University of Calgary
SENG 300 - Introduction to Software Engineering

Group Project Iteration 2

Finding Declarations and References

March 26, 2018

Team 2

Evan Quan 10154242
Marcello Di Benedetto 30031839
Mona Agh 30033301
Sunah Kim 10155604
Zahra Al Ibrahim 30020048

1 Structural Diagram

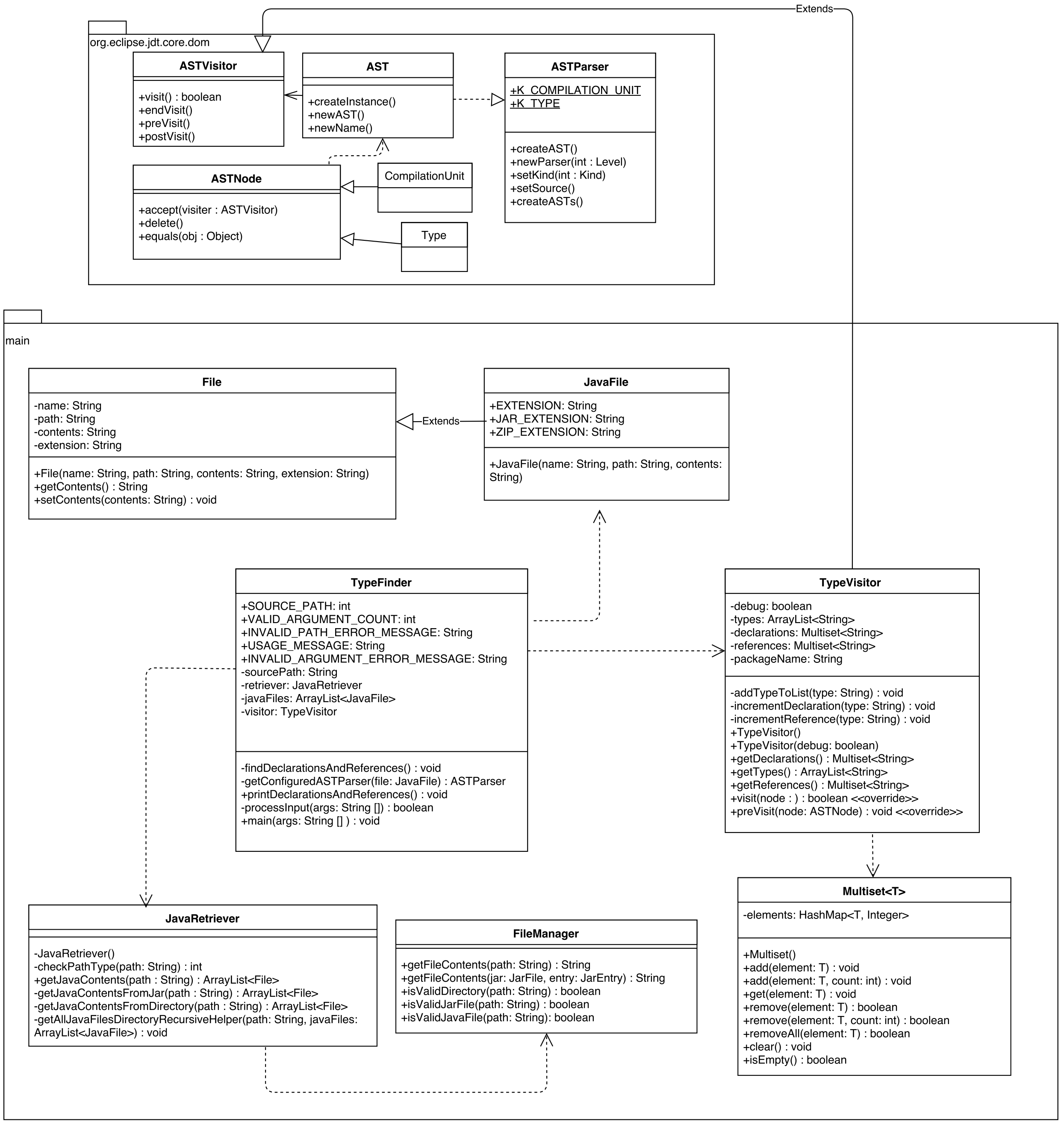
Extends
org.eclipse.jdt.core.dom
ASTVisitor
+visit() : boolean
+endVisit()
+preVisit()
+postVisit()
AST
+createInstance()
+newAST()
+newName()
ASTParser
+K_COMPILATION_UNIT
+K_TYPE
+createAST()
+newParser(int : Level)
+setKind(int : Kind)
+setSource()
+createASTs()
ASTNode
+accept(visiter : ASTVisitor)
+delete()
+equals(obj : Object)
CompilationUnit
Type
main
File
-name: String
-path: String
-contents: String
-extension: String
+File(name: String, path: String, contents: String, extension: String)
+getContents() : String
+setContents(contents: String) : void
Extends
JavaFile
+EXTENSION: String
+JAR_EXTENSION: String
+ZIP_EXTENSION: String
+JavaFile(name: String, path: String, contents: String)
TypeFinder
+SOURCE_PATH: int
+VALID_ARGUMENT_COUNT: int
+INVALID_PATH_ERROR_MESSAGE: String
+USAGE_MESSAGE: String
+INVALID_ARGUMENT_ERROR_MESSAGE: String
-sourcePath: String
-retriever: JavaRetriever
-javaFiles: ArrayList<JavaFile>
-visitor: TypeVisitor
-findDeclarationsAndReferences() : void
-getConfiguredASTParser(file: JavaFile) : ASTParser
+printDeclarationsAndReferences() : void
