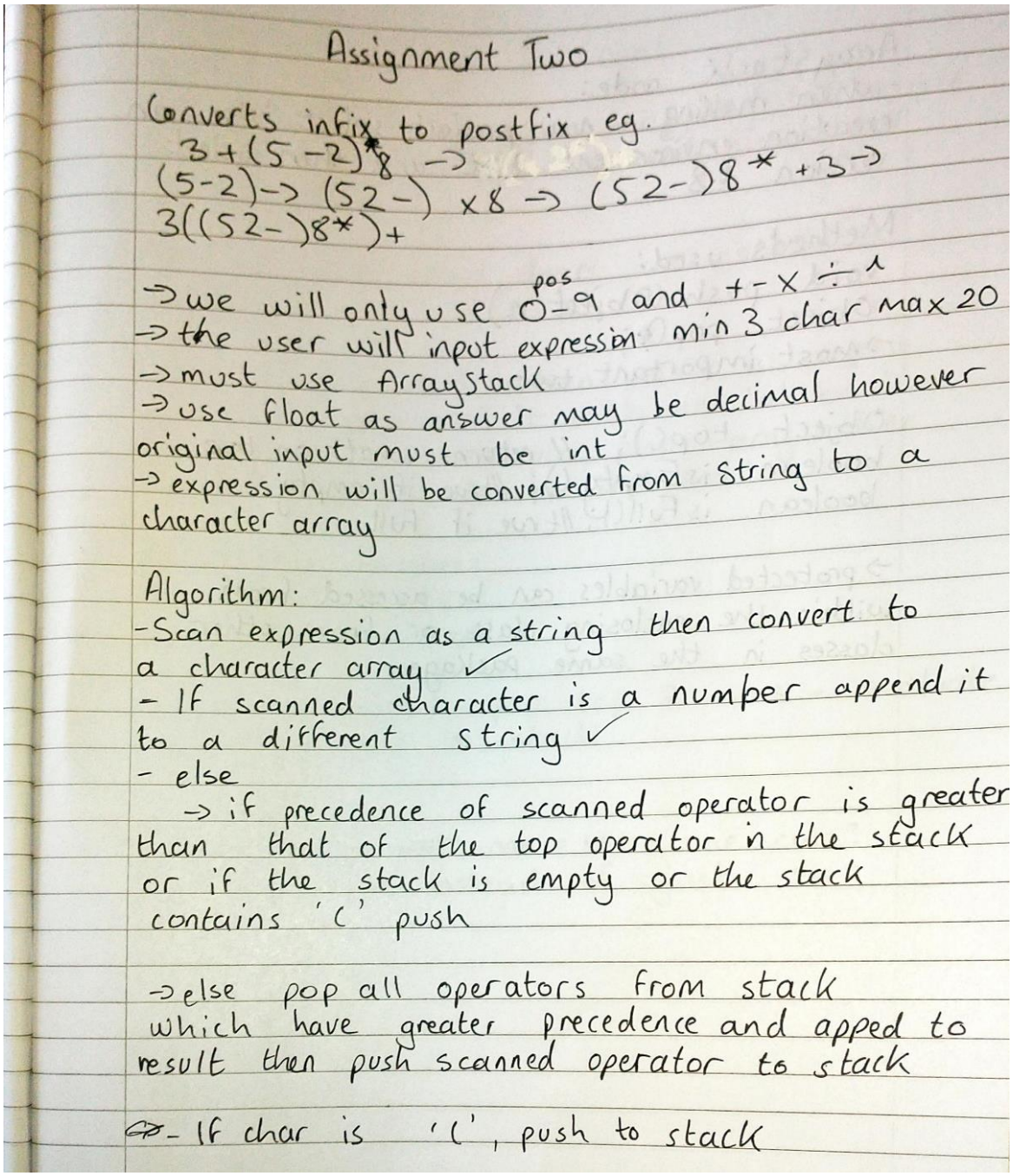
OOP Data Structures and Algorithms Assignment 2

Problem statement:

Create an infix to postfix converter that also evaluates the answer to the expression, using the ArrayStack code given. Only takes integers 0-9, operands +-*/^ and brackets (). Minimum 3 chars, maximum 20. Answer should be saved as a float as it may not be a whole number.

