SENG300-Iteration2

I. Running the program

Note: # SENG300-Iteration2 has been tested to run on MacOS and Windows on Eclipse

Steps.
1. Import the project into Eclipse
2. Set arguments to a *Directory* and a *Java Type* (right-click on project > Run As > Run Configurations)
3. Run the program

II. Developer Info

Group: 16

Members:
1)Andrew Jamison - 10132190
2)Bryan Lam -
3)Zachariah Albers - 10175133
4) Lisa Cho - 10108678
5) Tariq Rahmani –

See git-log for information regarding commits. Further information can be found on https://github.com/zachalbers/SENG300-Iteration1

III. Design

A. Purpose of SENG300-Iteration2 & Functionality

The purpose of this program is to be able to count declaration and reference type of java types in a directory, both provided by the user,
of java files recursively. The program is designed so that the directory and java type is provided by the user through the command line/terminal.

Project Iteration 1: Summary

Structural Diagram:

- We chose to include the package org.eclipse.JDT.core.dom in our model and chose to include the skelotons of these classes; ASTParser, ASTVisitor and ASTNode. Other classes were abstracted out.
- The Class TypeFinder makes use of these classes since its goal is to parse all java files in a given directory and generate the correct output to the console.
- The relationship between from TypeFinder to ASTParser has a multiplicity from 0..* since it'll have to parse multiple java files and therefore may need multiple parsers. Or there could be zero parsers given no java files are present.
- The relationship between TypeFinder and ASTNode also has a multiplicity from 0..* as there could be no java files hence no nodes or there could be multiple nodes for one java file.
- We chose to abstract the subclasses of all classes in org.eclipse.JDT.core.dom such as ASTParser, ASTVisitor, ASTNode and CompilationUnit as we believe the relationship were more important to display rather than risking cluttering our model.
- ASTNode only has one method which TypeFinder depends on to traverse.
- The relationship between ASTNode and ASTVisitor is dependency as they both are used as parameters in both
- TypeFinder overloads methods of ASTVisitor. The methods in ASTVisitor are the methods being overloaded by TypeFinder. They allow us to perform operations upon visiting certain nodes of the java file represented as an AST.
- TypeFinder can throw a IOException which has a dependency relationship to java.lang.exception. This class was also abstracted out and only the skeleton of the class was shown.

Sequential Diagram:

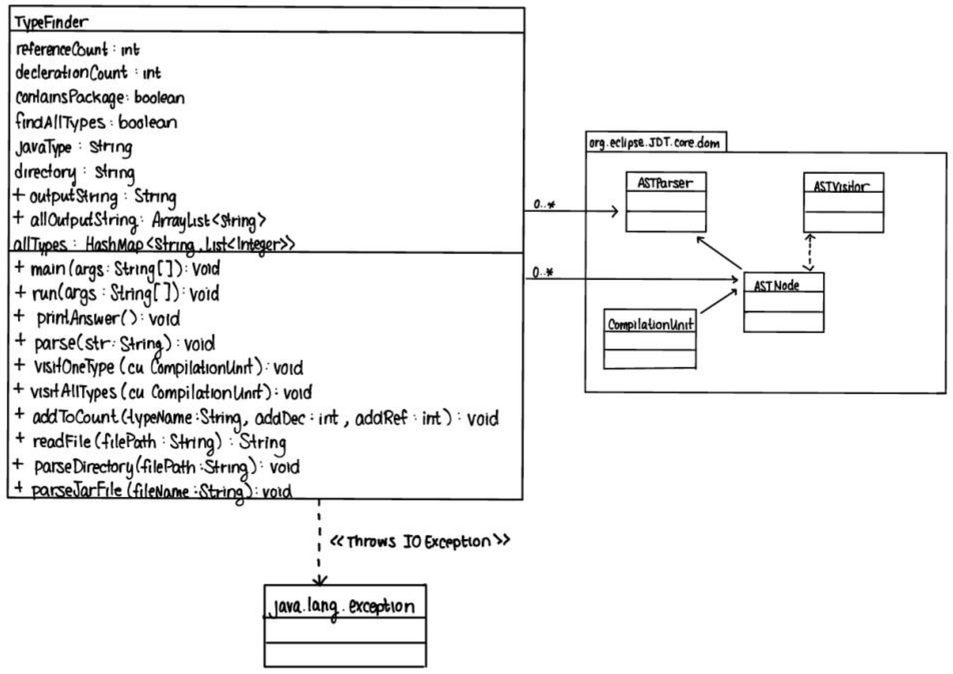
- We decided to use a sequential diagram to model a usage scenario.
- Shaded arrow heads represents synchronous messages (waits for return). Asynchronous messages are solid open arrow heads and dotted lines are returns.
- A user provides an existing directory and a java type – this the user input.
- TypeFinder makes the call to parseDirectory() using directory as a parameter. For each java file in the directory, it will make a call to AST parser and parse it as an AST. TypeParser will recursively call until it is not a directory anymore.
- Note that we chose not to include the newParser() method since it is used to instantiate the class and we do not normally care what caused these objects to exist, they just do from the beginning of this model.
- We also chose to abstract the initialization of the ASTParser. Things such as setSource, setResolvedBindings and setEnvironment were left out as to not clutter the diagram.
- TypeFinder then traverses the AST starting from the root node and decides whether to increases reference counts and declaration counts depending if the java type provided by the user matches with the node.

