Assignment 3

- Academic integrity is taken seriously in this course, which has a zero tolerance policy for cheating. All discovered incidents of cheating will be reported and result in an E in the course.
- You ARE allowed to:
  - discuss high level solutions with other students and
  - ask questions in the class slack channel and during office hours
  - use code presented in the lectures.
- You are NOT allowed to
  - Share your code with anyone else (except your official project partner).
  - Receive code from others (except your official project partner), including students in previous semesters.
- All work submitted should be the work of the submitting student(s), created this semester for this assignment.
- Submissions by groups must have approximately equal contributions from both members.

In this assignment, you will create an ASTVisitor that will check context constraints and decorate the AST returned by your parser.

This assignment was discussed in detail in class on 10/17.

1. Fix any problems with your lexer and parser from previous assignments.
2. Implement a symbol table for a block structured lexically scoped language as described in class. A LeBlanc-Cook symbol table is suitable.
3. Implement a class to perform type checking and decorate the AST.
   a. Your class should implement the ASTVisitor interface.
   b. Update ComponentFactory.makeTypeChecker as appropriate.
   c. Your visitor should check context constraints given in the grammar and throw a TypeCheckException if a constraint is violated. Error messages will not be graded, but you will appreciate useful ones later. Use the firstToken of the AST node exhibiting the error to get the location of the error in the source code.
   d. Your visitor should set the type and nameDef fields in the AST nodes where these have been added.

Provided code

- Modified version of edu.ufl.cise.cop4020fa23.ComponentFactory

- Package edu.ufl.cise.cop4020fa23.ast containing updated classes to use for the AST nodes. Some classes now contain fields Type type and/or nameDef nameDef which should be set by your visitor.
- Type.java – enum for representing types
- SyntheticNameDef – new ASTNode for implicitly declared variables (discussed in class on 10/19)
- TypeCheckTest_starter.java -- a few Junit tests.
- TypeCheckException – exception to throw if the type check visitor detects an error. (Errors detected by the Lexer should throw a LexicalException and errors detected by the Parser a SyntaxException as before)

Hints

- **Watch the lecture(s) before attempting this assignment**. How to do the assignment specifically was discussed on 10/17 and 10/19. Implementation of symbol tables is described in NamesScopesBindings3.
- Arrange for not yet implemented methods or branches in your code to throw a distinct exception (I use UnsupportedOperationException) and make sure that these are all gone in a complete implementation.
- The provided tests, which do NOT provide complete coverage, were automatically generated from the output of a reference implementation and are ugly and more detailed than you probably want to bother with when writing your own tests.

To turn in

- A jar file containing the source code (i.e .java files) of all classes necessary to test your parser with the appropriate directory structure. Do NOT include class files.
- Be aware that your IDE may provide a way to generate jar files that defaults to including .class files instead of .java files. Double check the contents of your jar file before submission.
- A zip file containing your git repository WITH history. To obtain this file, go to the parent of your local git repository and create a zip file. Your zip file should contain a directory called .git (dot git). Do NOT use the "download zip file" option on github--it does not include the history. Make sure that the name of your zip file does not start with . (dot). Canvas does not let us download filenames like .git.zip.