Assignment 4: Writing Concurrent Code

Overview

This assignment will give you the opportunity to write your own concurrent Java code. By this time, you should already be familiar with the process of lexing JavaScript code. Here, we will go a step further and actually perform *parsing* of the generated tokens.

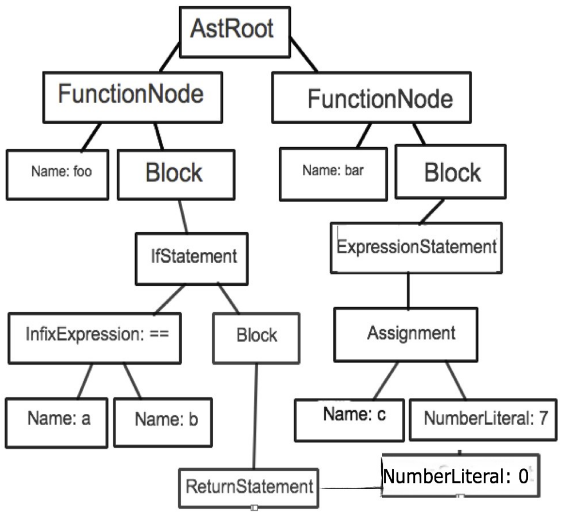
Background on JavaScript Parsing

As you already know, lexing is the process whereby tokens are generated for code - in our case, JavaScript code. The next step in the JavaScript code interpretation process is *parsing*, where the generated tokens themselves are analyzed to generate what is called an *abstract syntax tree (AST)*. The AST is basically a data structure that represents the syntax of the JavaScript source code as a tree. It is rooted at a root node (the **`AstRoot`** node in thecode), and each of the individual nodes represent some syntactic feature of the language. For example, a function is represented by a **`FunctionNode`**, an 'if' statement is represented by an **`IfNode`**, an assignment expression is represented by an **`Assignment`** node, a function or variable identifier is represented by a **`Name`** node, etc.

To make this more concrete, consider the following JavaScript code:

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

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



The AST representation of the above code is shown on the right. You can see that the AST not only represents the syntactic elements present in the JavaScript code, but it also represents the structure of the code.

For this assignment, your goal is to ***collect statistics*** regarding an AST. Specifically, you will use the AST to count the number of if statements and the number of assignments present in a JavaScript source code. The sequential, single-threaded implementation is already provided. Your task is to transform this sequential implementation into a concurrent implementation using what you know about threads in Java.

Instructions

1. Read and complete Lab 4, which gets you more familiar with writing concurrent Java code.

2. Download the file eece310_assn4.zip from the course website. Do NOT reuse code from previous assignments.
3. Unzip the above file and add the project to Eclipse.
4. Under the org.mozilla.javascript.ast package, you will see different classes defining the different kinds of AST nodes. Have a look through some of these files to give you an idea how AST nodes are created and what they contain (you might want to look at Assignment.java, AstNode.java, AstRoot.java, FunctionNode.java, and IfStatement.java in particular).

5. Under the org.mozilla.javascript package, you will see some of the files that you've already seen from previous assignments, as well as a few extra files, described below:

Parser.java: This class is used to parse the tokens generated by the lexer into an AST.

IfData.java: An object of this class contains information about an if statement, particularly the name of the function it belongs to, the if condition, and its name which contains a unique tag assigned by the statistics collector (i.e., unique amongst all if statements found)

AssnData.java: An object of this class contains information about an assignment expression, particularly the left hand side of the assignment, the right hand side of the assignment, the name of the function it belongs to, and its name which also contains a unique tag assigned by the statistics collector (i.e., unique amongst all assignment expressions found)

Tag.java: This static class contains the function newTag(), which is used to assign a unique tag to an IfData or AssnData object.

