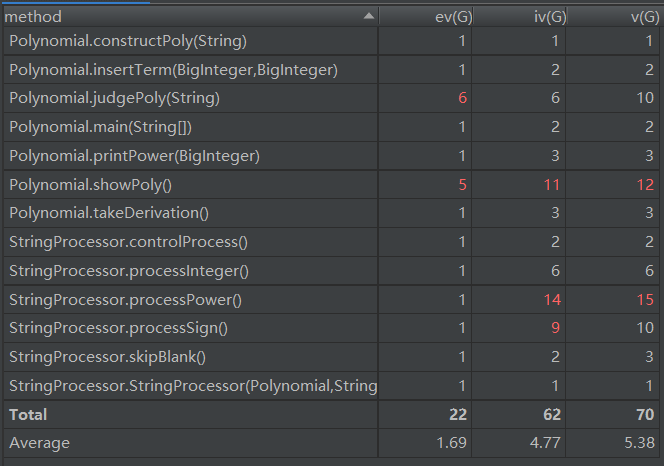
1. 程序实现

如上所言，本程序一共有两个类：Polynomial.class和StringProcessor.class。Polynomial中含有一个Hashmap数据结构，用来存储系数与指数的对应关系，并完成有关多项式求导、输出的一系列操作。StringProcessor主要是进行合法输入字符串的读取和有效信息的提取。下面是本程序的UML图。



1. 复杂度度量

复杂度度量如下所示。由于第一次作业类的耦合程度过高，每个方法的复杂度也很高，导致复杂度情况并不理想。尤其是显示多项式的方法，由于对多种情况考虑都综合在一起，比较影响程序性能。在读取字符串的指数部分没有对多种情况分开讨论而是写在了一个方法里面，也是影响程序的一个点。



1. Bugs修改、遇到的问题

Bug 1：（自测）由于一开始对正则表达式的理解不充分，使用了整个表达式全部匹配的方法。在测试的时候当项数达到200项以上，程序就出错了。主要原因是因为正则表达式匹配是一种贪婪匹配，整个表达式进行匹配导致复杂度过大，程序崩溃。

解决方法：修改为每项匹配并检查上次匹配末尾是否是本次匹配开头的方法能够有效处理这个问题。不仅如此，在交流过程中，我发现有很多同学是每项匹配结束后直接调用matcher的内部函数来处理当前项字符串的，也就是说讨论问题的范围缩小到了一项。这样的方法显然要比我现在的架构要好不少，并且更加的“面向对象”。但是因为最后没有时间修改自己的架构了，故维持原样。

Bug 2：（自测）在测试过程出现了大数运算溢出的问题。

尽管使用了long型变量，但原题并没有给出数据的范围。是我在看题目需求的时候疏忽了。后来经过查阅资料发现java中有内置的一个大数类BigInteger。这个类使用体验极好，给本次作业提供了很好的便利。

没有遇到特别困难的问题，主要是第一次作业还是对这种思想不熟悉，用起来感觉怪怪的。所以即使思路简单，但效率并不高。

1. 对第二次作业的分析
2. 设计思路

吸取上一次的教训，本次作业是使用了单项单项进行分析操作的方法。主要思路是读取字符串后，查看当前项是不是符合定义。如果不符合，则输出错误信息并跳出程序；如果符合，则使用StringProcessor将他们的指数和系数都存入到Polynomial中的Hashmap中去。值得注意的是，指数（也就是Hashmap的键值）已经不止是1个了，我们需要建立一个类来存放x、sin(x)、cos(x)三个函数的指数组成的整体，并把它作为Polynomial的Hashmap的键值。在Hashmap的Key值实现上需要对此方法的hashCode()方法和equals()方法进行改写，这是存储阶段。在求导阶段，使用链式求导法则，对于每一个项可以分离出一个只有一个指数的项以及一个混合项，（例如3\*x^3\*sin(x)^7\*cos(x)可以分离为x^3和剩余部分），使用链式法则递归求导。保存求导后的数据，进行合理优化的输出。

