```
/****
 *
 * FunctionEntry extends SymbolTableEntry by adding data fields to support
 * functions, procedures, and methods.  These forms of functional construct are
 * considered equivalent for the purposes of storing data in a symbol table.
 *                                                                          <p>
 * The public data fields of a FunctionEntry are a TreeNodeList of formal
 * parameters, a TreeNode body, and a SymbolTable scope.  The inherited type
 * field is used to hold the return type of the function.
 *                                                                          <p>
 * The scope field holds a reference to the function's own local scope.  All of
 * the function's formal parameters and local variables are entered in this
 * local table.  In this way, the table defines a scope that belongs to the
 * function, which is the standard semantics in block-structured programming
 * languages.
 *                                                                          <p>
 * In programming languages that allow nested function definitions, a
 * function's local scope may have further nested scopes.  These are
 * represented simply by having function entries in a parent function's scope
 * table.  Nested symbol tables are also used to represent anonymous inner
 * scopes, such as nested declaration/statement blocks, in languages that all
 * such constructs.  See the documentation of the SymbolTable class for a
 * large-grain picture and description of nested scope representation.
 *                                                                          <p>
 * A function's formal parameters are stored both in the formals list as well
 * as being entered in the local symtab scope.  The list is necessary when
 * parameters need to be accessed in left-to-right declared order.  The formals
 * are also entered in the function's local scope, so they have a storage
 * identity that is distinct to this scope.
 *                                                                          <p>
 * The body data field of a function is a reference to the entire parse tree
 * for its executable body.  This tree is used for back-end processing, which
 * can include one or more of the following phases: type checking,
 * interpretation, and/or code generation.
 *
 */

public class FunctionEntry extends SymbolTableEntry {

    /**
     * Construct this with null data fields.
     */
    public FunctionEntry() {
    }

    /**
     * Construct this with the given data field values.  Initialize memorySize
     * to 0.
     */
    public FunctionEntry(String name, TypeNode type, TreeNodeList formals,
            TreeNode body, SymbolTable scope) {
        super(name, type);
        this.formals = formals;
        this.body = body;
        this.scope = scope;
    }

    /**
     * Return the string rep of this.
     */
    public String toString(int level) {
        return super.toString(level) + formalsString(level) +
            scopeString(level);
    }

    /**
     * Called by toString to stringify the list of formal parameter names.
     */
    protected String formalsString(int level) {
        return formals == null ? "" : "\n" + indentString(level) +
            " Formals: " + formals.toString(level + 5);
    }

    /**
     * Called by toString to recursively stringify the scope, if non-null.
     */
    protected String scopeString(int level) {
        return scope == null ? "" : "\n " + indentString(level) +
            scope.toString(level);
    }


    /** Formal parameter list, in declared order. */
    public TreeNodeList formals;

    /** Function body, in the form of its raw parse tree. */
    public TreeNode body;

    /** Local scope for this function. */
    public SymbolTable scope;

}
```