Analysis and design notes:



Assignment Two

Converts infix to postfix eg.
3+(5-2)*8 ->
(5-2)-> (52-) x8 -> (52-)8* +3->
3((52-)8*)+

-> we will only use pos 0-9 and +-x÷^
-> the user will input expression min 3 char max 20
-> must use ArrayStack
-> use float as answer may be decimal however original input must be int
-> expression will be converted from string to a character array

Algorithm:
-Scan expression as a string then convert to a character array ✓
- If scanned character is a number append it to a different string ✓
- else
  -> if precedence of scanned operator is greater than that of the top operator in the stack or if the stack is empty or the stack contains '(' push

-> else pop all operators from stack which have greater precedence and apped to result then push scanned operator to stack

- If char is '(', push to stack

Assign 2 cont.

- If char is ')' pop + append to result until a '(' is encountered then get rid of both

- Pop any remaining chars on stack to result

* code doesn't work when stack is empty so I just populated it with an 'a' because that has a lower prec than any other characters that can be inputted

Calculating answer:
- ~~create new stack~~ can use same stack
- scan postfix expr.
  - If no., push to stack
  - If operator, pop top 2 no.s + evaluate
  - push result to stack

*error when ~~eate~~ more than one calc?
eg. (2+3)-4 instead of just 2+3
✓ fixed

Code:

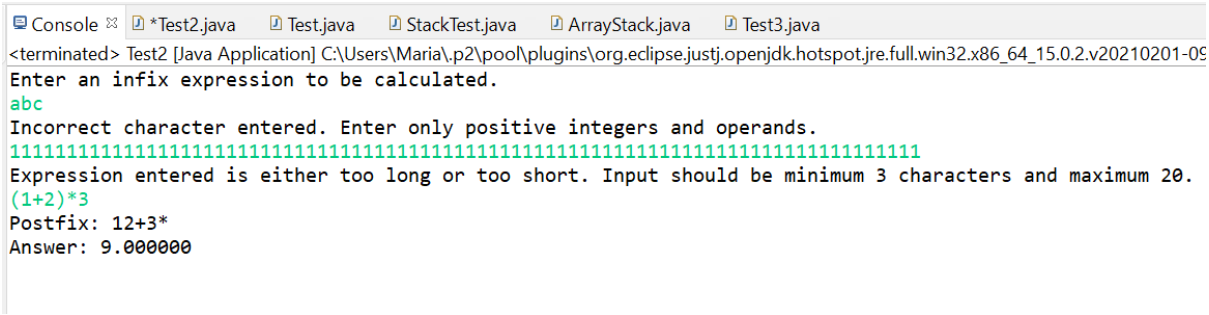
```
import java.util.Scanner; //need to import the scanner class to use a scanner

public class Test2
{
	static String input; //takes initial input from user
	static char[] expression = new char[20]; //input is put in a char array
	static String postfix = "";
	static float ans;
	static int num1;
	static int num2;
	static float floatNum1;
	static float floatNum2;

	static int CheckInput(String str1)
	{
		if(input.length()<3 || input.length()>20) {
			System.out.println("Expression entered is either too long or
too short. Input should be minimum 3 characters and maximum 20.");
			return 0;
			//makes sure the length of the expression is correct
		}

		else{
		for(int j=0; j < input.length(); j++) {
			char a = input.charAt(j);

if(a!='1'&&a!='2'&&a!='3'&&a!='4'&&a!='5'&&a!='6'&&a!='7'&&a!='8'&&a!='9'&&a!='+'&
&a!='-'&&a!='*'&&a!='/'&&a!='^'&&a!='('&&a!=')') {
			 System.out.println("Incorrect character entered. Enter only
positive integers and operands.");
			 return 0;
			//checks that only integers 0-9 and operands are entered
		}
		  if(j<input.length()-1 && Character.isDigit(input.charAt(j)) &&
Character.isDigit(input.charAt(j+1)))
		  {
			 System.out.println("Incorrect expression entered. Enter a new
one.");
			 return 0;
			 //checks that only single digit integers are entered, so no
numbers over 9
		  }

		}
		}
		return 1;
	}

	static int Prec(char ch)
    {
        switch (ch)
        {
        case '+':
        case '-':
            return 1;
```

