HOMEWORK #4

Muhammed Yasir Fidan

161044056

Part1

Infix to Postfix Algorithm:

1-)Create and empty StringBuilder called postfix and empty Stack called operators.

2-)Read the infix string from left to right token by token.

3-)If reading token is a operand append it to postfix StringBuilder.

4-)If reading token is a operator check operators stack top element.

5-)If reading token preference bigger than stack top element or stack is empty or top element is a paranthesis then just push the reading token to the stack.

6-)else reading token preference lower or equal than stack top element pop stack elements and appent postfix until encounter a paranthesis.After that push the reading operator into the operators stack.

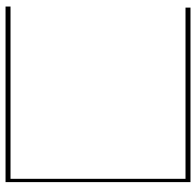
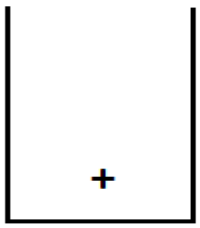
7-)If reading token is '(' push open paratheses to the operators stack.

8-)If reading token is ')' pop all operator from the stack until encounter a '(' operator.

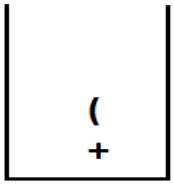
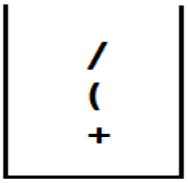
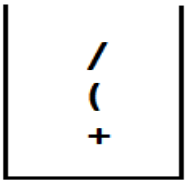
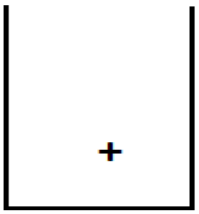
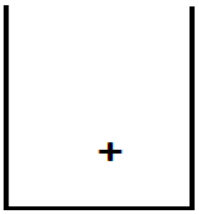
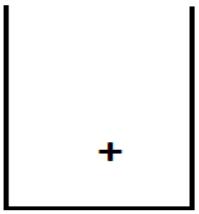
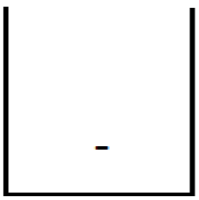
9-)Repeat these steps until infix string readed.

10-)If there is still some operators in operators stack, pop all operator and append postfix.

- **A+((B-C*D)/E)+F-G/H convert postfix**

Token	Action	Operators Stack	Postfix
A	Append A to postfix		A
+	Stack is empty,Push + to operators Stack		A

(	Push '(' to the Stack	(+	A
(	Push '(' to the Stack	((+	A
B	Append B to postfix	((+	AB
-	Push – to the Stack	- ((+	AB
C	Append C to postfix	- ((+	ABC
*****	* has bigger preference to – so push to the stack	* - ((+	ABC
D	Append D to postfix	* - ((+	ABCD

)	Pop stack until encounter a '(' token		ABCD*-
/	Push / to the Stack		ABCD*-
E	Append E to postfix		ABCD*-E
)	Pop Stack untill encounter a '(' token		ABCD*-E/
+	Pop stack because + has equal precedence with +.After pop push +		ABCD*-E/+
F	Append F to postfix		ABCD*-E/+F
-	Pop '+' because + and – has same precedence than push – into the stack		ABCD*-E/+F+

G	Append G to postfix	-	ABCD*-E/+F+G
/	Push / to the stack because it has highed precedence than -	/ -	ABCD*-E/+F+G
H	Append H to postfix	/ -	ABCD*-E/+F+GH
Infix is done.So pop all elements in the stack and append postfix			ABCD*-E/+F+GH/-

A+((B-C*D)/E)+F-G/H = ABCD*-E/+F+GH/- in postfix

- **! (A && ! ((B < C) || (C > D))) || (C < E) convert to postfix**

Token	Action	Operators Stack	Postfix
!	Push ! into the Stack	!	
(	Push (into the Stack	(!	
A	Append A Postfix	(!	A
&&	Push && into the Stack	&& (!	A
!	Push ! into the Stack	! && (!	A

