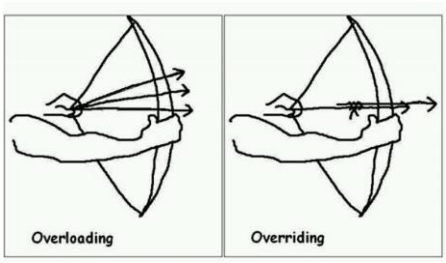
ASSIGNMENT 1

CS 474: Object-Oriented Languages and Environments / Fall 2020



Description

In this assignment, you will have to write a tool to detect methods that are overridden and overloaded given a set of Java classes as source code. You will use the Java Parser library[1] to access the structure of the Java source code provided to you.

For this assignment, you need to write a class that implements the interface `OverloadOverrideFinder`, as shown below. You will be provided with startup code that you cannot modify (except as described in this document), and you only have to submit an implementation to the interface described below.

```
interface OverloadOverrideFinder {
    Map<String, Map<String, Integer>> getOverloads(Map<String, ClassOrInterfaceDeclaration> classes);

    Map<List<String>, Set<String>> getOverrides(Map<String, ClassOrInterfaceDeclaration> classes);
}
```

Both methods take the same argument, a Java map from `String` to `ClassOrInterfaceDeclaration`.[2]

- The map contains all classes that you should process as entries.
- The **key** of each entry is the name of that class.
- The **value** of each entry is a JavaParser object that allows you to access the structure of the target class (i.e., all methods, fields, super classes, etc.)

[1] https://javaparser.org/
[2] https://www.javadoc.io/doc/com.github.javaparser/javaparser-core/3.3.2/com/github/javaparser/ast/body/ClassOrInterfaceDeclaration.html

The behavior of each method is as follows:

- `getOverloads`
  - List all the method overloads present in the Java classes passed as argument.
  - The return is a Java map from string to another map from string to integer.
  - The outer map has one entry with the name of each class present in the argument `classes`. The **key** is the name of each class.
  - The inner map has one entry per overloaded method, and how many times that method is overloaded. They **key** is the name of the method, the **value** is the number of overloads in that class.

  - Example: Calling `getOverloads` on the class below should return the map
    - `{ "A" -> { "overloadedMethod", 2 } }`

```
class A {
    void overloadedMethod() { }
    void overloadedMethod(int argument) { }
}
```

- `getOverrides`
  - List all the method overrides present in the Java classes passed as argument.
  - The return is a Java map from a list of string to a set of strings.
    - The **key** contains the name of the overridden method in the first position of the list, followed by the types of the arguments for that overridden method, in the same order they are declared in the Java source.
    - The **value** contains the name of all the classes that override that method in no particular order.
  - Examples: Calling getOverrides on the classes below should return the map
    - `{ [overridenAC, String, Top] -> { "A", "C" } }`

```
class Top {
    void overridenAC(String s, Top t) { }
    void notOverriden() { }
}
class A extends Top { void overridenAC(String a, Top t) { } }
class B extends A   { }
class C extends B   { void overridenAC(String a, Top t) { } }
```

Entry Point

You should create a new class, on a new file, where you will implement your solution. You should change method Main.getFinder so that it creates an instance of you're the class you added. You cannot change any other part of the code that is provided to you.

```
public abstract class Main {
    static Main getMain() {
        throw new Error("Not implemented");
    }
}
```

Due Date and Resubmission Policy

This assignment is due on **September 19 2020** (Saturday) at **5pm CST**. There is no late policy.

The code and date used for your submission is defined by the last commit to your Git repository.

To resubmit this assignment, your **original grade** (as defined by the autograder) should be **equal to or higher than 30%**. You can resubmit your assignment until **September 26 2020** (following Saturday) at **5pm CST**. Together with your resubmission, you will have to submit a written description of what you changed from the original submission (on Gradescope).

Bonus Points

This assignment has a total of **10% bonus points**, which you can earn by using Piazza as described in the syllabus. Your posts should be public, tagged with the `assignment1` label, and non-anonymous to the instructors to count towards the bonus.

Submission and Grading

This assignment is submitted through Github, and has an automatic grade component of 100%. You can check your current grade at any point by submitting your code and checking Travis. The automatic grade is determined by 10 tests, that will check if your project outputs the expected result. Each test is worth 10%.

Graduate students should also submit a video explaining their solution. For graduate students, each test is worth 9%, for a total of 90%, and the video is worth 10%. **The maximum length for the video is 5 minutes.** This video should be a screencast of their IDE open on the code submitted, and the student should highlight the code and narrate the purpose of the highlighted code. You can record such a video without installing any software by using the following website: https://screenapp.io/#/

The final grade for the assignment will be the grade of the original submission, for assignments without a resubmission; or the average between the original grade and the resubmission grade, for assignments with a resubmission. The grade of the original submission includes any bonus points.

Errors and Omissions

If you find an error or an omission, please post it on Piazza as soon as you find it.

Hardcoding and Academic Integrity

Any hardcoding will result in a 0% grade. Hardcoding is when you submit code that detects which test is being run, and simply outputs the expected result. For instance, detecting that test 22 is running, and replacing the usual execution of your submission with `System.out.println("expected result")`.

The academic integrity policy described in the syllabus applies to this assignment. You are responsible for writing all the code that you submit. We will use an automatic tool that detects plagiarism on all submitted code, and we will investigate all instances where plagiarism is more than likely.

Please refer to the syllabus for the full academic integrity policy.