University of Calgary
SENG 300 - Introduction to Software Engineering

Group Project Iteration 1

Finding Declarations and References

March 14, 2018

Evan Quan 10154242, Irene Chan 10103807, Patrick Gharibi 10137116

1 Structural Diagram

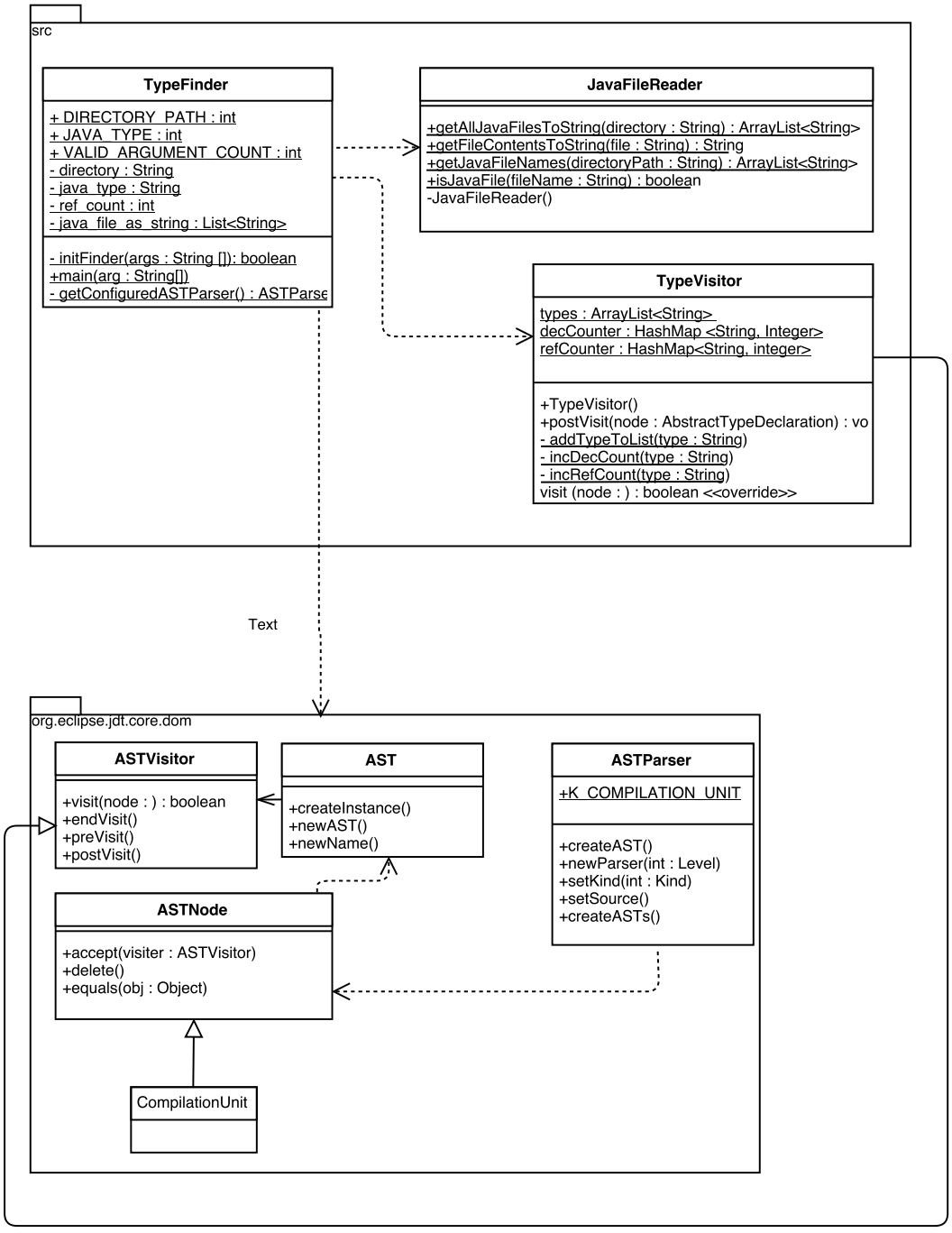

Figure 1: The relationship between our main package and relevant classes provided by org.eclipse.jdt.core.dom

