Assignment 1 - SOEN 6611 - Winter 2020

About Submission

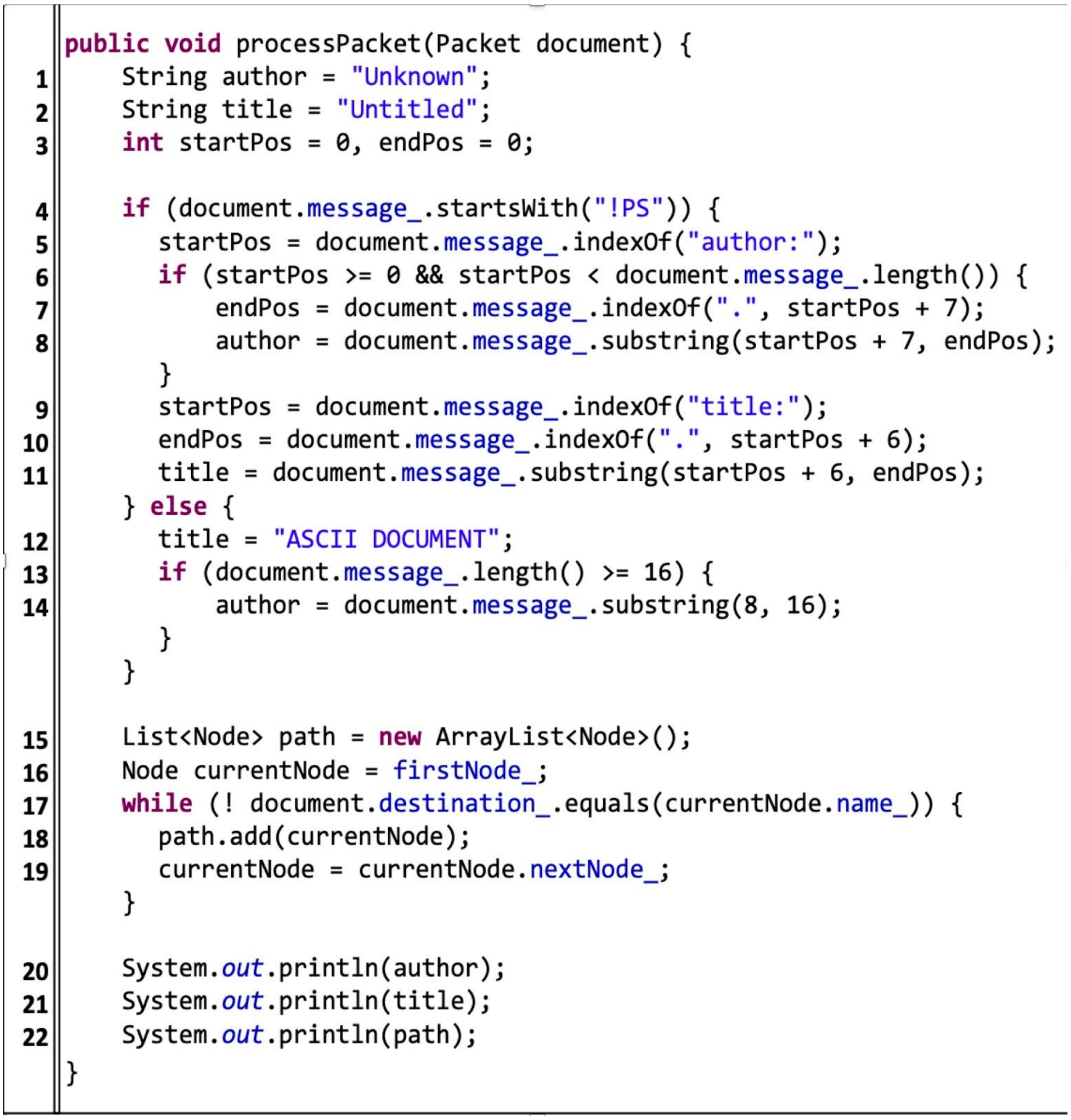
- You will write down your answers to Question 1, and 2in a pdf. The name of the pdf should be “<student id>-assignment-1.pdf”
- Question 3 is a coding problem, please refer to the submission section of Question 3 and follow the instructions.

