1. Design: Our design has two folders' worth of classes:
    - Intermediate Representation nodes (IR*), which are fairly directly converted from corresponding statements in code, and store tables for any fields or methods defined within them; and
    - Symbol Tables, which are hashmaps from the string name of a variable or method to its IR node. These allow easy lookup of methods or fields that are accessed, and automatically search in their parent tables if they don't have a corresponding method/field in them. These are ProgramTable, ClassTable, MethodTable, and VariableTable. VariableTable generically handles fields, parameters, or locals. ClassTable is never used except as a parent class for ProgramTable since decaf does not have classes, but we wanted to have the symbol table capacity to include classes.

    We then have a "trees" folder where we have code that converts Antlr's output into something that's actually a tree (ConcreteTree), and then transforms that into an Abstract Syntax Tree out of the IR nodes and Symbol Tables.
2. Extras: As of now, there have not been any substantial extras used in our project. We did write a class symbol table even though classes are not part of decaf, because we wanted to understand how to build the tables to make them work.
3. Difficulties: Up to now, we have not been able to focus on the semantic checks and have put most of the work into generating the intermediate representation, and so feedback would be most productive in these areas (though suggestions for better ways to structure the mostly-unimplemented SemanticChecker class would be useful too). As of now, we are still figuring out the best ways to represent types, as well as in general hashing out the exact split between representing something in the IR as opposed to the symbol table..
4. Contribution: Arkadiy mostly designed the IR nodes, Jackie built the symbol tables and integrated them into the AST, and Maya wrote the code to build the ConcreteTrees and ASTs. Jackie then wrote all of the existing SemanticChecker file. Currently, we plan on Arkadiy and Jackie focusing on implementing most of the semantic checks, while Maya continues on improving the IR generating code.