IfCollectorSequential.java: This is the sequential implementation of the statistics collector. In this class, we count the number of 'if' statements and assignment expressions contained in some input JavaScript source code file. For each if statement found, an IfData object is created. Similarly, for each assignment expression found, an AssnData object is created. **You should read through this file and try to understand what the code is doing. This will help you prepare to write the concurrent implementation**.

IfCollector.java: This is the class that you must implement, containing the concurrent implementation of the statistics collector. It should function similarly to IfCollectorSequential.java; however, whereas IfCollectorSequential has a single thread operating on the entire AST, IfCollector will have multiple threads operating on different **FunctionNode subtrees** of the AST, essentially dividing the task amongst these threads.

6. You are already provided with the skeleton code for IfCollector.java. Your task is to complete the implementation of this class based on what you know about writing concurrent Java code. In the IfCollector.java file, ***make sure you carefully read all the instructions marked EECE310_TODO***. Briefly, your main tasks are as follows:

    a. Implement the statistics collection scheme which creates an IfData object for each if statement encountered, and creates an AssnData object for each assignment expression encountered. The unique tags contained in the names of each IfData and AssnData object must be generated using the newTag() function in the Tag class (see TODO instructions for specific details).

b. Create and run multiple threads in the main function. Each thread you create will be assigned to work on one or more FunctionNode subtrees of the AST. It is your responsibility to figure out how to access a FunctionNode (*HINT: Every children of the AstRoot will be a FunctionNode in this assignment*). For a refresher on how to create and run a thread see Lab 4.
c. Manage synchronization so that your program executes correctly.

7. Run IfCollectorSequential.java by clicking on Run > Run Configurations... > Search..., selecting the IfCollectorSequential main function in the list that shows up, and then pressing Run. You will be prompted to specify the JavaScript file to parse (JavaScriptCode1.js in our case). You may also notice that the execution will take quite some time, depending on what machine you are using. Once the execution is finished, the following output will appear:

```
Finished in <time it took to finish collecting stats>
Number of ifs: <number of if statements found>
<IfData name> <if condition> <name of containing function>
.....

Number of assignments: <number of assignment expressions found>
<AssnData name> <left expression> <right expression> <name of containing function>
.....
```

Notice that the tags in the IfData name are all unique and are multiples of 10001 (the same goes for the AssnData tags). Your IfCollector implementation should result in the same output as IfCollectorSequential.java (although tags may not be in the same order). **In addition, your IfCollector implementation should have some speedup (larger than 1.0) over the sequential implementation to get full marks**. Speedup is calculated as follows:

Speedup = (time it takes IfCollectorSequential to finish) / (time it takes IfCollector to finish)

Note that speedup larger than 1.3 should be relatively easy to obtain. We set the bar of 1.0 intentionally low.

In order to attain a good speedup, you may want to experiment with the number of FunctionNode subtrees handled by each thread and the number of created threads.

Deliverables

You will submit only one file: IfCollector.java (see submission instructions). We will test your code both on the test files provided to you (JavaScriptCode1.js, JavaScriptCode2.js) and on new test files.

Submission Instructions

Send the file, IfCollector.java, as an attachment. Email to ece310w2011@gmail.com with the subject as follows, depending on how you are submitting:

If you are working in groups of 2

Subject will be "Assignment4: <StudentNumber1>_<StudentNumber2>".

Example Subject **Assignment4: 21779990_23001345**

Evaluation

Your mark will be based on two things: correctness and speedup. To check correctness, we will check whether your code generates the correct statistics for the two JavaScript files and new test files.

The marking breakdown is as follows:

- Correct stats for JavaScriptCode1.js - 20%
- Correct stats for JavaScriptCode2.js - 20%
- Correct results for new tests: 50%
- Speedup - 10% (Speedup larger than 1.0x will get these 10p; Speedup larger than 2x will get 10 bonus points). Note: we plan to test on a machine with a quad-core Intel i7 processor.

You will also have to add comments to your code explaining what you are trying to achieve.