SWE_261P_ Project_Part_3

Team Members:
· Yuxin Huang: yuxinh20@uci.edu
· Changhao Liu: liuc50@uci.edu
· Ruokun Xu: ruokunx@uci.edu
Team Name: OffersAreHere
Github link: https://github.com/yx-hh/commons-io

Structural Testing:

Structural testing is one of the four major software testing types defined by the International Software Testing Qualifications Board. As its name suggests, structural testing evaluates the structure of the code and guides developers to find out whether the test suite is sufficient to cover the codebase(Pezzè and Young, pg 211 - pg 215). Undoubtedly, a program is not sufficiently tested if some of its elements have never been executed. To assess the thoroughness of existing test cases, developers utilize structural testing to locate elements that are not executed, including statements, branches, conditions, and paths(Pezzè and Young, pg 212). It verifies aspects such as the correct implementation of the conditional statements of the code and whether each statement in the code executes and runs smoothly. Structural-based testing cases are usually provided by developers, as they understand how the system was built, to improve the thoroughness of the current test suite rather than examine if the function can output the correct result.

Example: As for a program that calculates the average value of an int array, structural testing focuses on evaluating steps of counting the average value, rather than checking the correctness of the result(*Structural Testing Tutorial, 2022*).

Advantages:

- Thoroughly tests the codebase to reduce the undetected errors that may occur.
- Has some existing tools to automate the process.
- Helps developers to deeply understand code's structure, fix any errors and implement higher quality code.
- It can be executed during the system implementation progress, saving time.
- Prunes unnecessary code.
- Runs by developers, and there is no need to wait for other testers such as QAs.

Code Coverage Analysis:

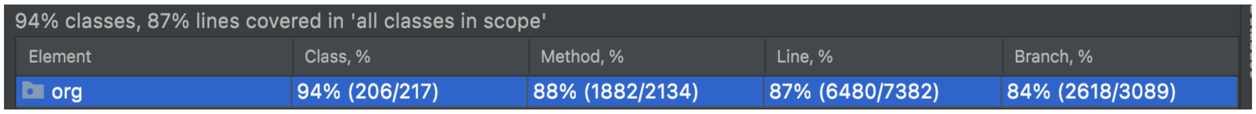
We utilized jacoco, a coverage report generator tool, to analyze the coverage of the existing test suite: Commons IO has already integrated with Jacoco. In the pom.xml file, there is a line with `<commons.jacoco.version>0.8.7</commons.jacoco.version>` , informing us current Jacoco version used in Commons IO is 0.8.7.

How to run Jacoco?

Jacoco automatically generates a coverage report when we start a build or run the test suite. To start a build, developers can simply run `mvn` in the command line and wait for the build to complete. After such build completes, we are able to see a detailed report generated by Jacoco about the test suite coverage.

Coverage of the existing test suite:

Jacoco provides a few coverage measures, including class, method, line, and branch, guiding us to understand the coverage of the existing test suite. As you can see from Figure 1 below, Commons IO has class coverage of 94%, method 88%, line 87%, and branch coverage 84%.



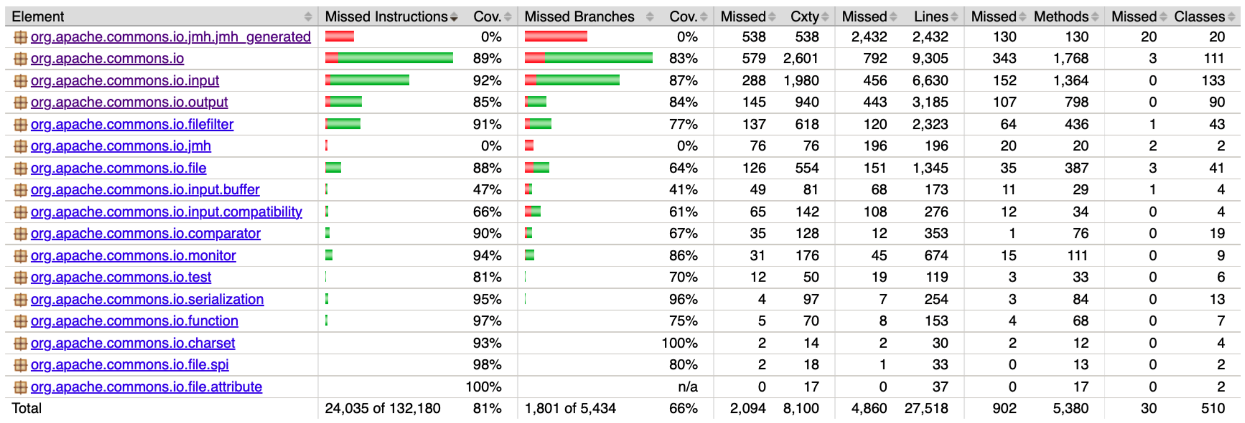
94% classes, 87% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %	Branch, %
org	94% (206/217)	88% (1882/2134)	87% (6480/7382)	84% (2618/3089)