-processInput(args: String []) : boolean
+main(args: String []) : void
TypeVisitor
-debug: boolean
-types: ArrayList<String>
-declarations: Multiset<String>
-references: Multiset<String>
-packageName: String
-addTypeToList(type: String) : void
-incrementDeclaration(type: String) : void
-incrementReference(type: String) : void
+TypeVisitor()
+TypeVisitor(debug: boolean)
+getDeclarations() : Multiset<String>
+getTypes() : ArrayList<String>
+getReferences() : Multiset<String>
+visit(node :) : boolean <<override>>
+preVisit(node: ASTNode) : void <<override>>
JavaRetriever
-JavaRetriever()
-checkPathType(path: String) : int
+getJavaContents(path : String) : ArrayList<File>
-getJavaContentsFromJar(path : String) : ArrayList<File>
-getJavaContentsFromDirectory(path : String) : ArrayList<File>
-getAllJavaFilesDirectoryRecursiveHelper(path: String, javaFiles: ArrayList<JavaFile>) : void
FileManager
+getFileContents(path: String) : String
+getFileContents(jar: JarFile, entry: JarEntry) : String
+isValidDirectory(path: String) : boolean
+isValidJarFile(path: String) : boolean
+isValidJavaFile(path: String): boolean
Multiset<T>
-elements: HashMap<T, Integer>
+Multiset()
+add(element: T) : void
+add(element: T, count: int) : void
+get(element: T) : void
+remove(element: T) : boolean
+remove(element: T, count: int) : boolean
+removeAll(element: T) : boolean
+clear() : void
+isEmpty() : boolean

