CS2430 SPRING 2015

Program #3 (30 points)

Dr. Lily Chang

Due Dates

1. Work Plan in SE Tools due: Friday, Feb. 27 @10pm
2. JUnit test for Complex class due on S drive: Wednesday, March 4 @10pm
3. Test document for PostfixEvaluator class due on S drive: Friday, March 6 @10pm
4. All Java files due on the Grader: Monday, March 9 @ 10pm
5. Grace date: Wednesday, March 11 @10pm

** Files due on S drive must be put into a single folder named your_login_name, and put it into 1Dropbox folder by the due date.

Program Description

In this program, you will be implementing an RPN expression calculator that computes the value of RPN expressions over complex numbers. Informally, complex numbers are numbers of the form a+bi, where a and b, for this program, are assumed to be integers and i is the imaginary number whose square is -1, i.e., $i^2 = -1$. We will evaluate RPN expressions of the form 1 2 i * + 3 4 i * + *, which is equivalent to: (1+2i) * (3+4i) infix.

You must write three classes for this program: (1) a **Complex class** to represent a complex number as described above, (2) a **PostfixEvaluator class** to read expressions, evaluate them and then output the answers, and (3) a **general Stack class** as we discussed in class.

For the sake of simplicity, you may assume that all operands given as input are valid. Valid operands are positive integers or the imaginary number i. You may assume that the only valid operators are +, -, * and ~, where ~ is the conjugate operator. The conjugate of a complex number a+bi is defined as the complex number a-bi. Note that ~ is a unary operator, meaning it operates on exactly one operand, whereas +, - and * are all binary operators, which means they operate on exactly two operands. You must figure out how to modify the RPN evaluation algorithm that we talked about in class to allow for the conjugate operator. Since division is not a valid operator, you do not have to check for division-by-zero in this program.

When given an input RPN expression, you must check for all possible error conditions but keep in mind that all operands given will be valid. Each input RPN expression will be input on its own line with spaces separating operands and operators. Evaluate and output (echo) each expression as you read it in. If you detect that an expression is invalid, then only output the expression starting from the beginning up to and including the point at which you detect the error. Thus, for valid expressions, you will output the entire expression. When you echo the expression, make sure there is exactly one space following each operator or operand and there should be no trailing spaces. If an expression is invalid, after you echo the (portion of the) expression, output "Invalid Expression!" on the next line. If the expression is valid, then output "The value is: X", where X is the result of invoking **toString()** on the Complex object that represents the value of the just-evaluated-expression.

As you evaluate each expression, store the result of every valid expression in an ArrayList, in the order in which the result was computed, that is, first result is stored at first position, second at second position, etc. You MUST use a **generic ArrayList** for this task. Have the declaration **private ArrayList<Complex> answers;** in your program (don't forget to instantiate it before you use it!). To add a Complex object to answers, use the **add()** method of ArrayList. When you are done evaluating all of the expressions, loop through your ArrayList of results and print each one out on its own line. You may use the **get()** method of ArrayList in the loop to access the Complex objects one by one. As you are printing out each of the answers, output whether each result is real, complex or imaginary. For a complex number z=a+bi, z is real if b=0, z is imaginary if a=0 and z is complex if neither of the previous two conditions hold. Print "Normal termination of program 3." before the program stops.

Program Requirement

1. Program 3 is an **individual assignment**. Sharing your solution could result in being accused of **plagiarism**.
2. You MUST follow the software development ground rules.
3. You are required to create a work plan for Prog3. The work plan is due **Feb. 27 @10pm**. Up to **-5 points** if you didn't create a work plan for this program. Log on to CSSE Hub/SE Tools, and add your plan of how you will work on this

program, together with what you plan to work on at that time, and how you plan to finish by the due date. It must show that you have a good understanding of what needs to be done and have specific parts of the program you plan to work on and when. For example,

Plan for Prog3 – must be entered into CSSE Hub/SE Tools
Feb. 25 1-2pm Read program description
Feb. 25 7-8pm Prepare work plan
……

