Damian Kryzia

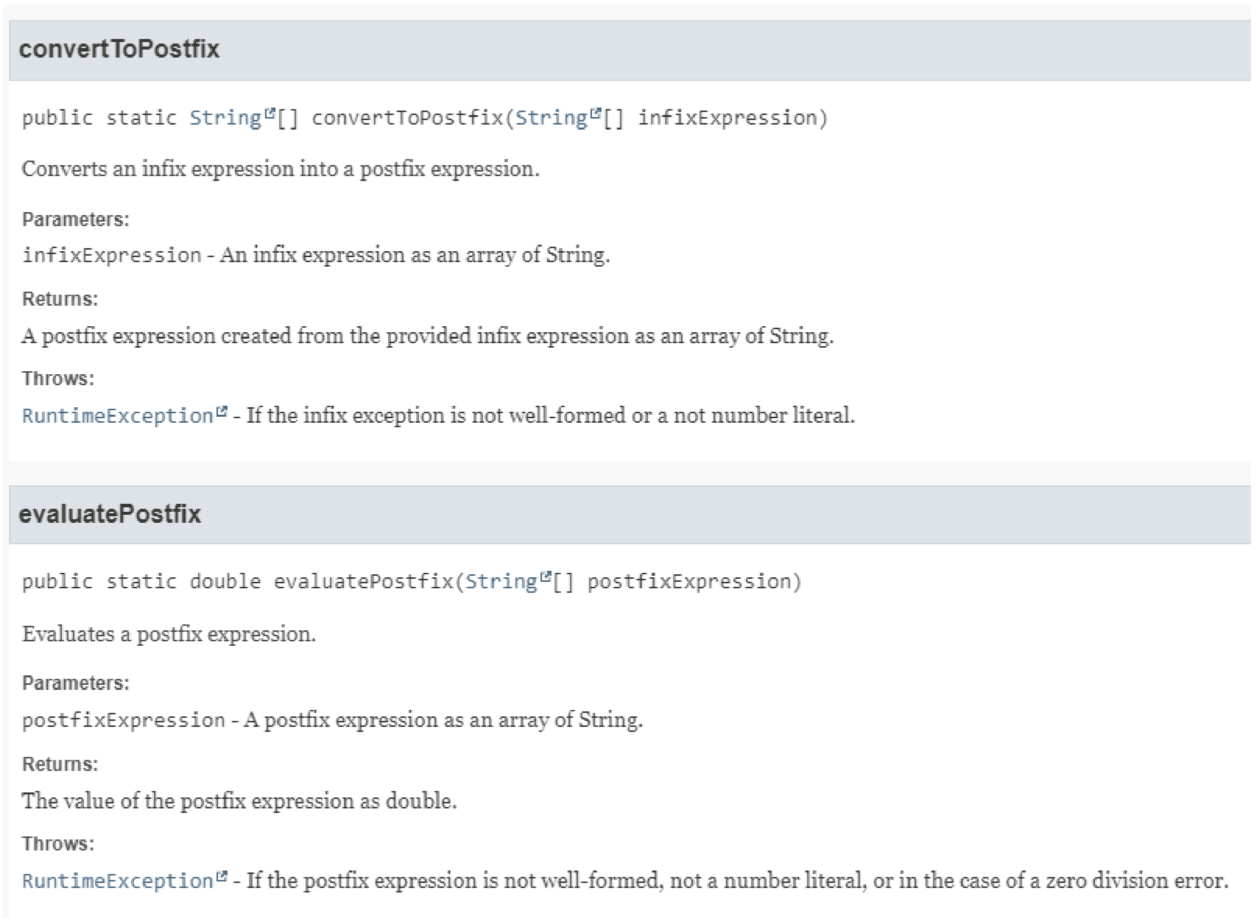
CS2400-02

Project 3

10/21/2022

## Project Specification

This project includes the Expression class that contains methods used to convert an infix expression into a postfix expression, as well as evaluate a postfix expression. They are static methods and the class's constructor is set to private in order to not allow the creation of Expression objects. The methods of the Expression class are as follows:



### convertToPostfix

```
public static String[] convertToPostfix(String[] infixExpression)
```

Converts an infix expression into a postfix expression.

Parameters:
`infixExpression` - An infix expression as an array of String.

Returns:
A postfix expression created from the provided infix expression as an array of String.