Figure 1: The relationship between our main package and relevant classes provided by org.eclipse.jdt.core.dom. TypeFinder acts as the main class that handles user input, manages other classes, and output results. JavaRetriever retrieves Java file contents from the directory or .jar of choice. TypeVisitor counts the references and declarations from the Java file contents.

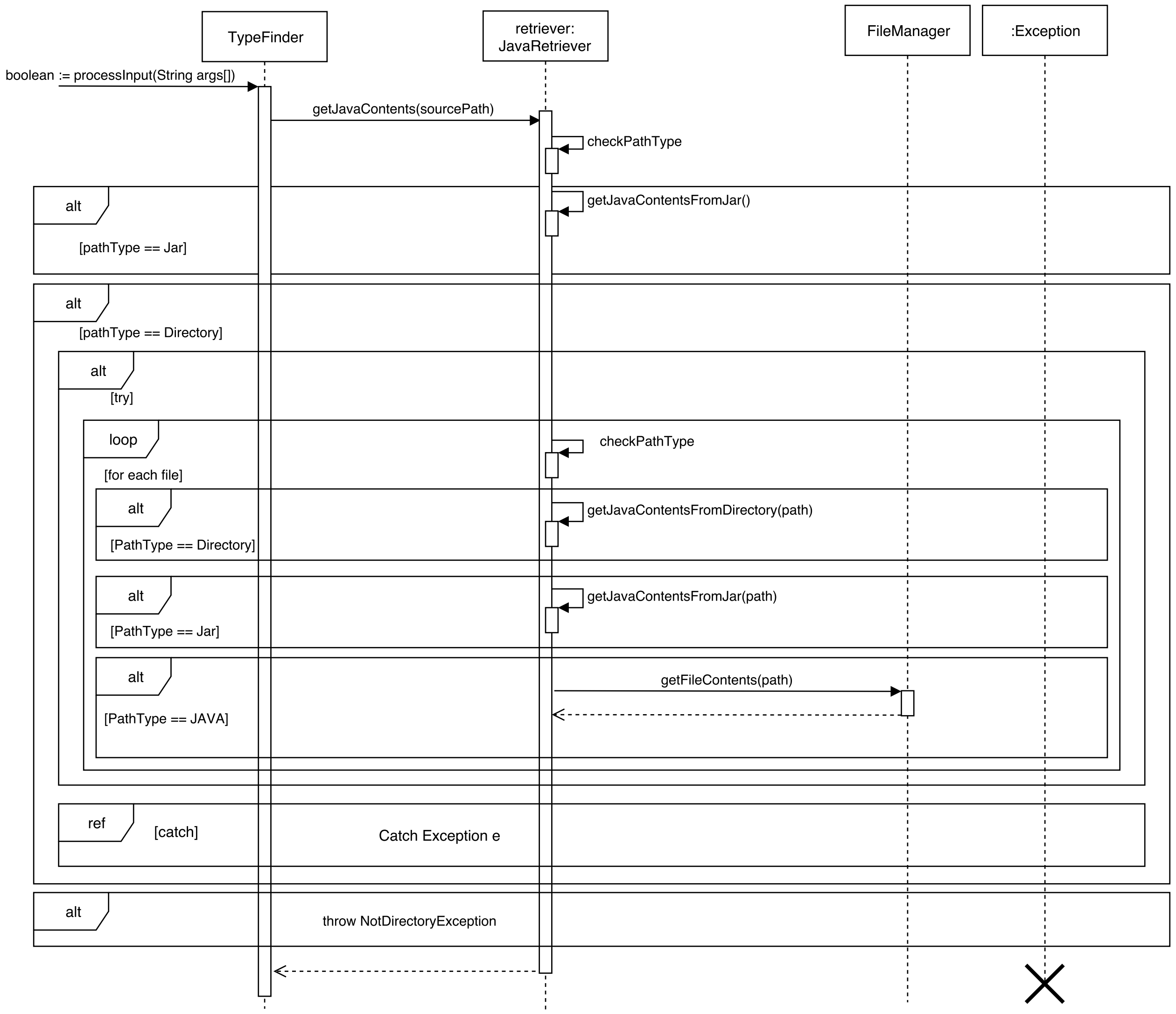

Figure 2: Sequence of JavaRetriever retrieving the contents of all Java files in a directory or .jar file recursively, so it can be later processed to find the references and declarations.

