

```

1  /****
2  *
3  * TreeNode is the abstract parent class for a parse tree node. It contains an
4  * integer ID data field that is common to all types of node. The ID defines
5  * what type of tree node this is, e.g., an IF node, a PLUS, etc. The ID
6  * values are those defined for symbols in <a href="sym.html">sym.java</a>.
7  *
8  * Extensions of TreeNode add additional data fields to hold information
9  * necessary for a particular node type. The TreeNode extensions are the
10 * following:
11 *
12 *   <a href="TreeNode1.html">TreeNode1</a> -- a node with one subtree
13 *   reference, used to define unary expressions, or other unary
14 *   constructs, such as a single declaration
15 *
16 *   <a href="TreeNode2.html">TreeNode2</a> -- a node with two subtree
17 *   references, used to define binary expressions, or other binary
18 *   constructs, such as an assignment statement
19 *
20 *   <a href="TreeNode3.html">TreeNode3</a> -- a node with three subtree
21 *   references, used to define trinary expressions, or other
22 *   trinary constructs, such as an if-then-else statement
23 *
24 *   <a href="TreeNodeList.html">TreeNodeList</a> -- a node with an
25 *   indefinite number of subtree references, used to define node
26 *   lists of any form, or equivalently, n-ary constructs
27 *
28 *   <a href="LeafNode.html">LeafNode</a> -- a leaf node with value
29 *   information, but no subtree references
30 *
31 *   <a href="TypeNode.html">TypeNode</a> -- a node with specific structure
32 *   for defining data types
33 *
34 * See the documentation for each of these extending classes for further
35 * detail.
36 *
37 */
38 public abstract class TreeNode {
39
40     /**
41     * Construct a tree node with id = 0. This is used, e.g., for nodes in a
42     * list, that don't need individual id's.
43     */
44     public TreeNode() {
45         this.id = 0;
46     }
47
48     /**
49     * Construct a tree node with the given id.
50     */
51     public TreeNode(int id) {
52         this.id = id;
53     }
54
55     /**
56     * Output the String representation of a pre-order tree traversal. The
57
58     * value of each node is written on a separate line, with subtree nodes
59     * indented two spaces per each level of depth, starting at depth 0 for the
60     * root.
61     *
62     * For example, the following tree
63     *
64     *   <img src= "images/expr-tree.gif">
65     *
66     * looks like this from TreeNode.toString
67     *
68     * +
69     * a
70     * *
71     * b
72     * c
73     *
74     * The implementation of toString() uses an int-valued overload to perform
75     * recursive traversal, passing an incrementing level value to successive
76     * recursive invocations. See the definitions of toString(int) in each
77     * TreeNode extension for further details.
78     */
79     public String toString() {
80         return toString(0);
81     }
82
83     /**
84     * This is the recursive work-doer for toString. See its definition in
85     * extending classes for details.
86     */
87     public abstract String toString(int level);
88
89     /**
90     * Print a readable string value for a numeric-valued tree ID.
91     */
92     public String symPrint(int id) {
93         switch (id) {
94             case sym.SEMI: return ";";
95             case sym.PLUS: return "+";
96             case sym.MINUS: return "-";
97             case sym.TIMES: return "*";
98             case sym.DIVIDE: return "/";
99         }
100        return "";
101    }
102
103    /** The ID of this node. Yea, it's public. Take that, you pain-in-the-xxx
104    * software engineers. */
105    public int id;
106
107 }

```