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 Gharib 10137116

1 Structural Diagram

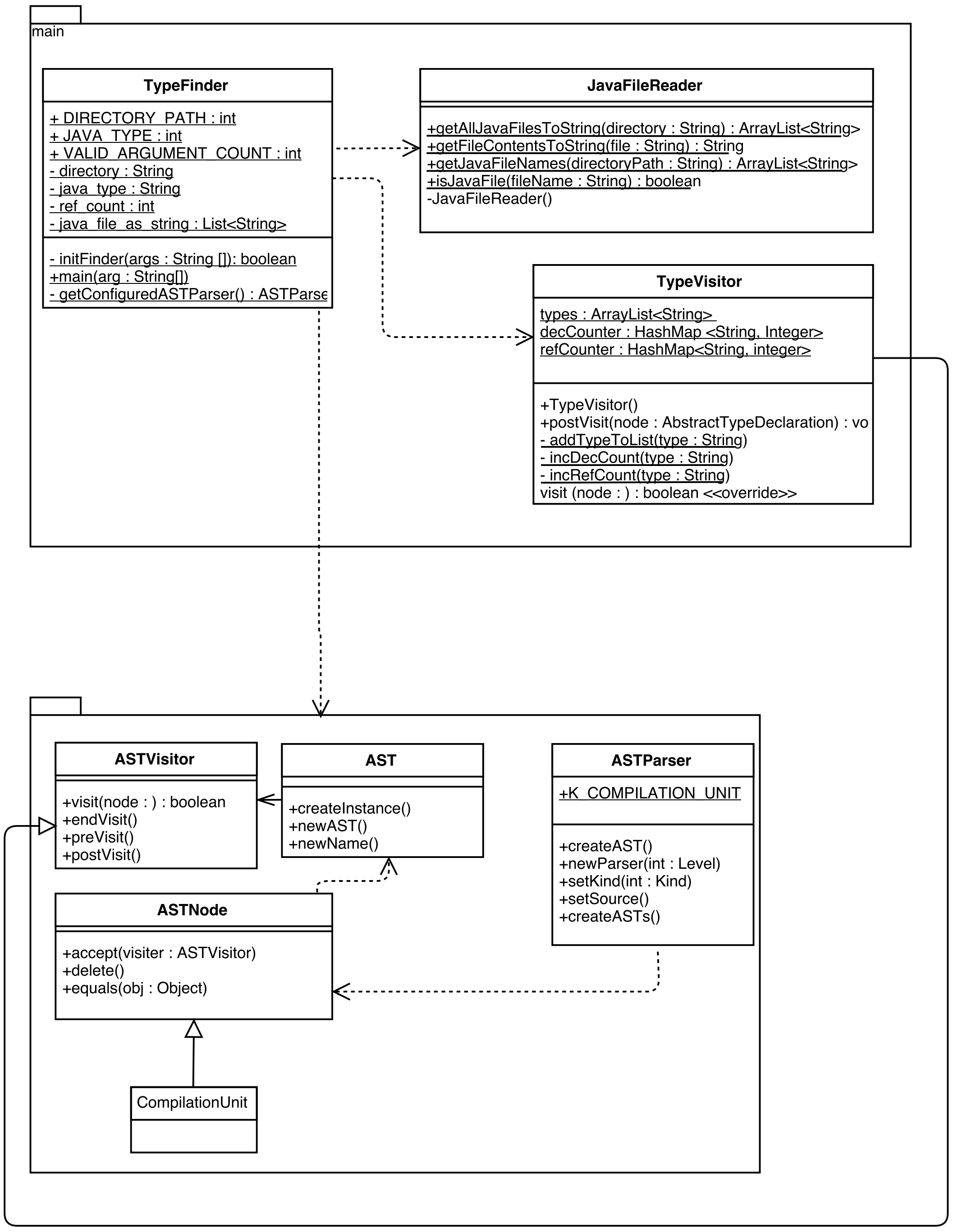
main
TypeFinder
+ DIRECTORY_PATH : int
+ JAVA_TYPE : int
+ VALID_ARGUMENT_COUNT : int
- directory : String
- java_type : String
- ref_count : int
- java_file_as_string : List<String>
- initFinder(args : String []): boolean
+main(arg : String[])
- getConfiguredASTParser() : ASTParse
JavaFileReader
+getAllJavaFilesToString(directory : String) : ArrayList<String>
+getFileContentsToString(file : String) : String
+getJavaFileNames(directoryPath : String) : ArrayList<String>
+isJavaFile(fileName : String) : boolean
-JavaFileReader()
TypeVisitor
types : ArrayList<String>
decCounter : HashMap <String, Integer>
refCounter : HashMap<String, integer>
+TypeVisitor()
+postVisit(node : AbstractTypeDeclaration) : vo
- addTypeToList(type : String)
- incDecCount(type : String)
- incRefCount(type : String)
visit (node :) : boolean <<override>>
ASTVisitor
+visit(node :) : boolean
+endVisit()
+preVisit()
+postVisit()
AST
+createInstance()
+newAST()
+newName()
ASTParser
+K_COMPILATION_UNIT
+createAST()
+newParser(int : Level)
+setKind(int : Kind)
+setSource()
+createASTs()
ASTNode
+accept(visiter : ASTVisitor)
+delete()
+equals(obj : Object)
CompilationUnit

