Homework 5

Djuro Radusinovic

171044095

Question 2

Problem-solution approach

Here I just extended the binary tree that was already implement by the book we are using. For making an expression tree I wrote a very short recursive code. When constructing a tree both prefix and postfix can be used and the readBinaryTree is used in the constructor. The main idea I used here was that the all leaf nodes of an expression tree are actually operands which was very useful here. For evaluating algorithm I just used again a very simple recursive method. When preorder traversal is used the prefix notation is built as a string and when postorder is used the postfix notation will be built.

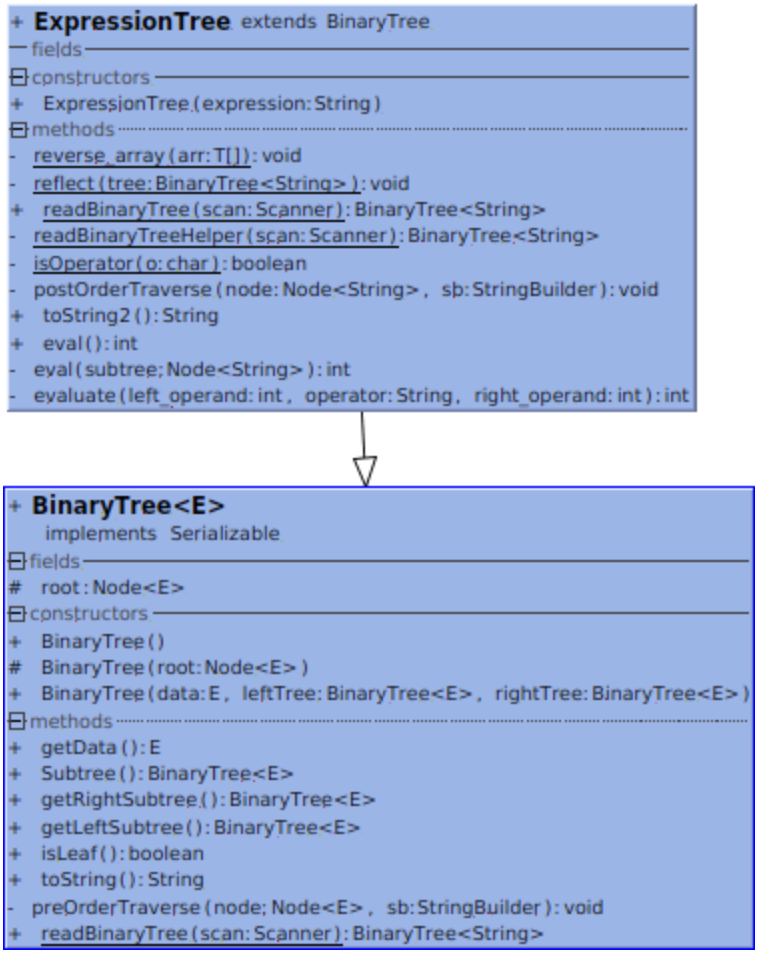
Tests cases

Test cases are run inside the virtual machine provided. Its actual results can be confirmed from the attached screenshots.

Test Scenario	Expected Results	Actual Results
Creating an expression tree from a prefix expression **+ + 10 * 5 15 20**	Tree should be created	As expected
Evaluating the created tree from the first case	Should evaluate to 105	As expected
Printing it using toString() should print its prefix notation	**+ + 10 * 5 15 20**	As expected
Printing it using toString2() should print its postfix notation	10 5 15 * + 20 +	As expected
Creating the same tree but now from the postfix notation 10 5 15 * + 20 +	Tree should be created	As expected
Evaluating the created tree from the postfix	Should evaluate to 105	As expected
Printing it using toString() should print its prefix notation	**+ + 10 * 5 15 20**	As expected
Printing it using toString2() should print its postfix notation	10 5 15 * + 20 +	As expected

