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>.

Extensions of TreeNode add additional data fields to hold information necessary for a particular node type. The TreeNode extensions are the following:

- <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
- <a href="TreeNode2.html">TreeNode2</a> -- a node with two subtree references, used to define binary expressions, or other binary constructs, such as an if-then-else statement
- <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
- <a href="TreeNode4.html">TreeNode4</a> -- a node with four subtree references, used to define quartinary constructs
- <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
- <a href="LeafNode.html">LeafNode</a> -- a leaf node with value information, but no subtree references
- <a href="LeafNode.html">LeafNode</a> -- an extension of TreeNode4 specialized to represent type trees

See the documentation for each of these extending classes for further detail.

public abstract class TreeNode {

    /*
     * Construct 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;
    }

    /*
     * Construct a tree node with the given id, with the given line and column positions.
     */
    public TreeNode(int id, int line, int column) {
        this.id = id;
        this.line = line;
        this.column = column;
    }

    /* 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.
     */
    public String toString() {
        return toString(0);
    }

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

    /*
     * Common method for subclasses’ toStrings to append line number and column positions, if they have been explicitly set for this node.
     */
    public String toStringLineAndColumn(String indent) {
        return toString(indent + "  " + (line >= 0 ? Integer.toString(line + 1) + "", col + " + ")
    }
}
** TreeNode.java **

```java
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    }
```