Code structure

semanticCheck – The main package of PA3, contains the following files:

3 classes implementing the visitor interface:

- **SymbolTableBuilder** – Visits the program AST nodes from its root, and creates a FrameScope tree of all scopes in the program, during which it connects each AST node's scope to its parent to form the hierarchy.
- **TypeTabelBuilder** – Creates a data structure of all primitive, user and method types used in the program by going over the AST nodes from its root down, including array types.
- **SemanticChecker** - Checks the code for semantic errors - all checking rules defined by PA3, and a bonus check for "a method with a non-void return type returns a value on every control path".

Additional classes:

- **FrameScope** –
  The class acts as the scope of each node of the AST, and by that obeys the hierarchy rules. It possess all necessary scope details of the scope such as methods, fields, local variables etc. Each scope is defined differently according to his scope type as in Section 10 of the IC specification.
  SemanticException –
  A unique exception to be thrown when a semantic error is discovered during the checking.

ic – Conatins the main Compiler class, which creates all the compiling phases: Lexing, parsing and semantic symbol table building & checking. Also contains all the basic enums used in the project – DataTypes, BinaryOps etc.
ic.ast – Contains all the AST node types used in the project, and the Visitor interface
ic.parser – Contains the Lexer & Parser files from previous PA1, PA2, after generation by the lex.CUP & IC.cup files.

Semantic analysis implementation

Each AST node implements an "accept" function, with a visitor as an argument. Each time this function is called, the node calls the visit function of the relevant visitor, with itself as an argument. That way, each different visitor conducts its relevant visit on each node.

When **building the symbol table**, each visit of the nodes attaches the current scope to the visited node and to its parent scope.

When **building the type table**, each visit of the nodes adds the type to the relevant data structure in the TypeTableBuilder, as primitive, arrayTypes, classes and methods have their own HashMap.

When **checking semantics**, each visit runs the required checking, and uses the built FrameScope hierarchy to validate the node's info with its relevant scope. For instance, if the node is an assignment to variable, the checker finds its scope, and checks that the variable is indeed defined in the scope/main scope (In case of a function – The variable can be an argument or a local defined variable), and that the type of the LHS and the RHS are valid and obey the assignment rules.
Note – All semantic checking rely on the AST node Top Down hierarchy, starting from root!!

Known problems:

- When running input with syntax errors through the JAR file, there are some problems caused by the parser function "expected_token_ids()" , unlike running from eclipse, where there is no such error – The actual syntax error is displayed.
- TypeTableBuilder scans the AST tree from the root downwards, each time going over the children and moving on to the next level. Therefore, the order of type prints is different than the example given.