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

2 Sequence Diagrams

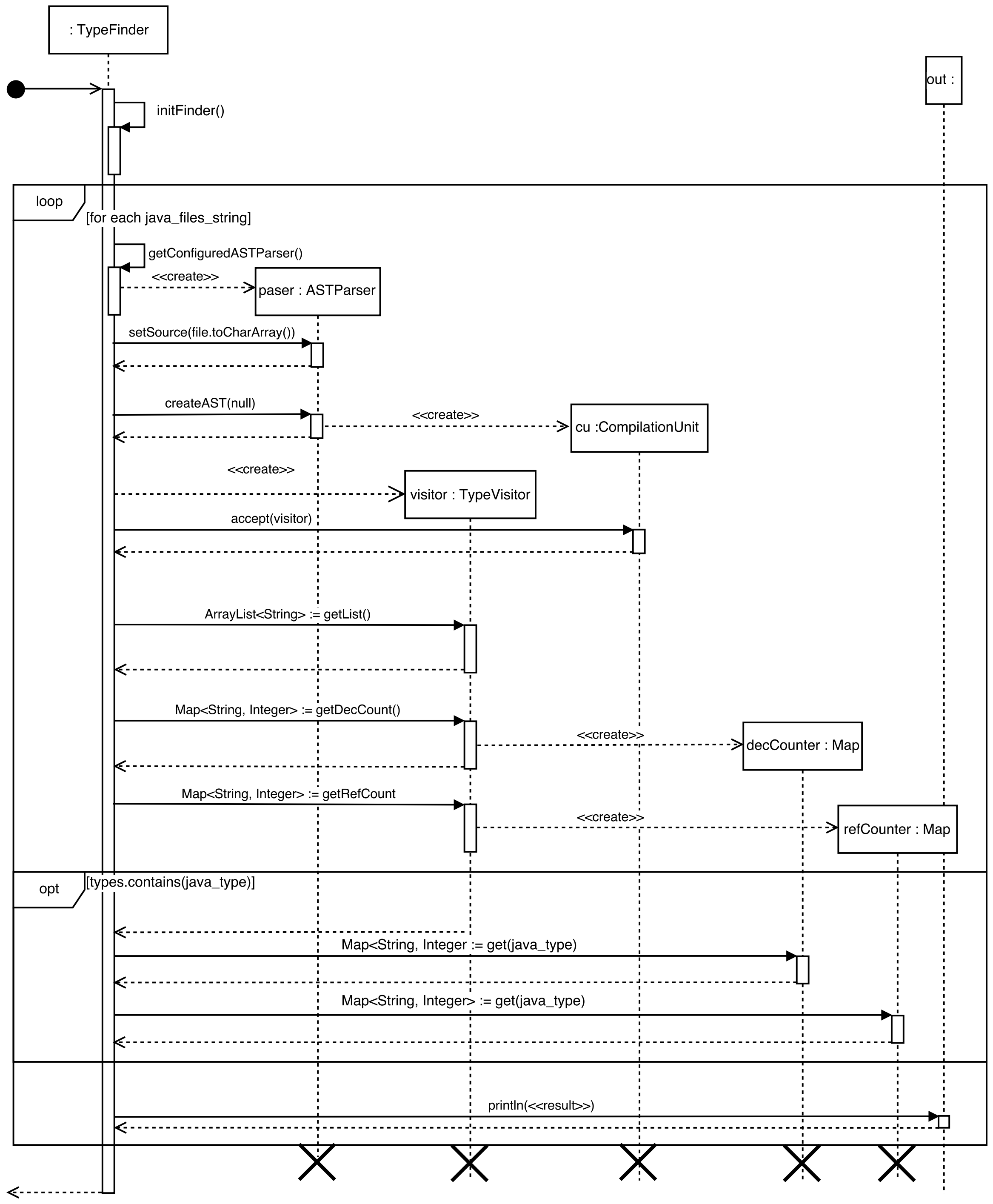
: TypeFinder
out :
initFinder()
loop
[for each java_files_string]
getConfiguredASTParser()
<<create>>
paser : ASTParser
setSource(file.toCharArray())
createAST(null)
<<create>>
cu :CompilationUnit
<<create>>
visitor : TypeVisitor
accept(visitor)
ArrayList<String> := getList()
Map<String, Integer> := getDecCount()
<<create>>
decCounter : Map
Map<String, Integer> := getRefCount
<<create>>
refCounter : Map
opt
[types.contains(java_type)]
Map<String, Integer := get(java_type)
Map<String, Integer> := get(java_type)
println(<<result>>)