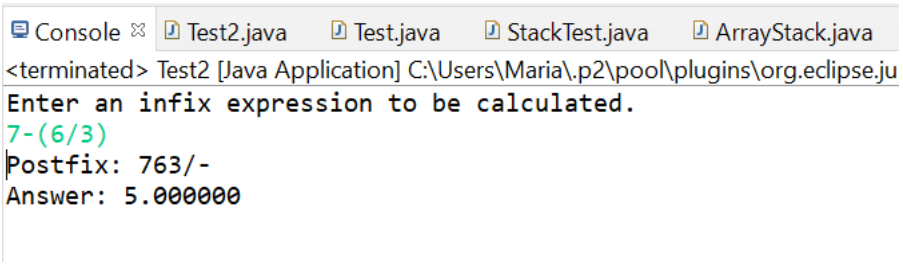
```
        case '*':
        case '/':
            return 2;

        case '^':
            return 3;
        }
        return -1;
    }
      //Prec takes the operand as a char and returns a number on a acale from -1
to 3 that represents its importance

      static void Evaluate(char ch2)
      {
      switch (ch2)
    {
    case '+':
        ans = floatNum1 + floatNum2;
        return;

    case '-':
        ans = floatNum2 - floatNum1;
        return;

    case '*':
        ans = floatNum1 * floatNum2;
        return;

    case '/':
        ans = floatNum2 / floatNum1;
        return;

    case '^':
      num1 = (int)floatNum1;
      num2 = (int)floatNum2;
        ans = (float)Math.pow(num2, num1);
        return;
    }
      }
      //Evaluate evaluates the answer of each individual expression and saves the
answer as ans

      public static void main(String[] args) {

            //evaluate postfix expression first

            Stack stack = new ArrayStack(20);
            stack.push('a'); //I pushed a onto the stack as my code wasn't
running when the stack was empty originally and it was easier
            //and I think more efficient to just add an a which would have
Prec('a')=-1 anyways

            Scanner scan = new Scanner(System.in); //creates scanner
            System.out.println("Enter an infix expression to be calculated.");
            input = scan.next(); //takes the entire input from user as a String

            while(CheckInput(input)<1) {
                  input = scan.next();
                  //checks that the input is correct
```

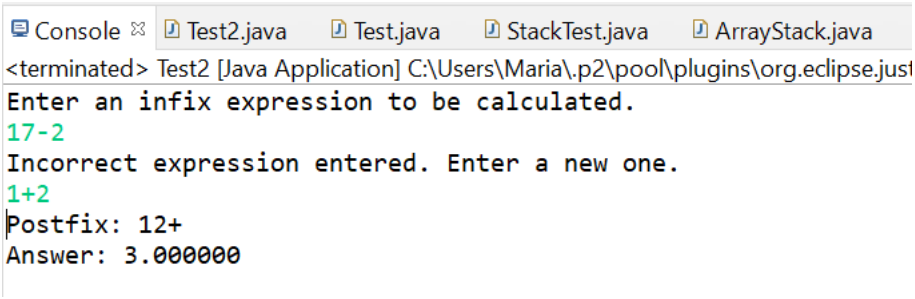
```
		}

		for(int j=0; j<input.length(); j++) {
			expression[j] = input.charAt(j);
			//populates the expression array with chars from input

		}

		for(int i=0; i < input.length(); i++) {
		  if(Character.isDigit(expression[i])) {
			postfix += expression[i];
			//if the char is an integer it is added to the postfix String
		  }


			else if(!Character.isDigit(expression[i]) &&
expression[i]!=')' && (Prec(expression[i])>Prec((char)stack.top()) ||
Prec(expression[i])==Prec((char)stack.top()) || ((char)stack.top())=='(')) {
			  stack.push(expression[i]);
			  //if the char is not an integer and satisfies the above
conditions it is pushed to the stack
			}

			else {

			  if(expression[i] == '(') {
				stack.push(expression[i]);
				//open brackets are always pushed to stack
			    }

			    if(expression[i] == ')') {
				while((char)stack.top() != '(' && !(stack.isEmpty()))
{
					postfix += (char)stack.pop();
					//pops to postfix everything in the stack until
an open bracket is encountered
				}

				if((char)stack.top()=='(') {
					stack.pop();
					//gets rid of open bracket
				}
			    }

			    while(Prec((char)stack.top())>Prec(expression[i]) &&
(expression[i] != '(' || expression[i] != ')')) {
			      postfix += (char)stack.pop();
			      //if the Prec of the operand is lower than the stack
pop the stack, appending to postfix, until the
			      //top is equal or lower in precedence

			  }

			   }

			}
		while ((char)stack.top() != 'a') {
			postfix+=(char)stack.pop();
```

