TreeNode.java

```java
public abstract class TreeNode {
  /*
   * TreeNode is the abstract parent class for a parse tree node. It contains an
   * integer ID data field that is common to all types of node. The ID defines
   * what type of tree node this is, e.g., an IF node, a PLUS, etc. The ID
   * values are those defined for symbols in <a href="sym.html">sym.java</a>.<p>
   * Extensions of TreeNode add additional data fields to hold information
   * necessary for a particular node type. The TreeNode extensions are the
   * following:<p>
   * <ul>
   *   <li> <a href="TreeNode1.html">TreeNode1</a> -- a node with one subtree
   *       reference, used to define unary expressions, or other unary
   *       constructs, such as a single declaration
   *   <p>
   *   <li> <a href="TreeNode2.html">TreeNode2</a> -- a node with two subtree
   *       references, used to define binary expressions, or other binary
   *       constructs, such an assignment statement
   *   <p>
   *   <li> <a href="TreeNode3.html">TreeNode3</a> -- a node with three subtree
   *       references, used to define trinary expressions, or other
   *       trinary constructs, such as an if-then-else statement
   *   <p>
   *   <li> <a href="TreeNode4.html">TreeNode4</a> -- a node with four subtree
   *       references, used to define quartinary constructs
   *   <p>
   *   <li> <a href="TreeNodeList.html">TreeNodeList</a> -- a node with an
   *       indefinite number of subtree references, used to define node
   *       lists of any form, or equivalently, n-ary constructs
   *   <p>
   *   <li> <a href="LeafNode.html">LeafNode</a> -- a leaf node with value
   *       information, but no subtree references
   *   </ul>
   * See the documentation for each of these extending classes for further
   * detail.<p>
   *
   */

  public abstract String toString();

  /**
   * Constructs a tree node with id = 0. This is used, e.g., for nodes in a
   * list, that don't need individual id's.
   */
  public TreeNode() {
    this.id = 0;
  }

  /**
   * Constructs a tree node with the given id.
   */
  public TreeNode(int id) {
    this.id = id;
  }

  /**
   * Output the String representation of a pre-order tree traversal. The
   * value of each node is written on a separate line, with subtree nodes
   * indented two spaces per each level of depth, starting at depth 0 for the
   * root.<p>
   * For example, the following tree
   * <p>
   * <img src="images/expr-tree.gif">
   * <p>
   * looks like this from TreeNode.toString
   * <pre>
   * +
   *  a
   *  *
   *  b
   *  *
   *  c<\/pre>
   * The implementation of toString() uses an int-valued overload to perform
   * recursive traversal, passing an incrementing level value to successive
   * recursive invocations. See the definitions of toString(int) in each
   * TreeNode extension for further details.<p>
   */
  public String toString() {
    return toString(0);
  }

  /**
   * This is the recursive work-doer for toString. See its definition in
   * extending classes for details.<p>
   */
  public abstract String toString(int level);

  /**
   * Print a readable string value for a numeric-valued tree ID. This method
   * uses the mapping defined in the symNames class.<p>
   */
  public static String symPrint(int id) {
    return symNames.map[id];
  }

  /**
   * The ID of this node. Yea, it's public. Take that, you pain-in-the-xxx
   * software engineers. */
  public int id;
}
```

Page 1

assignments/3/support-files

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>.<p>
6 * Extensions of TreeNode add additional data fields to hold information
7 * necessary for a particular node type. The TreeNode extensions are the
8 * following:<p>
9 * <ul>
10 *   <li> <a href="TreeNode1.html">TreeNode1</a> -- a node with one subtree
11 *       reference, used to define unary expressions, or other unary
12 *       constructs, such as a single declaration
13 *   <p>
14 *   <li> <a href="TreeNode2.html">TreeNode2</a> -- a node with two subtree
15 *       references, used to define binary expressions, or other binary
16 *       constructs, such an assignment statement
17 *   <p>
18 *   <li> <a href="TreeNode3.html">TreeNode3</a> -- a node with three subtree
19 *       references, used to define trinary expressions, or other
20 *       trinary constructs, such as an if-then-else statement
21 *   <p>
22 *   <li> <a href="TreeNode4.html">TreeNode4</a> -- a node with four subtree
23 *       references, used to define quartinary constructs
24 *   <p>
25 *   <li> <a href="TreeNodeList.html">TreeNodeList</a> -- a node with an
26 *       indefinite number of subtree references, used to define node
27 *       lists of any form, or equivalently, n-ary constructs
28 *   <p>
29 *   <li> <a href="LeafNode.html">LeafNode</a> -- a leaf node with value
30 *       information, but no subtree references
31 *   </ul>
32 * See the documentation for each of these extending classes for further
33 * detail.<p>
34 * */
35
36 public abstract class TreeNode {
37  /**
38   * Construct a tree node with id = 0. This is used, e.g., for nodes in a
39   * list, that don't need individual id's.<p>
40   */
41  public TreeNode() {
42    this.id = 0;
43  }
44
45  /**
46   * Construct a tree node with the given id.<p>
47   */
48  public TreeNode(int id) {
49    this.id = id;
50  }
51
52  /**
53   * Output the String representation of a pre-order tree traversal. The
54   * value of each node is written on a separate line, with subtree nodes
55   * indented two spaces per each level of depth, starting at depth 0 for the
56   * root.<p>
57   * For example, the following tree
58   * <p>
59   * <img src="images/expr-tree.gif">
60   * <p>
61   * looks like this from TreeNode.toString
62   * <pre>
63   * +
64   *  a
65   *  *
66   *  b
67   *  *
68   *  c<\/pre>
69   * The implementation of toString() uses an int-valued overload to perform
70   * recursive traversal, passing an incrementing level value to successive
71   * recursive invocations. See the definitions of toString(int) in each
72   * TreeNode extension for further details.<p>
73   */
74  public String toString() {
75    return toString(0);
76  }
77
78  /**
79   * This is the recursive work-doer for toString. See its definition in
80   * extending classes for details.<p>
81   */
82  public abstract String toString(int level);
83
84  /**
85   * Print a readable string value for a numeric-valued tree ID. This method
86   * uses the mapping defined in the symNames class.<p>
87   */
88  public static String symPrint(int id) {
89    return symNames.map[id];
90  }
91
92  /**
93   * The ID of this node. Yea, it's public. Take that, you pain-in-the-xxx
94   * software engineers. */
95  public int id;