CMPU-102 Polynomial Calculator Project
Alice Marbach

Notes & Instructions for running the program:

Starting:
For use with text input only, run the main(String[ ] args) from PolynomialCalc. For a GUI, run the first main(String[] args) from PolynomialGUI.

Features:
My program can take uppercase and lowercase for all input, and will accept either as the same.
Efficient running by use of TreeSet in Polynomial class.
The GUI can evaluate multiple lines without needing multiple clicks of Evaluate!

Notes for usage:
Names (variables) can only be one letter.
According to the flow diagram, once file mode or user mode is chosen, a different mode cannot be chosen. Therefore, when using the GUI, remember that once you have clicked on a button, you cannot choose a different mode (unless you exit and restart). This makes usage easier overall because for each user mode evaluation you don't need to keep clicking user mode.

Explanation:

A condensed explanation of how the program works can be found below.

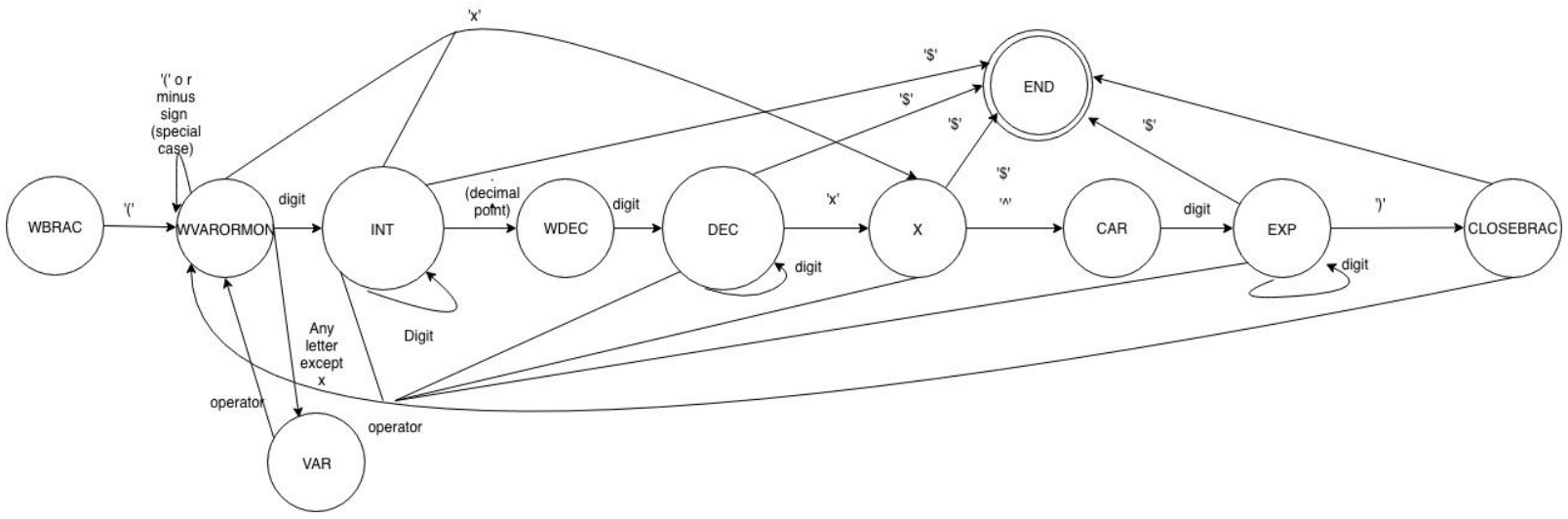

Figure 1: The Finite State Machine used in PolyReader

My project has many classes, so I will break them down into 3 categories:

- Building block classes
- User input evaluator classes
- Main[] strings programs (just where to run the program from).

Building block classes:

When thinking about building block classes, I had to consider how I would represent polynomials and variables to the computer, and which data structures I would use for this. I used tokens, with the interface "Token", as the most generic description of everything the user would be trying to parse in. I broke this down into Operators and Operands.

Operands:

I made Operand an interface because I did not require any instance methods or objects for the Operand class, but only needed one static method, isOperand(). Operand classes included NonXVariable, which was just a way to represent variables (e.g. a, b,c) in the list of Tokens for evaluation, and Monomial.