Question 1 - A correct CFG - McCabe complexity All linearly independent paths	**2.0** 1.5 0.5 0.5
Question 2 - Example - Description	**1.0** 0.5 0.5
Question 3 - Correct program behaviours and output - Meet the submission requirements	**5.0** 4.0 1.0
Total	**8.0**

Question 1 - CFG

1) Create the control flow graph for the method shown in the figure below. Note the presence of the logical AND operator (&&) in statement 6. Use the line number in each node of CFG, e.g., use 1-3 to represent ‘1,2,3’. You may use draw.io to draw the control flow graph.

2) Based on the graph, compute the McCabe complexity metric and list all the linearly independent paths.



```java
public void processPacket(Packet document) {
    String author = "Unknown";
    String title = "Untitled";
    int startPos = 0, endPos = 0;

    if (document.message_.startsWith("!PS")) {
       startPos = document.message_.indexOf("author:");
       if (startPos >= 0 && startPos < document.message_.length()) {
           endPos = document.message_.indexOf(".", startPos + 7);
           author = document.message_.substring(startPos + 7, endPos);
       }
       startPos = document.message_.indexOf("title:");
       endPos = document.message_.indexOf(".", startPos + 6);
       title = document.message_.substring(startPos + 6, endPos);
    } else {
       title = "ASCII DOCUMENT";
       if (document.message_.length() >= 16) {
           author = document.message_.substring(8, 16);
       }
    }

    List<Node> path = new ArrayList<Node>();
    Node currentNode = firstNode_;
    while (! document.destination_.equals(currentNode.name_)) {
       path.add(currentNode);
       currentNode = currentNode.nextNode_;
    }

    System.out.println(author);
    System.out.println(title);
    System.out.println(path);
}
```

Question 2 - McCabe

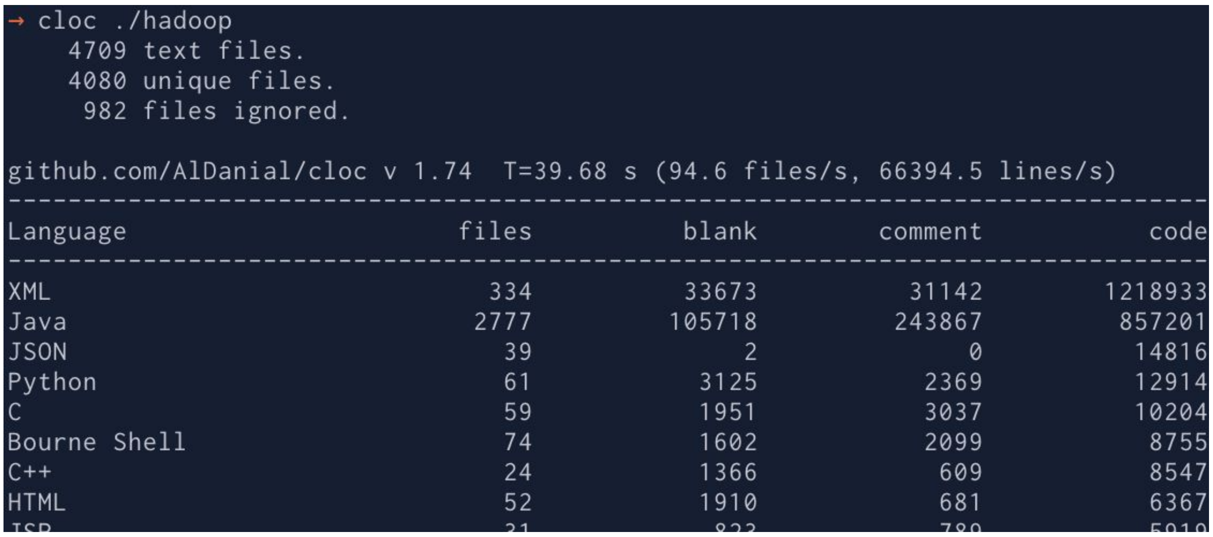
For a program, it is possible that its McCabe complexity is higher than the number of possible execution paths in the program. Provide an example to explain under which scenario the statement on McCabe is true. Note that you need to provide a code example, a short description such as drawing a simple CFG and list the possible execution paths.