4. You MUST use **SE tools** to keep an up-to-date log of the time spent working. Up to **-5 points** if this is not done or you do not log all your time or you do not provide specific comments as to what you were working on.
5. Each class must go in a separate file. I will provide **Prog3.java**.
6. You will get **0 points** if you submit Prog3 after **March 11, 10pm**, or submit Prog3 with differences. You MUST submit Prog3 with 0 differences to pass this course.
7. Submit the following files to the grader:
    - Stack.java
    - Complex.java
    - PostfixEvaluator.java
    - Prog3.java
8. If you need help from me, place your project in **S:\Courses\CSSE\changl\cs2430\your_login_id**. So I can access it from my office when you stop by.
9. The requirement for Stack class, Complex class and PostfixEvaulator class is as follows.
    - **Stack Class**
      This is a general Stack class and you MUST use an array of **Object** to implement the Stack class. You must have pop() and push() methods. Make sure you have a constructor that takes an integer that represents the capacity of the Stack. You may have a clear() method and isEmpty() method.
    - **Complex class**
      The Complex class represents a complex number of the form a+bi, where a and b are integers. Your Complex class MUST have the following--and ONLY the following--public methods. **-2 points** for each method not defined and/or not used.
        - public Complex(): this constructor creates the default complex number a+bi where a=b=0.
        - public Complex(int a): this constructor creates the complex number a+bi, where b=0.
        - public Complex(int a, int b): this constructor creates the complex number a+bi where a and b are integers.
        - public Complex plus(Complex cp): this method returns a new Complex that represents the sum of this and cp. If this represents a+bi and cp represents c+di, then the return value should represent (a+c)+(b+d)i.
        - public Complex minus(Complex cp): this method returns a new Complex that represents the difference between this and cp. Its implementation is similar to plus().
        - public Complex times(Complex cp): this method returns a new Complex that represents the product of this and cp. If this represents a+bi and rhs represents c+di, then the return value should represent (ac-bd)+(ad+bc)i.
        - public Complex conjugate(): this method should return a new Complex that represents the conjugate of this. The conjugate of a complex number a+bi is defined as the complex number a-bi.
        - public boolean equals(Object obj): this method returns true if the given Object represents the same Complex as this. Two Complex objects have the same value if they have the same real and imaginary parts.
        - public String toString(): this method returns a String representation of this. The return value should not contain any redundant symbols. For example, if z=5+0i, then you should return "5". If z=7+-3i, then you should return "7-3i". If z=0+0i, then you should output "0". If z=0+1i, then you should output "i". If z=0+-1i, then you should output "-i". Always be sure to match the output exactly!
        - You MUST use every method and constructor of the Complex class.

- You must create JUnit tests for all methods of Complex. The tests must be thorough! Complex.java and ComplexTest.java are **due March 4 @10pm** on S drive. You will **lose 1 point** per test (**up to 5 points**) for each test that is not thorough. You will also **lose 1 point per missing/failing test** (up to 5 points). You DO NOT have to test the constructors. Just assume that they are correct for the other tests.

- **PostfixEvaluator class**
  This class allows the user to input and evaluate RPN expressions either via the command line or from an input file. No input is prompted!

  You may have ONLY the following public method: **public void run() throws IOException**. Make sure you have the signature of this method correct, especially the throws IOException part. No method can have more than 30 lines of code (including the opening and closing braces) so you will have to define some private helper methods. You should have the following--and ONLY the following--private data members:

  ```
  private static final int STACK_SIZE = 100;
  private Stack operandStack;
  private String expression;
  private ArrayList<Complex> answers;    // import java.util.*;
  ```

  Do the input for this program using a BufferedReader. Start by putting: **import java.io.*;** at the top of your the class, then have the following lines in your run() method:

  ```
  //BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
  BufferedReader stdin = new BufferedReader(new FileReader("P3.in"));
  ```

  Having the two lines as above, your program will read from the standard input (you type in input). By commenting out the first line and uncommenting the second, your program will read from the file P3.in (put it at the top level of your Prog3 project folder next to but not inside of the "src" folder). When you submit to the GRADER, have it as above with the first line uncommented and the second line commented.