2 Sequence Diagrams

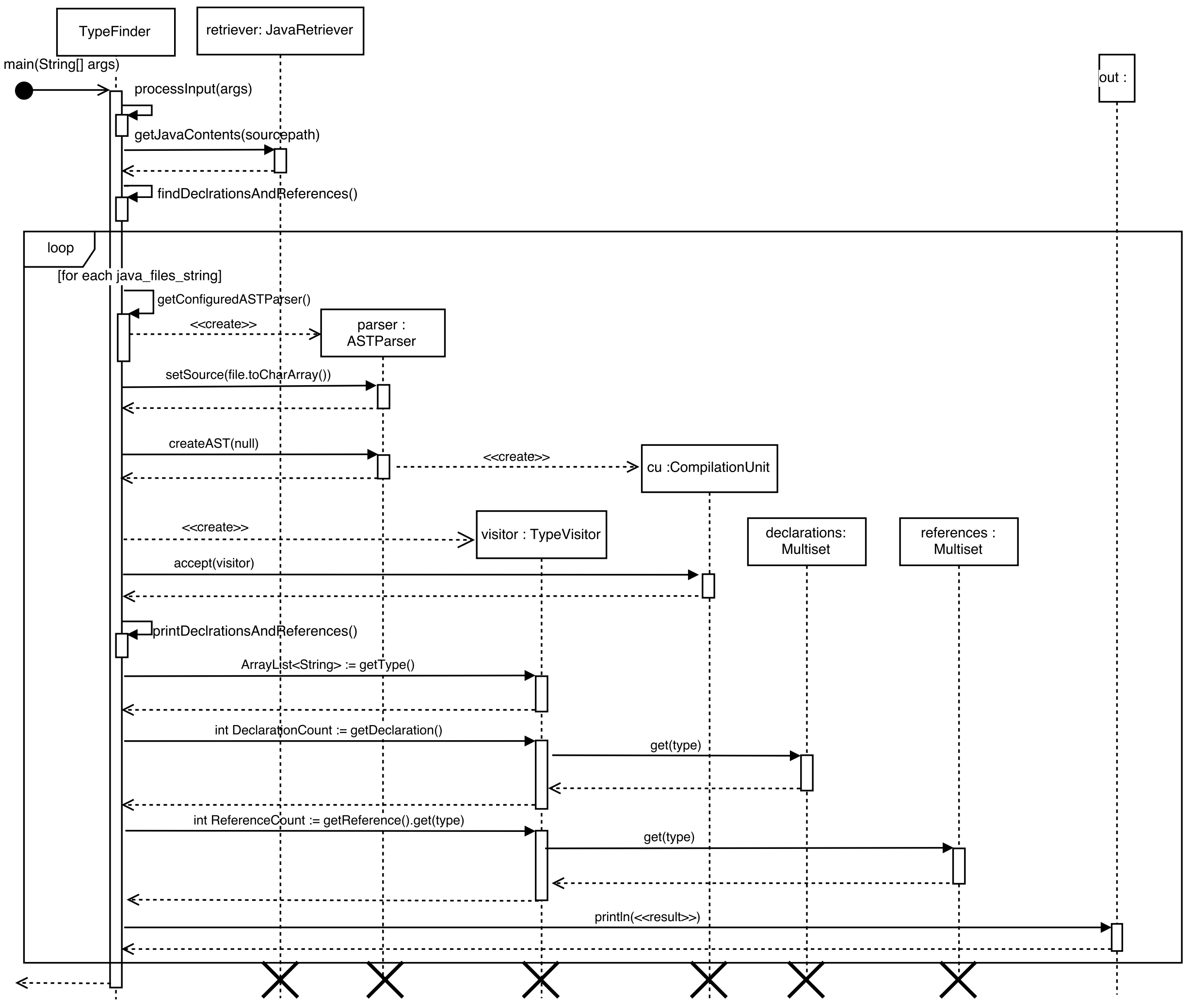

Figure 3: Sequence of TypeFinder program intialization and completion, emphasizing how the program processes finding the reference and declaration counts after the Java file contents have been acquired.