Figure 1: Overall Code Coverage for Apache Commons IO, From IntelliJ, By Changhao Liu

Detailed Report:

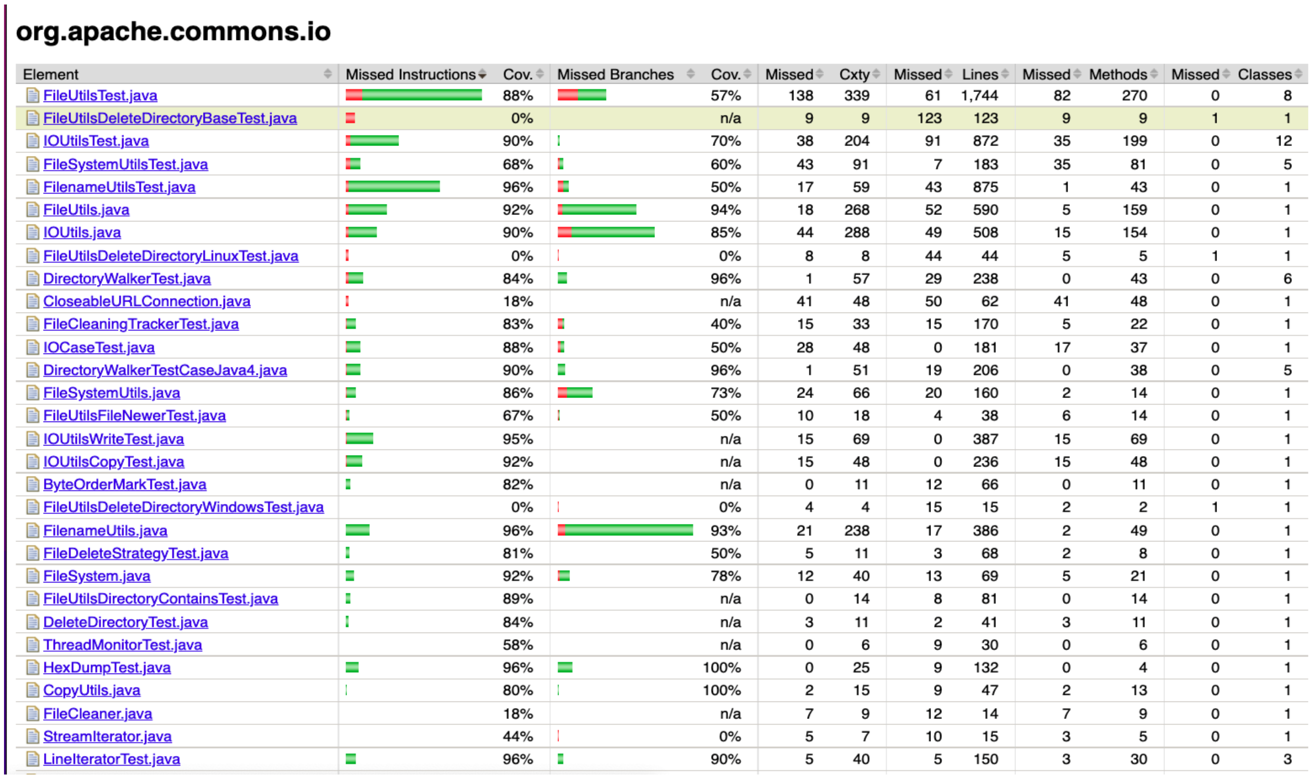
Jacoco is able to generate a detailed report into an html page. By clicking a specific class name or method name on the html page, we can examine its coverage information. As you can see from Figure 2 below, Commons IO has a variety of packages with different coverage statistics:



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.io.jmh.jmh_generated		0%		0%	538	538	2,432	2,432	130	130	20	20
org.apache.commons.io		89%		83%	579	2,601	792	9,305	343	1,768	3	111
org.apache.commons.io.input		92%		87%	288	1,980	456	6,630	152	1,364	0	133
org.apache.commons.io.output		85%		84%	145	940	443	3,185	107	798	0	90
org.apache.commons.io.filefilter		91%		77%	137	618	120	2,323	64	436	1	43
org.apache.commons.io.jmh		0%		0%	76	76	196	196	20	20	2	2
org.apache.commons.io.file		88%		64%	126	554	151	1,345	35	387	3	41
org.apache.commons.io.input.buffer		47%		41%	49	81	68	173	11	29	1	4
org.apache.commons.io.input.compatibility		66%		61%	65	142	108	276	12	34	0	4
org.apache.commons.io.comparator		90%		67%	35	128	12	353	1	76	0	19
org.apache.commons.io.monitor		94%		86%	31	176	45	674	15	111	0	9
org.apache.commons.io.test		81%		70%	12	50	19	119	3	33	0	6
org.apache.commons.io.serialization		95%		96%	4	97	7	254	3	84	0	13
org.apache.commons.io.function		97%		75%	5	70	8	153	4	68	0	7
org.apache.commons.io.charset		93%		100%	2	14	2	30	2	12	0	4
org.apache.commons.io.file.spi		98%		80%	2	18	1	33	0	13	0	2
org.apache.commons.io.file.attribute		100%		n/a	0	17	0	37	0	17	0	2
Total	24,035 of 132,180	81%	1,801 of 5,434	66%	2,094	8,100	4,860	27,518	902	5,380	30	510