(	Push (into the Stack	(! && (!	A
(	Push (into the Stack	((! && (!	A
B	Append B Postfix	((! && (!	B
<	Push < into the Stack	< ((! && (!	AB
C	Append C Postfix	< ((! && (!	ABC
)	Pop Stack until encouter (	(! && (!	ABC<
||	Push || into the Stack	|| (! && (!	ABC<
(	Push (into the Stack	(|| (! &&	ABC<

		(!	
C	Append C Postfix	(|| (! && (!	ABC<C
>	Push > into the Stack	> (|| (! && (!	ABC<C
D	Append D Postfix	> (|| (! && (!	ABC<CD
)	Pop Stack until encounter (	|| (! && (!	ABC<CD>
)	Pop Stack until encounter (	! && (!	ABC<CD>||
)	Pop Stack until encounter (	!	ABC<CD>||!&&
||	Pop ! after that push || because || prefence lower than !	||	ABC<CD>||!&&!
(	Push (into stack	(||	ABC<CD>||!&&!
C	Append C postfix	(||	ABC<CD>||!&&!C
<	Push < into the Stack	< (	ABC<CD>||!&&!C

		||	
E	Append E postfix	< (||	ABC<CD>||!&&!CE
)	Pop stack until encounter (	||	ABC<CD>||!&&!CE<
İnfix reading completed pop stack if its not empty	Pop || and make stack empty	Stack empty	ABC<CD>||!&&!CE<||

! (A && ! ((B < C) || (C > D))) || (C < E) postfix is A B C < C D > || ! && ! C E < ||

Infix to Prefix Algorithm:

1-)Reverse infix and make '(' to ')'.

2-)Use Profix algorithm.

3-)Reverse step 2 result.

- **A+((B-C*D)/E)+F-G/H convert prefix**

1-)reverse = H/G-F+(E/(D*C-B))+A

Now we must generete postfix version of step 1 result.

Token	Action	Operators Stack	Postfix
H	**Append H to postfix**	**Empty**	**H**
/	**Push / into the stack**	**/**	**H**
G	**Append H to postfix**	**/**	**HG**
-	**Pop / because – precedence lower after that push – into the stack**	**-**	**HG/**
F	**Append F to postfix**	**-**	**HG/F**
+	**Pop – from stack because – and + precedence equal than push +**	**+**	**HG/F-**
(	**Push (to stack**	**(** **+**	**HG/F-**
E	**Append E to postfix**	**(** **+**	**HG/F-E**
/	**Push / to stack**	**/** **(**	**HG/F-E**

		+	
(	Push (to stack	(/ (+	HG/F-E
D	Append D to postfix	(/ (+	HG/F-ED
*	Push * to stack	* (/ (+	HG/F-ED
C	Append C to postfix	* (/ (+	HG/F-EDC
-	Pop * fro stack because – precedence lower than * after that push – to stack	- (/ (+	HG/F-EDC*
B	Append B to postfix	- (/ (+	HG/F-EDC*B
)	Pop stack until encounter a '('	/ (+	HG/F-EDC*B-
)	Pop stack until encounter a '('	+	HG/F-EDC*B-/
+	Pop + from stack after that push +	+	HG/F-EDC*B-/+
A	Append A to postfix	+	HG/F-EDC*B-/+A
İnfix reading is completed.pop stack until its became empty	Pop + and make array empty	Empty	HG/F-EDC*B-/+A+

3-) Reverse HG/F-EDC*B-/+A+ = +A+/-B*CDE-F/GH

So; A+((B-C*D)/E)+F-G/H conversion of prefix is **+A+/-B*CDE-F/GH**

- **! (A && ! ((B < C) || (C > D))) || (C < E) convert to prefix**

1-) reverse = (E<C)|| (((D>C)|| (C<B))! && A)!

