SENG300-Iteration1

I. Running the program

Note: # SENG300-Iteration1 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:24

Members:

1)Andrew Jamison - 10132190
2)Bryan Lam -
3)Zachariah Albers - 10175133

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-Iteration1 & Functionality

1. 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. 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 these classes; ASTParser, ASTVisitor and ASTNode along with various subclasses belonging to ASTNode
- 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 it could be zero 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 and therefore no nodes and there could be multiple nodes for one java file.
- We chose to abstract the subclasses of ASTNode 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 ASTVistior 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.

Sequential Diagram:

- We decided to use a sequential diagram to model a usage scenario.
- Shaded in arrow head represents synchronous messages (waits for return). Asynchronous messages are solid open arrow heads and dotted are return messages.
- A user provides an existing directory and a java type.
- 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.
- 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 has 4 states, the starting state, a normal state and 2 error states (error1 and error2). The starting state is the state the program is in after being created. After using the run method it has the possibility of being in 3 states depending if there is an existing directory or not or if there were enough arguments provided.
- In the error states, you can make method calls too but you normally wouldn't want to since it may lead to an exception or an incorrect output.
- As a team, we are expecting to build on top of this software in the future, so even though this model is uninteresting as of now, there could be more things done in different states or have more states.