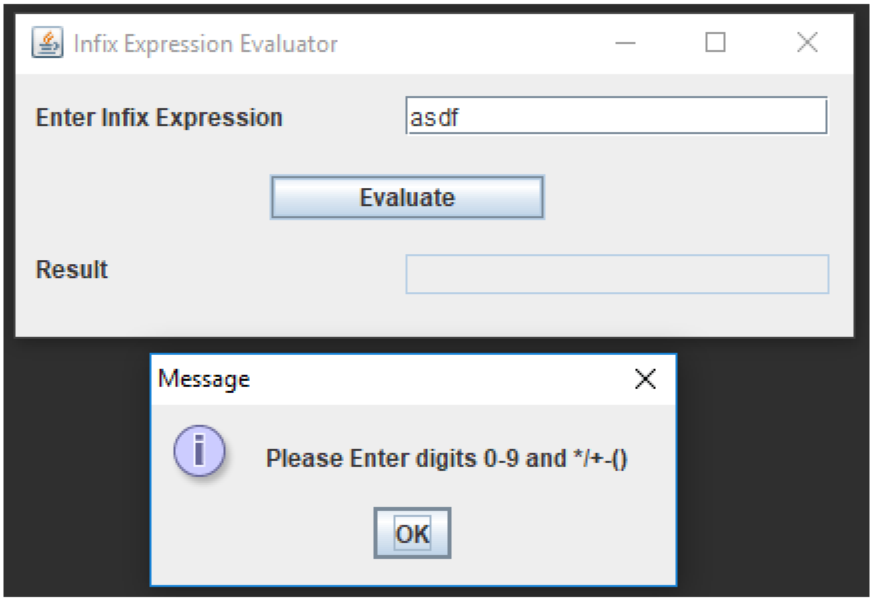
Kresimir Tokic
CMSC350 Project 1
Test Plan, UML diagram, Lessons Learned

TEST PLAN

Test Case 1: Testing user input invalid characters

1. Input “asdf”
2. Click Evaluate button
3. Verify Invalid character window pops up

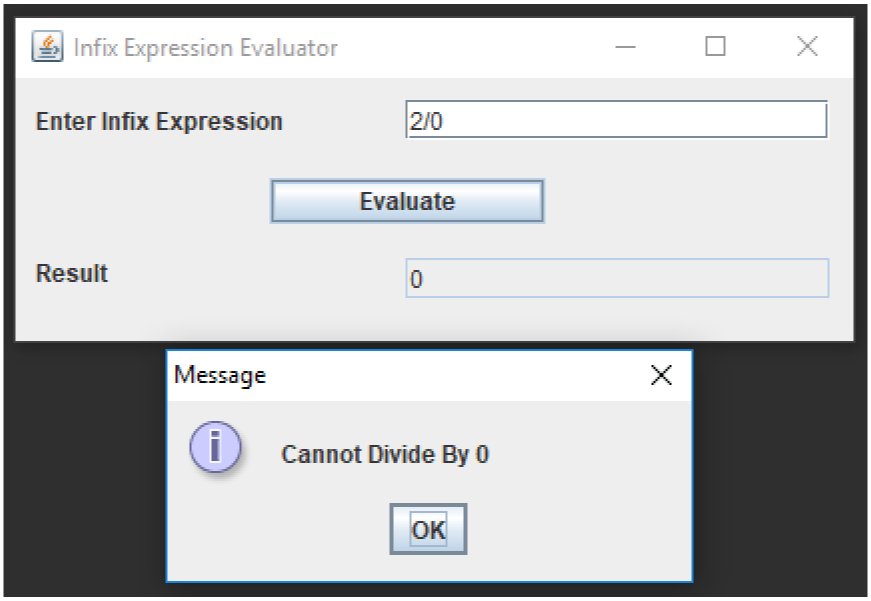


Enter Infix Expression: asdf

Please Enter digits 0-9 and */+-()

