

SOEN 345 - Software Testing, Verification, and
Quality Assurance
Assignment 1



Ali Hanni - 40157164

*Gina Cody School of Computer Science and Software Engineering
Concordia University, Montreal, QC, Canada*

Winter 2022

Question 1

SOENTutorials

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|-------------------------------|---|------|---|------|--------|------|--------|-------|--------|---------|--------|---------|
| soen.tutorial |  | 100% |  | 100% | 0 | 6 | 0 | 8 | 0 | 4 | 0 | 2 |
| Total | 0 of 44 | 100% | 0 of 4 | 100% | 0 | 6 | 0 | 8 | 0 | 4 | 0 | 2 |























Method Coverage: 4/4 → 100%

Line Coverage: 8/8 → 100%

Branch Coverage: 4/4 → 100%

Question 2

Apache Commons IO

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|------|---|------|--------|-------|--------|-------|--------|---------|--------|---------|
| org.apache.commons.io.input |  | 88% |  | 87% | 147 | 876 | 224 | 1,898 | 55 | 444 | 0 | 51 |
| org.apache.commons.io |  | 93% |  | 89% | 157 | 1,187 | 183 | 2,297 | 49 | 558 | 0 | 29 |
| org.apache.commons.io.file |  | 76% |  | 63% | 90 | 289 | 110 | 470 | 23 | 166 | 3 | 21 |
| org.apache.commons.io.output |  | 88% |  | 85% | 64 | 422 | 122 | 1,064 | 30 | 295 | 0 | 35 |
| org.apache.commons.io.input.buffer |  | 40% |  | 35% | 43 | 67 | 62 | 132 | 11 | 25 | 1 | 3 |
| org.apache.commons.io.filefilter |  | 92% |  | 87% | 48 | 386 | 46 | 655 | 16 | 247 | 1 | 29 |
| org.apache.commons.io.monitor |  | 88% |  | 84% | 29 | 126 | 30 | 239 | 13 | 66 | 0 | 4 |
| org.apache.commons.io.function |  | 88% |  | n/a | 2 | 23 | 5 | 24 | 2 | 23 | 0 | 2 |
| org.apache.commons.io.serialization |  | 97% |  | 100% | 0 | 29 | 2 | 56 | 0 | 19 | 0 | 4 |
| org.apache.commons.io.comparator |  | 99% |  | 93% | 3 | 66 | 1 | 132 | 0 | 42 | 0 | 10 |
| org.apache.commons.io.file.spi |  | 97% |  | 75% | 2 | 11 | 1 | 17 | 0 | 7 | 0 | 1 |
| Total | 3,249 of 29,969 | 89% | 478 of 3,176 | 84% | 585 | 3,482 | 786 | 6,984 | 199 | 1,892 | 5 | 189 |





















Method Coverage: 1693/1892 → 89.5%

Line Coverage: 6198/6984 → 88.7%









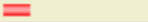
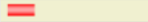












Branch Coverage: 2698/3176 → 84.9%

Question 3

org.apache.commons.io.comparator

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|--|---|------|---|------|--------|------|--------|-------|--------|---------|--------|---------|
| ReverseFileComparator |  | 85% |  | 50% | 1 | 4 | 1 | 7 | 0 | 3 | 0 | 1 |
| CompositeFileComparator |  | 100% |  | 100% | 0 | 12 | 0 | 32 | 0 | 5 | 0 | 1 |
| SizeFileComparator |  | 100% |  | 87% | 2 | 13 | 0 | 23 | 0 | 5 | 0 | 1 |
| ExtensionFileComparator |  | 100% |  | 100% | 0 | 6 | 0 | 16 | 0 | 5 | 0 | 1 |
| NameFileComparator |  | 100% |  | 100% | 0 | 6 | 0 | 14 | 0 | 5 | 0 | 1 |
| PathFileComparator |  | 100% |  | 100% | 0 | 6 | 0 | 14 | 0 | 5 | 0 | 1 |
| LastModifiedFileComparator |  | 100% |  | 100% | 0 | 5 | 0 | 9 | 0 | 3 | 0 | 1 |
| DirectoryFileComparator |  | 100% |  | 100% | 0 | 5 | 0 | 5 | 0 | 4 | 0 | 1 |
| AbstractFileComparator |  | 100% |  | 100% | 0 | 6 | 0 | 8 | 0 | 4 | 0 | 1 |
| DefaultFileComparator |  | 100% |  | n/a | 0 | 3 | 0 | 4 | 0 | 3 | 0 | 1 |
| Total | 5 of 582 | 99% | 3 of 48 | 93% | 3 | 66 | 1 | 132 | 0 | 42 | 0 | 10 |