  Have a loop that reads input until there is no more input to read. You should read a line of input by using the **readLine() method** of BufferedReader. This method reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed. NOTE: This method returns a String containing the contents of the line, not including any line-termination characters, or null if the end of the input has been reached.

  Compute and output each expression as you read it in. Output should be of the form "Expression X is: Y", where X is the number of the expression (the first expression is 1, the second 2, etc.), and Y is the expression with exactly one space after each token, except for the last token, which does not have any trailing spaces. If the expression is invalid, then Y is the portion of the expression, from the beginning, up to and including the point at which the expression was determined to be invalid. See the sample output for details.

  When you are evaluating each expression, you will need to push() and pop() your **operandStack**. You must **ONLY push instances of Complex.** Thus, when you pop(), you know exactly what to convert the return value to.

  After you output each expression, output the result of evaluating the expression (the answer). If the expression is valid, then output "The value is: X", where X is the value returned by invoking toString() on the Complex that represents the result of the expression. If the expression is invalid, then output "Invalid expression". See the sample output for details. For each valid expression, store the result in answers, which as stated above, is an ArrayList.

  When the return value of readLine() is null, then there are no more expressions. At this point, you should output all of the results, one per line. Output should be of the form "X is Y", where X is the result of invoking toString() on a Complex and Y is either "real", "imaginary" or "complex" depending on whether or not X is real, imaginary or complex. See the sample output for details. In order to determine whether each answer is real, imaginary or complex, you can use any of the public methods of Complex class but you **CANNOT use the toString() in your if/else statements**. Before terminating the program, print the phrase: "Normal termination of program 3.".

10. System-level Test Document for PostfixEvaluator class.
    You must write a test document for PostfixEvaluator class specifying tests to achieve a correct program. You MUST use a table format for your test document. Put it in Landscape mode. You must specify explicit tests, consisting of explicit inputs and precise expected outputs. Each test case must be stand-alone in the sense that no single test should depend on the success or failure of any previous test. When you have completed your test document (on or before its due date), put it on S drive in 1Dropbox **by March 6 @10pm**. **Failure to submit a test document will result in an automatic 5 point deduction.**
    Furthermore, do not simply throw this document together at the last minute. Take it serious. In fact, you should probably create this document before implementing your program and so that you may use it as a guide during the implementation phase. The following is an example of how the test document should be formatted assuming that input expressions are entered one after another during a single "run" of Prog3:

Test Case	Description	Input	Expected result / output
1	Valid input expression	Expression 1 is: 37 64 i * -	The value is: 37-64i
2	Invalid operator in input expression	Expression 2 is: 1 2 i * /	Invalid Expression!
...			

Keep in mind that the above example IS NOT a thorough test document. Your test document will contain many more test cases. Don't forget to put your name on the test document as well.

Sample Input

```
0
0 ~
5 ~
i ~
2 3 *
3 2 *
5 7 +
7 5 +
17 20 -
20 17 -
i
i 2 +
2 i +
i 2 *
2 i *
i 10 -
i 0 -
0 i -
12 34 i * +
12 i 34 * +
37 64 i * -
13 i 1 * -
13 i 1 * +
0 100 3 i * + -
0 99 i 7 * - -
31 2 3 i * + +
63 5 i 7 * + -
2 13 17 i * + *
2 3 i * + 31 +
5 i 7 * + 63 -
13 17 i * + 2 *
```

```
1 2 i * + 2 i 3 * - +
1 2 i * + 2 i 3 * + *
13 27 i * + 12 i 13 * + -
43 17 i * + ~ 15 i 7 * - +
2 3 i * + 2 3 i * + ~ -
5 i 7 * + 14 7 i * + -
14 7 i * + 5 i 7 * + -
8 9 i * + ~ ~
i 1 + 2 3 4 5 6 7 8 9 10 i * - + + + + + + * +
1 2 i * / 3 i * 4 +
+
~
197 24 i * + 211 133 i * - ~
%
1 2 i * + 3 i 4 * - * *

i 1 2 3
```