3 State Diagram

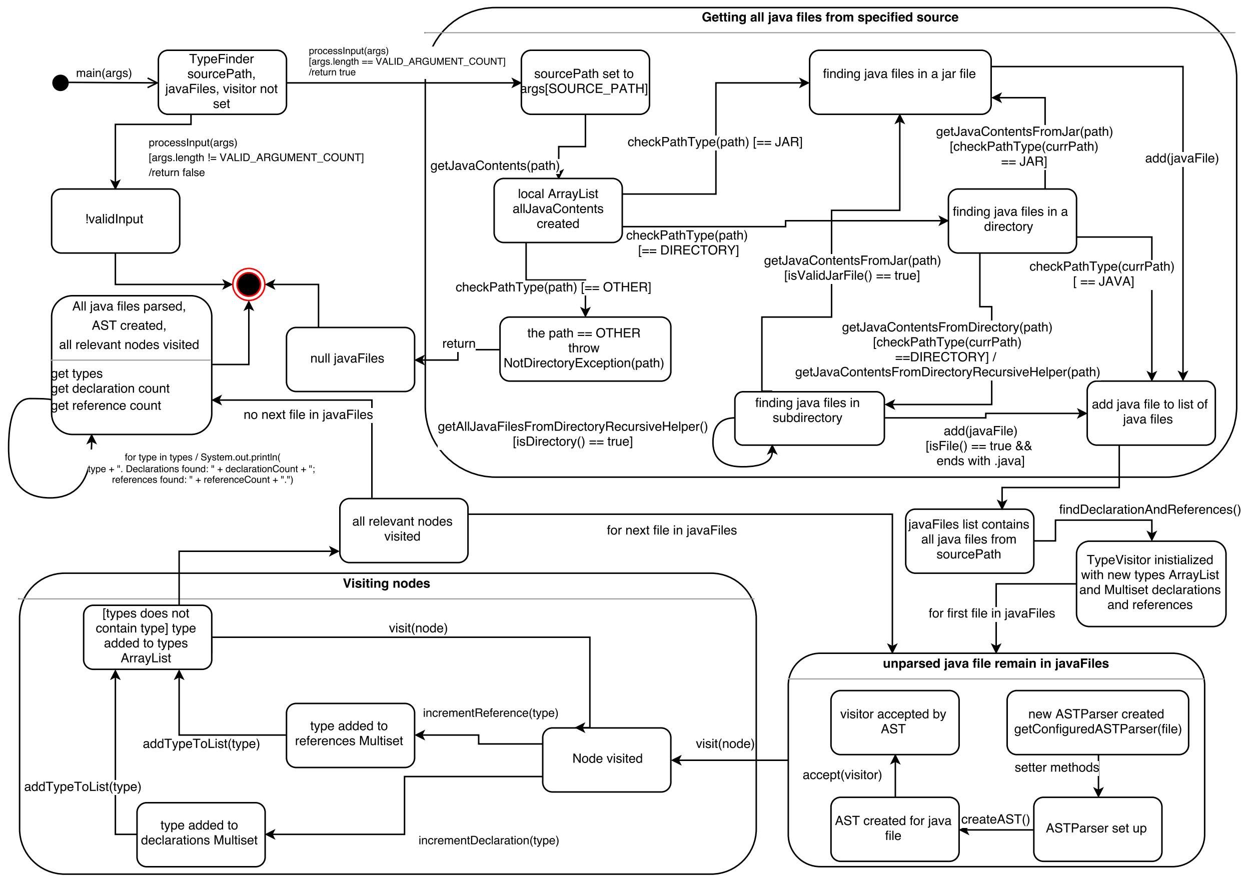
Getting all java files from specified source
main(args)
TypeFinder sourcePath, javaFiles, visitor not set
processInput(args) [args.length == VALID_ARGUMENT_COUNT] /return true
sourcePath set to args[SOURCE_PATH]
finding java files in a jar file
processInput(args) [args.length != VALID_ARGUMENT_COUNT] /return false
!validInput
getJavaContents(path)
checkPathType(path) [== JAR]
getJavaContentsFromJar(path) [checkPathType(currPath) == JAR]
add(javaFile)
local ArrayList allJavaContents created
finding java files in a directory
checkPathType(path) [== DIRECTORY]
getJavaContentsFromJar(path) [isValidJarFile() == true]
checkPathType(currPath) [== JAVA]
checkPathType(path) [== OTHER]
All java files parsed, AST created, all relevant nodes visited
get types
get declaration count
get reference count
null javaFiles
return
the path == OTHER throw NotDirectoryException(path)
getJavaContentsFromDirectory(path) [checkPathType(currPath) ==DIRECTORY] / getJavaContentsFromDirectoryRecursiveHelper(path)
no next file in javaFiles
finding java files in subdirectory
add java file to list of java files
getAllJavaFilesFromDirectoryRecursiveHelper() [isDirectory() == true]
add(javaFile) [isFile() == true && ends with .java]
for type in types / System.out.println(type + ". Declarations found: " + declarationCount + "; references found: " + referenceCount + ".")
all relevant nodes visited
for next file in javaFiles
javaFiles list contains all java files from sourcePath
findDeclarationAndReferences()
TypeVisitor initialized with new types ArrayList and Multiset declarations and references
Visiting nodes
[types does not contain type] type added to types ArrayList
visit(node)
for first file in javaFiles
unparsed java file remain in javaFiles
type added to references Multiset
incrementReference(type)
visitor accepted by AST
new ASTParser created getConfiguredASTParser(file)
addTypeToList(type)
Node visited
visit(node)
accept(visitor)
setter methods
addTypeToList(type)
AST created for java file
createAST()
ASTParser set up
type added to declarations Multiset
incrementDeclaration(type)