Apache Commons IO

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|-------------------------------------|---|------|---|------|--------|-------|--------|-------|--------|---------|--------|---------|
| org.apache.commons.io |  | 2% |  | 1% | 1,166 | 1,187 | 2,229 | 2,297 | 539 | 558 | 26 | 29 |
| org.apache.commons.io.input |  | 0% |  | 0% | 876 | 876 | 1,898 | 1,898 | 444 | 444 | 51 | 51 |
| org.apache.commons.io.output |  | 0% |  | 0% | 422 | 422 | 1,064 | 1,064 | 295 | 295 | 35 | 35 |
| org.apache.commons.io.filefilter |  | 0% |  | 0% | 386 | 386 | 655 | 655 | 247 | 247 | 29 | 29 |
| org.apache.commons.io.file |  | 0% |  | 0% | 289 | 289 | 470 | 470 | 166 | 166 | 21 | 21 |
| org.apache.commons.io.monitor |  | 0% |  | 0% | 126 | 126 | 239 | 239 | 66 | 66 | 4 | 4 |
| org.apache.commons.io.input.buffer |  | 0% |  | 0% | 67 | 67 | 132 | 132 | 25 | 25 | 3 | 3 |
| org.apache.commons.io.serialization |  | 0% |  | 0% | 29 | 29 | 56 | 56 | 19 | 19 | 4 | 4 |
| org.apache.commons.io.function |  | 0% |  | n/a | 23 | 23 | 24 | 24 | 23 | 23 | 2 | 2 |
| org.apache.commons.io.file.spi |  | 0% |  | 0% | 11 | 11 | 17 | 17 | 7 | 7 | 1 | 1 |
| org.apache.commons.io.comparator |  | 99% |  | 93% | 3 | 66 | 1 | 132 | 0 | 42 | 0 | 10 |
| Total | 29,119 of 29,969 | 2% | 3,110 of 3,176 | 2% | 3,398 | 3,482 | 6,785 | 6,984 | 1,831 | 1,892 | 176 | 189 |

Method Coverage: 61/1892 → 3.2%

Line Coverage: 199/6984 → 2.8%

Branch Coverage: 66/3176 → 2.1%

The *pom.xml* file used to run modify surefire parameters in order to exclude certain packages is joined with the report as *q3.xml*.

Question 4

Before using OClver, we first need to modify *pom.xml* file in order to add the plugin to the project. In order to do so, we add the following lines in the *plugin* section of the *pluginManagement* section of the *pom.xml* file.

```
<plugin>
  <groupId>org.openclover</groupId>
  <artifactId>clover-maven-plugin</artifactId>
  <version>4.4.1</version>
</plugin>
```

Using the terminal, from within the project folder, the command

```
mvn clean clover:setup test clover:aggregate clover:clover
```

will run the code coverage tool on the project and generate a report as for JaCoCo. The reports can be found in the target *../target/site/clover/* folder accessible as an *.html* or *.xml* file.

The *pom.xml* file used to run OClver tool on this project is joined with the report as *q4.xml*.


