Design of your symbol table:

- Our symbol table is a Stack of Scopes. Scope is implemented with HashMap of Symbols.
    - In global scope Symbols are functions, types, “static” variables.
    - In other scopes (let and function) we only have “var” variables and types as Symbols.
- Whenever we enter a new scope, we push a new scope object in SymbolTable Stack. Whenever we exit the scope, we pop the last scope from the Stack.
- Whenever the tiger code declares a new Symbol, we put this symbol in current scope (top of Stack).
- Symbol table also gives us ability to generate temporary variables of any base type
- Each Symbol can be converted to NakedVariable which removes all the parts from Symbol that IR does not need.
- All the Symbols carry a single TypeStructure which tells us how it’s going to be represented in memory. And we use it for type checking as well.
- each symbol knows how to format itself for .st file

Description of method of semantic checking including any listeners/visitors used:

- We do semantic checking and constructing the symbol table in the same iteration.
- We iterate over the code with visitor pattern (however we don't extend any existing Visitors).
- If we detect an error, we log it, because of this we have to constantly check if we are in a correct state.
- BREAK keyword needs to know last loop scope so we store label for BREAK to jump to
- Type checking is implemented using bunch of if’s :D
- We are not super worried about generating too many temporary variables. Our IR optimizer better be good.

Description of method of IR generation including any listeners/visitors used:

- IRGenerator class is the only class actually aware of IR instruction names.
- It’s the one responsible for generating unique labels
- It also merges all the variables from scopes that are generated for a particular function.(since in IR there’s only one scope per function)

Bugs or features that are not fully operational:

- Forward calls work ;) no test actually tests it. It’s a feature in our opinion.
- Test 24 is wrong on your side