# Assignment 4

- 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.
  - share Junit test cases
- 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 generate Java code to implement a subset of our programming language. The rest of the language will be handled in Assignment 5.

Your task is to create an ASTVisitor class to traverse the type checked and decorated AST created with code from previous assignments. The output will be a String containing a valid Java program that implements the desired semantics for our programming language. See the code generation document for a detailed specification. *The lecture on 10/31 will also present important information about how to do the project.*

Provided routines in the edu.ufl.cise.cop4020fa23.DynamicJavaCompileAneExecute package compile the string to create a Java class file (residing in memory as a byte[]) and routines to load and execute the class.

You will need to do the following:

- Correct any mistakes in your Lexer, Parser, or type checking Visitor.
- To work around a mismatch between the scoping rules of our language and Java's scoping rules, implement a way to rename variables in the generated Java code so that every variable has a unique name. It is up to you to determine when and how to do this. (See the lecture on 10/31)
- Implement an ASTVisitor (CodeGenVisitor) to generate Java code. The visitProgram method of this visitor should accept a package name as an argument and return a String containing a Java program implementing the semantics of the language. The package name may be null or an empty String. In either such case, the generated program should be in the default package (i.e. has no package declaration).
- Add the following method (with correct implementation) to ComponentFactory

- `public static ASTVisitor makeCodeGenerator() {…}`

- Write statements should be implemented using routines in the provided class ConsoleIO.java
- It is suggested that attempts to compile unimplemented features throw an UnsupportedOperationException.
- It is suggested that you include '\n' at obvious places (for example after declarations or statements) to make your generated Java code more readable, but you will not be graded on this.

## Provided Code

- Package edu.ufl.cise.cop4020fa23.DynamicJavaCompileAndExecute
  - Several classes which contains routines to dynamically compile Java code and to load classfiles from a byte[]. The basic functionality is part of the standard Java distribution; the provided package provides some classes that are more convenient than using the Java classes directly.
  - Class PLCLangExec contains a method runCode that accepts a String containing source code in the PLC language, a package name for the generated code, and a variable length list of parameters. It will compile the PLC language code into a Java class, then compile the Java class to obtain a classfile, then execute its "apply" method with the given parameters. This is the routine that will be invoked in the Junit test cases. It would also allow little programs in PLC language to be invoked from any Java program.
- edu.ufl.cise.cop4020fa23.runtime.ConsoleIO.java
  - Routines in this class should be used to implement PLC language write statements.
  - Do not modify this class.
- edu.ufl.cise.cop4020fa23.exceptions.CodeGenException.java
  - Exception to be used for errors found during code generation.
- edu.ufl.cise.cop4020fa23.CodeGenTest_starter.java
  - Some Junit tests. Note that write statements are not checked in any of the examples, but will be during grading. You should verify for yourselves that appropriate values are printed to the console. Make sure to use routines in ConsoleIO.java for all output.

## To Turn in

- A jar file containing the source code (i.e .java files) of all classes necessary to test your compiler components in 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.