Monomial(double coefficient, double exponent):
The basic representation of a monomial. Each Monomial object has 2 instance variables initialized in the constructor. The Monomial class also contains add, subtract, multiply and divide instance methods which take in a Monomial and return the result Monomial. Monomial also implements the Comparable interface, overriding the compareTo method, so that Monomials would be ordered only according to their exponents (i.e. so that when 3x^2 and 2x^3 were compared, 2x^3 would be ordered first). I also overrode Object's equals method so that only exponents were compared. There is also a toString() class which added the 'x' and caret '^' signs appropriately, and getter methods.

Polynomial():
(Not officially an operand since did not implement Operand, but is still an operand by math.) The data structure I used for Polynomial() was quite complex, TreeSet, which implements Comparable (this is why I had to override compareTo above). For adding, subtracting, and multiplying, I made 2 instance methods each, with input of objects of either Monomial or Polynomial class. When adding a monomial to the list, I couldn't just add the object to the TreeSet, because if there was the same exponent, the coefficients needed to be summed instead of a new object being put in, so in add, subtract, and part of the multiply methods I used the equals method from Monomial and iterated through the set to check for this. Also I made sure that if the set was empty after any of the methods, a Monomial(0,0) was added to the set; otherwise 0 values were not included in the sets. All these methods output a new Polynomial because I did not want to alter the value of the input Polynomials/Monomials. Also included an Arithmetic Exception class for dividing by 0.

Figure 1: The Finite State Machine used in PolyReader

Benefits & Downsides of TreeSet implementation:
It is very efficient adding because of TreeSet's tree layout for comparing objects. But it cannot easily get a value from the set, because unlike ArrayList, there is no way to easily retrieve a Monomial from the set. There is a contains method, but no indexOf(Object o) like in ArrayList. This meant I had to see if there was a subSet from the input monomial to the input monomial (a way to see if it was there and retrieve it) every time I wanted add something.

Operators:

I made Operator an abstract class for two reasons. **First**, I knew that there would be no instances of an Operator object, but there would be of specific operators. These included AddOperator (+), SubtractOperator (-), MultiplyOperator(*), DivideOperator(/), RightParenOperator( )), and LeftParenOperator ((). **Second**, I could have the instance variables 'sign' and 'priority' to be initialized differently by each subclass.

Operator contained 2 static classes: isOperator(char c), outputCorrectOperator(char c), which checked the type of the input in the former, and output an instance of the operator matching the input character in the latter. Each of the Operator implemented operate(Polynomial poly1, Polynomial poly2), a non-static method which outputted the result of applying the operator that the method was called on to the two inputs.

User input evaluation:

There are 3 components that takes a String to a fully-evaluated polynomial.

1. **PolyReader:**

This class uses a Finite State Machine to a parse a String into a list of Token objects. To store the states of the machine, I used an enum, with the only accepting state being "END". First I split the input string left and right of the = sign, and split the string to the right according to whitespace. If there was an =, I assigned that as the name which was used as the key in the polyMap, with the value being the token list returned by step 3. I made a methods compileIntSaver() and addFromIntSaver() as a way to parse integers of decimals of multiple digits.

2. **InfixtoPostfix:**

Put token list (from PolyReader) from infix notation, (a + 3) * 4x^2), to postfix, a 3 + 4x^2 * for ease of evaluating according to the correct order of operations. This was done by adding operators onto a stack (a LIFO queue of type ArrayDeque), and popping them into a new token list with the operands according to their priority. This is also removed all parentheses.

3. **PostfixEvaluator:**

Takes a postfix String and evaluates it, also using a stack implementation. This is where the Operator abstract class method "operate" was useful, as for every 2 operands added onto the stack, the output of the operate method was used to replace them with the result of the operation. Added the result with the name from step 1 into the polyMap (a map with keys that

Figure 1: The Finite State Machine used in PolyReader

are variables (characters), and values that are Polynomials); this map is found in PolynomialCalc.

main(String[ ] args) classes:

PolynomialCalc and PolynomialGUI are the two “bootstrapping” methods.

Figure 1: The Finite State Machine used in PolyReader