Question 4 - Calculating LOC

Description

You are asked to implement a tool (in **Java**) that measures the physical length of a Java project. **Source lines of code** (**SLOC**), also known as **lines of code** (**LOC**), is a common software metric to indicate the efforts for developing and maintaining software.

There are several tools (e.g. *CLOC*, *pygount*, *Understand*, etc.) that can measure LOC. Below is an example report that is generated by CLOC for the Hadoop project. The report shows the number of files, the number of blank lines, the number of comments, and the number of code lines for all the programming languages in Hadoop.



```
→ cloc ./hadoop
    4709 text files.
    4080 unique files.
     982 files ignored.

github.com/AlDanial/cloc v 1.74  T=39.68 s (94.6 files/s, 66394.5 lines/s)
--------------------------------------------------------------------------------
Language                     files          blank        comment           code
--------------------------------------------------------------------------------
XML                            334          33673          31142        1218933
Java                          2777         105718         243867         857201
JSON                            39              2              0          14816
Python                          61           3125           2369          12914
C                               59           1951           3037          10204
Bourne Shell                    74           1602           2099           8755
C++                             24           1366            609           8547
HTML                            52           1910            681           6367
```

Information to Count

Here, you need to implement a tool that can count a total of 5 kinds of information. The descriptions of the information are listed below:

- *The total number of Java files*. Given a directory, the tool needs to count how many Java files in this directory, if given a directory. Here, we assume the files end with *.java* are Java files.
- *The total number of unique Java files*. Given a directory, the tool needs to count how many unique Java files in this directory. Uniqueness means that the contents of the files are different. If there are several files with the same content, only count once.
- *The total number of blank lines*. Given a directory or a Java file, the tool needs to count the total number of blank lines in all the unique Java files (If there are

several files with the same content, only count one file.). Blank lines are the lines that are neither the comment lines or code lines.

- *The total number of comment lines.* The tool needs to count the total number of comment lines in all the unique Java files (If there are several files with the same content, only count one file.). Note: if the line does not start with optional whitespace followed by // or between /* and */, then it is not a comment line. For example, the following line is not a comment line:
  `for (i = 0; i < 100; i++) printf("hello"); /* How many lines of code is this? */`
- *The total number of code lines*. The tool needs to count the total number of code lines in all the unique Java files (If there are several files with the same content, only count one file.). Comment lines do not count as code lines.. It can be a curly brace, semicolon, etc.

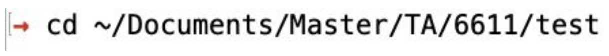
Requirements

Please make sure your project meets the following requirements:

- Your should implement the tool in Java.
- You need to implement the core code logics yourself. You may use third-party libraries however such libraries should not count LOC *already*. **If you are not sure whether you are allowed to use one certain library, please contact the POD for clarifications.**
- You need to use Maven to manage your project and libraries.
  - By typing `*mvn install*` in the terminal/command line, an executable JAR with dependencies named *assignment-1.0.jar* will be generated in the *target* folder.
    Hints: 1. You can use Maven Assembly Plugin to generate the executable JAR. 2. In order to generate the JAR with the given name, modify <artifactId> and <version> of your project in maven configuration to *assignment and 1.0.*
- We will test your tool with latest Maven (3.6.3) and Java (13.0.2) versions, so please make sure your tool will work fine in this environment.
- The program should take one parameter, which can be either a directory or a file and the output should be the 5 counts, separated by -. For example, the output can be 1-1-1-1-1, it means there is only one java file, one unique java file and one blank line, one comment line and one line of code. If the parameter is a directory, the program should count all the java files in that directory.