Sample Output

```
Expression 1 is: 0
The value is: 0
Expression 2 is: 0 ~
The value is: 0
Expression 3 is: 5 ~
The value is: 5
Expression 4 is: i ~
The value is: -i
Expression 5 is: 2 3 *
The value is: 6
Expression 6 is: 3 2 *
The value is: 6
Expression 7 is: 5 7 +
The value is: 12
Expression 8 is: 7 5 +
The value is: 12
Expression 9 is: 17 20 -
The value is: -3
Expression 10 is: 20 17 -
The value is: 3
Expression 11 is: i
The value is: i
Expression 12 is: i 2 +
The value is: 2+i
Expression 13 is: 2 i +
The value is: 2+i
Expression 14 is: i 2 *
The value is: 2i
Expression 15 is: 2 i *
The value is: 2i
Expression 16 is: i 10 -
The value is: -10+i
Expression 17 is: i 0 -
The value is: i
Expression 18 is: 0 i -
```

```
The value is: -i
Expression 19 is: 12 34 i * +
The value is: 12+34i
Expression 20 is: 12 i 34 * +
The value is: 12+34i
Expression 21 is: 37 64 i * -
The value is: 37-64i
Expression 22 is: 13 i 1 * -
The value is: 13-i
Expression 23 is: 13 i 1 * +
The value is: 13+i
Expression 24 is: 0 100 3 i * + -
The value is: -100-3i
Expression 25 is: 0 99 i 7 * - -
The value is: -99+7i
Expression 26 is: 31 2 3 i * + +
The value is: 33+3i
Expression 27 is: 63 5 i 7 * + -
The value is: 58-7i
Expression 28 is: 2 13 17 i * + *
The value is: 26+34i
Expression 29 is: 2 3 i * + 31 +
The value is: 33+3i
Expression 30 is: 5 i 7 * + 63 -
The value is: -58+7i
Expression 31 is: 13 17 i * + 2 *
The value is: 26+34i
Expression 32 is: 1 2 i * + 2 i 3 * - +
The value is: 3-i
Expression 33 is: 1 2 i * + 2 i 3 * + *
The value is: -4+7i
Expression 34 is: 13 27 i * + 12 i 13 * + -
The value is: 1+14i
Expression 35 is: 43 17 i * + ~ 15 i 7 * - +
The value is: 58-24i
Expression 36 is: 2 3 i * + 2 3 i * + ~ -
The value is: 6i
Expression 37 is: 5 i 7 * + 14 7 i * + -
The value is: -9
Expression 38 is: 14 7 i * + 5 i 7 * + -
The value is: 9
Expression 39 is: 8 9 i * + ~ ~
The value is: 8+9i
Expression 40 is: i 1 + 2 3 4 5 6 7 8 9 10 i * - + + + + + + * +
The value is: 85-19i
Expression 41 is: 1 2 i * /
Invalid Expression!
Expression 42 is: +
Invalid Expression!
Expression 43 is: ~
Invalid Expression!
Expression 44 is: 197 24 i * + 211 133 i * - ~
Invalid Expression!
```

```
Expression 45 is: %
Invalid Expression!
Expression 46 is: 1 2 i * + 3 i 4 * - * *
Invalid Expression!
Expression 47 is:
Invalid Expression!
Expression 48 is: i 1 2 3
Invalid Expression!
The list of good answers is:
0 is real
0 is real
5 is real
-i is imaginary
6 is real
6 is real
12 is real
12 is real
-3 is real
3 is real
i is imaginary
2+i is complex
2+i is complex
2i is imaginary
2i is imaginary
-10+i is complex
i is imaginary
-i is imaginary
12+34i is complex
12+34i is complex
37-64i is complex
13-i is complex
13+i is complex
-100-3i is complex
-99+7i is complex
33+3i is complex
58-7i is complex
26+34i is complex
33+3i is complex
-58+7i is complex
26+34i is complex
3-i is complex
-4+7i is complex
1+14i is complex
58-24i is complex
6i is imaginary
-9 is real
9 is real
8+9i is complex
85-19i is complex
Normal Termination of Program 3.
```