Test Case 2: Testing for divide by zero error

1. Input “2/0”
2. Click Evaluate button
3. Verify divide by zero error window pops up



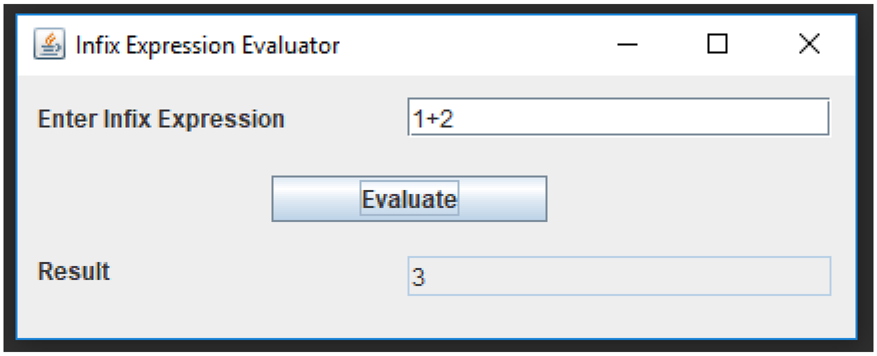
Enter Infix Expression 2/0

Result 0

Cannot Divide By 0

Test Case 3: Testing an expression without white spaces

1. Input “1+2”
2. Click evaluate button
3. Verify the correct result

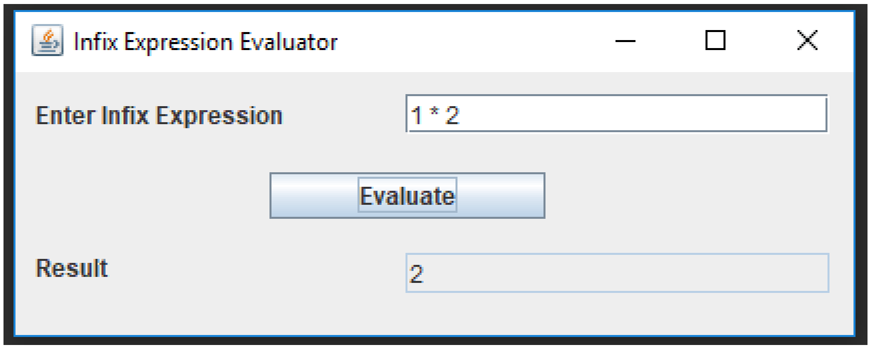


Enter Infix Expression 1+2

Result 3

Test Case 4: Testing an expression with white spaces

1. Input “ 1 * 2 ”
2. Click evaluate button
3. Verify the correct result

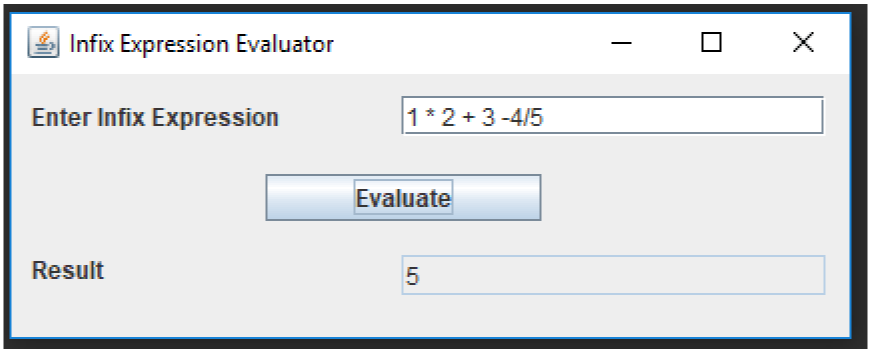


Enter Infix Expression 1 * 2

Result 2

Test Case 5: Testing precedence

1. Input “1 * 2 + 3 - 4/5”
2. Click Evaluate button
3. Verify the correct result

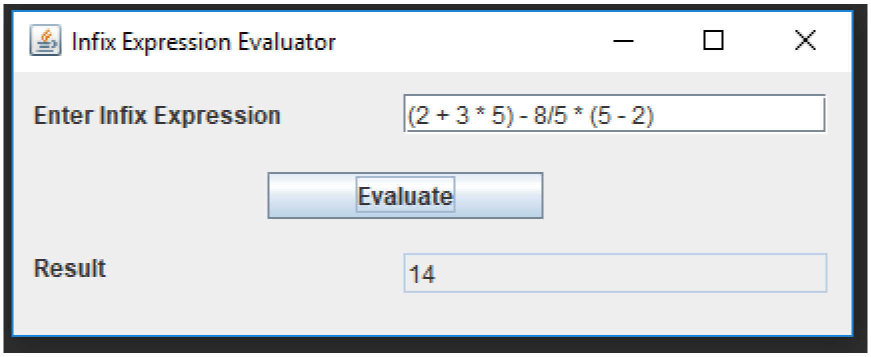


Enter Infix Expression 1 * 2 + 3 -4/5

Result 5

Test Case 6: Testing expressions with parenthesis

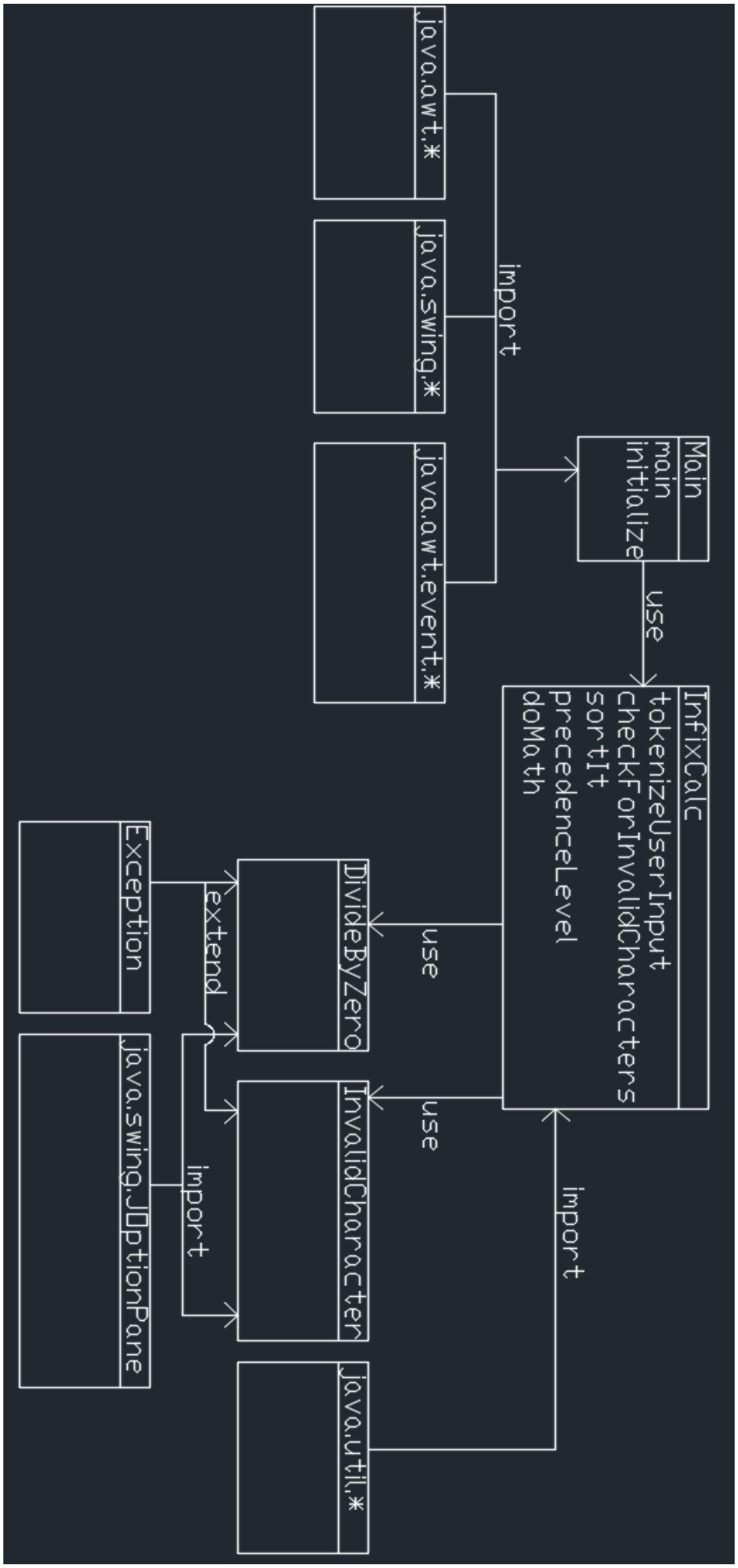
1. Input “(2 + 3 * 5) - 8/5 * (5 - 2)”
2. Click Evaluate button
3. Verify the correct result



Enter Infix Expression (2 + 3 * 5) - 8/5 * (5 - 2)

Result 14

UML Diagram


java.awt.*
java.swing.*
java.awt.event.*
import
Main
main
initialize
use
InfixCalc
tokenizeUserInput
checkForInvalidCharacters
sortIt
precedenceLevel
doMath
use
use
import
DivideByZero
InvalidCharacter
java.util.*
Exception
extend
java.swing.JOptionPane
import

Lessons Learned

The Character wrapper class has some really useful methods for this project that helped me minimize the lines of code, such as .isWhiteSpace() and .isDigit(). I was having a hard time determining if the user input contained proper operators and/or invalid characters until I figured out how to use Set<> for my operator symbols which cleaned up a lot of lines of code as well. These two things were probably biggest lessons learned. I was already familiar with .push() and .pop() from my minimal experience with JavaScript. I originally built my algorithm slightly differently than the provided pseudocode and had difficulty getting the correct results from my evaluations. I decided to rewrite much more closely to the pseudocode and it immediately cleared up my issues. I frequently use System.out.print() to help me debug my code. This was especially useful creating my tokens, making sure they're being sorted into the appropriate stacks and tracking the results of the evaluations. This was my first-time creating UML diagrams, which I drew in AutoCAD because I'm comfortable with it. I'm not sure if I've captured the requirements of the UML diagram, but I'm sure I'll get more adept at it as I complete more of them. Another challenge I found with this project was converting data types. If I were do this project over, I would reconsider using characters for my operand stack.