Question 5

Part E - Questions












1. In *part E*, we need to generate a coverage report on the apache-commons-io project using two different code coverage tools, JaCoCo and OCLover so we can compare how the project is covered by each. We are looking for differences in code coverage, or in other words, if the code coverage tools do not cover the project code in an identical way. In case of difference in code coverage, we must understand why the code coverage tools behave differently.
2. In order to do so, we duplicate the project folder and modify pom.xml file for each so we can use JaCoCo for one and OCLover for the other. We then need to modify pom.xml file to make the plugin work. For JaCoCo, we need to specify the JaCoCo version in the *properties* as well as add the JaCoCo plugin in the *plugins* section. For clover, we have to add the plugin in the *plugins* section. Running the command `mvn test` will automatically generate coverage results in an .xml format for OCLover and .csv for JaCoCo. These report files can then be analyzed to try and notice differences between the code coverage tools under study.

Code metrics

Branches: 2,316 Statements: 5,717 Methods: 1,793 Classes: 198 Files: 180 Packages: 11 LOC: 39,515
NCLOC: 14,015 Total complexity: 3,357 Complexity density: 0.59 Statements/Method: 3.19 Methods/Class: 9.06
Classes/Package: 18 Average method complexity: 1.87

| Project | Packages | % Filtered | Average Method Complexity | Uncovered Elements | TOTAL Coverage |
|--|----------|---------------|---------------------------------|-----------------------|---|
| Clover database Wed. Jan. 19 2022 13:19:53 EST | 11 | 0% | 1.87 | 1,092 | 88.9%  |

3.

| Package ▾ | Files | % Filtered | Average Method Complexity | Uncovered Elements | TOTAL Coverage |
|---|-------|---------------|------------------------------|-----------------------|---|
| org.apache.commons.io.input.buffer | 3 | 0% | 2.56 | 109 | 48.1%  |
| org.apache.commons.io.file | 14 | 0% | 1.68 | 181 | 74.4%  |
| org.apache.commons.io.monitor | 5 | 0% | 1.73 | 48 | 84.7%  |
| org.apache.commons.io.function | 3 | 0% | 1 | 5 | 84.8%  |
| org.apache.commons.io.input | 49 | 0% | 2.01 | 275 | 89.6%  |
| org.apache.commons.io.output | 35 | 0% | 1.55 | 118 | 90.5%  |
| org.apache.commons.io | 25 | 0% | 2.17 | 282 | 91.9%  |
| org.apache.commons.io.file.spi | 1 | 0% | 1.5 | 2 | 92%  |
| org.apache.commons.io.filefilter | 30 | 0% | 1.53 | 70 | 92.6%  |
| org.apache.commons.io.comparator | 10 | 0% | 1.73 | 2 | 98.6%  |
| org.apache.commons.io.serialization | 5 | 0% | 1.21 | 0 | 100%  |

Tables above show code coverage of the project using OCLover. Metrics like the number of statements, branches methods, classes in the project as well as

more interesting ones like number of uncovered elements per package and total coverage percentage can be easily viewed.