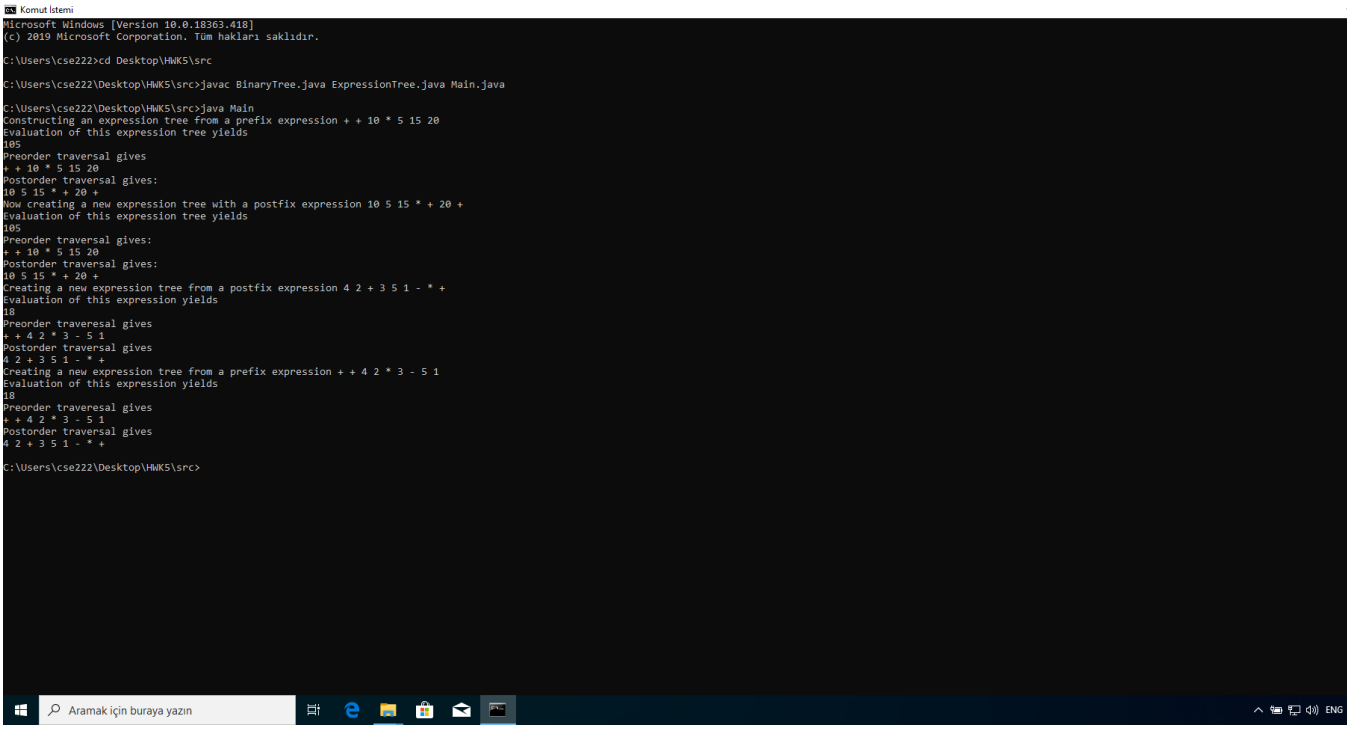
Creating a tree from postfix notation **4 2 + 3 5 1 - * +**	Tree should be created	As expected
Evaluating the created tree from the **4 2 + 3 5 1 - * +**	Should evaluate to 18	As expected
Printing it using toString() should print its prefix notation	**4 2 + 3 5 1 - * +**	As expected
Printing it using toString2() should print its postfix notation	**+ + 4 2 * 3 - 5 1**	As expected
Creating a tree from prefix notation **+ + 4 2 * 3 - 5 1**	Tree should be created	As expected
Evaluating the created tree from the **+ + 4 2 * 3 - 5 1**	Should evaluate to 18	As expected
Printing it using toString() should print its prefix notation	**4 2 + 3 5 1 - * +**	As expected
Printing it using toString2() should print its postfix notation	**+ + 4 2 * 3 - 5 1**	As expected

Class diagram of ExpressionTree and its helper classes



Running command and results

Output of the code executed in the virtual machine is provided in here


Komut İstemi
Microsoft Windows [Version 10.0.18363.418]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.
C:\Users\cse222>cd Desktop\HW5\src
C:\Users\cse222\Desktop\HW5\src>javac BinaryTree.java ExpressionTree.java Main.java
C:\Users\cse222\Desktop\HW5\src>java Main
Constructing an expression tree from a prefix expression + + 10 * 5 15 20
Evaluation of this expression tree yields
105
Preorder traversal gives
+ + 10 * 5 15 20
Postorder traversal gives:
10 5 15 * + 20 +
Now creating a new expression tree with a postfix expression 10 5 15 * + 20 +
Evaluation of this expression tree yields
105
Preorder traversal gives:
+ + 10 * 5 15 20
Postorder traversal gives:
10 5 15 * + 20 +
Creating a new expression tree from a postfix expression 4 2 + 3 5 1 - * +
Evaluation of this expression yields
18
Preorder traveresal gives
+ + 4 2 * 3 - 5 1
Postorder traversal gives
4 2 + 3 5 1 - * +
Creating a new expression tree from a prefix expression + + 4 2 * 3 - 5 1
Evaluation of this expression yields
18
Preorder traveresal gives
+ + 4 2 * 3 - 5 1
Postorder traversal gives
4 2 + 3 5 1 - * +
C:\Users\cse222\Desktop\HW5\src>
Aramak için buraya yazın
ENG