Assignment 5: Data and Iteration Abstractions

Overview

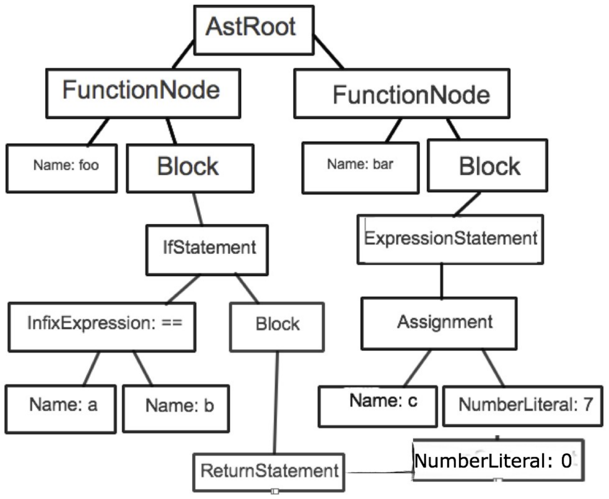
For this assignment, you will get the chance to create and implement your own iteration abstraction. More specifically, you will create an iterator for the abstract syntax tree (AST), which was introduced in Assignment 4. This abstraction can help simplify the process of collecting statistics from the AST.

Creating a Linked List of AST Nodes

Recall the sample JavaScript code shown in the Assignment 4 instruction sheet:

```
function foo() {
        if (a == b) {
                return 0;
        }
}

function bar() {
        c = 7;
}
```



Your goal is to implement an iteration abstraction that will help iterate over the nodes of the AST. The iteration should be ordered by 'depth'[1] (in ascending order – i.e., nodes closer to the root are iterated over first). In the given code, each node in the AST is of type AstNode.

Instructions

1. Download the file eece310_assn5.zip from the course website. Do NOT reuse code from previous assignments - use only code contained within this zip file.
2. Unzip the above file and add the project to Eclipse
3. As before, the org.mozilla.javascript.ast package contains classes that define various kinds of AST nodes (which are subclasses of the AstNode class). Look through these briefly to familiarize yourself with some of the node names.
4. Under the org.mozilla.javascript package, you will encounter the following new file:

    TestIterator.java: This is the class that contains the main function. You should **not** modify this class, as we will be using it to test the code you will write for this assignment.

5. Go to AstRoot.java (from the org.mozilla.javascript.ast package) and find the instructions marked **EECE310_TODO**. Read all the TODO instructions carefully. Your main tasks will be as follows:

1 The term *depth* describes how far down a node is from the root. In the tree above the AstRoot has depth 0, the FunctionNodes have depth 1, etc.

a. Implement the astIterator_depth() function, which merely returns a new instance of an AstNodeGenerator
b. Implement the hasNext() and next() functions of AstNodeGenerator. The method hasNext() returns true if the iterator still has elements to iterate over. The method next() returns the next AstNode to iterate over (it throws a NoSuchElementException if there are no more elements in the list).

Your generator should iterate over the AST nodes in order of depth.

For this assignment, you are free to include whatever extra members, methods, or private classes you wish to add the AstNodeGenerator class (*HINT: You may find Java's Comparator class useful*).

6. Run TestIterator.java to test your implementation. The expected outputs for JavaScript-Code1.js are described in JavaScriptCode1_Correct.txt (*Note: If two nodes have the same depth, the order of these two nodes in the list does not matter, so your output may still be considered correct even if it does not exactly match the contents of this text file*).

Deliverables

You will submit only one file: AstRoot.java, as this is the only file you will need to modify for this assignment.

Submission Instructions
Attach your code file: AstRoot.java.
Once you have done the above, submit the file as an attachment to ece310w2011@gmail.com with the subject as follows:

If you are working in groups of 2

Subject will be "Assignment 5: <StudentNumber1>_<StudentNumber2>".
Example Subject
Assignment5: 21779990_23001345

Evaluation

Your mark will be based on the correctness of your implementation. You will have access to JavaScriptCode1.js, but not JavaScriptCode2.js. The marking breakdown is as follows:

Correct stats for JavaScriptCode1.js - 45%
Correct stats for JavaScriptCode2.js - 35%
Good style (comments, overall code structure, etc.) - 20%

BONUS: 20 Bonus marks will be rewarded to students who can extend the JavaScript lexer and parser to support the parsing of loops (either for loops or while loops).