2-) now try to convert this postfix

Token	Action	Operators Stack	Postfix
(	Push (into empty stack	(	
E	Append E to postfix	(	E
<	Push < into stack	< (	E
C	Append C to postfix	< (	EC
)	Pop stack until encounter '(' token	Empty	EC<
||	Push || into stack	||	EC<
(	Push (into stack	(||	EC<
(	Push (into stack	((||	EC<
(	Push (into stack	(((||	EC<
D	Append D to postfix	(((||	EC<D
>	Push > into stack	>(((||	EC<D
C	Append C to postfix	>(((||	EC<DC

)	Pop stack until encounter ‘(‘ token	((||	EC<DC>
||	Push || into stack	|| ((||	EC<DC>
(	Push (into stack	(|| ((||	EC<DC>
C	Append C to postfix	(|| ((||	EC<DC>C
<	Push < into stack	< (|| ((||	EC<DC>C
B	Append B to postfix	< (|| ((||	EC<DC>CB
)	Pop stack until encounter ‘(’ token	|| ((||	EC<DC>CB<
)	Pop stack until encounter ‘(’ token	(||	EC<DC>CB<||
!	Push ! into stack	! (||	EC<DC>CB<||
&&	Pop ! token because its precedence bigger than &&.After that push && into stack	&& (||	EC<DC>CB<||!

A	Append A to postfix	&& (||	EC<DC>CB<||!A
)	Pop stack until encounter '(' token	||	EC<DC>CB<||!A&&
!	Push ! into stack	! ||	EC<DC>CB<||!A&&
Reading infix is completed.Make Stack empty	Pop Stack while its became empty	empty	EC<DC>CB<||!A&&!||

3-)Reverse EC<DC>CB<||!A&&!|| = **||!&&A!||<BC>CD<CE**

Evaluate Postfix Algorithm:

1-) Read given string

2-) If reading element is operand push it to stack

3-)If reading element is operator pop operands calculate and then push result again.

4-)In the end result will be in the stack.

Evaluate postfix ABCD*-E/+F+GH/-

Token	Action	Stack
A	Push A to Stack	A
B	Push B to Stack	B A
C	Push C to Stack	C B A
D	Push D to Stack	D C B A
*	Pop D and C from stack then push C*D	C*D B A
-	Pop C*D and B from Stack then push B-C*D	B-C*D A
E	Push E to Stack	E B-C*D A
/	Pop E and B-C*D then push (B-C*D)/E to Stack	(B-C*D)/E A

+	Pop (B-C*D)/E and A from Stack then push (B-C*D)/E+A	(B-C*D)/E+A
F	Push F to Stack	F (B-C*D)/E+A
+	Pop F and (B-C*D)/E+A from stack then push F+(B-C*D)/E+A	F+(B-C*D)/E+A
G	Push G to Stack	G F+(B-C*D)/E+A
H	Push H to Stack	H G F+(B-C*D)/E+A
/	Pop G and H then push G/H to stack	G/H F+(B-C*D)/E+A
-	Pop G/H and F+(B-C*D)/E+A Then push F+(B-C*D)/E+A-G/H	F+(B-C*D)/E+A-G/H

ABCD*-E/+F+GH/- postfix evaluation is **F+(B-C*D)/E+A-G/H**

Evaluate postfix ABC<CD>||!&&!CE<||

Token	Action	Stack
A	Push A to stack	A
B	Push B to Stack	B A
C	Push C to Stack	C B A
<	Pop C and B then push B<C	B<C A
C	Push C to Stack	C B<C A
D	Push D to Stack	D C B<C A
>	Pop D and C from Stack then push C>D	C>D B<C A
||	Pop C>D and B<C then push (B<C)||(C>D)	(B<C)||(C>D) A

!	Pop (B<C)||(C>D) then push !((B<C)||(C>D))	!((B<C)||(C>D)) A
&&	Pop !((B<C)||(C>D)) and A then push A&&(!((B<C)||(C>D)))	A&&(! (B<C)||(C>D)))
!	Pop A&&(!((B<C)||(C>D))) then push !(A&&(!((B<C)||(C>D)))	!(A&&!((B<C)||(C>D)))
C	Push C to Stack	C !(A&&!((B<C)||(C>D)))
E	Push E to Stack	E C !(A&&!((B<C)||(C>D)))
<	Pop E and C from stack then C<E to stack	C<E !(A&&!((B<C)||(C>D)))
||	Pop C<E and !(A&&(!((B<C)||(C>D)))) then push stack !(A&&(!((B<C)||(C>D))))||(C<E)	!(A&&!((B<C)||(C>D)))||(C<E)

ABC<CD>||!&&!CE<|| postfix evaluation is **!(A&&!((B<C)||(C>D)))||(C<E)**

Evaluate Prefix Algorithm

It is same as Evaluate profix only difference is we start to read string from the last element.