Throws:
`RuntimeException` - If the infix exception is not well-formed or a not number literal.

### evaluatePostfix

```
public static double evaluatePostfix(String[] postfixExpression)
```

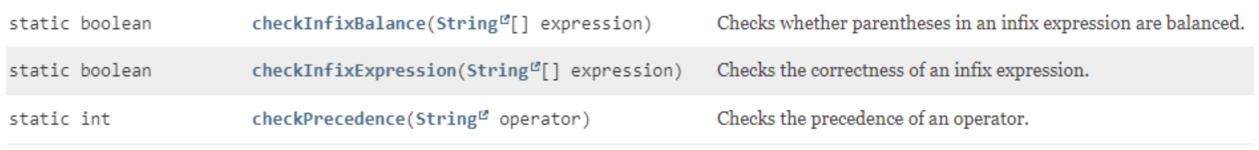
Evaluates a postfix expression.

Parameters:
`postfixExpression` - A postfix expression as an array of String.

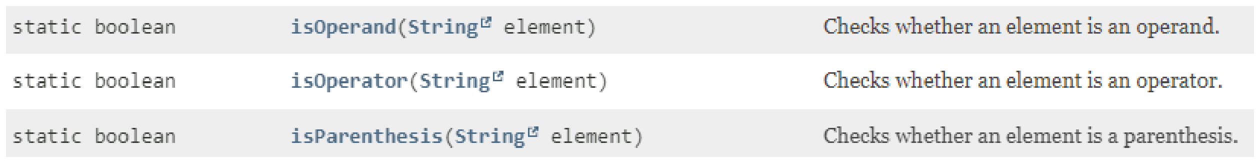
Returns:
The value of the postfix expression as double.

Throws:
`RuntimeException` - If the postfix expression is not well-formed, not a number literal, or in the case of a zero division error.

The class Expression also uses private helper methods that check the correctness of the infix expression as well as the precedence of used operators. They are as follows:

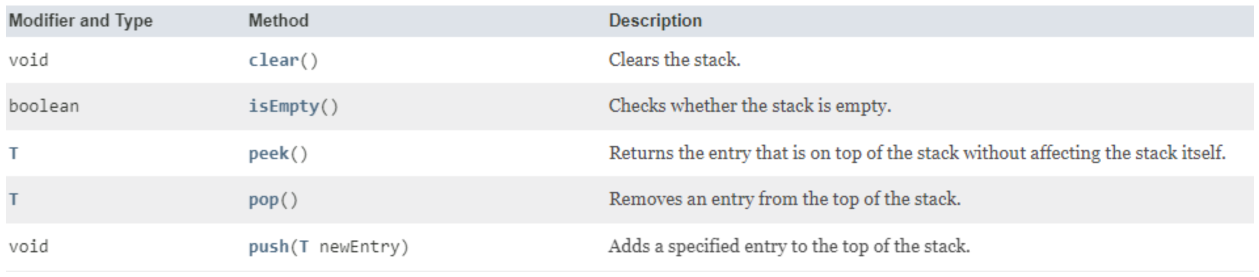


| | | |
|---|---|---|
| `static boolean` | `checkInfixBalance(String[] expression)` | Checks whether parentheses in an infix expression are balanced. |
| `static boolean` | `checkInfixExpression(String[] expression)` | Checks the correctness of an infix expression. |
| `static int` | `checkPrecedence(String operator)` | Checks the precedence of an operator. |



| | | |
|---|---|---|
| static boolean | isOperand(String element) | Checks whether an element is an operand. |
| static boolean | isOperator(String element) | Checks whether an element is an operator. |
| static boolean | isParenthesis(String element) | Checks whether an element is a parenthesis. |

The class Expression uses the generic LinkedStack class which is an implementation of the generic Stack interface, therefore the stack data type. LinkedStack uses a singly-linked list of nodes to store and manipulate data. The methods of LinkedStack<T> specified by Stack<T> are as follows:



| Modifier and Type | Method | Description |
|---|---|---|
| void | clear() | Clears the stack. |
| boolean | isEmpty() | Checks whether the stack is empty. |
| T | peek() | Returns the entry that is on top of the stack without affecting the stack itself. |
| T | pop() | Removes an entry from the top of the stack. |
| void | push(T newEntry) | Adds a specified entry to the top of the stack. |

LinkedStack uses a private Node class defined within it in order to implement a singly-linked list. The Node class is as follows:

```
private class Node {

        private T data;
        private Node next;

        private Node(T dataPortion)
        {
                this(dataPortion, null);
        }

        private Node(T dataPortion, Node nextNode)
        {
                data = dataPortion;
                next = nextNode;
        }

}
```

## Testing Methodology

This project is tested using the ExpressionTest.java file that contains the main() method.

