CS 2336 – PROJECT 2 – Whovian Calculator (Standalone Project 1)

Pseudocode Due: 2/5 by 11:59 PM
Project Due: 2/19 by 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- **The file containing `main` must be named Main.java. (-5 points)**
- **The project files must be in a package named Calculator. (-5 points)**
- All project deliverables are to be submitted in eLearning.
- Zip the contents of the `src` directory into a single zipped file
    - Make sure the zipped file has a .zip extension (not .tar, .rar, .7z, etc.) (-5 points)
    - Please review the submission testing information in eLearning on the Course Homepage
- The pseudocode should be submitted as a Word or PDF document and is not accepted late.
- Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Programs must compile and run with JDK 8.
- Each submitted program will be graded with the rubric provided in eLearning as well as a set of test cases. These test cases will be posted in eLearning after the due date. Each student is responsible for developing sample test cases to ensure the program works as expected.
- **Type your name and netID in the comments at the top of all files submitted.** (-5 points)

Objectives:

- Implement basic class concepts in Java
- Implement inheritance with super and sub-classes
- Implement basic polymorphism concepts

Problem: The TARDIS has been infected by a virus which means it is up to Doctor Who to manually enter calculations into the TARDIS interface. The calculations necessary to make the TARDIS work properly involve real, imaginary and complex numbers. The Doctor has asked you to create a program that will analyze numerical expressions so that he can quickly enter the information into the TARDIS.

Details:

- **Classes**
    - Number class
        - Attributes
            - Real number (double)
        - Methods
            - Constructor – pass in value for number
            - Accessor

            - Mutator
            - toString
            - equals
    - Complex number class
        - Extends number class (-5 points)
        - Attributes:
            - Imaginary number (double)
        - Methods
            - Constructor – pass in real and imaginary
                - Call super constructor
            - Accessor
            - Mutator
            - toString
            - equals
- Read in the entire expression as a string and parse it into the proper objects
    - Store each part of the in the correct attribute of the object
- Use the toString function to display the object when necessary
- Both the real and imaginary parts may be floating point values
- Validate that each expression contains no invalid characters
    - The only valid letter in a complex number is `i` (lower case)
    - Validate that each expression contains a valid operator
- If a line contains invalid data, ignore the line
- Numbers may be represented in 3 ways
    - Complex (real + imaginary)
        - `<number><+ or -><number>i`
    - Real only
        - `<number>`
    - Imaginary only
        - `<number>i`
- Numbers may be positive or negative
- Results for arithmetic operators will be numerical
- Results for relational operators will be Boolean
- Calculating less/greater than of a complex or imaginary number will be determined by analyzing the modulus (or absolute value) of each complex number
- All functions in Main.java that would normally require a real or complex number parameter object must use Object parameters instead (-10 points)
    - Use the instanceof operator to determine the type of object and perform the proper actions based on the object type
        - For example, adding complex numbers would add both the real and imaginary parts, where as adding real numbers would only add the real parts
- Comment your code generously. Refer to the grading rubric for more details

- Use as few variables as possible. Don't waste memory holding a calculation just so you can print it out one time.

User Interface: There will be no user interface for this program. All I/O will be performed with a files

Input:

- All input will come from a file named `expressions.txt`.
- Each line in the file will contain an expression to be evaluated
- Valid line format: `<number><space><operator><space><complex number>`
- Valid operators
    - + (add)
    - – (subtract)
    - * (multiply)
    - / (divide)
    - < (less than)
    - > (greater than)
    - = (equal)
    - /= (not equal)
- There will be a newline at the end of each line except for the last line (which may or may not have a newline)

Output:

- All output will be written to a file named `results.txt`.
- Format all output into a table with 2 columns
    - Column 1: original expression
        - Use toString to print out the numbers in the expression
    - Column 2: value
- All values in the columns will be left justified
- Numerical values will be rounded to 2 decimal places.
- Relational operators should evaluate to true or false
- Each expression will be listed on separate lines