Source Dependency Graph

02242: Assignment 01

You have just been hired at a new place, and they have a huge legacy code base in Java. You can't figure out which pieces of the code base depends on which others. It would be nice to have a graph over the dependencies. Your assignment is to:

> Create a **source dependency graph** tool, that can create graphs over a Java source code repository.

You are allowed to use any regular language framework or lexer to get the job done, but refrain from using a parser library (more about this next time).

Definition 1 (Source Dependency Graph)**.** *A source dependency graph (V, E), is a directed graph where Java-files are nodes V and dependencies between them are edges $E \subseteq V \times V$.*

$$E = \{(a, b) \mid a, b \in V, \textit{the class a cannot compile without b}\}$$

Present your solution as a single slide in the slide deck. The slide should contain:

- ☐ one graph generated from your analysis;
- ☐ your wildest regular expression; and
- ☐ answer the following questions:
    1. Is your approach (in theory) sound and/or complete.
    2. How did you evaluate the correctness of your approach.
    3. Which corner cases did/didn't you handle.
    4. What cool feature did you think of which could improve the analysis.

Hints! Here are some hints and resources that might be nice to explore or make use of. You can break the problem down into four problems.

1. Find a project to analyse.

- Start here[1]. Pull-requests are appreciated.

2. Find classes in the project V.

   - Using glob patterns like `**/*.java`, (Java[2] and Python[3]) you can often search through an entire file structure.

3. For each class $v \in V$, figure out the dependencies to other classes.

   - There exist regular expression libraries in most languages, e.g., Java[4] and Python[5].
   - Look for the `import <package>;` statements, but don't forget about comments, strings and wildcards. And what about direct imports?
   - Consult the specification[6] if something is unclear.
   - The game is about covering as much of the corner cases as possible, your are not expected to cover all.

4. Render the graph.

   - Graphviz[7] is a great tool for quickly visualizing graphs.

```
% "source-graph.dot"
digraph SourceGraph {
  a [label="my.package.Factory"];
  b [label="my.other.Item"];
  a -> b;
}
% $ dot -Tpng source-graph.dot -o source-graph.png
```

References

[1] https://gitlab.gbar.dtu.dk/chrg/course-02242-examples/
[2] https://docs.oracle.com/javase/tutorial/essential/io/find.html
[3] https://docs.python.org/3/library/glob.html
[4] https://docs.oracle.com/javase/8/docs/api/java/util/regex/package-summary.html
[5] https://docs.python.org/3/library/re.html
[6] https://docs.oracle.com/javase/specs/jls/se20/html/jls-3.html#jls-3.10.1
[7] https://graphviz.org/

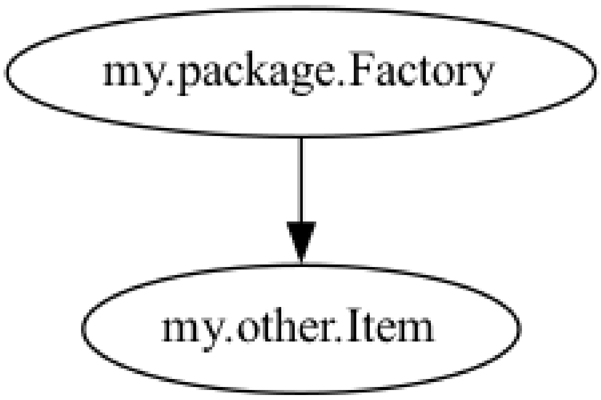

Figure 1: The output “source-graph.png”