The testing goes as follows:

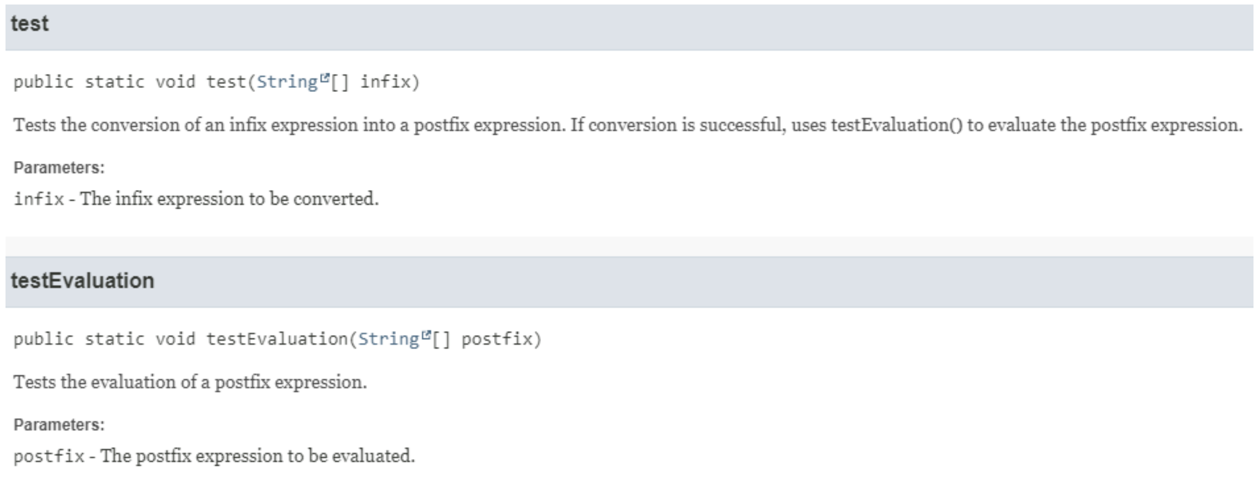
1. main() tests the Expression class methods using the command line parameter that is an infix expression.

   It is to be provided in the form of e.g. "a + - ( b * c ) ^ d" where a, b, c, and d are number literals.

   main() converts the infix expression to its postfix form using Expression.convertToPostfix(String[] infixExpression), evaluates the postfix expression using Expression.evaluatePostfix(String[] postfixExpression), and outputs its value.

2. main() then tests different cases of infix and postfix expressions, some valid, some invalid (provided purposefully) in order to ensure the correctness of the Expression class methods. In the case of invalid expressions, errors are caught using a try{} catch(){} expression.

   For testing purposes, the following methods are used by the main() method:



**test**

```
public static void test(String[] infix)
```

Tests the conversion of an infix expression into a postfix expression. If conversion is successful, uses testEvaluation() to evaluate the postfix expression.

**Parameters:**
`infix` - The infix expression to be converted.

**testEvaluation**

```
public static void testEvaluation(String[] postfix)
```

Tests the evaluation of a postfix expression.

**Parameters:**
`postfix` - The postfix expression to be evaluated.

3. The following cases are tested:
    - Valid infix expression: 2 + 4
    - Valid, longer infix expression: 2 + 3 * 2 / 12 ^ 8 - 14
    - Valid, longer infix expression with parentheses: (2 + 3) * (2 / 12) ^ (8 - 14)
    - Invalid infix expression with incorrect operands or operators: a + 10
    - Invalid infix expression with unbalanced parentheses: (2 + 2) - 3)
    - Invalid infix expression with missing operands: 2 * 3 / (7 - )
    - Invalid postfix expression with incorrect operands or operators: 2 a +
    - Invalid postfix expression with missing operands or operators: 4 5 + -
    - Invalid postfix expression with incorrect order (prefix expression is provided instead of a postfix expression): + 2 3
    - Invalid postfix expression with zero division: 10 0 /