COP4020 Programming Assignment 6

Download the CalcAST.java and AST.java sources from:

http://www.cs.fsu.edu/~engelen/courses/COP4020/CalcAST.java
http://www.cs.fsu.edu/~engelen/courses/COP4020/AST.java

The CalcAST program constructs an abstract syntax tree (AST) representation of arithmetic expressions. For example, when the expression that you input is 1+2; the program constructs the following AST:

```
+  
/ \  
1  2
```

This tree structure is constructed with the AST class, which has a tree node structure that contains an optional operator (e.g. +), an optional value (e.g. 1), and optional left and right subnodes for the operands of unary and binary operators. The AST class has a toLisp method. When invoked it will output the expression in Lisp form, such as (+ 1 2).

Compile the sources with:

```
javac CalcAST.java
```

And run this program:

```
java CalcAST
```

Enter 1+2; for example. The program output is the Lisp expression (+ 1 2).

Modify the AST.java source (and CalcAST.java program if necessary) to evaluate constant parts of expressions. That is, arithmetic operations are performed by the AST class when the operands are constants. When one of the operands is non-constant (symbolic), an AST node must be created to represent the non-constant expression (nodes are created using the AST class). The output of the program will be the partially evaluated expressions translated into Lisp.

In addition, your task is also to add productions and code to implement the ^ power operator. For the implementation, you need to use the static Math.pow method of class Math computes powers. This operator must be evaluated when possible: Example:

```
java CalcAST
2*(1+3)-2^3+x;
```

The output x is a simplified Lisp expression of the input.

Note that the AST node structure includes a val member that can be used to store a node’s value and to pass values as part of the AST instances that returned from methods (as a synthesized attribute values) and passed to methods (as an inherited attribute values). The type of val is Object, so to create an AST node with an integer value, say 7, you need: new AST(new Integer(7)).
Your program should at least be able to compute the following values:

\[
\begin{align*}
1+2+3; & \quad 6 \\
1*2*3; & \quad 6 \\
1*-2*(3-6); & \quad 6 \\
1+2+x+3; & \quad (+ (+ 3 x) 3) \\
x+1+2; & \quad (+ (+ x 1) 2) \\
x+0; & \quad x \\
1*x; & \quad x \\
x^1; & \quad x
\end{align*}
\]

Note that the semantic rules of the grammar enforce associativity, so \(1+2+x+3\) is evaluated from left to right. The evaluation process does not consider commutativity, so this expression does not need to be simplified down to \(x+6\).