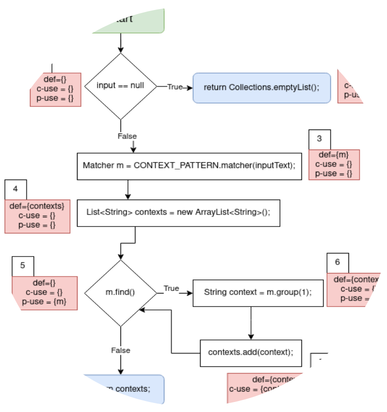
Assignment 8

Dataflow Testing

In this assignment we perfomed Dataflow testing to analyze the flow between a variable definition and its uses. Also, even if Jacoco plugin is only used to check for the line and branch coverages, we used it to make sure that a path of a variable was being exercised, by commenting the other related tests, and thus isolating it.

`List<String> parse(String inputText)`

This function from the `ContextParser` class is responsible for parsing the contexts defined in a task text. It defines and uses the variables `m`, `contexts` and `context`, inside a loop. Its dataflow is represented by the following diagram:



Flow Analysis

Variable **m**: | pair id | def | use | path | | -------- | -------- | -------- | -------- | | 1 | 3 | (5, True)| <3, 4, 5, 6>| | 2 | 3 |(5, False)| <3, 4, 5, 8>|

- All-def paths: 1, 2
- All-c-uses: -
- All-p-uses: 1, 2
- All-uses: 1, 2

Variable **contexts**: | pair id | def | use | path | | -------- | -------- | -------- | -------- | | 1 | 4 | 7 | <4, 5, 6, 7>| | 2 | 4 | 8 | <4, 5, 8>| | 3 | 7 | 7 | <7, 7>| | 4 | 7 | 8 | <7, 5, 8>|

- All-def paths: 1, 2, 3
- All-c-uses: 1, 2, 3, 4
- All-p-uses: -

- All-uses: 1, 2, 3, 4

Variable **context**:

pair id	def	use	path
1	6	7	<6, 7>

- All-def paths: 1
- All-c-uses: 1
- All-p-uses: -
- All-uses: 1

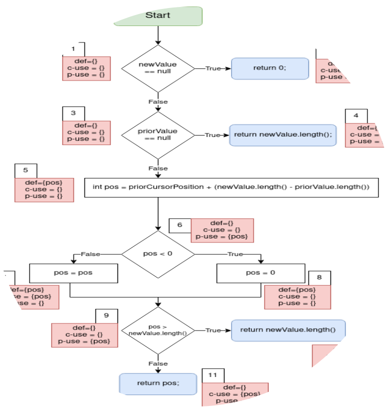
Unit tests

Because each variable is closely associated, only 3 tests were created in order to reduce redundancy.

- **oneContextInText()**: Tests pair 1 of the variable **m**, path 1 of **context**, and paths 1, 2, 3 of **contexts**.
  - Passed.
- **multipleContextsInText()**: Tests paths 1 and 2 of **m**, and particularly path 4 of **contexts**.
  - Passed.
- **noContextInText()**: Tests pair 2 of the variable **m**.
  - Passed.

`int calculate(int priorCursorPosition, String priorValue, String newValue)`

This utility method calculates the cursor position based on 3 inputs, and defines a variable **pos**, that is updated and returned at the end of the its flow.



Variable **pos**:

pair id	def	use	path
1	5	(6, True)	<5, 6, 8>
2	5	(6, False)	<5, 6, 7>
3	7	7	<7, 7>
4	7	(9, True)	<7, 9, 10>
5	7	(9, False)	<7, 9, 11>
6	8	(9, True)	<8, 9, 10>
7	8	(9, False)	<8, 9, 11>

- All-defs: 1, 2
- All-c-uses: 3

- All-p-uses: 1, 2, 4, 5, 6, 7
- All-uses: 1, 2, 3, 4, 5, 6, 7

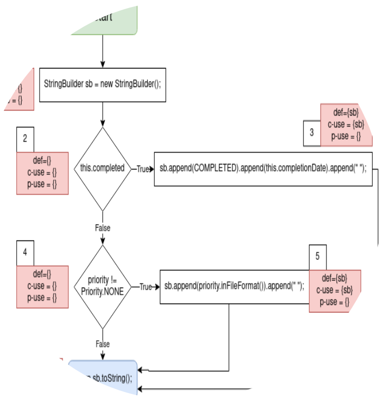
Unit tests

- negativeCursorPosition(): tests a **pos** with negative value. This includes the pair 1.
    - Passed
- positiveCursorPositionGreaterThanNewValueLength(): tests cursor **pos** with a value that is higher than the length of the related string. Tests the pairs 2, 3, 4 and 6.
    - Passed
- positiveCursorPositionLowerThanNewValueLength(): In this case, **pos** is lower than the new string length. Tests the pairs 2, 3, 5 and 7.
    - Passed

Also, it is impossible to reach the node 10 from the pair 1 when going to other paths afterwards.

`String inFileFormatHeaderNoDate()`

This method from the `Task` class, return a string with the tasks header. This header could be the priority or the completion mark. It is not clear if both should be returned, but looking to the code, it is not possible, so we assumed only one field from priority/completion is returned. The only variable used is `sb`, that builds the returning string.



StringBuilder sb = new StringBuilder();
2
def={}
c-use = {}
p-use = {}
this.completed
True
sb.append(COMPLETED).append(this.completionDate).append(" ");
3
def={sb}
c-use = {sb}
p-use = {}
False
4
def={}
c-use = {}
p-use = {}
priority != Priority.NONE
True
sb.append(priority.inFileFormat()).append(" ");
5
def={sb}
c-use = {sb}
p-use = {}
False
sb.toString();

Variable **sb**: | pair id | def | use | path | | -------- | -------- | -------- | -------- | | 1 | 1 | 3 | <1, 2, 3>| | 2 | 1 | 5 | <1, 2, 4, 5>| | 3 | 1 | 6 | <1, 2, 4, 6>| | 4 | 3 | 3 | <3, 3> | | 5 | 3 | 5 | <3, 6> | | 6 | 5 | 5 | <5, 5> | | 7 | 5 | 6 | <5, 6> |

- All-defs: 1, 2, 3, 4, 7
- All-c-uses: 1, 2, 3, 4, 5, 6, 7, 8
- All-p-uses: -
- All-uses: 1, 2, 3, 4, 5, 6, 7, 8

Unit tests

- **inFileFormatHeaderNoDateTaskCompleted()**: Task with the completion mark. Tests pairs 1, 4, 5 and 6.

    - Passed
- **inFileFormatHeaderNoDateTaskWithPriority()**: Task with the priority specified. Tests pairs 2, 6 and 7
    - Passed
- **inFileFormatHeaderNoDateTaskWithoutCompletionAndPriority()**: Task without the completion mark and priority. Tests pair 3.
    - Passed
- **inFileFormatHeaderNoDateTaskCompletedAndWithPriority()**: Task with both the completion mark and the priority. Tests pair 5.
    - Passed