Class Diagram

02242: Assignment 02

Realizing that regular expressions alone would not be enough to solve the problem alone, you sought to use more powerful tools, namely syntax trees. Your boss, inspired by your relative success, suggest if you could make the tool into a class diagram tool istread?

> Create a **Class Diagram** tool, that can create class diagrams over a Java source code repository.

You are allowed to use any parser to get the job done.

Definition 1 (Class Diagram)**.** *A Class Diagram (V, E, F), is a directed graph where Java classes are nodes V and dependencies between them are annotated edges $E \subseteq V \times A \times V$, where A is a set of edege annotations (see defintion of E). $F \subseteq V \times E$ is the fields of the classes*

$$E = \left\{ (a, x, b) \;\middle|\; a, b \in V, x = \begin{cases} \texttt{inheritance} & \textit{if a extends b} \\ \texttt{realization} & \textit{if a implements b} \\ \texttt{aggregation} & \textit{if b is a field in a} \\ \texttt{composition} & \textit{if b is non-static innerclass of a} \\ \texttt{dependency} & \textit{if a cannot compile without b} \end{cases} \right\}$$

the first statisfying condition from the top is accepted.

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

- ☐ one graph generated from your analysis;
- ☐ the craziest corner case you have found; 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.
   - Like last time
3. For each class $v \in V$, figure out the fields and annotate the dependencies to other classes.
   - Use the java tree-sitter bindings[2]
   - Consult the specification[3] 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[4] is a great tool for quickly visualizing graphs.
   - Consider the demo[5]

1 Getting started with Tree-Sitter

General instructions, getting started with using python or java to analyse java programs with tree-sitter. Install python3 and pip3, with `pip3` install `tree-sitter`. For working with java you should install Java 11 and `gradle`.

Download the tree-sitter grammar for java:

```
$ git clone https://github.com/tree-sitter/tree-sitter-java
```

Sometimes the version of the grammar is important, in which case you can `git switch` to the one that works.

```
$ pushd tree-sitter-java; git switch -c v0.19.0; popd
```

[1] https://lab.compute.dtu.dk/chrg/example-dependency-graphs
[2] https://github.com/serenadeai/java-tree-sitter
[3] https://docs.oracle.com/javase/specs/jls/se20/html/jls-3.html#jls-3.10.1
[4] https://graphviz.org/
[5] https://graphviz.org/Gallery/directed/UML_Class_diagram.html

1.1 Vagrant

If you use a windows or MacOS (intel) machine, one way to get started is to use Vagrant and Virtualbox to get everything going; see install instructions here[6]. It is uses python as the analysing language.

Add the following code as a `Vagrantfile` in your directory:

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/focal64"
  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    # if using python
    apt-get install -y python3-pip
    pip3 install tree_sitter
    # if using java
    apt-get install -y openjdk-11-jdk gradle
  SHELL
end
```

And run

```
vagrant up
vagrant ssh
```

1.2 Docker

Create a `Dockerfile`:

```
FROM ubuntu:focal
COPY . /vagrant
ENV LANG C.UTF-8
RUN apt-get update && apt-get install -y python3-pip
RUN pip3 install tree_sitter
```

You should be able to build it use it like this:

```
docker build -t tree-sitter .
docker container run -it tree-sitter
```

1.3 Python

You can now create a file `run.py`:

[6] https://developer.hashicorp.com/vagrant/tutorials/getting-started/getting-started-install

```
from tree_sitter import Language, Parser

FILE = "./languages.so" # the ./ is important
Language.build_library(FILE, ["tree-sitter-java"])
JAVA_LANGUAGE = Language(FILE, "java")

parser = Parser()
parser.set_language(JAVA_LANGUAGE)

with open("Test.java", "rb") as f:
    tree = parser.parse(f.read())

# the tree is now ready for analysing
print(tree.root_node.sexp())
```

And run it like this:

```
# in the vagrant box:
cd /vagrant
python3 ./run.py
```

Happy hacking.

1.4 Java

For instructions on how to set it up for java look at the following repository[7]

The bindings are lacking a little behind so you need to use version `v0.19.0` of the `https://github.com/tree-sitter/tree-sitter-java` repository. You can get this running the `git switch -c v0.19.0` command inside the repository.

I used the following `build.gradle` file

```
apply plugin: 'java'
apply plugin: 'application'

repositories {
    maven { url 'https://jitpack.io' }
}
sourceCompatibility = 1.8
targetCompatibility = 1.8
```

[7] https://github.com/serenadeai/java-tree-sitter

```
mainClassName = 'Test'

dependencies {
    implementation "com.github.serenadeai:java-tree-sitter:1.1.2"
}
```

And the file `src/main/java/Test.java`:

```
import ai.serenade.treesitter.*;
import java.nio.file.*;
import java.nio.charset.*;
import java.io.*;

public class Test {
    static {
        System.load("/vagrant/java-tree-sitter/libjava-tree-sitter.so");
    }
    public static void main(String[] main) throws Exception {
        try (Parser parser = new Parser()) {
            parser.setLanguage(Languages.java());
            Path p = FileSystems.getDefault().getPath("src/main/java/Test.java");
            String string = Files.readString(p, StandardCharsets.UTF_16LE);
            try (Tree tree = parser.parseString(string)) {
                Node root = tree.getRootNode();
                System.out.println(root.getChildCount());
            }
        }

    }
```

In total I ran the following commands:

```
$ git clone https://github.com/tree-sitter/tree-sitter-java
$ pushd tree-sitter-java; git switch -c v0.19.0; popd
$ git clone https://github.com/serenadeai/java-tree-sitter.git --recursive
$ pushd java-tree-sitter; JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64 ./build.py ..
$ gradle wrapper --gradle-version=5.1.1
$ ./gradlew run
```

References