Apache Commons IO

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|-------------------------------------|------------------------|------|------------------------|------|--------|-------|--------|-------|--------|---------|--------|---------|
| org.apache.commons.io.input | <div><div></div></div> | 88% | <div><div></div></div> | 87% | 147 | 876 | 224 | 1,898 | 55 | 444 | 0 | 51 |
| org.apache.commons.io | <div><div></div></div> | 93% | <div><div></div></div> | 89% | 157 | 1,187 | 183 | 2,297 | 49 | 558 | 0 | 29 |
| org.apache.commons.io.file | <div><div></div></div> | 76% | <div><div></div></div> | 63% | 90 | 289 | 110 | 470 | 23 | 166 | 3 | 21 |
| org.apache.commons.io.output | <div><div></div></div> | 88% | <div><div></div></div> | 85% | 64 | 422 | 122 | 1,064 | 30 | 295 | 0 | 35 |
| org.apache.commons.io.input.buffer | <div><div></div></div> | 40% | <div><div></div></div> | 35% | 43 | 67 | 62 | 132 | 11 | 25 | 1 | 3 |
| org.apache.commons.io.filefilter | <div><div></div></div> | 92% | <div><div></div></div> | 87% | 48 | 386 | 46 | 655 | 16 | 247 | 1 | 29 |
| org.apache.commons.io.monitor | <div><div></div></div> | 88% | <div><div></div></div> | 84% | 29 | 126 | 30 | 239 | 13 | 66 | 0 | 4 |
| org.apache.commons.io.function | <div><div></div></div> | 88% | <div><div></div></div> | n/a | 2 | 23 | 5 | 24 | 2 | 23 | 0 | 2 |
| org.apache.commons.io.serialization | <div><div></div></div> | 97% | <div><div></div></div> | 100% | 0 | 29 | 2 | 56 | 0 | 19 | 0 | 4 |
| org.apache.commons.io.comparator | <div><div></div></div> | 99% | <div><div></div></div> | 93% | 3 | 66 | 1 | 132 | 0 | 42 | 0 | 10 |
| org.apache.commons.io.file.spi | <div><div></div></div> | 97% | <div><div></div></div> | 75% | 2 | 11 | 1 | 17 | 0 | 7 | 0 | 1 |
| Total | 3,249 of 29,969 | 89% | 478 of 3,176 | 84% | 585 | 3,482 | 786 | 6,984 | 199 | 1,892 | 5 | 189 |

In the table above we can see an overview of the project coverage as presented by JaCoCo. Both tools offer a way to easily view approximately the same metrics.

4. a) Yes, when analyzing the results presented by each tool, we can clearly see that for many files, the number of covered statements is not the same when using JaCoCo compared to when using OClover.

A script is used to compile the project's statement coverage obtained with each tool. Both the script (*script.ipynb*) and the compiled results (*report.txt*) are joined with this report. By looking at the compiled results, it is obvious that JaCoCo and OClover have different ways to compute metrics since coverage results differ for a majority of files.

Question 5 - Questions

- a) Yes, when analyzing the results presented by each tool, we can clearly see that for many files, the number of covered statements is not the same when using JaCoCo compared to when using OClover.

A script is used to compile the project's statement coverage obtained with each tool. Both the script (*script.ipynb*) and the compiled results (*report.txt*) are joined with this report. By looking at the compiled results, it is obvious that JaCoCo and OClover have different ways to compute metrics since coverage results differ for a majority of files.

```
org.apache.commons.io.input.BoundedInputStream.java: {'JaCoCo': '22', 'Clover': '17'}
org.apache.commons.io.input.BoundedReader.java: {'JaCoCo': '26', 'Clover': '21'}
org.apache.commons.io.input.BrokenInputStream.java: {'JaCoCo': '10', 'Clover': '7'}
org.apache.commons.io.input.BrokenReader.java: {'JaCoCo': '9', 'Clover': '7'}
org.apache.commons.io.input.BufferedFileChannelInputStream.java: {'JaCoCo': '65', 'Clover': '58'}
org.apache.commons.io.input.CharSequenceInputStream.java: {'JaCoCo': '75', 'Clover': '65'}
org.apache.commons.io.input.CharSequenceReader.java: {'JaCoCo': '65', 'Clover': '58'}
```

For example, the `org.apache.commons.io.input.BrokenReader` file has 9 statements covered when analyzed using JaCoCo and only 7 with OClover.