Evaluate prefix +A+/-B*CDE-F/GH

Token	Action	Stack
H	Push H to Stack	H
G	Push G to Stack	G H
/	Pop G and H from the Stack thne push G/H	G/H
F	Push F to Stack	F G/H
-	Pop F and H/G from the stack then push F-H/G	F-G/H
E	Push E to Stack	E F-G/H
D	Push D to Stack	D E F-G/H

C	Push C to Stack	C D E F-G/H
*	Pop C and D then push C*D	C*D E F-G/H
B	Push B to Stack	B C*D E F-G/H
-	Pop B and C*D then push B-(C*D)	(B-C*D) E F-G/H
/	Pop B-(C*D) and E then push ((B-C*D)/E)	((B-C*D)/E) F-G/H
+	Pop ((B-C*D)/E) and F-G/H Then push ((B-C*D)/E)+F-G/H	((B-C*D)/E)+F-G/H
A	Push A to Stack	A ((B-C*D)/E)+F-G/H
+	Pop A and ((B-C*D)/E)+F-G/H then push A+ ((B-C*D)/E)+F-G/H	A+ ((B-C*D)/E)+F-G/H

Evaluate prefix +A+/-B*CDE-F/GH is **A+ ((B-C*D)/E)+F-G/H**

Evaluate prefix ||!&&A!||<BC>CD<CE

Token	Action	Stack
E	Push E to Stack	E
C	Push C to Stack	C E
<	Pop C and E from stack then push C<E	C<E
D	Push D to Stack	D C<E
C	Push C to Stack	C D C<E
>	Pop D and C from stack then push C>D	C>D C<E
C	Push C to Stack	C

		C>D C<E
B	Push B to Stack	B C C>D C<E
<	Pop B and C then push B<C	B<C C>D C<E
||	Pop B<C and D>C then push (B<C)||(D>C)	(B<C)||(C>D) C<E
!	Pop (B<C)||(D>C) then push !((B<C)||(D>C))	!((B<C)||(C>D)) C<E
A	Push A to Stack	A !((B<C)||(C>D)) C<E
&&	Pop A abd !((B<C)||(D>C)) from stack than push (A&&!((B<C)||(D>C)))	(A&&!((B<C)||(C>D))) C<E
!	Pop (A&&!((B<C)||(D>C))) then push ! (A&&!((B<C)||(D>C)))	! (A&&!((B<C)||(C>D))) C<E
||	Pop ! (A&&!((B<C)||(D>C))) and C<E then push !(A&&!((B<C)||(D>C)))||(C<E)	!(A&&!((B<C)||(C>D)))||(C<E)

Evaluate prefix ||!&&A!||<BC>CD<CE is **!(A&&!((B<C)||(C>D)))||(C<E)**

Part-2:

In part2 I create myDeque class, this class extend AbstractCollection and implement Deque interface.Then I implement All unimplemented methods for my class.Also I create a Node class and Iterator class for myDeque class.My deque class hold 2 Node ref called head and tail also I hold 2 more node ref for Removed_node_list.When I remove a node from the list I add this node to the removed_node_list so after I add a node to my deque class first I look removed_node_list. If there is some node I add this node to my deque class so I didn't create a node. But if removed_node_list is empty so I create a node.I use all methods and iterator in main.

```
7   public static void main(String[] args) {
8
9       try{
10          Deque<Integer> x=new myDeque<>();
11
12          x.addFirst( e: 2);
13          x.addFirst( e: 4);
14          x.addFirst( e: 5);
15          x.addLast( e: 1);
16          x.offerFirst( e: 3);
17          x.offerLast( e: 11);
18          x.removeFirst();
19          x.removeFirst();
20          x.addFirst( e: 0);
21          x.addFirst( e: 3);
22          x.removeLast();
23          x.addFirst( e: 7);
24          x.pollFirst();
25          x.pollLast();
26          x.addFirst( e: 1);
27          x.addLast( e: 5);
28          x.removeLast();
29          x.offer( e: 8);
30          x.remove();
31          x.poll();
32          x.addLast( e: 0);
33          x.addFirst( e: 2);
34          x.removeFirstOccurrence( o: 0);
35          x.removeLastOccurrence( o: 2);
36          //Using my iterator with calling myDeque class iterator method.
37          Iterator<Integer> myIter=x.iterator();
```

```
"C:\Program Files\Java\jdk-13.0.2\bin
2
3
4
0
myDeque size: 4
first element: 2
Last element: 0
first element peekFist : 2
Last element peekLast: 0

Process finished with exit code 0
```

