```
1  /****
2   *
3   * TreeNode2 extends TreeNode by adding two child components, which are
4   * references other TreeNodes.  Hence, TreeNode2 is used to represent binary
5   * syntactic constructs in a parse tree.
6   *
7   */
8  public class TreeNode2 extends TreeNode {
9
10     /**
11      * Construct this with the given id and child TreeNode references.  Set the
12      * line and column fields to -1 to indicate that they have not been
13      * explicitly initialized.
14      */
15     public TreeNode2(int id, TreeNode child1, TreeNode child2) {
16         super(id);
17         this.child1 = child1;
18         this.child2 = child2;
19     }
20
21     /**
22      * A la the other constructor, but with line and column numbers.
23      */
24     public TreeNode2(int id, TreeNode child1, TreeNode child2, int line,
25             int column) {
26         super(id, line, column);
27         this.child1 = child1;
28         this.child2 = child2;
29     }
30
31     /**
32      * Return the String representation of this subtree, which is the String
33      * value of its ID, followed on the next two indented lines by the
34      * recursive toString of its two children.  See the documentation for <a
35      * href= "TreeNode.html#toString()"> TreeNode.toString() </a> for a general
36      * description the way trees are represented as strings.
37      */
38     public String toString(int level) {
39         String indent = "";
40         for (int i = 0; i < level; i++) {
41             indent += "  ";
42         }
43         return symPrint(id) + toStringLineAndColumn(" ") + "\n" +
44             indent + "  " +
45                 (child1 == null ? "null" : child1.toString(level+1)) + "\n" +
46             indent + "  " +
47                 (child2 == null ? "null" : child2.toString(level+1));
48     }
49
50     /** Reference to the left child of this node. */
51     public TreeNode child1;
52
53     /** Reference to the right child of this node. */
54     public TreeNode child2;
55
56 }
```