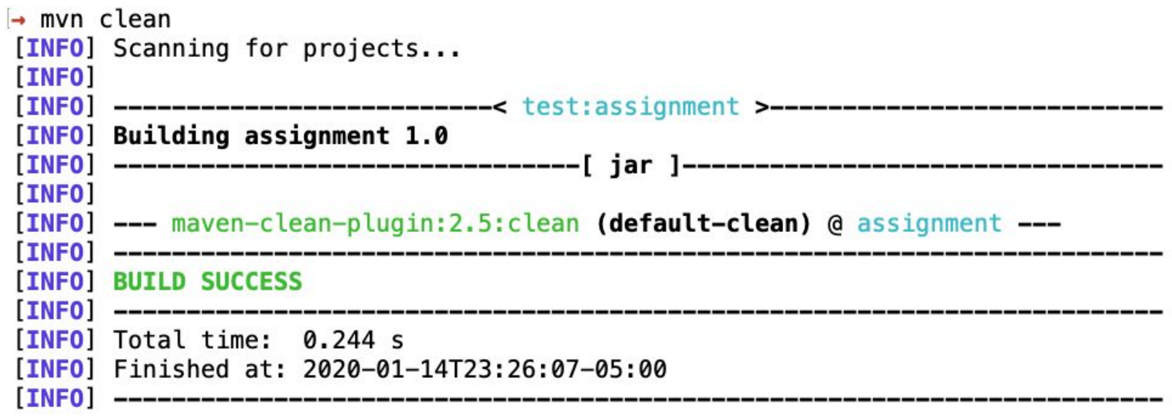
Here is an example for the requirements. We use command lines to run the following steps.

1. Change the current working directory to the root directory (The directory should be changed to the root directory of your program).



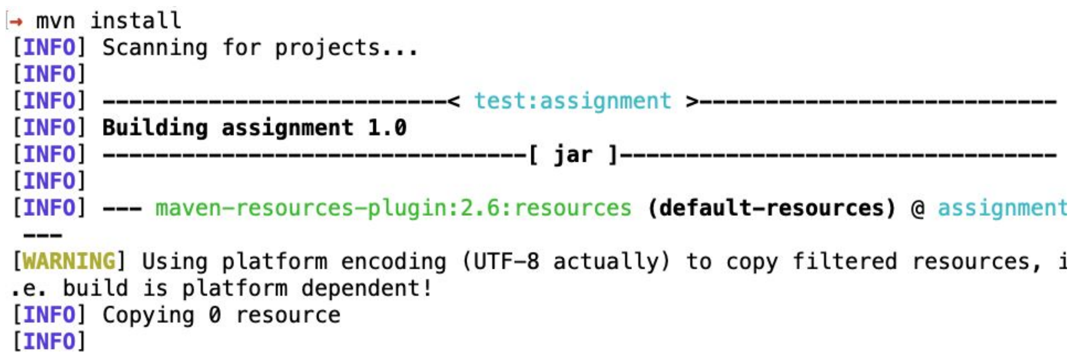
```
➜ cd ~/Documents/Master/TA/6611/test
```

2. Clean the project and generate an executable JAR with dependencies.
   a. `mvn clean`



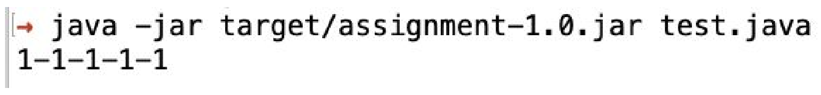
```
➜ mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------------< test:assignment >--------------------------
[INFO] Building assignment 1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ assignment ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.244 s
[INFO] Finished at: 2020-01-14T23:26:07-05:00
[INFO] ------------------------------------------------------------------------
```

   b. `mvn install`



```
➜ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------------< test:assignment >--------------------------
[INFO] Building assignment 1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ assignment
 ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i
.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
```

3. Now, there should be a new folder called *target* and the executable JAR with dependencies named *assignment-1.0.jar* should be inside the *target* folder.
4. Now, run the jar file and get the results.



```
➜ java -jar target/assignment-1.0.jar test.java
1-1-1-1-1
```

5. We would use the result of CLOC as the ground truth to evaluate your tool. For example, given the project common-io, the output should be
   `248-248-6017-21839-30455`

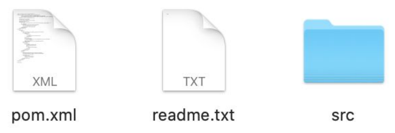
Note: we will run your source code automatically, please make sure that your code can work like the given example.

Here are some useful links:

1. https://maven.apache.org/
2. https://github.com/AlDanial/cloc
3. http://maven.apache.org/plugins/maven-assembly-plugin/

Submission

You need to submit both the project and a readme.txt, compress them in a zip file with the name *assignment-1.zip* and upload it to moodle (“assignment1-code”). In “readme.txt” you should explain the design of your project. Once assignment-1.zip is unzipped, the directory structure should be as follows.


XML
TXT
pom.xml
readme.txt
src