Ilias Kontonis
1115201700055

Compilers - Project 2 - Semantic Analysis

General

This project has two main tasks. The first one is the implementation of a class that would be used as the symbol table and the second one is the handling of semantic errors while vising the parse-tree. When a semantic error is detected the program throws exception with a message. All the test cases that were given were passed.

The symbol table class

This class consists other classes: the symbol table is an array of classEntry objects. Class classEntry extends Entry that is a class that holds (Name, Type) and has a Table object and classEntry object. The first one has the fields and methods of the class while the classEntry object points to the super class (if it exists). Table class has an array of Entry, and an array of MethodEntry. The last one has two arrays of Entry, one for the parameters and one for the fields, and a field Entry for the method's return type and name. And that is the data struct that makes the scope resolution work!

The visitors

For this implementation two visitors are made. The SymbolVisitor visits the parse-tree nodes that note a declaration and makes the respective insert to the table. The TypeVisitor visits the parse-tree nodes that note a statement. For this one the visit() methods return the type of the expression so at the upper level a type check is made. Also GJDepthFirst $<$ String, Void $>$ is changed to $<$ String, Boolean $>$ because at the visitor of Identifier in some cases the name is needed and in other the type. There is one check that is not in a visitor. Lets say that in the main class we have A a. First we must search the whole file for A and produce an error if it is not defined. So when the SymbolVisitor visits all the nodes and the symbol table is complete we can check that every type of identifier, that it is not int/int[]/etc, exists. About the offset, it is a string that the SymbolTable class holds. In addition, each classEntry has a method counter and variable counter. If class A extends class B, A's counters continue from B's, otherwise the start from zero.

Usage

Only the files that were changed or new ones are included in the tar.gz file. The developing was done on the existing files of minijava-example, so the Makefile and Main.java are almost the same. Use this Makefile to compile the new class files.