Figure 4: The state of the program in finding declaration and reference counts.

4 Explanation

4.1 Program description

This program finds all Java types all .java files in a directory and all subdirectories (recursively) or .jar file and outputs to the console the type name and with the number of declarations and references of each type. It makes use of classes in the org.eclipse.core.dom package such as the ASTParser in order to parse a .java file, from which an AST is created. The declarations and references of Java types are found visiting specific nodes of the AST through an ASTVisitor.

4.2 Usage

Run the Java `TypeFinder.class` file through command line:

```
java TypeFinder <path of directory or jar>
```

4.3 Structure

The UML diagram shows the important components of the software. Most of the methods in the Type-Finder class were represented in the diagram since they drive the software during execution. The classes JavaRetriever, JavaFile, File, and FileMangaer are used to read files and get their contents so that we can parse the files. Because of that, it was essential to represent those. Multiset<T> was used to store the declarations and references found. Therefore, the TypeVisitor class depends on it. 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.

4.4 Sequence Diagram

The sequence diagram for this software has shown in two separate diagrams. The first one depicts an overall view of how the software works. It first start by receiving the argument from the user which is the pathname of a directory or JAR file and check if the arguments is valid. Then, it calls an instance of JavaRetriever class which has the responsibility of iteration through the directory or JAR files recursively and reading all Java files. After that, it enters into a loop which creates ASTParser for each Java files, sets the configuration and create AST as CompilationUnit. Following from last step, it creates a visitor to visit each node of the AST and count the number of declaration and references for each node and finally, print the result for all types on the console. The second diagram show in more detail how the software recursively goes through the directory or JAR files of given pathname. It checks the pathname of interest to find its type, and cording to that, if it's a directory, a JAR file or a Java file takes the appropriate action. We used abstraction and we did not show all the details in the diagram specifically the steps that it takes to read the content of files to make easier for the client to understand the functionality of the software.

4.5 State diagram

The diagram is divided into three major states: 1) Getting java files, where the states are based on the path type (directory, jar, java, other) that the program is looking at. 2) Parsing Java file, where states are based on setting up/using ASTParser and creating AST. 3) Visiting nodes for each AST created to get declarations and references of types. The program reaches the end after all files are parsed, ASTs created for each parsed file, all relevant nodes visited for each AST, and all types with declaration and reference count are printed.