Figure 2: Sequence of TypeFinder program intialization and completion

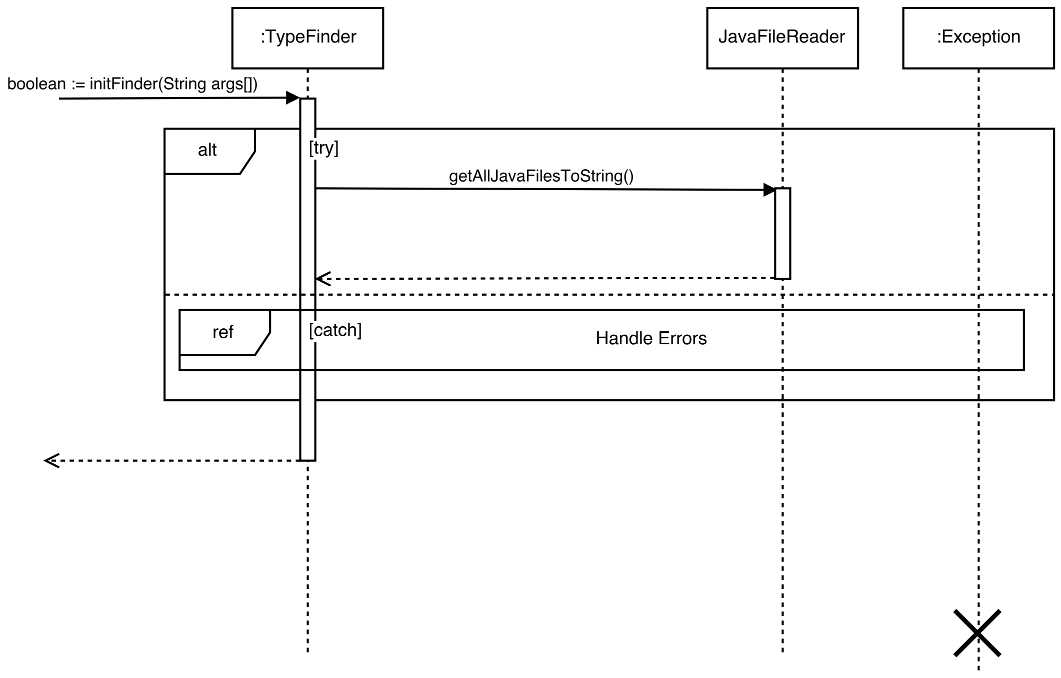

Figure 3: Initializing TypeFinder involving checking for valid user input. If valid, it acqures the Java file contents and setting up the information necessary for parsing. If invalid, prompt the user with an error message.

