Project 10 Report

Michael Li, Iris Lian, Zeb Keith-Hardy, and Kevin Zhou

December 7, 2018

Overview

For this project we built the parser for our compiler. This parser reads in tokens from our scanner from last week and then uses recursive descent to generate an abstract syntax tree for a program. As long as there are no errors to this program, the Scan and Parse button then generates a AST drawing in a new Frame using a TreeDrawer object. If there is an error found by the parser, the Parser will throw a CompilationException and the parsing will exit.

Solution to Extensions

To build the parser, we split up the project and just worked method by method to complete it. We used the notes given above each method to know exactly what kind of tokens each method needed to handle and what type of ASTNode the method needed to generate. By reading the Bantam manual, we were able to to handle the special cases for each of the methods. When methods hit an inappropriate token, the parser will register an error with the errorHandler and exit from the parsing by throwing a Compilation exception. Since we are using threading for this project, this causes the Scan & Parse thread to die, and then the errors are printed to the Console. We then created a legal bantam test file to test our parser. This test file handles all of the possible cases for our parser so we are pretty sure that the parser works perfectly.

Why Our Code is Elegant

- We added threading to our Scan and our Scan & Parse methods

Why Our Code is Inelegant

- At least for this project, the Parser is still generating the AST rather than using a Builder or a Listener

Division of Work

- We all split up the methods from the Parser and worked on a few of them each. We then checked each other's work for possible errors
- We all bug fixed the parser and made sure we were appropriately hitting tokens and directing them to the correct methods

- Zeb fixed the errors in the Scanner from last week (very brief).