本次实现的优化，在上次作业的基础上增进了对于三角函数简单合并的一些操作，主要说来是对以下三个恒等式的处理：

①sin(x)^2 + cos(x)^2=1

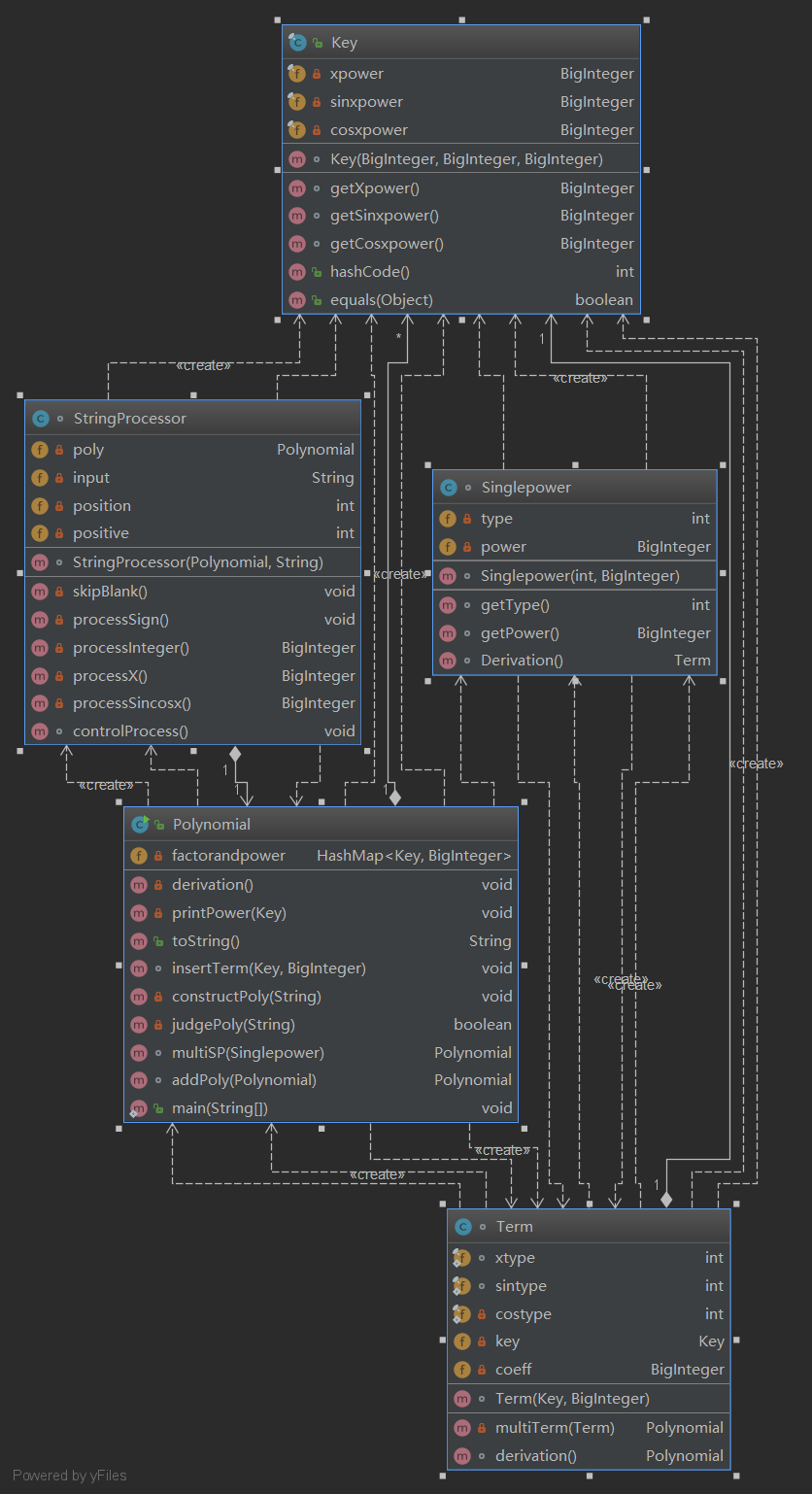
②1-sin(x)^2=cos(x)^2

③1-cos(x)^2=sin(x)^2

同样，这三个恒等式的化简，我们只需要在输出阶段进行处理即可，不影响我们的架构。

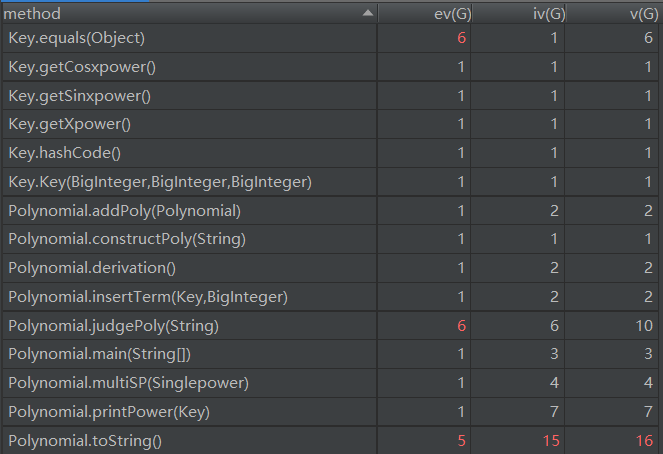
1. 程序实现

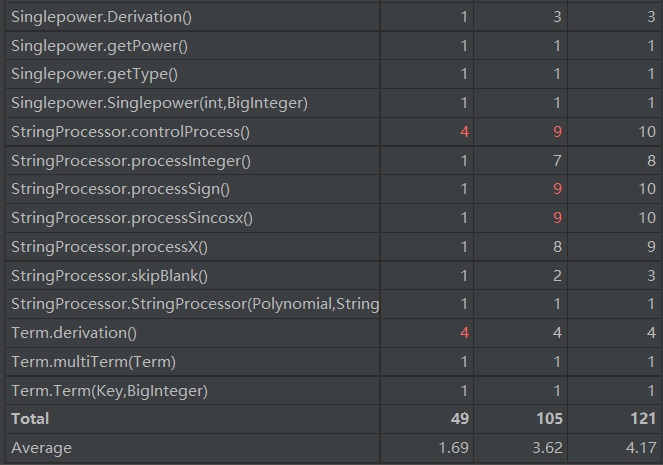
根据以上的想法，本程序主要分为以下五个类：Polynomial负责控制整个程序的入口，判断表达式中项的合法性，建立HashMap存储当前的指数-系数对，整个多项式的求导控制等；StringProcessor对一项的内容进行处理，分解为指数、系数，并反馈给Polynomial；Key类是Polynomial中HashMap的键值类型，存储了三个系数；Term是实例化一项的备用类，里面的属性是一个Key值和一个系数值；SinglePower是在求导时分离出来的只有单独一项的特殊类，是求导的基本单位。下图是本次作业的UML图。



1. 复杂度度量

复杂度度量如下图所示，由于Polynomial.toString涉及了多种输出方式的优化，所以其复杂度非常高。事实上，我们可以将三种方法分别在polynomial中实现最后合并在toString函数里面，这种降低耦合度的设计以后可以多考虑一下。





1. Bug修改、遇到的问题

Bug 1：（自测）数组越界问题

修复方法：本次作业首次使用了try-catch手段进行异常的检测、处理、抛出。

在编写Key.class时修改了Key函数的hashCode()和equals()方法。事实上，在比较两个类相等的时候，机制如下：

①先比较两个类的hashCode()，相等则转②，不等则二者不等

②调用equals()方法，返回bool值。

在使用类作为键值或者进行比较的时候，一般来说都要重写这样两个方法。第二次作业因为主要沿用了第一次作业的想法和架构，所以感觉难度反而降低了一些，对于机制有了更好理解。但是实际上现在看来SinglePower完全可以作为Term的一个子类，并且可以直接把Polynomial的hashmap改成term的一个set进行存储。这样可能会在逻辑上和实现上更为清晰、简洁一些。

