Homework Turnin

Account:	6G_06 (rgalanos@fcps.edu)
Section:	6G
Course:	TJHSST APCS 2016-17
Assignment:	07-03
Receipt ID:	44c3ccade4e95f1c7ba7f94c5cf0f2b3

Turnin Successful!

The following file(s) were received:

BXT_Driver.java (4591 bytes)

```
1. //name:     date:
2. import java.util.*;
3.
4. /******************
5. Represents a binary expression tree.
6. The BXT can build itself from a postorder expression.  It can
7. evaluate and print itself. It also prints an inorder string and a preorder string.
8. **********************/
9. class BXT
10. {
11.    private int count;
12.    private TreeNode root;
13.    public BXT()
14.    {
15.       count = 0;
16.       root = null;
17.    }
18.
19.
20.   /*  enter your instance methods here.  */
21.
22.    public void buildTree(String str)   //buildTree
23.    {
24.       StringTokenizer st = new StringTokenizer(str);
25.       Stack<TreeNode> stack = new Stack<TreeNode>();
26.       while(st.hasMoreTokens())
27.       {
28.          String s = st.nextToken();
29.          if("*/+-".indexOf(s)!=-1)
30.          {
31.             TreeNode temp = stack.pop();
32.             TreeNode t = new TreeNode(s, stack.pop(), temp);
33.             stack.push(t);
34.          }
35.          else
36.          {
37.             TreeNode t = new TreeNode(s);
38.             stack.push(t);
39.          }
40.       }
41.       root = stack.pop();
42.    }
43.
44.    public String display()   //display
45.    {
46.       return display(root, 0);
47.    }
48.    private String display(TreeNode t, int level)
49.    {
50.       String toRet = "";
```

```
51.       if(t == null)
52.          return "";
53.       toRet += display(t.getRight(), level + 1);
54.       for(int k = 0; k < level; k++)
55.          toRet += "\t";
56.       toRet += t.getValue() + "\n";
57.       toRet += display(t.getLeft(), level + 1);
58.       return toRet;
59.    }
60.
61.    public String inorderTraverse()  //inorderTraverse
62.    {
63.       return inorderTraverse(root);
64.    }
65.    private  String inorderTraverse(TreeNode t)
66.    {
67.       String toReturn = "";
68.       if(t == null)
69.          return "";
70.       toReturn += inorderTraverse(t.getLeft());
71.       toReturn += t.getValue() + " ";
72.       toReturn += inorderTraverse(t.getRight());
73.       return toReturn;
74.    }
75.
76.    public String preorderTraverse()  //preorderTraverse
77.    {
78.       return preorderTraverse(root);
79.    }
80.    private String preorderTraverse(TreeNode t)
81.    {
82.       String toReturn = "";
83.       if(t == null)
84.          return "";
85.       toReturn += t.getValue() + " ";
86.       toReturn += preorderTraverse(t.getLeft());
87.       toReturn += preorderTraverse(t.getRight());
88.       return toReturn;
89.    }
90.
91.    public double evaluateTree()  //evaluateTree
92.    {
93.       return evaluateTree(root);
94.    }
95.    private double evaluateTree(TreeNode t)
96.    {
97.       if("+-*/".indexOf(t.getValue()+"")==-1)
98.          return Double.parseDouble(t.getValue()+"");
99.       else
100.      {
101.         if(t.getValue().equals("+"))
102.            return evaluateTree(t.getLeft()) + evaluateTree(t.getRight());
103.         else if(t.getValue().equals("-"))
104.            return evaluateTree(t.getLeft()) - evaluateTree(t.getRight());
105.         else if(t.getValue().equals("*"))
106.            return evaluateTree(t.getLeft()) * evaluateTree(t.getRight());
107.         else if(t.getValue().equals("/"))
108.            return evaluateTree(t.getLeft()) / evaluateTree(t.getRight());
109.      }
110.      return -1l;
111.   }
112. }
113. /******************
114. Driver for a binary expression tree class.
115. Input: a postfix string, each token separated by a space.
116. ********************/
117. public class BXT_Driver
118. {
119.    public static void main(String[] args)
120.    {
121.       ArrayList<String> postExp = new ArrayList<String>();
122.       postExp.add("14 -5 / ");
123.       postExp.add("20 3 -4 + * ");
124.       postExp.add("2 3 + 5 / 4 5 - *");
125.
126.       for( String postfix : postExp )
127.       {
128.          System.out.println("Postfix Exp: "  + postfix);
129.          BXT tree = new BXT();
130.          tree.buildTree( postfix );
131.          System.out.println("BXT: ");
```

```
132.          System.out.println( tree.display() );
133.          System.out.print("Infix order:  ");
134.          System.out.println( tree.inorderTraverse() );
135.          System.out.print("Prefix order:  ");
136.          System.out.println( tree.preorderTraverse() );
137.          System.out.print("Evaluates to " + tree.evaluateTree());
138.          System.out.println( "\n------------------------");
139.       }
140.    }
141. }
142.
143. /**************************************
144.  Postfix Exp: 14 -5 /
145.  BXT:
146.    -5
147.  /
148.    14
149.  Infix order:  14 / -5
150.  Prefix order:  / 14 -5
151.  Evaluates to -2.8
152.  ------------------------
153.  Postfix Exp: 20 3 -4 + *
154.  BXT:
155.       -4
156.    +
157.       3
158.  *
159.    20
160.  Infix order:  20 * 3 + -4
161.  Prefix order:  * 20 + 3 -4
162.  Evaluates to -20.0
163.  ------------------------
164.  Postfix Exp: 2 3 + 5 / 4 5 - *
165.  BXT:
166.       5
167.    -
168.       4
169.  *
170.       5
171.    /
172.          3
173.       +
174.          2
175.  Infix order:  2 + 3 / 5 * 4 - 5
176.  Prefix order:  * / + 2 3 5 - 4 5
177.  Evaluates to -1.0
178.  ------------------------
179.  */
```