PROJECT #4: 100 points, CSE 274 – Fall 2019
Stack, Array-based Implementation
Algebraic Expressions: Infix, Postfix, Evaluation

Outcomes:

- Implement a Stack using an array (resizable).
- Implement an algorithm to convert infix expressions into postfix expressions
- Implement an algorithm to evaluate infix expressions
- Implement an algorithm to evaluate postfix expressions

Preliminaries:

Review the algorithms in chapter 5 for:

- Determining whether parentheses in an algebraic expression are balanced correctly
- Converting an infix expression to a postfix expression
- Evaluating a postfix expression
- Evaluating an infix expression directly

Read chapter 6 to understand how to implement a stack using an array.

Overall goal:

Write a program that allows the user to type an infix expression. Check that the expression is valid. Then, if it is valid, give the user the equivalent postfix expression, and evaluate the expression. Here are some sample runs to show how your program should interact with the user (user input is shown in red):

Infix expression: 4 +7* 6 – 10 **Enter 1 to get postfix expression, 2 to evaluate postfix expression and 3 to directly evaluate the infix expression: 1** **4 7 6 * + 10 -**
Infix expression: (5 + 4) / (4 - 1) **Enter 1 to get postfix expression, 2 to evaluate postfix expression and 3 to directly evaluate the infix expression: 1** **5 4 + 4 1 - /**
Infix expression: (5+(4 *3 - 6) **Enter 1 to get postfix expression, 2 to evaluate postfix expression and 3 to directly evaluate the infix expression: 1** **Fail: Invalid expression**
Infix expression: 10 ^ 2 ^ 3 **Enter 1 to get postfix expression, 2 to evaluate postfix expression and 3 to directly evaluate the infix expression: 1** **10 2 3 ^ ^**
Infix expression: 10 - 2 - 3 **Enter 1 to get postfix expression, 2 to evaluate postfix expression and 3 to directly evaluate the infix expression: 1** **10 2 - 3 -**
Infix expression: - 2 - 3 **Enter 1 to get postfix expression, 2 to evaluate postfix expression and 3 to directly evaluate the infix expression: 1** **Fail: Invalid expression**
Infix expression: 4 +7* 6 – 10 **Enter 1 to get postfix expression, 2 to evaluate postfix expression and 3 to directly evaluate the infix expression: 2** **4 7 6 * + 10 -** **36**
Infix expression: 4 +7* 6 – 10 **Enter 1 to get postfix expression, 2 to evaluate postfix expression and 3 to directly evaluate the infix expression: 3** **36**

PROJECT #4: 100 points, CSE 274 – Fall 2019
Stack, Array-based Implementation
Algebraic Expressions: Infix, Postfix, Evaluation

Naming requirements (not following any of these may result in a score of 0):

- You will have exactly four source code files: **StackInterface.java** (an interface you are provided), **ArrayStack.java (**your array-based implementation of StackInterface), **InfixExpression.java**, and **Tester.java**
- The exception, *EmptyStackException* you need to throw is from the standard java class library. https://docs.oracle.com/javase/7/docs/api/java/util/EmptyStackException.html
- You will use the **default package** (this means there should be no package statements in any of your files).

Requirements:

class ArrayStack - Does not require Javadoc since the interface is commented

- Implement the Stack interface provided in StackInterface.java in the project folder using generics. This is the first time that you have implemented an interface by using generics. The idea here is simple. Imagine how you would implement a stack of Strings… you would find **String** throughout your implementation. Here, instead of using String, you will use **T** instead. The first line of your implementation must begin:
  public class ArrayStack<T> implements StackInterface<T> {

- Use an array to implement the ArrayStack class. It has to be a resizable array-based implementation, that is, double the capacity each time you occupy all the available spaces. You don't need to reduce the capacity at any time.
- Implement only one constructor, which constructs an empty stack.
- Throw an **EmptyStackException** whenever a peek or pop is called on an empty stack.

InfixExpression - Requires Javadoc

Implement the following class

InfixExpression
-infix : String
+InfixExpression(String) +toString() : String -isBalanced() : boolean -isValid() : boolean -clean() : void +getPostfixExpression() : String +evaluatePostfix() : int +evaluate(): int

- **public InfixExpression(String):** This should construct a new InfixExpression object, but will store it as a cleaned up expression. It should make use of the private clean() method to do the cleaning. The constructor should throw an IllegalArgumentException if the String parameter is not valid, calling on the isValid() method for help.

PROJECT #4: 100 points, CSE 274 – Fall 2019
Stack, Array-based Implementation
Algebraic Expressions: Infix, Postfix, Evaluation

- **public String toString()** simply returns the infix expression. This is a one-liner that will make it easy for you and me to print and test your code.

- **private boolean isBalanced()** This private method should return true if the infix expression has balanced parentheses, and false otherwise. It does not check for any other kinds of invalid expressions. So, for example, isBalanced() should return true for the invalid infix expression "(3 ((* * 4) 8) 7 7) 6" because the parentheses are balanced, even though other things are a mess. Use the algorithm described in chapter 5 for checking balanced parentheses. This algorithm should use an instance of your ArrayStack class.

