

```

1  /****
2  *
3  * TypeNode is a specialized extension of TreeNode4 intended for use in tree
4  * evaluation contexts where the node is known to be a type. The point of this
5  * is to allow users of a TypeNode value to assume specific properties about
6  * the node, without having to cast more generic TreeNodes in various ways.
7  *
8  * One specific property of a TypeNode is that its ID should be one of a fixed
9  * set of values that are legal for identifying types. These ID values can
10 * vary among different languages, but should be limited in scope, and have a
11 * specific meaning in the context of a TypeNode, even if they have another
12 * meaning in the context of some other type of TreeNode.
13 *
14 * Another known property of a TypeNode is that it has four TreeNode children,
15 * zero or more of which can be used to hold data for different types of node.
16 * For example, built-in atomic types typically use none of the children,
17 * relying on the ID to uniquely identify the type. As another example, a
18 * composite array type will typically use two children -- one for the base
19 * type of the array, the other for the dimensions.
20 *
21 * A final specialized component of TypeNode is a data field of type
22 * SymbolTable. This is used for types that need a symbol table reference,
23 * such as struct, record, and class types.
24 */
25
26 public class TypeNode extends TreeNode4 {
27
28     /**
29      * Construct this with the given id and null children.
30      */
31     public TypeNode(int id) {
32         super(id, null, null, null, null);
33     }
34
35     /**
36      * Construct this with the given id and given single child.
37      */
38     public TypeNode(int id, TreeNode child1) {
39         super(id, child1, null, null, null);
40     }
41
42     /**
43      * Construct this with the given id and given two children.
44      */
45     public TypeNode(int id, TreeNode child1, TreeNode child2) {
46         super(id, child1, child2, null, null);
47     }
48
49     /**
50      * Construct this with the given id and given three children.
51      */
52     public TypeNode(int id, TreeNode child1, TreeNode child2,
53                     TreeNode child3) {
54         super(id, child1, child2, child3, null);
55     }
56
57     /**
58      * Construct this with the given id and given four children.
59      */
60     public TypeNode(int id, TreeNode child1, TreeNode child2,
61                     TreeNode child3, TreeNode child4) {
62         super(id, child1, child2, child3, child4);
63     }
64
65     /**
66      * A la the other constructor, but with line and column numbers.
67      */
68     public TypeNode(int id, int line, int column) {
69         super(id, null, null, null, null, line, column);
70     }
71
72     /**
73      * A la the other constructor, but with line and column numbers.
74      */
75     public TypeNode(int id, TreeNode child, int line, int column) {
76         super(id, child, null, null, null, line, column);
77     }
78
79     /**
80      * A la the other constructor, but with line and column numbers.
81      */
82     public TypeNode(int id, TreeNode child1, TreeNode child2, int line,
83                     int column) {
84         super(id, child1, child2, null, null, line, column);
85     }
86
87     /**
88      * A la the other constructor, but with line and column numbers.
89      */
90     public TypeNode(int id, TreeNode child1, TreeNode child2,
91                     TreeNode child3, int line, int column) {
92         super(id, child1, child2, child3, null, line, column);
93     }
94
95     /**
96      * A la the other constructor, but with line and column numbers.
97      */
98     public TypeNode(int id, TreeNode child1, TreeNode child2,
99                     TreeNode child3, TreeNode child4, int line, int column) {
100        super(id, child1, child2, child3, child4, line, column);
101    }
102
103    /**
104     * Return the String representation of this subtree, which is the String
105     * value of its ID, followed on the next zero to four indented lines by the
106     * recursive toString of its four children. Null children are not printed
107     * at all. See the documentation for <a href= "TreeNode.html#toString()">
108     * TreeNode.toString() </a> for a general description the way trees are
109     * represented as strings.
110     */
111    public String toString(int level) {
112        String indent = "";

```

```
113     for (int i = 0; i < level; i++) {
114         indent += " ";
115     }
116     return symPrint(id) + toStringLineAndColumn(" ") +
117         (child1 == null ? "" : ("\n" + indent + " " +
118         child1.toString(level+1))) +
119         (child2 == null ? "" : ("\n" + indent + " " +
120         child2.toString(level+1))) +
121         (child3 == null ? "" : ("\n" + indent + " " +
122         child3.toString(level+1))) +
123         (child4 == null ? "" : ("\n" + indent + " " +
124         child4.toString(level+1)));
125 }
126
127 /** Reference to a symbol table, for struct, record, and class types. */
128 public SymbolTable symtab;
129
130 }
```