State Diagram:

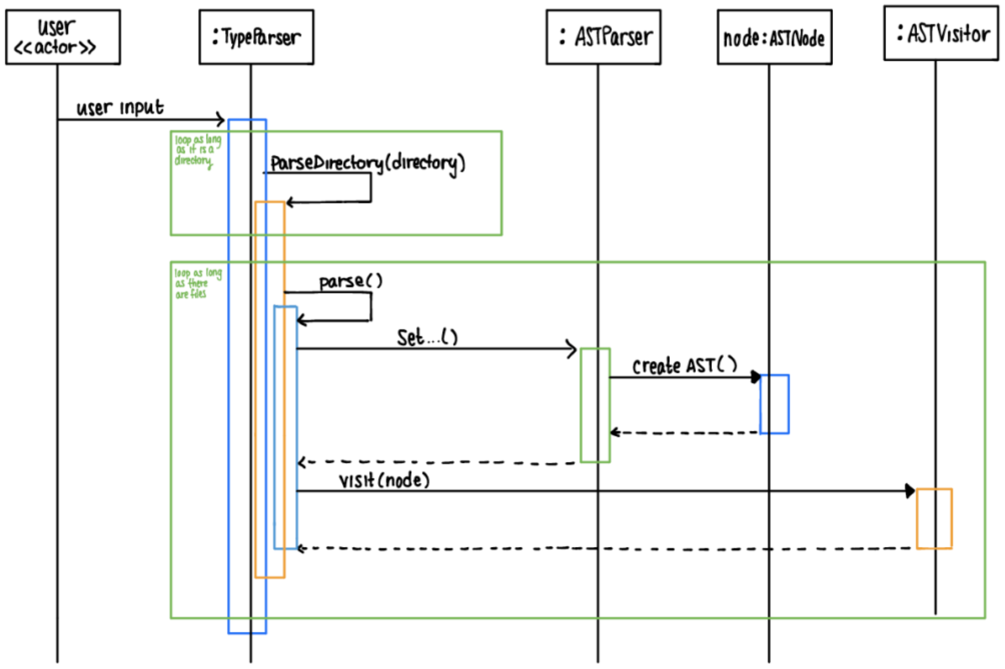
- We chose to use a state diagram to model the different states of TypeFinder.

- TypeFinder starts in the starting state where it waits for user input. If the user input length is 1 it enters the find all types state. If the user input is 2 it enters the state where it looks for a specific type indicated by the user. Any other input that are not caught by the first two, will enter an illegal argument exception state.
- If it enters the first two state mentioned above where the input length is either 1 or 2, it will enter a series of states to obtain the directories and the files that are in the directories. If the directory contains a directory, it will recursively go back to the state to set the directory and list the files.
- Once it gets a file path, it will then diverge into different states according to what type of file it is. If it is a .jar file, it will keep recursively unpack until a .java file is found.
- If it is a .java file, it can now enter the ASTParser states. This then get taken to states not shown due to simplicity where it involves the ASTParser in org.eclipse.JDT.core.dom and consecutive states.