1. 对第三次作业的分析
2. 设计思路

根据作业的提示，建立了加减法类、乘法类、嵌套类、三角函数类、幂函数类、常数类，并对他们分别建立了词法分析函数。

**合法性检查**：这次作业显然不能再使用正则表达式去匹配一项了。这次采用了一种类似于词法分析和语法分析器的功能，自顶向下分析。从加减法类进入，根据各种情况跳转到相应的类中进行词法检查。在匹配的过程中可以利用正则表达式，简化了匹配过程。

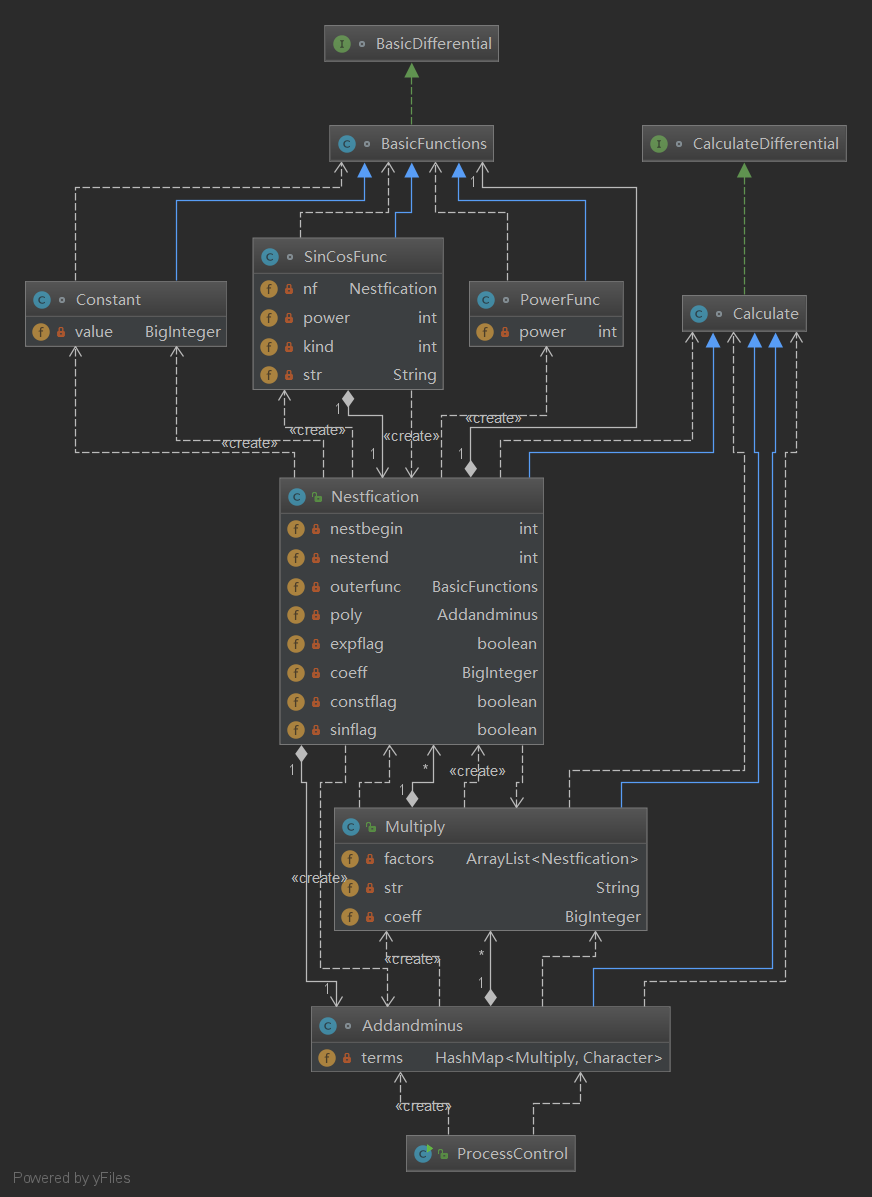
**多项式信息存储：**合法性检查过程中，经过每一个类将其直接下层类的信息变为线性结构存储起来。例如：加减类就要存储下面的乘除类的信息，方便之后进行求导操作。这样建立一棵多项式树。

**求导**：自顶向下求导过程。同样是上层类调用下层类的求导以完成自身求导。遵循链式法则。

本次的优化内容较复杂，我做了有关于去掉括号和合并常数的优化，事实上并没有做全，优化效果也比较差，根据分数显示在强测中基本上都没有达到最优方案的3倍这个限制，并且还因为优化出了一个强测的bug，影响了程序的正确性，确实是没做好的一个点。我能想到的剩下一个提高程序优化性能的但是没有做的点是合并同类项，例如对于x+x+x+x这样的函数，我的程序最后的输出为1+1+1+1，不做同类项合并的操作确实效果不好。但是总体来说做了以上两种优化也是锻炼了自己的一些能力的。