2 Sequence Diagrams

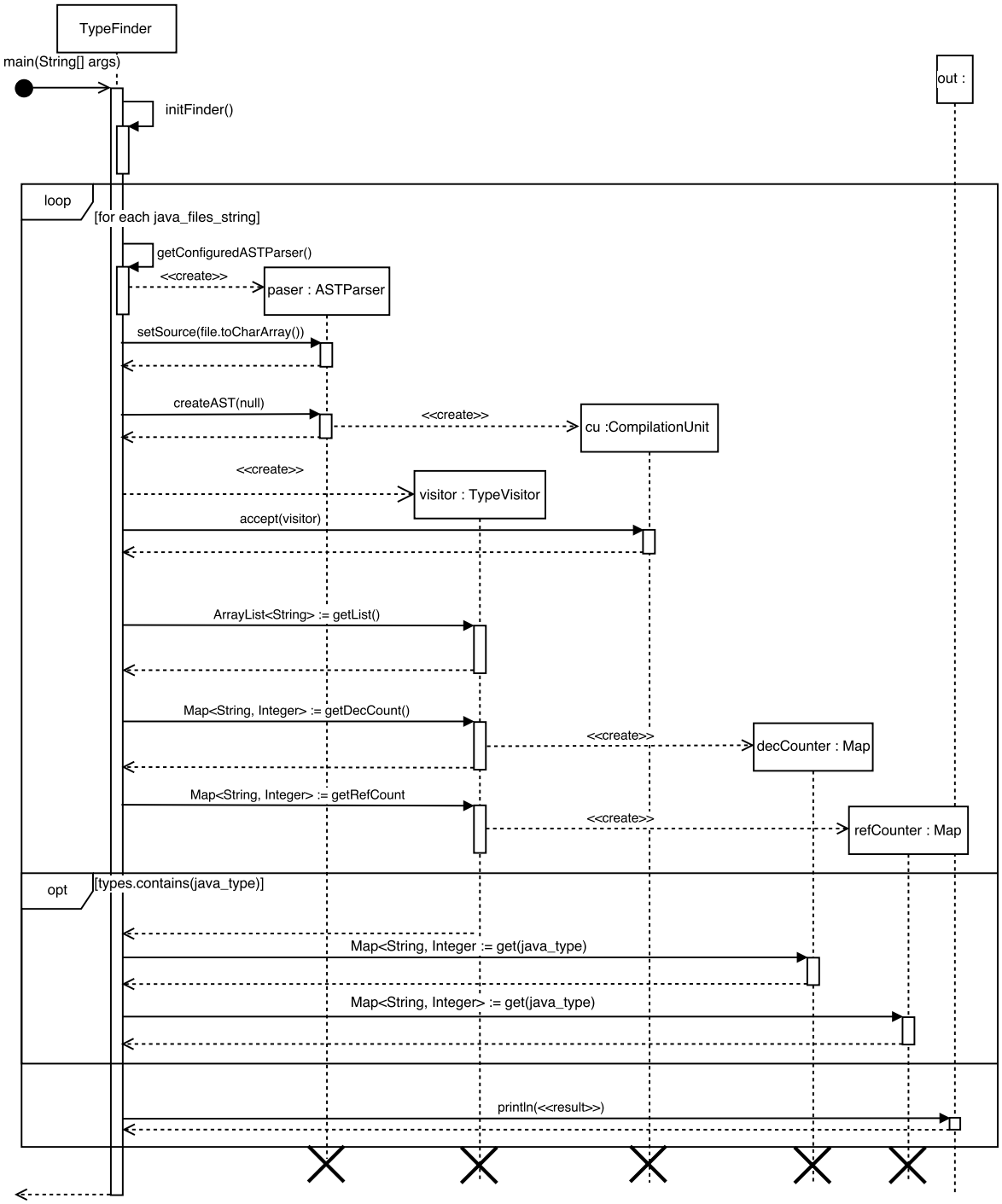

Figure 2: Sequence of TypeFinder program intialization and completion

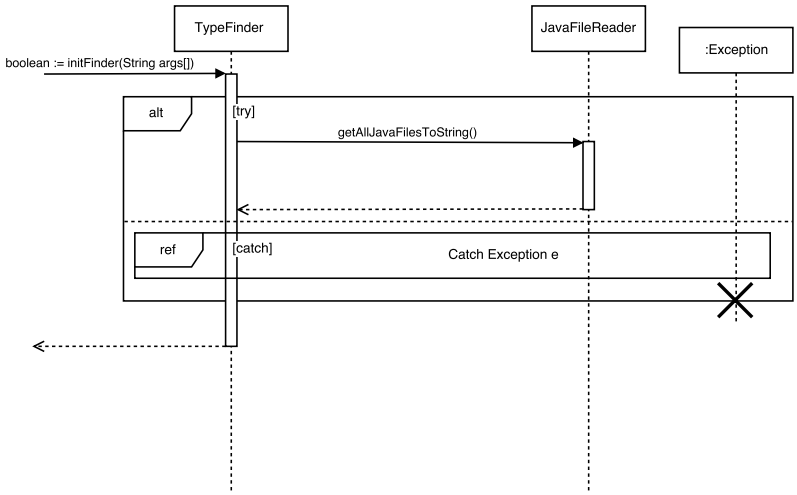

Figure 3: Initializing TypeFinder involving checking for valid user input. If valid, it acqures