**Assignment 2 – Oisin Mc Laughlin – 22441106**

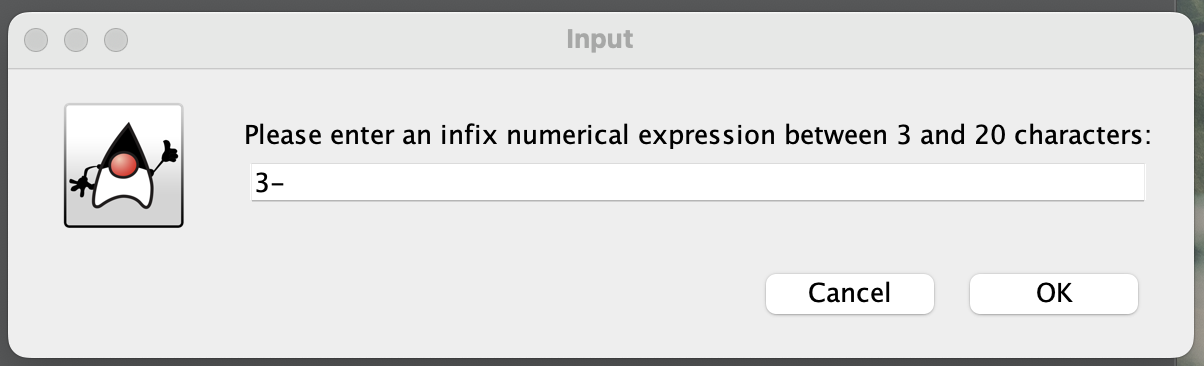
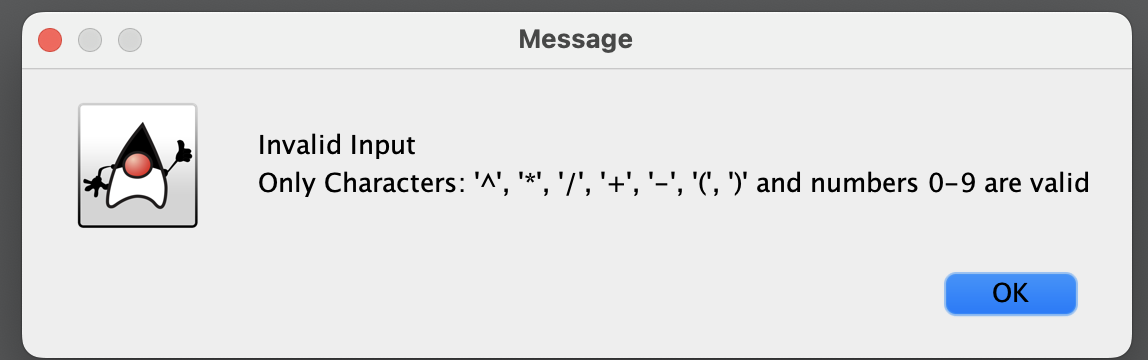
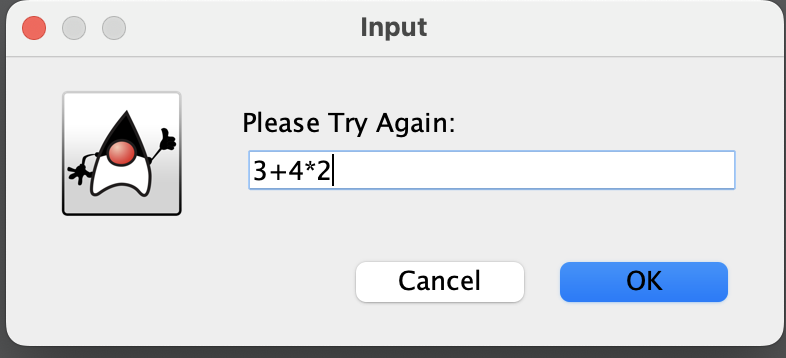
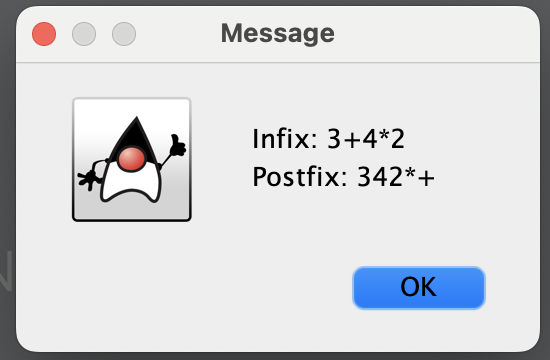
**Problem Statement**

This assignment will take an infix expression which takes the form <operand> <operator> <operand> and convert it to a postfix expression which will take the form <operand> <operand> <operator>. The assignment must only scan in single digits 0-9 or the operators +, -, \*, /, (, ), ^. This assignment implements the arraystack class which is on canvas. From my analysis for this assignment I’m going to have 5 methods including main. checkInput will ensure the user input is correct by using to char array and checking if the entered value is between 3 and 20 characters, returning false if it is out of the range. It also checks to see if the characters are valid by comparing the operators and operands to an array for valid operands and operators. The isOpp array will check if the character is an operand once again by comparing it to the valid operators array and returns true if it finds a match. The precedence method will return an integer value depending on what the precedence of the operator is, for example for + or – it will return 1, for \* or / it will return 2 etc. Main will prompt a user for an entry and then run a while loop until the correct input is entered. If the correct input is entered, it will run the infToPos method where the real bread and butter happens. This method will create a string builder and check if the character is a digit, if it is, the digit is added to the postfix expression straight away to maintain the order. It then checks if there is an open bracket, if there is, it is also pushed to the stack. Next it checks if there a is a closed bracket, if there is, the method will pop the operators from the stack and append to the postfix string builder until an open bracket is found which will handle the idea with brackets in the first place. The next check is for operators, when one is found the precedence method returns how important the operator is, if the precedence of the operator is lower or equal, the other operators are popped and added to the postfix string builder until that condition is no longer met. This makes sure that the postfix expression is the correct order. After this the string builder is returned.