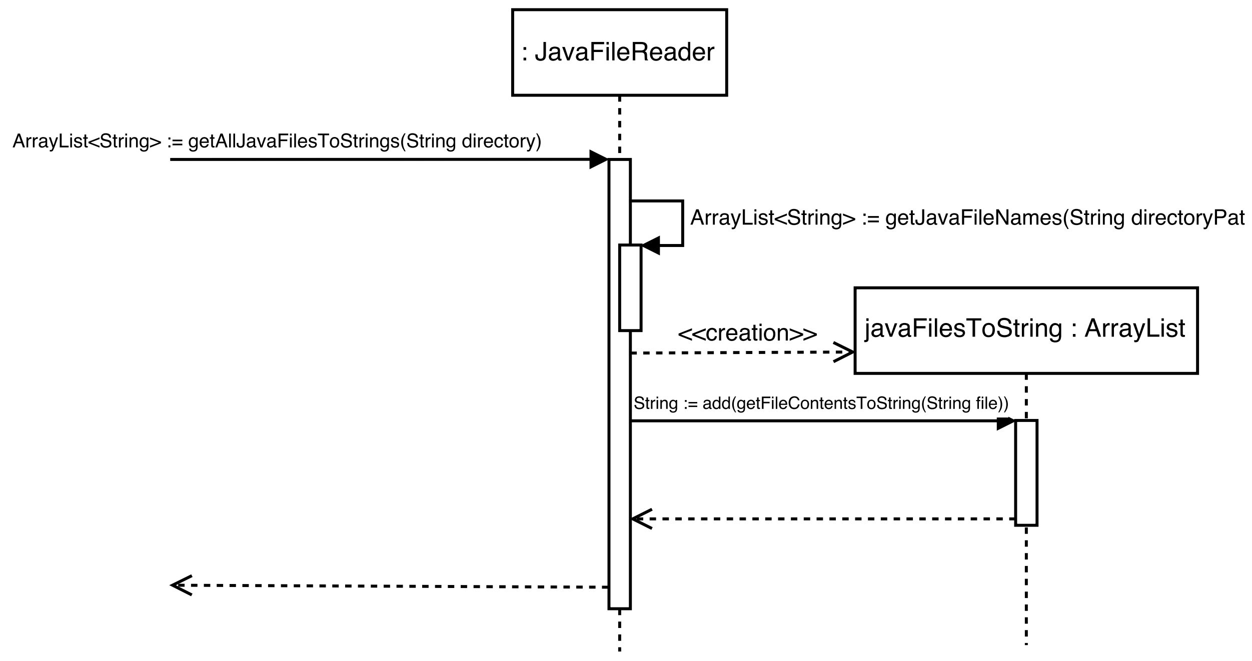
: JavaFileReader
ArrayList<String> := getAllJavaFilesToStrings(String directory)
ArrayList<String> := getJavaFileNames(String directoryPat
<<creation>>
javaFilesToString : ArrayList
String := add(getFileContentsToString(String file))

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

3 State Diagram

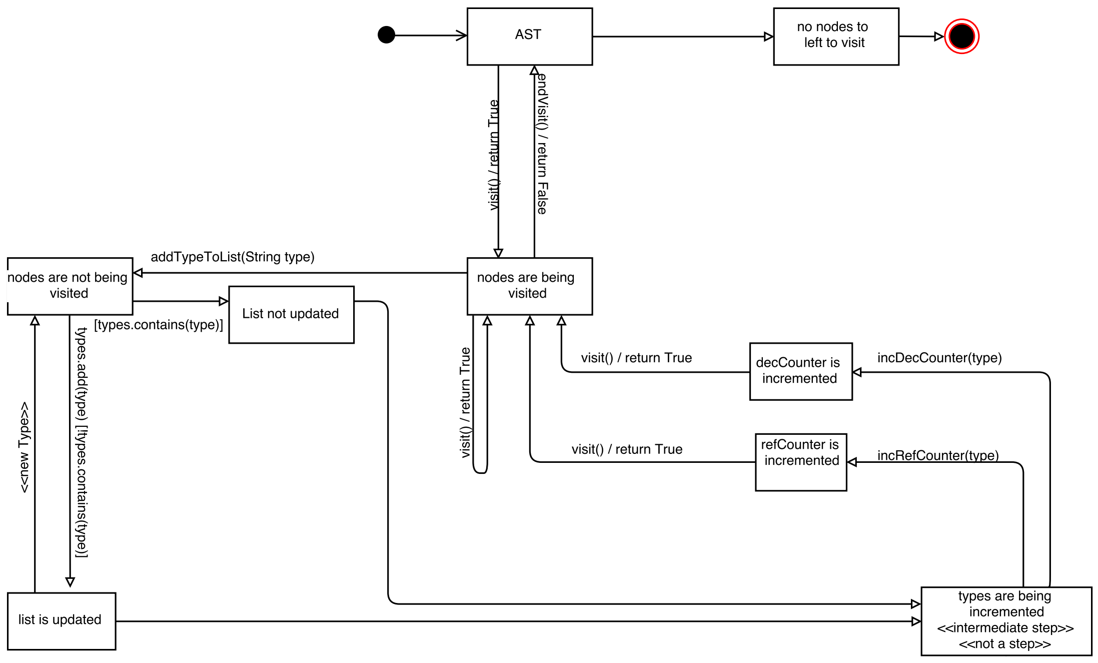
AST
no nodes to left to visit
visit() / return True
endVisit() / return False
addTypeToList(String type)
nodes are not being visited
List not updated
nodes are being visited
[types.contains(type)]
types.add(type) [!types.contains(type)]
<<new Type>>
visit() / return True
visit() / return True
decCounter is incremented
incDecCounter(type)
visit() / return True
refCounter is incremented
incRefCounter(type)
list is updated
types are being incremented <<intermediate step>> <<not a step>>

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

4 Explanation

4.1 Struture

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 formating was done this way due to the multiple overriden 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.

4.2 Sequence

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.