the Java file contents and setting up the information necessary for parsing. If invalid, prompt the user with an error message.

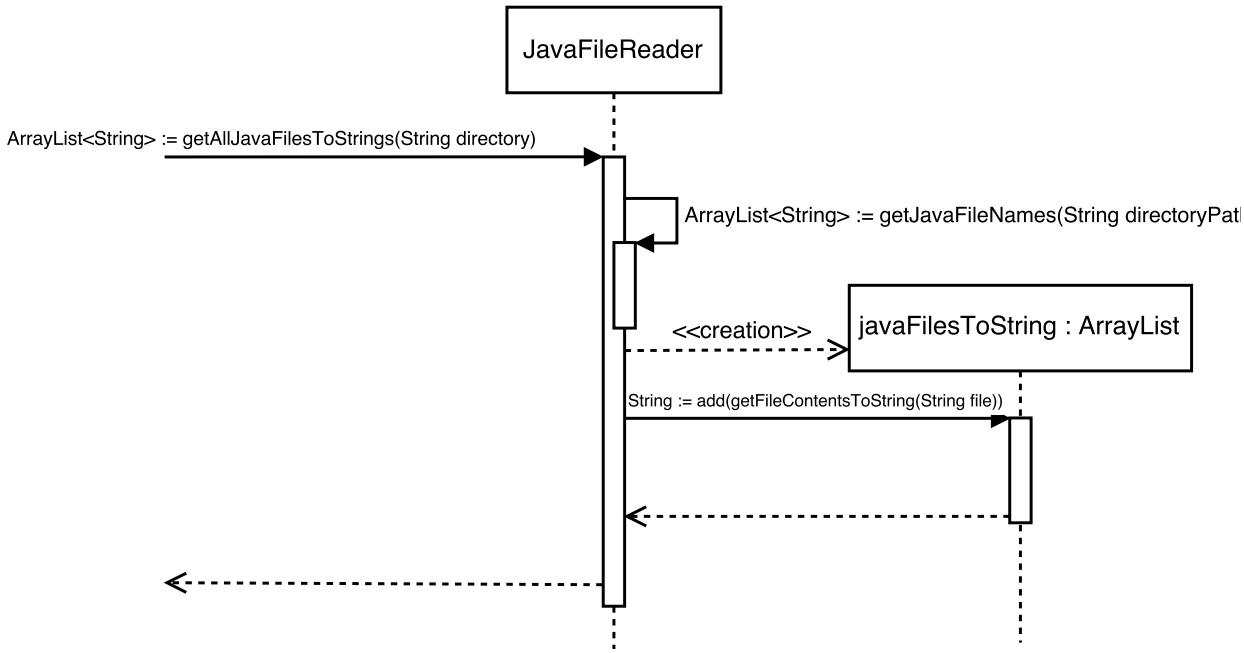

Figure 4: JavaFileReader retrieves the contents of all Java files in a directory, one file at a time