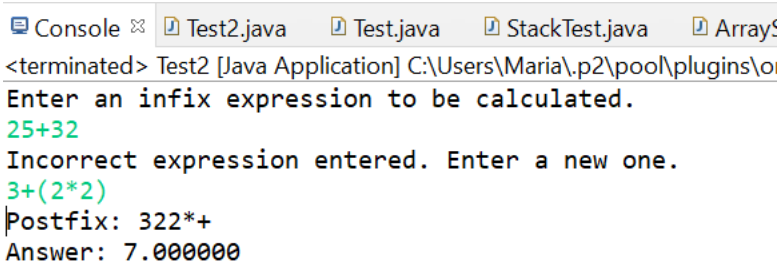
```
                    //pop the rest of the stack when all characters have been run
through the code
             }

       System.out.printf("Postfix: %s \n", postfix);

       //now evaluate answer

       for(int i=0; i < postfix.length(); i++) {
                if(Character.isDigit(postfix.charAt(i))){
                     stack.push(postfix.charAt(i));
                     //if an integer, push to stack
                }
                else {
                     String a = (stack.pop()).toString();
                     String b = (stack.pop()).toString();
                     floatNum1=Float.parseFloat(a);
                     floatNum2=Float.parseFloat(b);
                     //pops the top 2 no.s from the stack and converts to float

                     Evaluate(postfix.charAt(i));
                     //sends the operand at i through Evaluate, which takes
floatNum1 and floatNum2,
                     //operates on them and saves the answer as ans
                     stack.push(ans);
                     //ans is pushed to the stack so it can be used in the next
sum if needed

                }
             }

       System.out.printf("Answer: %f \n", (float)stack.top());
              scan.close(); //scanner closed
              System.exit(0);
       }

}
```

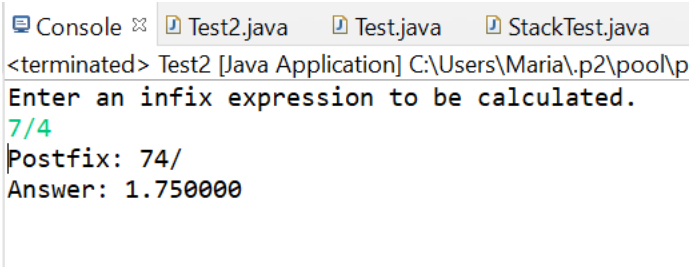
Testing:

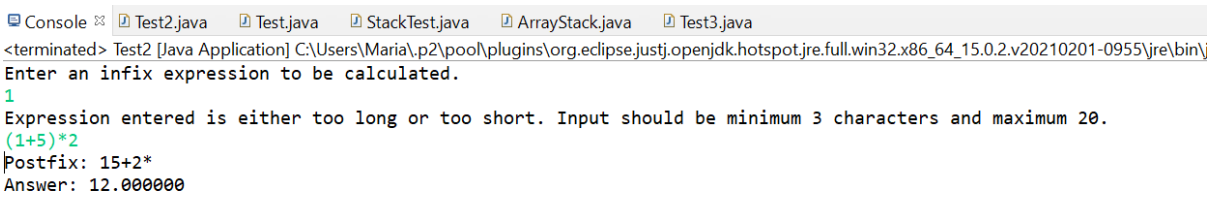

Console
*Test2.java
Test.java
StackTest.java
ArrayStack.java
Test3.java
<terminated> Test2 [Java Application] C:\Users\Maria\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-09
Enter an infix expression to be calculated.
abc
Incorrect character entered. Enter only positive integers and operands.
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
Expression entered is either too long or too short. Input should be minimum 3 characters and maximum 20.
(1+2)*3
Postfix: 12+3*
Answer: 9.000000


Console
Test2.java
Test.java
StackTest.java
ArrayStack.java
<terminated> Test2 [Java Application] C:\Users\Maria\.p2\pool\plugins\org.eclipse.ju
Enter an infix expression to be calculated.
7-(6/3)
Postfix: 763/-
Answer: 5.000000


Console
Test2.java
Test.java
StackTest.java
ArrayStack.java
<terminated> Test2 [Java Application] C:\Users\Maria\.p2\pool\plugins\org.eclipse.just
Enter an infix expression to be calculated.
17-2
Incorrect expression entered. Enter a new one.
1+2
Postfix: 12+
Answer: 3.000000


Console
Test2.java
Test.java
StackTest.java
<terminated> Test2 [Java Application] C:\Users\Maria\.p2\pool\plugins\o
Enter an infix expression to be calculated.
25+32
Incorrect expression entered. Enter a new one.
3+(2*2)
Postfix: 322*+
Answer: 7.000000


Console
Test2.java
Test.java
StackTest.java
<terminated> Test2 [Java Application] C:\Users\Maria\.p2\pool\p
Enter an infix expression to be calculated.
7/4
Postfix: 74/
Answer: 1.750000


Console
Test2.java
Test.java
StackTest.java
ArrayStack.java
Test3.java
<terminated> Test2 [Java Application] C:\Users\Maria\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\j
Enter an infix expression to be calculated.
1
Expression entered is either too long or too short. Input should be minimum 3 characters and maximum 20.
(1+5)*2
Postfix: 15+2*
Answer: 12.000000