APCS-A Java Expression Evaluator Coding Assignment

Write a single-file Java program that parses and evaluates arithmetic expressions. Your program should prompt the user to enter a mathematical expression on a single line. That is, the user will type the entire expression at one time. Each element of the expression, whether it is a number, operator, or symbol, may be separated by zero or more spaces. After the expression has been fully entered, your program should compute and display the numerical result. The following are sample entries, along with the answers that should be displayed by the computer. Note that the examples below do not show every type of expression required or needed for testing.

Examples:

User Enters	Computer Displays	Comment
7 – 2 + 1	6	Expressions are evaluated from left to right
7 – 9 / 2 + 5 * 3 + 1	18.5	Operator precedence must always be followed
(8 + 3) * 2	22	Parentheses take priority over everything
107 * 4.2 – 13.59	435.81	Each number may contain multiple digits and a decimal point
8000000 + 500 % 101	8000096	Numbers can consist of any amount of digits (don't worry about overflow)
0.4 * 5.27 – .591 / 8.	2.034	A decimal point can appear anywhere in a number
42 %14 / 4^ 2	0	Numbers, operators, and symbols may be separated by zero or more spaces
(12 + -5) ^ (-8 + 11)	343	Expressions may contain negative numbers
3 ^ 2 ^ 4 / 36	182.25	Decimal answers should be rounded to the nearest thousandth, if necessary
9 – 3/4 – 1/4	8	Decimal values produced within expressions may produce integer answers
-12 % (7 / 2)	-1.5	The modulus operator may be used with negative and decimal numbers
2 ^ -1.6 + 5 ^ (15 / 4)	418.293	Exponents may be positive or negative integers or decimal numbers
-54.0	-54	Expressions can consist of one number, positive or negative, integer or decimal
(-101) * (32.5)	-3282.5	Parenthetical expressions can consist of a single number

NOT Required:

Component	Examples			
Nested Parentheses	2 * (13 % (8 – 3)) + 7	(-(3) (4)) ^ 2	8 * (((12 + 9) / 3) – 2)	((5))
Implied Multiplication	6(3 – 4)	-(16)	(20 + 8) (5 ^ 2)	(352) 4

Your program should accept mathematical expressions consisting of positive and negative integers and decimal numbers of any size. The addition (+), subtraction (–), multiplication (*), division (/), and modulus (%) arithmetic operators are allowed. The exponent (^) symbol is also permitted, as are both left and right parentheses. Exponents can be positive or negative integers or decimal numbers.

Your program must trap for syntactical errors in each expression and display an appropriate message. This includes missing operators, symbols, or numbers, illegal characters, unmatched parentheses, and out-of-place operators or symbols. If an expression is otherwise correct, but cannot be properly evaluated (e.g., dividing or modding by zero), your program must give a detailed error message. Non-integer answers should be rounded to (at most) three digits after the decimal point, but only if necessary (see the examples above).

When evaluating expressions entered by the user, your program must take into account the precedence (priority) of the operators. The computer should first evaluate the parts of the expression that are parenthesized. The parts of the expression containing the exponent symbol should be evaluated next. The computer should then evaluate the parts of the expression containing multiplication, division, and modulus operators (in whatever order they appear, from the beginning of the expression to the end of the expression). Finally, the addition and subtraction operators should be handled in the order in which they appear in the expression (from beginning to end).

After the computer shows the result of each expression, your program should prompt the user to enter another expression. The user should be able to enter new expressions indefinitely until pressing ENTER by itself (with no expression), at which point your program should end gracefully. Make sure each expression is parsed and evaluated entirely within your program. In other words, you may not issue any "shell" or other commands to have the user's expression evaluated by other applications or Java classes not written by you.

Your program does not have to accept sets of nested parentheses, and it does not need to handle implied multiplication (see above).

To earn credit for a component above, it must be observable to the user when your program is executed. Non-working code gets you no points. Build your program in pieces, adding features one-by-one. Make sure each new feature works completely before coding the next feature. Test your program extremely well, entering all of the above examples, as well as lots of your own math expressions, including ones that should produce errors. Try to break your program, because that's what the instructor will try to do during grading.

You must use your CodeHS account to write, modify, and test all of your code; you may not create code elsewhere and then paste that code into your CodeHS editor. All code must be written entirely and only by you and not obtained from, or modified by, other sources.