Assignment Description

In this programming assignment, you will implement the semantic analysis phases for IC. We expect you to build upon the code that you wrote in the previous programming assignments. You are required to implement the following:

Symbol Tables and Types. Design a hierarchy of symbol tables and a hierarchy of types. Your design should allow each AST node to access the symbol table corresponding to its current scope (e.g. class, method, or block scope), and each entry in the symbol table should have information about the type of the identifier stored in that entry. Any errors which occur during symbol table construction (such as multiply declared identifiers) are considered semantic errors. Your constructed symbol tables should be available to all remaining phases of the compiler. In the rest of your compiler, you will refer to program symbols (e.g., variables, methods, etc.) using references to their symbol table entries, not the actual text.

Semantic Checks. After you have constructed the AST and the symbol tables, your compiler will analyse the program and perform semantic checks. These semantic checks include: (1) scope rules (Section 10 in the IC specification); (2) type-checking rules (Section 15), including a check that the class hierarchy is a tree and checking correct overriding of instance methods in subclasses; (3) checking that the program contains a single `main` method with the correct signature, (4) that `break` and `continue` statements appear only inside loops, (5) that the `this` keyword is only used in instance methods, and (6) that the library class has the correct name (`Library`).

Bonus checks. You are **not** required to check that a local variable is used only after it has been initialized and that a method with a non-void return type returns a value on every control path. If you implement these checks you will get 5 points for each check (be sure to tell us in the documentation that you have implemented these checks).

Error Handling. When your compiler encounters an error it should report information about the error, such as the token and the line number where the error has occurred, or a message describing the violated semantic rule. It is not required to report more than one error; the execution may terminate after the first lexical, syntactic, or semantic error. You must start the error message for semantic errors by `semantic error at line XXX:` and a message of your choice describing the error.

Command line invocation: Your compiler must be invoked with a single file name as argument:

`java` *bin* ⟨*program-filename*⟩ `[options]`

With this command, the compiler will parse the input file, construct the AST and symbol tables, will perform the semantic checks, and will report any error it encounters. Your compiler must also support two command-line options to dump internal information about the AST and the symbol tables:

1. The "`-print-ast`" option: the compiler will print at `System.out` a textual description of the AST for the input file. This is the same print as the one in the previous assignment, with the addition of **type** and **symbol table information** for each AST node.
2. The "`-dump-symtab`" option: the compiler will print a textual description of the symbol tables and the (global) type table. Each entry in the symbol table will be printed on a separate line; the information for each entry should include the name of the identifier, its kind (variable, method, etc.), and its type. Different symbol tables will be separated by blank lines. You must indicate, for each symbol table, what are its children tables. When printing out the type table, print each type on a separate line. For each type include its name, its id, and the id of its superclass if it exists. Order should be primitives

then classes then arrays then methods. In a category the order is according to appearance. Each additional dimension of array is a new type (first one dimension arrays then 2 and so on).

We do not formally specify the output format for the symbol tables and type table. Instead, the web-site contains an input program and an output example. Your compiler's output should match the output for this example when it is activated with the arguments `example1.ic -Llibic.sig -dump-symtab`.

You are also expected to design a suite of tests (IC programs) that demonstrate the correctness of your semantic analysis and detection of various kinds of errors.

Package Structure: You will implement the new components of the compiler as sub-packages of the IC package. You will have a sub-package for each of the following: 1) the symbol tables, 2) the representation of types, and 3) the semantic checks.

What to turn in

You must send your code electronically to **compilation15a@gmail.com** due date, including a documentation write-up. **Please include only a zip file of the src directory in your submission, not the compiled class files or any other temporary files.**

As in any other large program, much of the value in a compiler is in how easily it can be maintained. For this reason, a high value will be placed here on both clarity and brevity – both in documentation and code. Make sure your code structure is well-explained in your write-up and in your javadoc documentation.

Turn in a document PA3-DOC.XXX (XXX can be one of txt/doc/pdf) with the following information:

- A brief, clear, and concise description of your code structure and testing strategy.
- A description of the class hierarchy in your src directory, brief descriptions of the major classes, any known bugs, and any other information that we might find useful when grading your assignment. Documented bugs will be graded more forgivingly than non-documented bugs.
- A high-level description of how you implement the semantic analysis, e.g., what is the order in which you build/check the symbol tables and types?

GOOD LUCK!