

```

1  /****
2  *
3  * A SymbolTableEntry holds semantic information for a symbol declared in a
4  * program. The information is the string name of the symbol and its declared
5  * data type.
6  *
7  * SymbolTableEntry is an abstract class, with extensions for variable data
8  * entries and functional entries. These are defined, respectively, in the <a
9  * href= VariableEntry.html> VariableEntry </a> and <a href=
10 * FunctionEntry.html> FunctionEntry </a> classes.
11 *
12 */
13 public abstract class SymbolTableEntry {
14
15     /**
16      * Construct this with null data fields.
17      */
18     public SymbolTableEntry() {
19     }
20
21     /**
22      * Construct this with the given data field values.
23      */
24     public SymbolTableEntry(String name, TypeNode type) {
25         this.name = name;
26         this.type = type;
27     }
28
29     /**
30      * Return toString(0).
31      */
32     public String toString() {
33         return toString(0);
34     }
35
36     /**
37      * Return toStringDeep(0).
38      */
39     public String toStringDeep() {
40         return toStringDeep(0);
41     }
42
43     /**
44      * Return the string representation of this' two fields, with other fields
45      * added by this' extensions. The fields are output on a single line,
46      * indented level * 2 blanks. Only the shallowest type string is output,
47      * namely just its root tree ID. The toStringDeep method outputs the full
48      * type field.
49      */
50     public String toString(int level) {
51         return doToString(level,
52             type != null ? TreeNode.symPrint(type.id) : "null");
53     }
54
55     /**
56      * Version of toString that dumps out the full type structure, not just its
57      * root ID.
58      */
59     public String toStringDeep(int level) {
60         return doToString(level, "\n" + type.toString(level + 30));
61     }
62
63     /**
64      * Common work doer for other toStrings.
65      */
66     protected String doToString(int level, String type) {
67         return indentString(level) + "Symbol: " + name + ", Type: " + type;
68     }
69
70     /**
71      * Convenience method for creating an indent string.
72      */
73     protected String indentString(int level) {
74         String indent = "";
75
76         for (int i = 0; i < level; i++) {
77             indent += " ";
78         }
79
80         return indent;
81     }
82
83     /** This symbol's name */
84     public String name;
85
86     /** Data type, represented as a parse tree. */
87     public TypeNode type;
88
89
90 }

```