

```

1  /****
2  *
3  * FunctionEntry extends SymbolTableEntry by adding data fields to support
4  * functions, procedures, and methods. These forms of functional construct are
5  * considered equivalent for the purposes of storing data in a symbol table.
6  *                                     <p>
7  * The public data fields of a FunctionEntry are a TreeNodeList of formal
8  * parameters, a TreeNode body, and a SymbolTable scope. The inherited type
9  * field is used to hold the return type of the function.
10 *                                     <p>
11 * The scope field holds a reference to the function's own local scope. All of
12 * the function's formal parameters and local variables are entered in this
13 * local table. In this way, the table defines a scope that belongs to the
14 * function, which is the standard semantics in block-structured programming
15 * languages.
16 *                                     <p>
17 * In programming languages that allow nested function definitions, a
18 * function's local scope may have further nested scopes. These are
19 * represented simply by having function entries in a parent function's scope
20 * table. Nested symbol tables are also used to represent anonymous inner
21 * scopes, such as nested declaration/statement blocks, in languages that all
22 * such constructs. See the documentation of the SymbolTable class for a
23 * large-grain picture and description of nested scope representation.
24 *                                     <p>
25 * A function's formal parameters are stored both in the formals list as well
26 * as being entered in the local symtab scope. The list is necessary when
27 * parameters need to be accessed in left-to-right declared order. The formals
28 * are also entered in the function's local scope, so they have a storage
29 * identity that is distinct to this scope.
30 *                                     <p>
31 * The body data field of a function is a reference to the entire parse tree
32 * for its executable body. This tree is used for back-end processing, which
33 * can include one or more of the following phases: type checking,
34 * interpretation, and/or code generation.
35 *
36 */
37
38 public class FunctionEntry extends SymbolTableEntry {
39
40     /**
41      * Construct this with null data fields.
42      */
43     public FunctionEntry() {
44     }
45
46     /**
47      * Construct this with the given data field values.
48      */
49     public FunctionEntry(String name, TreeNode type, TreeNodeList formals,
50                          TreeNode body, SymbolTable scope) {
51         super(name, type);
52         this.formals = formals;
53         this.body = body;
54         this.scope = scope;
55     }
56
57     /**
58      * Return the string rep of this.
59      */
60     public String toString(int level) {
61         return super.toString(level) + formalsString(level) +
62             scopeString(level);
63     }
64
65     /**
66      * Called by toString to stringify the list of formal parameter names.
67      */
68     protected String formalsString(int level) {
69         return formals == null ? "" : "\n" + indentString(level) +
70             " Formals: " + formals.toString(level + 5);
71     }
72
73     /**
74      * Called by toString to recursively stringify the scope, if non-null.
75      */
76     protected String scopeString(int level) {
77         return scope == null ? "" : "\n " + indentString(level) +
78             scope.toString(level);
79     }
80
81     /** Formal parameter list, in declared order. */
82     public TreeNodeList formals;
83
84     /** Function body, in the form of its raw parse tree. */
85     public TreeNode body;
86
87     /** Local scope for this function. */
88     public SymbolTable scope;
89
90
91 }

```