实验3:从计算器到微积分

6月15日星期六晚上11:59截止。

在这次的作业中，将编写各种实现计算器行为的类。这个作业包含三个部分：在第1部分中，你将编写实现常见计算器操作(如加法和减法)的对象；在第2部分中，你将使用第1部分中编写的操作来完成一系列表示算术表达式的类；第3部分结合了函数式编程和面向对象编程的思想。在第3部分中，你将编写一个类来表示一个变量，该变量的值可以更改，此时表达式可以表示代数函数。然后你需要编写一个类来表示任意函数的导数，并编写一个简短的程序，使用牛顿法找到任意函数的零点。

本次作业的目标是:

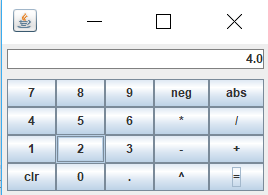
* 理解并应用多态性、封装和约束等概念，还包括合理的使用Java接口。
* 能够基于非正式的规范来解释、设计和实现软件，并展示对基本软件设计原则的理解。
* 编写单元测试，并使用JUnit测试。
* 理解代码覆盖率度量的好处和限制，并解释覆盖率度量的结果。
* 良好的Java编码规范，测试实践及风格。

本文档包含了本次作业中三个部分的具体任务分配和测试需求。

**第1部分:使用多态性实现计算器操作**

在src.lab3.operator中，本课程提供了表示算术运算符的接口（src.lab3.operator.Operator）和表示二元运算符(如加减乘除和取幂)以及一元运算符(如取负和绝对值)的子接口。 要完成此部分，你必须：1）至少为这七个运算符操作提供具体的实现（即实现src.lab3.operator.BinaryOperatorImp和src.lab3.operator.UnaryOperatorImp）；2）在src.lab3.guicalc.Main中完成该程序， 即将运算符当作参数传递给src.lab3.guicalc.GuiCalculator的构造函数并启动计算器。

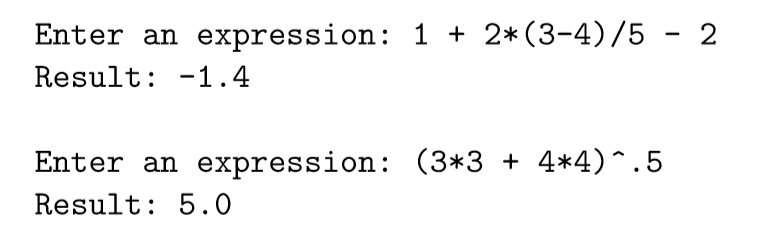
运行src.lab3.guicalc.Main主程序，确保你的运算符能够在GUI界面中正常工作。本课程建议(但不要求)你编写更多的运算符(例如，sin、cos、tan、log等)来检查该程序的扩展能力。



**第2部分:实现计算器表达式**

在src.lab3.expression中，本课程提供了一个表达式接口（src.lab3.expression.Expression）来表示算术表达式。在这一部分中，你的任务是实现表示数字、一元运算符表达式和二元运算符表达式的类（数字，2个一元运算符，5个二元运算符；共8个类）。你也可以通过手工构造更多熟悉的算术表达式来测试你的实现（例如，），然后检查它们的值是否正确。

助教提供了一个基于终端的计算器来帮助你测试你的解决方案。它有一个解析器，该解析器接受字符串形式的算术表达式，并使用你对表达式接口（src.lab3.expression.Expression）的实现将它们转换成表达式树。要让解析器访问你的实现，你需要：1）实现表达式生成器（src.lab3.termcalc.ExpressionMaker）接口，即需要实现src.lab3.termcalc.ExpressionMakerImp，该实现要求返回八种表达式类型（src.lab3.expression.Expression）的实例；2）完成这些之后，在主函数src.lab3.termcalc.Main中，将你实现的表达式生成器（src.lab3.termcalc.ExpressionMaker）的实例传递给src.lab3.termcalc.TerminalCalculator的构造函数；3）在Main中，使程序能够从键盘输入中读取算术表达式，然后解析它们，计算它们，并打印结果。换句话说，它将把你的表达式实现变成一个基于文本的可以理解运算符优先级的计算器。示例如下：