3 State Diagram

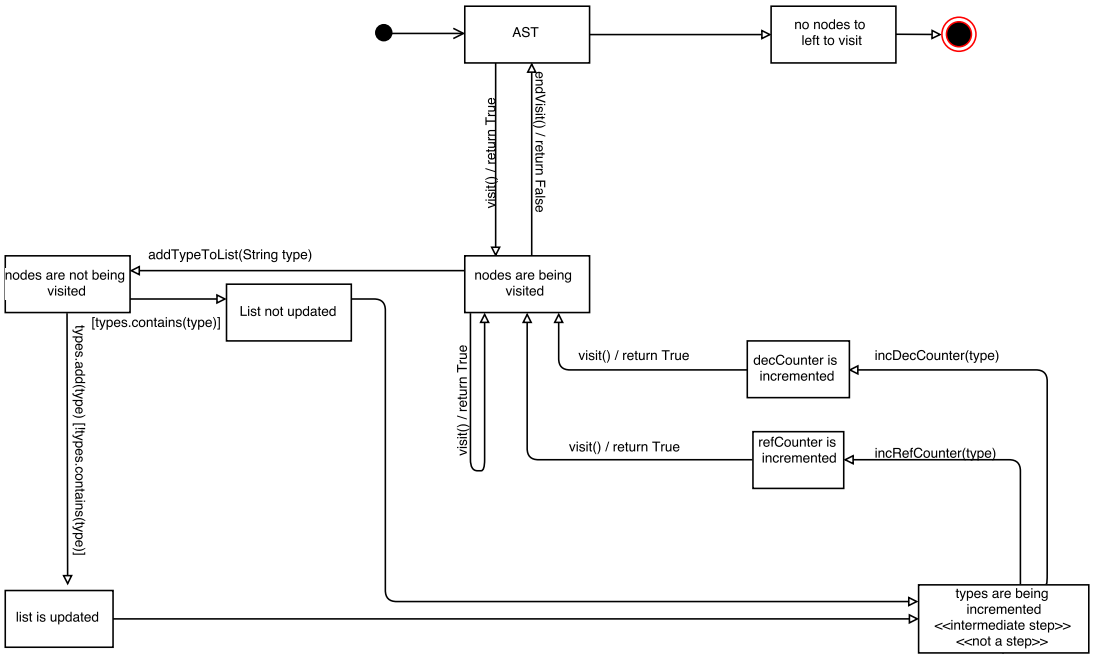

Figure 5: The state of the program in finding declaration and reference counts.

4 Explanation

4.1 Usage

Run the Java `TypeFinder.class` file through command line:
`java TypeFinder <directory> <Java type>`
`directory` is the path of the directory (either absolute or relative).

4.2 Structure

The UML diagram was kept as simple as possible and only represent main components of the software. All the methods in the Type Finder Class were included since they drive the software during execution. The JavaFileReader class is reads the all files in the directory and allows for parsing via AST. We Did not include all the methods for the TypeVisitor. The abstracted details were only supporting features that did not aid in the understanding of the functionality. As for visit (node :) : boolean, the formatting was done this way due to the multiple overridden methods with different parameter types, such as SingleVariableDecleration, TypeDecleration...etc. like the rest only the key components of the ASTParser, AST, ASTVistor, ASTNode were maintained in the Uml Diagram. Everything that did not aid or was none essential in understanding the relationships between the TypeFinder and the AST parser was left out.

4.3 Sequence Diagram

The software works by receiving arguments from the user, those being the directory of interest and the fully qualified name of a java type. It the enters a loop which executes per file read. After entering the loop, the ASTParser is configured with the correct specifications and created. after that the file to be read is set and the AST is created along with a visitor. we then begin to visit the nodes of the AST counting the number of declaration/references of types. finally, at the end of the loop we sum the total number of counts. the loop is exited and the number of references and declarations in printed to the console.
The inner workings of the getConfiguredASTParser () method is non-essential to the client's understanding of the software. Thus, the details were abstracted away and not expanded on. Another aspect of the code that was abstracted away was most of the methods in JavaFileReader. We modeled a very broad overview, this informs anyone viewing the model (provided some basic knowledge of java) of what is happening. Had we made our diagram anymore specific in this area, it would draw attention away from the more important functionality of the software and would only overwhelm the viewer. A very basic diagram shows the execution of initFinder, which was only meant to serve the purpose of showing that the javaFileReader Class was being used to read the files. A more Specific sequence diagram was provided to show how getAllJavaFilesToStrings () was receiving information about the directory and files, as well as how it returned the result for ASTParser to use.

4.4 State diagram

The state diagrams show the transition of states while visiting the nodes, within the AST, the