- **private boolean isValid()** This private method should return true if the infix expression is valid in all respects, and false otherwise. For example, isValid() should return false for the infix expression "(3 ((* * 4) 8) 7 7) 6", even though the parentheses are balanced. isValid() should check all of the following:
    - Whether the expression contains illegal characters (the only valid characters are the digits 0 through 9, the 6 operators and parentheses symbols + - * / % ^ ( ), and spaces.
    - Whether the parentheses are valid (by calling on isBalanced())
    - Whether there is an improper order of operators and operands. Observe that each operator must have an operand before it and after it (an operand can be a number, or an expression in parentheses). For example, the following are NOT valid infix expressions:
        - "* 3 4"
        - "3 4 +"
        - "+"
        - "3 (4 + 5)" // note that this is implied multiplication...something we will consider NOT valid
        - "3 * * 4"
        - "4 10"
    - Note that these ARE valid infix expressions:
        - "3"
        - "3 * (5)"
        - "(4 + 9)"
        - "((4 + 9))"

If you are stuck on isValid() or isBalanced(), then save them for last and then only test your code with valid expressions. But then be sure to write stubs for any unimplemented methods.

- **private void clean()** cleans the expression, cleaning up the provided string by putting a single space between adjacent tokens, and no leading or trailing spaces. So, for example, if the parameter is " 3+4* 83 / 6 ", it should be stored as "3 + 4 * 83 / 6". The only reason this method exists is to help the constructor. That's why it is private. One fairly simple approach to this could be to use the replace() method of the String class to insert spaces where needed, and to convert multiple spaces into single spaces.

PROJECT #4: 100 points, CSE 274 – Fall 2019
Stack, Array-based Implementation
Algebraic Expressions: Infix, Postfix, Evaluation

- **public String getPostfixExpression()** This method should return the postfix expression that corresponds to the given infix expression. For example, if the infix expression is "3 + 4" then getPostFixExpression() should return "3 4 +"
    - The operands (numbers) will all be non-negative integer values.
    - Valid operators are + - * / % ^ (the % is the modulus operator, and has the same level of precedence as * and /). Parentheses may appear in the expression. This method should assume that the infix expression is valid because the constructor is supposed to be throwing an exception if it is not.
    - The method should use the algorithm described in section 5.16, along with your ArrayStack class, to compute the postfix expression. The postfix expression should be returned with a single space separating each operand or operator from each adjacent operand or operator, and with no leading or trailing spaces. **Note** that the algorithm described in 5.16 assumes operands are single letter variables. For us, our operands will be non-negative integer values that could contain multiple digits (such as 47 or 0 or 999). You will need to modify the algorithm to accommodate for this.

- **public int** evaluatePostfix**()** This method should evaluate the infix expression, returning the integer result of the calculation. For example, if the infix expression is "13 + 4" then evaluatePostfix() should return 17.
    - All calculations should be integer calculations. So, for example, the value of the infix expression "7 / 2" should be 3, and not 3.5.
    - The method should use the **algorithm described in section 5.18**, along with your ArrayStack class, first using the getPostfixExpression and then evaluating that postfix expression.
    - Do not do anything special to handle dividing by 0 or computing modulus where the second operand is 0. Just let Java handle that the way that Java likes to handle that.

- **public int** evaluate**()** This method should evaluate the infix expression directly without converting it to postfix expression, returning the integer result of the calculation. For example, if the infix expression is "13 + 4" then evaluate() should return 17. In this method we are going to convert infix expression directly by using two stacks.
    - All calculations should be integer calculations. So, for example, the value of the infix expression "7 / 2" should be 3, and not 3.5.
    - The method should use the **algorithm described in section 5.21**, along with your ArrayStack class, using two stacks (an operator stack and a value stack) to evaluate infix expression.
- Do not do anything special to handle dividing by 0 or computing modulus where the second operand is 0. Just let Java handle that the way that Java likes to handle that.

PROJECT #4: 100 points, CSE 274 – Fall 2019
Stack, Array-based Implementation
Algebraic Expressions: Infix, Postfix, Evaluation

Tester

- **public static void main(String[] args)** This method is the one that should interact with the user. It can call on other helper methods as needed.
    - Assume that the user might type an expression with either balanced or unbalanced parentheses, but that the expression will be valid in all other ways.
    - Assume that the user will only use non-negative integers.
    - If the user enters a valid infix expression, you should display the corresponding postfix expression and evaluate the expression.
    - If the user enters an invalid expression, you should simply indicate that the expression is not valid, and not show the postfix expression or the value. Note that the InfixExpression class throws an exception if you try to instantiate a new object using an invalid expression. However, in this main method, you should simply catch that exception and print a friendly message to the user (see sample output).

Scoring:

Outcome	**Max score**
ArrayStack<T> implemented correctly	15
constructor, clean(), getInfixExpression() implemented correctly	10
isBalanced() implemented correctly	10
isValid() implemented correctly	10
getPostfixExpression() implemented correctly	15
evaluatePostfix() implemented correctly	15
evaluate() implemented correctly	15
Tester tests all the methods thoroughly	10
Code formatted according to generally accepted standards. Note that Javadoc comments are required for the InfixExpression class but not for ArrayStack.	0 (deductions only)