Homework 1

CS 474 2019

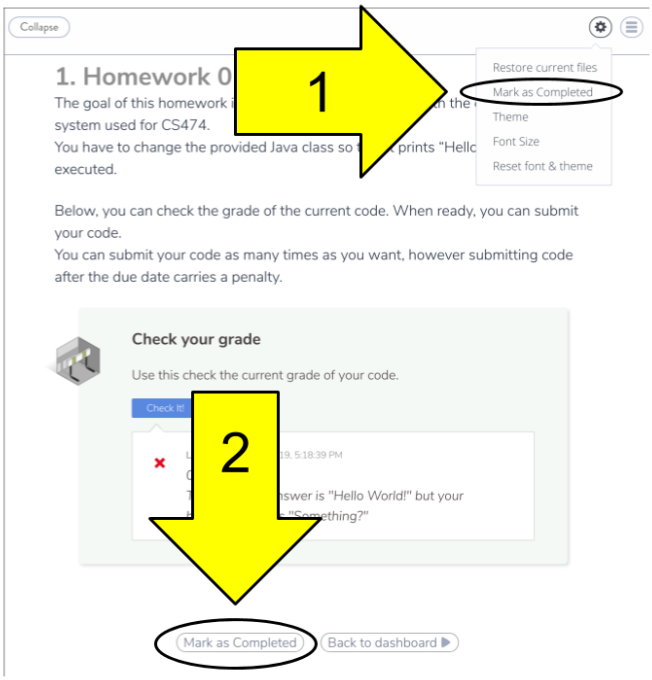
Description

In this homework, you will use a Java source parsing library to add methods `toString` to all the classes that do not have one already in a project. The methods you generate must print the fields declared in a particular order. For each field, your code should print the name of the field and the value for the current object on which `toString` is being invoked on. This document specifies the order and format of the fields and values.

Due Date and Late Policy

- This homework is due on **September 21** (Saturday) by **5pm CST**.
- Homeworks delivered by **September 22** (Sunday) by **5pm CST** will have a 10% penalty.
- Homeworks delivered by **September 23** (Monday) by **5pm CST** will have a 25% penalty.
- All homeworks started will be **marked as complete (i.e., delivered automatically)** by September 23 (Monday) at 5pm CST, with a 25% penalty.

To deliver a homework, you have to mark it as complete on Codio. The figure below shows the two ways to do that. Note that option (1) is preferred, as option (2) does not work reliably.



Java Parsing Library

For this project you will use the JavaParser library version 3.14.5. You cannot use any other libraries besides the basic Java Development Kit (JDK) and the JUnit testing library. All the libraries will be provided in your Codio environment.

You will be provided with plenty of code that uses JavaParser, and you will have to write very little new code for JavaParser.

JavaParser: https://javaparser.org/

Format of the Output

The `toString` method should generate a string with a single line (i.e., no line breaks) with the following format:

0 fields	{}
1 field	`{ name SEP value }`
2 or more fields	`{ name SEP value ; name SEP value }`

Where:

- `name` means the name of the field
- `SEP` is either:
  - Equals "=" for final fields
  - Colon ":" otherwise
- `value` is the value of the field (see below)
- Two or more fields are separated by a semi-colon: ";"
- Unless stated, there are no spaces

Name of Fields

Names should be formatted as follows:

```
visibility class . name
```

(name description continues on next page)

Where:

- `Visibility` one character as follows:
  - Plus "`+`" for public fields
  - Minus "`-`" for private fields
  - Pound/Hash "`#`" for protected fields
  - Space " " for default fields
- `class.` is only present for static fields, and it shows the full class name that declares the field including packages separated by a dot
- `name` is the declared name of the field
- Examples:
  - Public static field: `+edu.uic.cs474.MyClass.field`
  - Private instance field: `-parent`

Format of Values

Values should be formatted as:

- Primitive values:
  - int: The number (e.g., 10)
  - long: The number followed by L (e.g., 10L)
  - short: The number followed by S (e.g., 10S)
  - float: The number always with decimals (e.g., 10.0, 11.56)
  - double: Same as float followed by D (e.g., 10.0D, 11.56D)
  - char: The character surrounded by single quotes (e.g., 'a')
  - byte: The byte in hexadecimal starting with 0x (e.g., 0xDE)
- "`null full-type`" for null values, where the type is the full name of declared field type when that type is present in an import. For instance, if there is an import of `java.util.List` and a field is declared as "`List field`", you should print "`null java.util.List`"
- "`null declared-type`" for null values, where the type is the name of declared field type and there is no direct import for that particular type. For instance, if there is an import of "`java.util.*`" and a field is declared as "`List field`", you should print "`null List`"
- Strings should be surrounded with double quotes
- "`[] type`" for arrays of size zero, where type is the type of the elements of the array.
- "`[0: value]`" for arrays with one element
- "`[0: value,1: value,..]`" for arrays with multiple elements
  - Array values should be formatted as defined in this section
- The result of the existing/default `toString` method otherwise
- "..." for fields which value references the object itself
- The generated `toString` method must print only one line. All new line characters present in any value should be replaced by the sequence of characters "`\n`".
- This homework will not be evaluated with multi-dimensional arrays (e.g., `int[2][2]`)

Order of Fields

The fields should be ordered by the following rules, in the order given:

1. Static fields, then instance fields
2. Final fields, then non-final fields
3. Visibility:
    a. Public
    b. Protected
    c. Default
    d. Private
4. Name in alphanumeric order

Classes that already have their own `toString`

If a class already implements method `toString`, you should not generate any new method or change the existing method.

Submission and Grading

This homework should be submitted through Codio, and it is automatically graded. You can check your current grade at any point by pressing a button. The project is graded through 50 tests, that will check if your project outputs the expected result for a particular class. Each test is worth 2%, and you will be provided with 25 tests (i.e., 50%) from the start. You will have access to the name and expected/observed output for all 50 tests.

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 homework with `System.out.println("expected result")`.

(continues on next page)

The academic integrity policy described in the syllabus applies to this homework. 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.