Software Design Documumentation

Benji Altman

April 27, 2017

Contents

1 Introduction

This is the software design documentation, it will describe on a high level view how each package works. The entire program is based off the grammer rules defined in this `pdf`.

2 General

The `general` package contains non-program specific classes, as well as the main class `Main`.

2.1 Main

The `Main` class is static and has only general static functions, as well as the `main` function, the program's entry point.

2.2 Unique Identifier Generator

Each instance of this class keeps track of a list of given IDs, and makes sure that it never gives the same ID twice. When geting a new ID you may make a request, if you make a request it will try and give you a string close to what you gave it. In the current implementation it will first attempt to return the string you requested, and if that is not available it will append something to that string to make it unique.

By default the UID considers all alphanumeric characters and underscore to be valid characters in an identifier. All strings where you don't request a specific string are prepended with an underscore so that it does not produce an identifier that begins with a number. In future iterations of this it should only produce strings that match a regex given at creation time.

2.3 Pair

This very basic class simply is a pair of two genericed objects.

3 Scanner

This package maintains classes that allow you to scan through a file and produce tokens.

3.1 TokenType

This is an enum that contains a value for all symbols, and then a special value for each type of literal as well as a special value for identifiers.

3.2 Token

This class represents tokens that are parsed from an input Pascal file. It contains the string that was parsed to produce this value, a type which is a `TokenType`, and a line and column number which is used in error messages.

3.3 Lookup

This class acts much like a static class. It can not be instantiated as it's default constructor is private, although it does contain a static instance of itself named `LOOKUP` that can be used. It allows you to lookup TokenTypes based on their strings, and allows you to do the reverse. The lookup may be done with the `get` function and reverse lookup is done with `teg`, although if you call `teg` on any special values[1], it will break the program.

[1]Special values are: any types representing literals, and identifier

3.4 Scanner

This class is computer generated from the file `Scanner.jflex` by the program `jflex-1.6.1.jar`. It turns text in a file into tokens as difined in the grammer. With the listed exceptions:

1. **id** is defined as **(letter | _) (letter | _ | digit)***
2. There are seperate `TokenType`'s for both integer and real literals, instead of just a num `TokenType`.
   (a) Integer literals are defined as [+-λ] **digit digit***
   (b) Real literals are defined as [+-λ] **digit digit* . digit digit***

4 Parser

The parser package maintains support for taking tokens from the scanner and populating symbol tables and generates a syntax tree.

5 Symbol Table

This package contains support for a symbol table specific to the needs of this program.

5.1 Scope

This