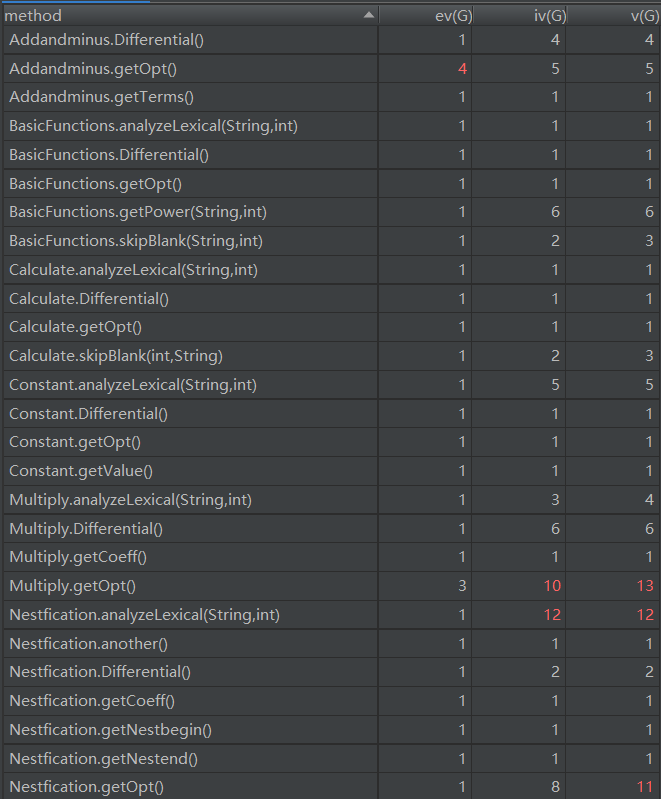
1. 程序实现

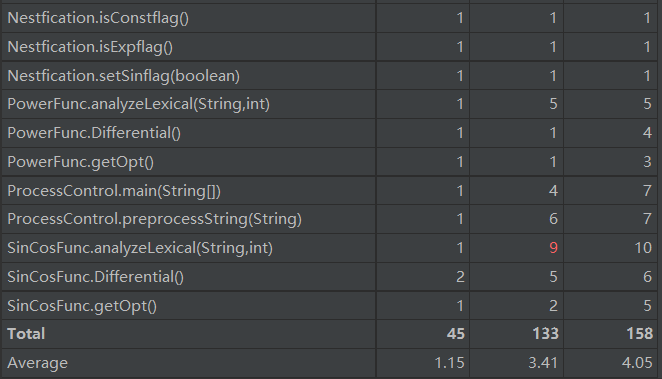
本次程序实现使用了接口和继承特性，将整个表达式从上到下建立了一个树，架构还算比较清晰。主要问题是对于返回值的考虑并不是很清晰，一开始设计的时候有问题，导致返回的是一个求导后的多项式，这使优化变得比较艰难，只能通过重新读取求导后的式子来进行优化。并且对于优化要单独实现，丧失了一定的功能分散的特性。



1. 复杂度度量

由下图可见，复杂度主要集中在优化函数、词法分析函数上面。在这两个函数上的内聚程度过高。





1. Bug修改、遇到的问题

Bug 1：（强测）数组越界

Try-catch运用的还不够好，不能理解其深层次的原理。并且是优化带出来的bug，之后应当注重边界检查。

Bug 2：（中测）三角函数里面的表达式不加括号

解决方法就是加上括号，并不引入其他括号。

1. Applying Creational Pattern

第三次作业中实际上使用了Factory Pattern。使用这种Pattern的好处是使得各个类功能分明，在逻辑上更加清晰。更重要的是，扩展性更强。事实上，因为前两次作业是第三次作业的子问题，我们当然也可以用工厂模式去套用。但是在问题发布的当周，感觉没有写到这种程度必要，这种想法同样使得前两次作业的程序仅仅能够解决前两周的问题，可扩展性几乎为0。这提醒我在设计的时候要有一定的远见，保证程序的扩展性。同时坚持高内聚低耦合的设计原则。

1. 总结

通过这三次的学习，首先是对面向对象程序设计思想有了一个大致的认识。其次对于java语言的各种概念和机制有了大致的了解并进行了应用（如类、方法、继承、接口、try-catch机制等）。通过自己课下练习、与同学讨论和在课上通过老师对作业的总结讲解，对于面向对象思想有了一个初步的认识，并且能够明确感受到这样一种思想与之前接触的面向过程的一些程序语言的区别。与此同时，通过规范化的训练，我也了解到了代码风格对于程序设计的重要作用，并重新审视了之前自己写过的不忍直视的代码。期待后续的学习能够更加更深层次的挖掘这门思想的奥秘。