**Code**

import javax.swing.\*;  
//import java.util.Scanner;  
  
public class Main {  
 private static char[] validChars = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '-', '\*', '/', '^', '(', ')'};  
 private static char[] validNums = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};  
 private static char[] validOperators = {'^', '\*', '/', '+', '-'};  
  
 public static void main(String[] args) {  
// Scanner sc = new Scanner(System.in);  
// System.out.println("Please enter an infix numerical expression between 3 and 20 characters:");  
 String input;  
 String postFix;  
  
 //Prompts the user to enter their infix expression  
 input = JOptionPane.showInputDialog(null, "Please enter an infix numerical expression between 3 and 20 characters:");  
 while (true) {  
 //This loop will constantly run until the user enters a correct expression by calling the checkInput method with the user input  
 if (checkInput(input)) {  
 break;  
 }  
 else {  
 //Prompt again if incorrect expresssion  
 JOptionPane.showMessageDialog(null, "Invalid Input\nOnly Characters: '^', '\*', '/', '+', '-', '(', ')' and numbers 0-9 are valid");  
 input = JOptionPane.showInputDialog(null, "Please Try Again:");  
 }  
 }  
 //Calling the infix to posfix method with the user input and displays their infix and postfix expression  
 postFix = infToPos(input);  
 JOptionPane.showMessageDialog(null, "Infix: " + input + "\nPostfix: " + postFix);  
 }  
  
 public static boolean checkInput(String in) {  
 //Converts to character array  
 char[] inCheck = in.toCharArray();  
// System.out.println(inCheck);  
  
 //If input is less than 3 or greater than 20 return false  
 if (in.length() < 3 || in.length() > 20) {  
 return false;  
 }  
  
 //Loops through input char array  
 for (int i = 0; i < inCheck.length; i++) {  
 //Creates a flag with value false for the char  
 boolean flag = false;  
 //It then loops through the array of valid characters  
 for (int j = 0; j < validChars.length; j++) {  
 //Once the char is found in the valid char array, the char is valid and the flag is switched to true and the loop moves on to the next char in the array  
 if (inCheck[i] == validChars[j]) {  
 flag = true;  
 break;  
 }  
 }  
 //If a matching char in the valid chars array is never found, the flag is never set to true and false is returned  
 if (!flag) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 public static String infToPos(String in) {  
 StringBuilder postF = new StringBuilder();  
 ArrayStack stack = new ArrayStack(); // Assuming ArrayStack can handle Character objects  
 char[] inArr = in.toCharArray();  
  
 for (int i = 0; i < inArr.length; i++) {  
 //Correctly reference the character from the input array  
 //Also done since had to keep putting (char) before i to stop errros  
 char c = inArr[i];  
  
 //Appends to the string builder if the char is a digit  
 if (Character.isDigit(c)) {  
 postF.append(c);  
 }  
 //Push char if open bracket  
 else if (c == '(') {  
 stack.push(c);  
 }  
 //If close bracket then while loop will run while stack isn't empty and top of stack isn't opening bracket  
 else if (c == ')') {  
 while (!stack.isEmpty() && (char)stack.top() != '(') {  
 //Append each popped char to string builder  
 postF.append((char)stack.pop());  
 }  
 //If stack isn't empty pop, this should remove the open bracket from the stack  
 if (!stack.isEmpty()) {  
 stack.pop();  
 }  
 }  
 //If character is an operator  
 else if (isOpp(c)) { // Check if the character is an operator  
 //While stack isn't empty and the precedence of that char is less than or equal to the char at the top of the stack  
 while (!stack.isEmpty() && precedence(c) <= precedence((char)stack.top())) {  
 //Append each popped char to string builder  
 postF.append((char)stack.pop());  
 }  
 //Push that char to stack then  
 stack.push(c);  
 }  
 }  
 while (!stack.isEmpty()) {  
 //While there is still chars in stack, append them to the string builder to ensure the correct postfix result  
 postF.append((char)stack.pop());  
 }  
 //Return the string builder result after loop has finished  
 return postF.toString();  
 }  
  
 //This method loops through and compares the input to the valid operator array and returns true if it finds a match, otherwise it returns false  
 public static boolean isOpp(char in) {  
 for (int i = 0; i < validOperators.length; i++) {  
 if (in == validOperators[i]) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 //This method checks the precedence, it assigns each operator a value depending on where it lays in the "BIMDAS" rule otherwise it returns -1 if theres no match  
 public static int precedence(char in) {  
 if ((in == '+') || (in == '-')) {  
 return 1;  
 }  
 else if ((in == '\*') || (in == '/')) {  
 return 2;  
 }  
 else if (in == '^') {  
 return 3;  
 }  
 else {  
 return -1;  
 }  
 }  
}

**Testing**

****