Here in line 10 I create my class.in line 12-15 I call addFirst method.That method add elements at the beginning.so my deque became 5-4-2 after that I call addLast, this method add element at the end of the deque.So my list became 5-4-2-1 ,after that I call offer first and offer last methods.This methods similer addFirst and addLast methods only difference is offerFirst and offerLast return a boolean.After these 2 method my deque became

3-5-4-2-1-11 Then I call removeFirst method 2 times.As you can imagine this method remove the first element of the deque.If there is no element its return false. So after 2 removeFirst it remove 3 and 5 so my list became 4-2-1-11 then again I call addlist with 0 and 3 input,my deque became 3-0-4-2-1-11 then I call removeLast method, again as you can imagine this method remove the last element of the deque so it remove 11 and list became 3-0-4-2-1 then in line 23 I call addfirst(7) so my deque became 7-3-0-4-2-1 then I call pollFirst and polLast methods. This methods similer to remove methods but only difference is that if deque size is 0 poll methods return null but remove methods throw a Exception.So pollfirst remove first element then pollLast remove last element and my list became 3-0-4-2 then I call addFirst(1) and addLast(5) so my queue became 1-3-0-4-2-5 and call removeLast so list became 1-3-0-4-2.Then I call offer(8) method ,this method work like offerFirst so list became 8-1-3-0-4-2. After that I call new method called remove and poll. I have removeFirst removeLast, pollFirst and pollLast methods but because of I imlepement deque class I had to implement remove and poll methods too . Remove method work similar like RemoveFirst and poll method work similer as pollFirst. So after these methods my deque became like

3-0-4-2 then I call addLast(0) and addFirst(2). My deque is 2-3-0-4-2-0 after these methods. Then I call removeFirstOccurence(0) this method start from the head and goes to the tail if its encounter a 0 remove it. So list became 2-3-4-2-0. Lastly I call removeLastOccurence(2) this method start from the tail and goes to the head it remove first encountered element. So list became 2-3-4-0

```
25            x.pollLast();
26            x.addFirst( e: 1);
27            x.addLast( e: 5);
28            x.removeLast();
29            x.offer( e: 8);
30            x.remove();
31            x.poll();
32            x.addLast( e: 0);
33            x.addFirst( e: 2);
34            x.removeFirstOccurrence( o: 0);
35            x.removeLastOccurrence( o: 2);
36            //Using my iterator with calling myDeque class iterator method.
37            Iterator<Integer> myIter=x.iterator();
38
39            while(myIter.hasNext())
40                System.out.println(myIter.next());
41
42            System.out.println("myDeque size: "+x.size());
43            System.out.println("first element: "+x.getFirst());
44            System.out.println("Last element: "+x.getLast());
45            System.out.println("first element peekFist : "+x.peekFirst());
46            System.out.println("Last element peekLast: "+x.peekLast());
47
48        }
49        catch (NoSuchElementException x){
50            System.out.println("Error Catched: No such Element");
51        }
52        catch (UnsupportedOperationException y){
53            System.out.println("Error Catched: You call a method which I dont implement");
54        }
55
```

```
"C:\Program Files\Java\jdk-13.0.2\bin
2
3
4
0
myDeque size: 4
first element: 2
Last element: 0
first element peekFist : 2
Last element peekLast: 0

Process finished with exit code 0
```

So after this methods my deque is 2-3-4-0. In line 37 I create I iterator with using iterator method. This is the iterator that I implement in myDeqeu class.With this iterator I travers list with hasNext() and next() methods and I print myDeque elements so it sprint 2-3-4-0.

Also I call size,getfirst(),getlast(), peekfirst() and peeklast() methods too.

