Project Part 3
Semantic processing hints

The semantic analysis part can be divided into two phases: the first involves gathering information, whereas the later involves using the information for semantic analysis. Note that two phases can occur at the same time; that is, as you are storing information in the symbol tables, you also perform semantic analysis. I have separated them here in order to emphasize the two different aspects of your project.

1 Data collection

The first phase will involve accumulating information about the entities (class, instance variables, methods, parameters) of your program. This information is usually available in the grammar rules that specify how entities can be declared. You can gather this information by annotating these rules with appropriate semantic rules and actions.

1.1 Kinds of information

You need to store information about classes, variables (instance, class, parameter, and local), and methods:

- Class-specific information: (i) class variables, (ii) instance variables, (iii) methods, and (iv) super class.
- Instance variable, local variable, or parameter specific information: (i) identifier, and (ii) type of identifier.
- Method: (i) identifier, (ii) its signature (or type), and (iii) result type, and (iv) body

1.2 Organization of information

You will need to store this information in symbol tables so that your compiler can access information about various entities when there is a need. Here is a suggested organization for symbol tables. You may choose a different one.

- Keep a global symbol table that will store identifier for each class that is defined in class, and a pointer to a symbol table storing information about the class.
- Keep a symbol table for each class. The symbol table will contain (i) entries for class and instance variables, and (ii) entries for methods

This will lead to a hierarchy of symbol tables. So as you move from one scope to another (for instance, from global scope to class scope to a method scope), you push the various symbol tables on the stack (project 1), use the symbol tables to access symbols, and pop off as you come out of a scope. Note that you should not delete the symbol table as you come out of the scope. The symbol tables remain attached with their corresponding entities.

2 Semantic analysis

Once the information has been gathered and organized in symbol tables, this information can now be used for performing semantic checking. I suggest that you implement the various semantic checking in the following order:

1. Create parse tree: Create parse tree using the technique that we discussed in the class. Also, I will suggest that you choose a few core constructs first. Create parse trees for them, and do some of the semantic analysis on them. Once they all work, keep adding constructs one at a time.

2. Type checking and type information distribution: We have seen much of it in the class. To summarize:

   (a) Find a way for representing different types. You may represent them using integers: Represent each primitive type by an integer, and every time you create a new reference type, check if it already exists, give error if it does else assign it a new type by giving it a new integer value.

   (b) Type identification: Traverse the parse tree to find all identifiers and constants.
   - Resolve all constants and their values; Also assign a type.
   - Resolve all names by searching for them in symbol tables and connecting the identifiers with their entries in symbol tables.
     You will need to be careful about resolving names such as a.b.c.d. Make sure that you search the symbol tables in the right order.

   (c) Type equivalence is merely comparing integer values here. (Because of name equivalence). You will also need to take care of type-subtyping relationship for classes and subclasses.

   (d) Look at each expression/statement rule, and propagate type information + perform type checking.

   (e) Note that you may not be able to complete your type checking before you have parsed the complete file. So in the first pass, build the symbol table. Do the type checking in the second pass by traversing the parse tree left-to-right during a depth search traversal.

3. Check usage of variables and methods:

   (a) For every identifier usage in the context of expressions, assignments, etc., check if this is a valid usage. You will have to search the symbol table to do so.

(b) For methods, you will need to construct a signature from method invocation expression and match this signature with method's signature.

Note: remember that methods and instance variables do not need a full name (and can be used with this and super). So you will need to ensure that you check a name in its valid scope.

4. Check valid usage of `this` and `super`.