This file has 8 'methods' including constructors. If we take a closer look at the `BrokenReader(IOException)` constructor, we see that in JaCoCo, all three lines of the method are counted as covered methods whereas using OClover, only the header and the statement are (not the final closing curly-bracket).

```

38.
39.
40.    /**
41.     * Creates a new reader that always throws the given exception.
42.     *
43.     * @param exception the exception to be thrown
44.     */
45.    public BrokenReader(final IOException exception) {
46.        this.exception = exception;
47.    }

```

Above is line coverage for the `BrokenReader(IOException)` constructor using JaCoCo.

| | | |
|----|---|---|
| 44 | 7 | <pre> public BrokenReader(final IOException exception) { this.exception = exception; } </pre> |
| 45 | 7 | |
| 46 | | |
| 47 | | |

Above is line coverage for the `BrokenReader(IOException)` constructor using OClover.

These results show that while OClover only counts statements, JaCoCo tends to count lines of code based on the actual source lines. For example, if a method ends because a `return` statement or a `throw` statement is reached, the final closing curly bracket is not counted as a line (since the execution never reached it) however, if the method is completely executed and the final closing curly bracket is reached, then it is counted as a covered line by JaCoCo. OClover does not operate this way. OClover counts the number of statement that have been executed through a test run. Let's cover another example using the `../org.apache.commons.io.comparator.NameFileComparator` file. While analysis with both tools return a 100% test coverage, we see that the number of covered lines/statements differ:

```

org.apache.commons.io.comparator.ExtensionFileComparator.java: {'JaCoCo': '16', 'Clover': '6'}
org.apache.commons.io.comparator.LastModifiedFileComparator.java: {'JaCoCo': '9', 'Clover': '6'}
org.apache.commons.io.comparator.NameFileComparator.java: {'JaCoCo': '14', 'Clover': '4'}
org.apache.commons.io.comparator.PathFileComparator.java: {'JaCoCo': '14', 'Clover': '4'}

```

This difference is due to the fact that JaCoCo counts the class variables initialization lines as covered lines because statements are executed (namely `new` statement), while OClover only counts the statements associated with methods.

```

51. public class NameFileComparator extends AbstractFileComparator implements Serializable {
52.
53.     private static final long serialVersionUID = 8397947749814525798L;
54.
55.     /** Case-sensitive name comparator instance (see {@link IOCase#SENSITIVE}) */
56.     public static final Comparator<File> NAME_COMPARATOR = new NameFileComparator();
57.
58.     /** Reverse case-sensitive name comparator instance (see {@link IOCase#SENSITIVE}) */
59.     public static final Comparator<File> NAME_REVERSE = new ReverseFileComparator(NAME_COMPARATOR);
60.
61.     /** Case-insensitive name comparator instance (see {@link IOCase#INSENSITIVE}) */
62.     public static final Comparator<File> NAME_INSENSITIVE_COMPARATOR = new NameFileComparator(IOCase.INSENSITIVE);
63.
64.     /** Reverse case-insensitive name comparator instance (see {@link IOCase#INSENSITIVE}) */
65.     public static final Comparator<File> NAME_INSENSITIVE_REVERSE = new ReverseFileComparator(NAME_INSENSITIVE_COMPARATOR);
66.
67.     /** System sensitive name comparator instance (see {@link IOCase#SYSTEM}) */
68.     public static final Comparator<File> NAME_SYSTEM_COMPARATOR = new NameFileComparator(IOCase.SYSTEM);
69.
70.     /** Reverse system sensitive name comparator instance (see {@link IOCase#SYSTEM}) */
71.     public static final Comparator<File> NAME_SYSTEM_REVERSE = new ReverseFileComparator(NAME_SYSTEM_COMPARATOR);
72.