**第三部分:函数式编程:导数与牛顿法**

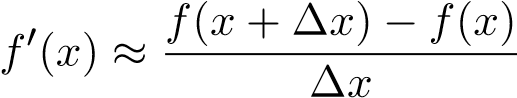
作业的这一部分有三个子部分。你*将首先将第2部分的工作扩展到支持包含命名变量的表达式(例如：)；然后在此基础之上编写一个导数表达式，对一个函数的导数进行数值计算；*最后，你将在牛顿法的一个实现中使用你的导数表达式，牛顿法是一个求函数零点的数值算法。

**3.1带命名变量的表达式**

在src.lab3.expression.Variable中，本课程提供了一个表示变量表达式的表达式接口（src.lab3.expression.Expression）的实现，本质上是一个表示值的命名框，就像代数中的变量一样。你需要实现这个框架，并通过创建一个名为x的变量表达式来测试它（注意：你不应该需要任何新的构造函数来实现这一点，在第2部分的基础之上，加上你的src.lab3.expression.Variable应该就足够了）。然后将x设置为某个值，并验证整个表达式的计算结果是否为正确的数值。如果愿意，可以编写一个小程序，通过反复设置x并对函数求值，生成函数的值表（如：）。你还可以(但不要求)使用多变量函数(如：)。

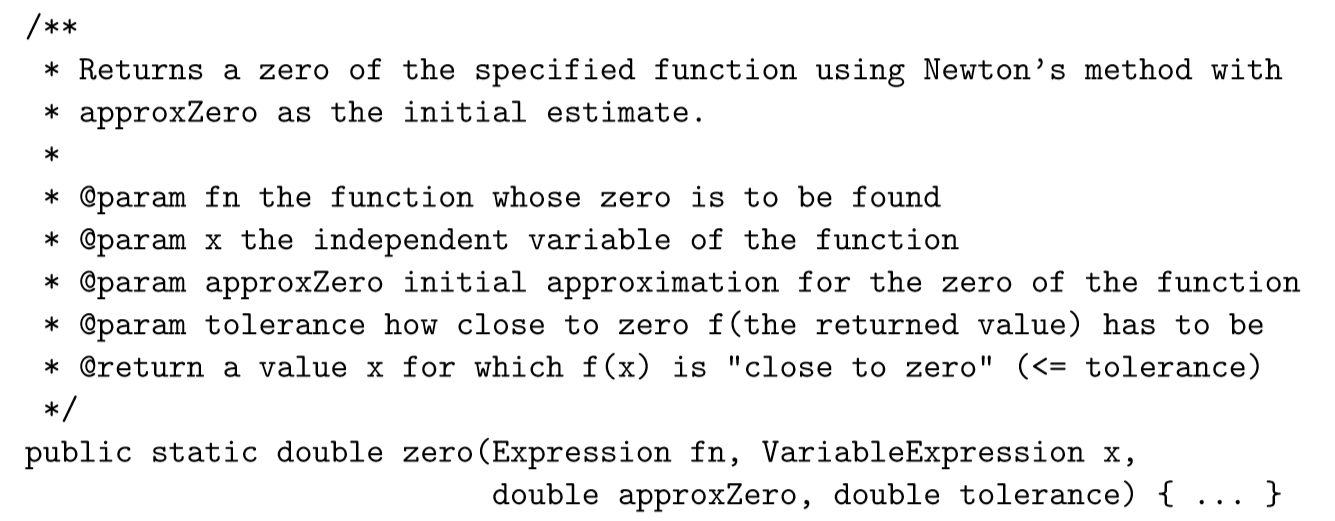
**3.2求导数的表达式**

在微积分中，函数的导数是另一个函数，其中每一点x的值就是在x处的斜率。因此，一个函数的导数在x处可以近似为:



实现表达式接口（src.lab3.expression.Expression），使得其实例表示某个指定函数的导数，即实现src.lab3.expression.DerivativeExpression。

实现的构造函数应该类似于:



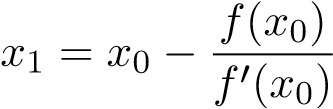
注：此处的VariableExpression即src.lab3.expression.Variable。

DerivativeExpression的eval方法应该返回fn在indepedentVar设置为某个值时的近似导数。 为了得到这个近似导数，设∆x为一个私有常数值：DELTA\_X=1e-9(即，)。通过评估在不同点的导数是否近似等于来验证你的程序；类似地，sin(x)的导数应该近似等于cos(x)。