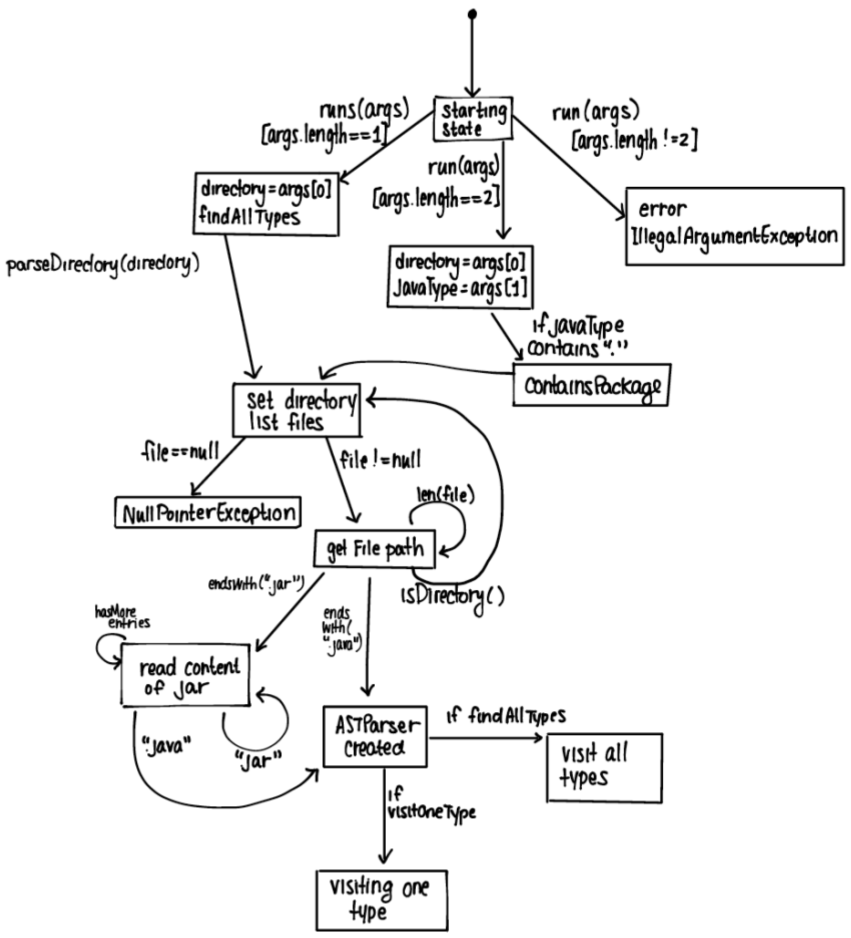
STRUCTURAL DIAGRAM


TypeFinder
referenceCount : int
declerationCount : int
containsPackage : boolean
findAllTypes : boolean
JavaType : String
directory : String
+ outputString : String
+ allOutputString : ArrayList<String>
allTypes : HashMap<String, List<Integer>>
+ main(args : String[]) : void
+ run(args : String[]) : void
+ printAnswer() : void
+ parse(str : String) : void
+ visitOneType(cu CompilationUnit) : void
+ visitAllTypes(cu CompilationUnit) : void
+ addToCount(typeName : String, addDec : int, addRef : int) : void
+ readFile(filePath : String) : String
+ parseDirectory(filePath : String) : void
+ parseJarFile(fileName : String) : void
0..*
0..*
org.eclipse.JDT.core.dom
ASTParser
ASTVisitor
ASTNode
CompilationUnit
<<Throws IO Exception>>
java.lang.exception

SEQUENCE DIAGRAM



STATE DIAGRAM


starting state
runs(args) [args.length==1]
run(args) [args.length==2]
run(args) [args.length !=2]
error IllegalArgumentException
directory=args[0] findAllTypes
directory=args[0] JavaType=args[1]
parseDirectory(directory)
if JavaType contains "."
ContainsPackage
set directory list files
file==null
file !=null
NullPointerException
get File path
len(file)
isDirectory()
endsWith(".jar")
ends with(".java")
hasMore entries
read content of jar
"Java"
"Jar"
ASTParser Created
if findAllTypes
visit all types
if visitoneType
visiting one type