FunctionEntry.java

```java
public class FunctionEntry extends SymbolTableEntry {

    public FunctionEntry() {
    }

    public FunctionEntry(String name, TypeNode type, TreeNodeList formals, 
                          TreeNode body, SymbolTable scope) {
        super(name, type);
        this.formals = formals;
        this.body = body;
        this.scope = scope;
    }
```

---

1. FunctionEntry extends SymbolTableEntry by adding data fields to support
2. functions, procedures, and methods. These forms of functional construct are
3. considered equivalent for the purposes of storing data in a symbol table.
4. The public data fields of a FunctionEntry are a TreeNodeList of formal
5. parameters, a TreeNode body, and a SymbolTable scope. The inherited type
6. field is used to hold the return type of the function.
7. The scope field holds a reference to the function's own local scope. All of
8. the function's formal parameters and local variables are entered in this
9. local table. In this way, the table defines a scope that belongs to the
10. function, which is the standard semantics in block-structured programming
11. languages.
12. In programming languages that allow nested function definitions, a
13. function's local scope may have further nested scopes. These are
14. represented simply by having function entries in a parent function's scope
15. table. Nested symbol tables are also used to represent anonymous inner
16. scopes, such as nested declaration/statements blocks, in languages that all
17. such constructs. See the documentation of the SymbolTable class for a
18. large-grain picture and description of nested scope representation.
19. A function's formal parameters are stored both in the formals list as well
20. as being entered in the local symtab scope. The list is necessary when
21. parameters need to be accessed in left-to-right declared order. The formals
22. are also entered in the function's local scope, so they have a storage
23. identity that is distinct to this scope.
24. The body data field of a function is a reference to the entire parse tree
25. for its executable body. This tree is used for back-end processing, which
26. can include one or more of the following phases: type checking,
27. interpretation, and/or code generation.
28. ***/
29. public class FunctionEntry extends SymbolTableEntry {
30.    /**
31.     * Construct this with null data fields.
32.     */
33.    public FunctionEntry() {
34.    }
35.  }
36.  /**
37.  * Construct this with the given data field values. Initialize memorySize
38.  * to 0.
39.  */
40.  public FunctionEntry(String name, TypeNode type, TreeNodeList formals,
41.                        TreeNode body, SymbolTable scope) {
42.      super(name, type);
43.      this.formals = formals;
44.      this.body = body;
45.      this.scope = scope;
46.  }```