TreeNode.java

1 /*****
2 * TreeNode is the abstract parent class for a parse tree node. It contains an
3 * integer ID data field that is common to all types of node. The ID defines
4 * what type of tree node this is, e.g., an IF node, a PLUS, etc. The ID
5 * values are those defined for symbols in <a href="sym.html">sym.java</a>.
6 */
7 public abstract class TreeNode {
8     /**
9      * Construct a tree node with id = 0. This is used, e.g., for nodes in a
10      * list, that don't need individual id's.
11      */
12      public TreeNode() {
13          this.id = 0;
14      }
15
16      /**
17      * Construct a tree node with the given id.
18      */
19      public TreeNode(int id) {
20          this.id = id;
21      }
22
23      /**
24      * Output the String representation of a pre-order tree traversal. The
25      * value of each node is written on a separate line, with subtree nodes
26      * indented two spaces per each level of depth, starting at depth 0 for the
27      * root.
28      */
29      public abstract String toString();
30
31      /**
32      * This is the recursive work-doer for toString. See its definition in
33      * extending classes for details.
34      */
35      public abstract String toString(int level);
36
37      /**
38      * See the documentation for each of these extending classes for further
39      * detail.
40      */
41      
42      public abstract class TreeNode {
43          
44          /**
45          * Construct a tree node with id = 0. This is used, e.g., for nodes in a
46          * list, that don't need individual id's.
47          */
48          public TreeNode() {
49              this.id = 0;
50          }
51
52          /**
53          * Construct a tree node with the given id.
54          */
55          public TreeNode(int id) {
56              this.id = id;
57          }
58
59          /**
60          * Print a readable string value for a numeric-valued tree ID.
61          */
62          public String symPrint(int id) {
63              switch (id) {
64                  case sym.SEMI: return ";
65                  case sym.PLUS: return "+
66                  case sym.MINUS: return "-
67                  case sym.TIMES: return "*
68                  case sym.DIVIDE: return "/
69              }
70              return "
71          }
72
73          /**
74          * The ID of this node. Yea, it's public. Take that, you pain-in-the-xxx
75          * software engineers. */
76          public int id;
77          
78          /**
79          * For example, the following tree
80          */
81          <p>
82              <img src="images/expr-tree.gif">
83          </p>
84          * looks like this from TreeNode.toString
85          <pre>
86              +
87              a
88              *
89              b
90              c
91          </pre>
92          * The implementation of toString() uses an int-valued overload to perform
93          * recursive traversal, passing an incrementing level value to successive
94          * recursive invocations. See the definitions of toString(int) in each
95          * TreeNode extension for further details.
96          */
97          public String toString() {
98              return toString(0);
99          }
100      }
101  
102  /**
103  * This is the recursive work-doer for toString. See its definition in
104  * extending classes for details.
105  */
106  public abstract String toString(int level);