```

Class variable initialization are counted as line covered when using JaCoCo as seen in the picture above.

```

51 public class NameFileComparator extends AbstractFileComparator implements Serializable {
52
53     private static final long serialVersionUID = 8397947749814525798L;
54
55     /** Case-sensitive name comparator instance (see {@link IOCase#SENSITIVE}) */
56     public static final Comparator<File> NAME_COMPARATOR = new NameFileComparator();
57
58     /** Reverse case-sensitive name comparator instance (see {@link IOCase#SENSITIVE}) */
59     public static final Comparator<File> NAME_REVERSE = new ReverseFileComparator(NAME_COMPARATOR);
60
61     /** Case-insensitive name comparator instance (see {@link IOCase#INSENSITIVE}) */
62     public static final Comparator<File> NAME_INSENSITIVE_COMPARATOR = new NameFileComparator(IOCase.INSENSITIVE);
63
64     /** Reverse case-insensitive name comparator instance (see {@link IOCase#INSENSITIVE}) */
65     public static final Comparator<File> NAME_INSENSITIVE_REVERSE = new ReverseFileComparator(NAME_INSENSITIVE_COMPARATOR);
66
67     /** System sensitive name comparator instance (see {@link IOCase#SYSTEM}) */
68     public static final Comparator<File> NAME_SYSTEM_COMPARATOR = new NameFileComparator(IOCase.SYSTEM);
69
70     /** Reverse system sensitive name comparator instance (see {@link IOCase#SYSTEM}) */
71     public static final Comparator<File> NAME_SYSTEM_REVERSE = new ReverseFileComparator(NAME_SYSTEM_COMPARATOR);
72

```

Class variable initialization are not counted as statements covered when using OCLover as seen in the picture above.

Many of these differences come down to the way the source formatting is done. Indeed, a single line of a source code can refer to another class or another function with its own number of source lines that are going to be covered as well since it was invoked. Because of that, it is impossible to simply add the number of statements covered within a particular class. JaCoCo calculates line coverage based on the actual number of lines covered during the execution of tests.

- b) The first file used for a) was `org.apache.commons.io.input.BrokenReader` since all methods are simple with no decision points and no branch, each one

has a cyclomatic complexity of 1. If we follow the theory, $CC = B - D + 1$ where B is the number of branches and D the number of decision points. Since the class has 8 methods a theoretical Cyclomatic complexity of 8 is expected. Both JaCoCo and OCLover find that same answer.

For the second file used in a), namely

`../org.apache.commons.io.comparator.NameFileComparator` the tools return two different numbers when it comes to cyclomatic complexity. Indeed, JaCoCo finds a cyclomatic complexity of 6 while OCLover finds 5. This is due to the fact that JaCoCo considers the class variables definitions as having a cyclomatic complexity of 1 while as previously stated, OCLover does not consider these lines. The method `NameFileComparator(IOCase)` is interesting since it includes an `if` statement. Both JaCoCo and OCLover find a cyclomatic complexity of 2 for this method as it contains a single `if` statement, allowing two branches. If we compute the theoretical cyclomatic complexity for this method we find: $CC = B - D + 1 = 2 - 1 + 1 = 2$. Hence, the values returned by both tools are equal to the theoretical value.

```

88 7
89 7
90
public NameFileComparator(final IOCase caseSensitivity) {
    this.caseSensitivity = caseSensitivity == null ? IOCase.SENSITIVE : caseSensitivity;
}

```

However, if we compute the theoretical cyclomatic complexity for this whole class, we obtain a value of 5. Only OCLover returns the right value as JaCoCo's cyclomatic complexity takes into account the class variables.

- c) In order to have more meaningful results, I would suggest OCLover over JaCoCo. Even though a lot of the metrics are similar, while completing part E, I realised that while OCLover returns the number of statements covered by the tests, JaCoCo finds the number of lines covered which will make it count meaningless lines in term of code coverage like closing curly braces for example. This can boost the line coverage percentage and hide some issues. On the other side, OCLover deals with executed (or missed) statements that express some action to be carried out by the program. I also prefer the OCLover web interface which I think provides the user with more details. That being said, both tools are complete and seamless to use.
- d) The metrics provided by both tools are in my opinion very complete. However, it would be even better if the tools provided the user with a user interface that would let the user chose what test to run, what folder to work on etc (even though additional plugins will let you do so). Also, it would be a good addition to allow the user to manipulate the metrics via the web-interface after the report is generated, to allow for an direct comparison and analysis of the data.