```
  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   *                                                     <p>
  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   *                                               <ul><li>
 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   *                                               <p><li>
 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 an assignment statement
 19   *                                               <p><li>
 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   *                                               <p><li>
 24   *     <a href="TreeNode4.html">TreeNode4</a> -- a node with four subtree
 25   *             references, used to define quartinary constructs
 26   *                                               <p><li>
 27   *     <a href="TreeNodeList.html">TreeNodeList</a> -- a node with an
 28   *             indefinite number of subtree references, used to define node
 29   *             lists of any form, or equivalently, n-ary constructs
 30   *                                               <p><li>
 31   *     <a href="LeafNode.html">LeafNode</a> -- a leaf node with value
 32   *             information, but no subtree references
 33   *                                               <p><li>
 34   *     <a href="LeafNode.html">LeafNode</a> -- an extension of TreeNode4
 35   *             specialized to represent type trees
 36   *                                               </ul><p>
 37   * See the documentation for each of these extending classes for further
 38   * detail.
 39   *
 40   */
 41  public abstract class TreeNode {
 42
 43      /**
 44       * Construct a tree node with id = 0.  This is used, e.g., for nodes in a
 45       * list, that don't need individual id's.
 46       */
 47      public TreeNode() {
 48          this.id = 0;
 49      }
 50
 51      /**
 52       * Construct a tree node with the given id.  Set the line and column fields
 53       * to -1 to indicate that they have not been explicitly initialized.n
 54       */
 55      public TreeNode(int id) {
 56          this.id = id;
 57          line = -1;
 58          column = -1;
 59      }
 60
 61      /**
 62       * Construct a tree node with the given id, with the given line and column
 63       * source file character positions.
 64       */
 65      public TreeNode(int id, int line, int column) {
 66          this.id = id;
 67          this.line = line;
 68          this.column = column;
 69      }
 70
 71      /**
 72       * Output the String representation of a pre-order tree traversal.  The
 73       * value of each node is written on a separate line, with subtree nodes
 74       * indented two spaces per each level of depth, starting at depth 0 for the
 75       * root.
 76       *
 77       * For example, the following tree
 78       *
 79       *     <img src= "images/expr-tree.gif">
 80       *                                                         <p>
 81       * looks like this from TreeNode.toString
 82       *                                                         <pre>
 83       * +
 84       *   a
 85       *   *
 86       *     b
 87       *     c
 88       *                                                         </pre>
 89       * The implementation of toString() uses an int-valued overload to perform
 90       * recursive traversal, passing an incrementing level value to successive
 91       * recursive invocations.  See the definitions of toString(int) in each
 92       * TreeNode extension for further details.
 93       */
 94      public String toString() {
 95          return toString(0);
 96      }
 97
 98      /**
 99       * This is the recursive work-doer for toString.  See its definition in
100       * extending classes for details.
101       */
102      public abstract String toString(int level);
103
104      /**
105       * Common method for subclasses' toStrings to append line number and column
106       * positions, if they have been explicitly set for this node.
107       */
108      public String toStringLineAndColumn(String indent) {
109          return
110              (line >= 0) ?
111                  (indent + " (at line " +
112                  Integer.toString(line + 1 ) + ", col " +
```

```
113                  Integer.toString(column)) + ")"
114             : "";
115     }
116
117     /**
118      * Print a readable string value for a numeric-valued tree ID.  This method
119      * uses the mapping defined in the symNames class.
120      */
121     public static String symPrint(int id) {
122         return symNames.map[id];
123     }
124
125     /** The ID of this node.  Yea, it's public.  Take that, you pain-in-the-xxx
126      * software engineers. */
127     public int id;
128
129     /** The line number position of this node in the input file. */
130     int line;
131
132     /** The column position of this node in the input file, relative to the
133      * line. */
134     int column;
135
136 }
```