All remove method add removed node to removed_nodes list.

```
import java.util.Deque;

import static org.junit.jupiter.api.Assertions.*;

class myDequeTest {

    @Test
    void addFirst() {
        Deque<Integer> x=new myDeque<>();
        x.addFirst( e: 1);
        x.addFirst( e: 2);
        x.addFirst( e: 3);
        assertEquals( expected: "3\n2\n1\n",x.toString());
    }

    @Test
    void addLast() {
        Deque<Integer> x=new myDeque<>();
        x.addLast( e: 1);
        x.addLast( e: 2);
        x.addLast( e: 3);
        assertEquals( expected: "1\n2\n3\n",x.toString());
    }

    @Test
    void offerFirst() {
        Deque<Integer> x=new myDeque<>();
        x.offerFirst( e: 1);
        x.offerFirst( e: 2);
        x.offerFirst( e: 3);
        assertEquals( expected: "3\n2\n1\n",x.toString());
    }
```

```
Test Results 134 ms
myDequeTest 134 ms
Tests passed: 24 of 24 tests – 134 ms
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" ...

Process finished with exit code 0
```

I use Junit5 test for testing my methods and all method pass the test cases.They work correctly.

Part-3:

- **Reverse String**

In reverse string method I pass a string argument and empty StringBuilder argument and recursivyly reverse my string argument and append this reversed string to StringBuilder argument.I use substring method for separete smaller strings to my string .With substring method I traverse string at the end to beginning when my substring last element encounter a ' '(space) character that means it is the end of a Word.I append this Word to my StringBuilder and print then clear StringBuilder.This steps goes to until S.substring length equals 0.When there is no substring(length=0) I print last Word and return recursive(this is my base case).

- **Elfish Word**

In Elfish Word method there are 4 argument one for my String to determine elfish or not,the other 3 of them for counter character e,f and l . In this method I use substring method for seperate string at the beginning.and check beginning letter is a l,f or e and increment their correspanding counter arguments.At the end my substring length is 0(this is my base case) I check the counters , if they are bigger than 1 it means my string is elfish,so I print "Elfish word" and return.

- **Selection Sort**

In selection sort method I have 5 argument.First argument for my integer array,2 of them my indexes and other 2 for array minimum value and this minimum value index. In this method Index argument hold a index and with minIndex value I traverse the list with increment minIndex +1 every recursive call and with this traverse I find minimum value and hold this minimum value in min argument and its index in holdMinIndex argument.After that I compare this min value with my index value if its smaller than my index I swap them.After that I increment my index +1 and again call my method recursivly and traverse again with call my method with minIndex+1 every time and so on.In the end when my Index equal to the array length-1 I print last array element and return(This is my base case).So I print all array element with sorted way.

- **Evaluate Postfix**

This method has 2 argument one for string that I will evaluate and the other argument is my Integer stack.In this method I call my recursive function with seperate my string.I do my Works with string first letter then with substring method I remove first letter and create a new string with the rest of the string.And do same things to this string first element and so on.If this first element is a digit i push it to my stack, if its a operator(+,-,*,/) than I pop 2 element from the stack and implement my operator with this 2 element.then I push the result into the stack again.In the end substring method only

return a string with length 1(my base case) and when its happen I pop last element into the stack and print it after that I return.

- **Evaluate Prefix**

This method has 2 agrument one for my prefix string that will evaluate and integer stack.

Like postfix method I use substring method for separete my string into the small pieces.But I remove last character with substring and do my work with this last character.With substring I remove last elements and return the other part of the string until my string length become 1(my base case).For string last elements I check if its a digit or not if its a digit I push it to stack, if its not I pop 2 element from the stack evaluate the operator with these 2 element and push the result back to stack and so on.After my length became 1 due to substring recursive calls I pop stack print this element and return.

- **Print 2D array elemets**

My Print array has 8 argument.First one for array,2 of them for array collumn and row,2 of them my x,y coordinant,2 of them a counter and last one is a way argument.With this way argument I traverse my 2D array in 4 way(left,right,up,down).For this 4 way there is 4 recursive call,I change x and y coordinants in this recursive call for traverse 2D array .For example if my way is right so I call my recursive method with y, x+1 every time so I go right way.When there is no right place it must go to down direction so I change way to down I call my recursive method with x,y+1 and it same with other directions too.When my counter is middle of the collumns or my counter2 is middle of the rows(my base case) it prints last element and return.