**Mod 4: Critical Thinking**

Alex Mehler

CSU Global

CSC400-1

Santosh Gottipamula

December 15, 2024

**Code:**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Stack;

public class Calc {

//evaluate a postfix expression

public static int evaluatePostfix(String expression) {

Stack<Integer> stack = new Stack<>();

String[] tokens = expression.split("\\s+"); // Split by spaces, support multi-digit numbers

for (String token : tokens) {

if (*isOperator*(token)) {

//Ensure there are enough operands in the stack

if (stack.size() < 2) {

System.***out***.println("Error: Invalid expression, not enough operands for operator '" + token + "'");

return Integer.***MIN\_VALUE***; // Error indication

}

//Pop the top two operands from the stack

int b = stack.pop();

int a = stack.pop();

//Perform the operation and push the result back on the stack

stack.push(*applyOperator*(a, b, token));

} else {

//Ensure the token is a valid number (operand)

try {

stack.push(Integer.*parseInt*(token)); //Push the operand onto the stack

} catch (NumberFormatException e) {

System.***out***.println("Error: Invalid number '" + token + "' in the expression");

return Integer.***MIN\_VALUE***; //Error indication

}

}

}

//If the stack has more than one element, the expression is invalid

if (stack.size() != 1) {

System.***out***.println("Error: Invalid expression, too many operands left.");

return Integer.***MIN\_VALUE***; // Error indication

}

//The result will be the only value left in the stack

return stack.pop();

}

//Check if the token is an operator

private static boolean isOperator(String token) {

return token.equals("+") || token.equals("-") || token.equals("\*") || token.equals("/") || token.equals("%");

}

//Apply the operator to the two operands

private static int applyOperator(int a, int b, String operator) {

switch (operator) {

case "+": return a + b;

case "-": return a - b;

case "\*": return a \* b;

case "/":

if (b == 0) {

System.***out***.println("Error: Division by zero.");

return Integer.***MIN\_VALUE***; //Error indication

}

return a / b;

case "%":

if (b == 0) {

System.***out***.println("Error: Modulo by zero.");

return Integer.***MIN\_VALUE***; //Error indication

}

return a % b;

default: throw new IllegalArgumentException("Unknown operator: " + operator);

}

}

//read file and evaluate each

public static void evaluateExpressionsFromFile(String fileName) {

try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {

String line;

while ((line = br.readLine()) != null) {

System.***out***.println("Evaluating expression: " + line);

try {

// Evaluate and print

int result = *evaluatePostfix*(line);

if (result != Integer.***MIN\_VALUE***) {

System.***out***.println("Result: " + result);

}

} catch (Exception e) {

System.***out***.println("Error: " + e.getMessage());

}

}

} catch (IOException e) {

System.***out***.println("Error reading file: " + e.getMessage());

}

}

public static void main(String[] args) {

//Read from a file and evaluate

String fileName = "C:\\Users\\maste\\eclipse-workspace\\Calculator\\src\\expressions.txt"; // Specify the path to your text file

*evaluateExpressionsFromFile*(fileName);

}

}

**Screenshot:**

**A screenshot of a computer

Description automatically generated**

**Git:**

https://github.com/SevRnce/csc400\_mod4