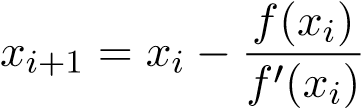
**3.3牛顿法**

最后，在牛顿法中使用你实现的导数表达式（src.lab3.expression.DerivativeExpression）计算函数的零点。*函数f(x)的零点是f(x)取值为零时的x值。比如的零点是1和2。*

牛顿法是一种数值算法，它迭代地改进零点的粗略逼近，直到逼近足够精确为止。详情请参阅维基百科上牛顿方法的文章。*给定零点的初始估计值，牛顿法计算出一个改进的估计值*为: (本段包含链接：[Wikipedia article](https://en.wikipedia.org/wiki/Newton)，[Newton’s method](https://en.wikipedia.org/wiki/Newton))

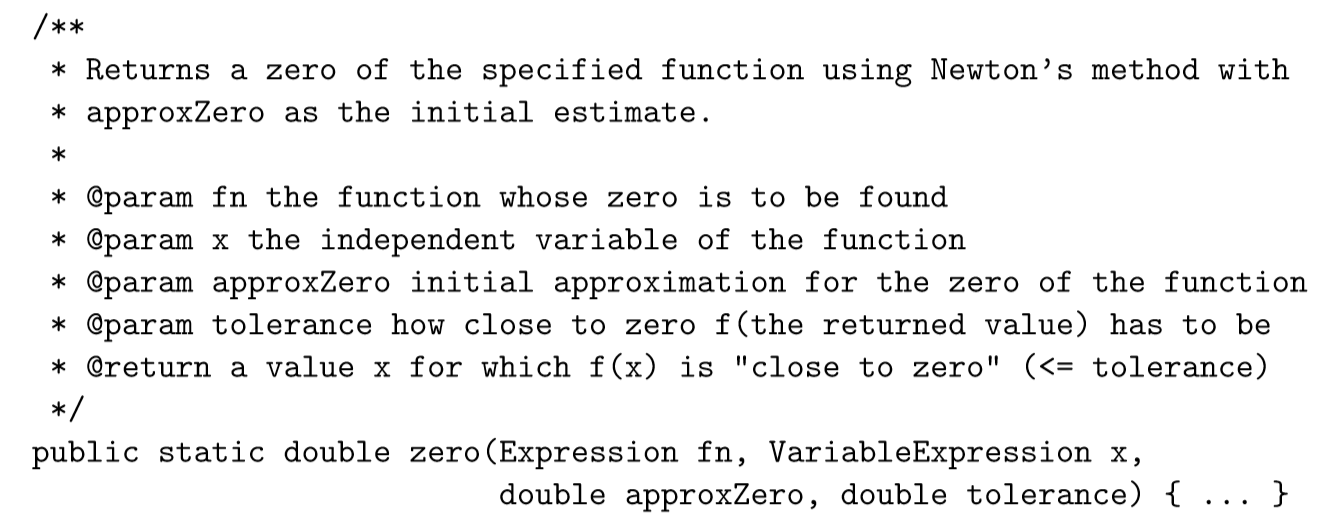


同样的,可以得到一个连续的估计值序列:



这个过程可以重复，直到估计值足够接近实际的零点。*牛顿法对某些函数f(x)不成立，但已知对许多函数收敛。*

为了完成这部分，你需要在src.lab3.expression.NewtonsMethod中写一个方法来计算任意函数的零点（给定初始粗略近似值和目标精度）:



如果你已经成功地完成了第1部分和第2部分以及你的派生表达式，那么这个方法的主体应该相对简短和简单。*在多个函数上测试你的求零点方法是否正确，比如函数为，且初始估计值为1。*

**测试实现**

本课程必须使用JUnit测试对解决方案进行正式测试。如果可能，应该检查你的实现在一般情况和边缘情况下的正确性。理想情况下，你的代码应该实现近100%的覆盖率，不包括用户界面代码。如果没有达到100%的覆盖率，请在每个未覆盖的区域用简短的注释解释为什么不能达到100%的覆盖率。

本课程建议你在完成解决方案时开始为解决方案编写测试代码; 不要等到你的程序都实现了才来编写单元测试。在实现的早期之前进行测试(并发现其中的bug)要容易得多。一种常见的策略是在编写代码时编写单元测试，当你要设计软件规范时，甚至要在编写代码之前编写单元测试。