This part of the project contains the following stages:

Environment building
Type checking
Static Analysis

each one of these, along with Weeder from assignment 1, utilizes the Visitor pattern. Each task above obtains context by traversing (visiting) each node on the AST.

Environment building:

Symbols:
Each entity in the joos language is a symbol: classes, interfaces, fields, methods, local variables formal parameters etc,.
So a generic class called Symbol is created containing an identifier and its declaration (which is something everything entities above have). Then each entity have their own class which extends Symbol.

Symbol Table:
Each AST node now has a symbol table containing local variables within that scope.
Each file (or you can say a class) has a more special symbol table. It contains symbols that not every scope can have: fields, constructors, methods. It also keeps track of the symbol tables from the classes and interfaces that this class has inherited.

The environment building visitor then parses those symbols and add them to the appropriate symbol tables. Then, reference types are linked with the appropriate files (classes). Finally a set of inheritance rules are checked.

Type checking:
This stage focuses on resolving the type of expressions. So when the visitor visits any class that contains an “Expression”, it will use a helper function “resolveType” which recursively determines the type of that Expression.

Static analysis
This visitor only does three things: check if statement is reachable, check if return statement is there when required, and check if a variable occurs in its own initialization. The assignment description requires the static analysis to check if all local variables have an initializer. After some clarification from Piazza we think that it is easier to just change the grammar to force every local variable to come with an initializer.
Each statement is given a reachable and a completable flag. Reachable means the code can be reached, completable means that the code will terminate, which implies that the statement following the current will be reachable.
Return statement was initially thought to be always required when the type of the method is not void. But the test case “J1_7_Reachability_WhileTrue_ConstantFolding.java” is apparently an exception. It suggests that return statement is not required when the while statement cannot terminate.
When the visitor visits LocalVariableDeclaration, it will set the currentLocalVariable to the current id of the variable. Then, during the subsequence visit of the local variable’s initialization, should

the visitor ever come to another id, it will check if this id matches the currentLocalVariable id, if so then this violates the rule that “variable must not occur in its own initializer”