Figure 2: Code Coverage for each package in Apache Commons IO, From Jacoco, By Changhao Liu

For example, as you can see from Figure 3, after clicking "org.apache.commons.io", we were routed to a page that contains coverage information for that specific package. Furthermore, not only can we check the number of lines, methods, or classes that the test suite misses, but also we can rank classes regarding their coverage ascendingly or descendingly.



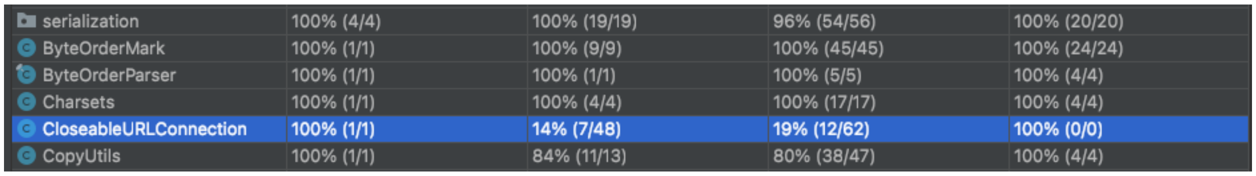
org.apache.commons.io

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
FileUtilsTest.java		88%		57%	138	339	61	1,744	82	270	0	8
FileUtilsDeleteDirectoryBaseTest.java		0%		n/a	9	9	123	123	9	9	1	1
IOUtilsTest.java		90%		70%	38	204	91	872	35	199	0	12
FileSystemUtilsTest.java		68%		60%	43	91	7	183	35	81	0	5
FilenameUtilsTest.java		96%		50%	17	59	43	875	1	43	0	1
FileUtils.java		92%		94%	18	268	52	590	5	159	0	1
IOUtils.java		90%		85%	44	288	49	508	15	154	0	1
FileUtilsDeleteDirectoryLinuxTest.java		0%		0%	8	8	44	44	5	5	1	1
DirectoryWalkerTest.java		84%		96%	1	57	29	238	0	43	0	6
CloseableURLConnection.java		18%		n/a	41	48	50	62	41	48	0	1
FileCleaningTrackerTest.java		83%		40%	15	33	15	170	5	22	0	1
IOCaseTest.java		88%		50%	28	48	0	181	17	37	0	1
DirectoryWalkerTestCaseJava4.java		90%		96%	1	51	19	206	0	38	0	5
FileSystemUtils.java		86%		73%	24	66	20	160	2	14	0	1
FileUtilsFileNewerTest.java		67%		50%	10	18	4	38	6	14	0	1
IOUtilsWriteTest.java		95%		n/a	15	69	0	387	15	69	0	1
IOUtilsCopyTest.java		92%		n/a	15	48	0	236	15	48	0	1
ByteOrderMarkTest.java		82%		n/a	0	11	12	66	0	11	0	1
FileUtilsDeleteDirectoryWindowsTest.java		0%		0%	4	4	15	15	2	2	1	1
FilenameUtils.java		96%		93%	21	238	17	386	2	49	0	1
FileDeleteStrategyTest.java		81%		50%	5	11	3	68	2	8	0	1
FileSystem.java		92%		78%	12	40	13	69	5	21	0	1
FileUtilsDirectoryContainsTest.java		89%		n/a	0	14	8	81	0	14	0	1
DeleteDirectoryTest.java		84%		n/a	3	11	2	41	3	11	0	1
ThreadMonitorTest.java		58%		n/a	0	6	9	30	0	6	0	1
HexDumpTest.java		96%		100%	0	25	9	132	0	4	0	1
CopyUtils.java		80%		100%	2	15	9	47	2	13	0	1
FileCleaner.java		18%		n/a	7	9	12	14	7	9	0	1
StreamIterator.java		44%		0%	5	7	10	15	3	5	0	1
LineIteratorTest.java		96%		90%	5	40	5	150	3	30	0	3

Figure 3: Code Coverage for package: Commons IO, From Jacoco, By Changhao Liu

Uncovered Test Cases:

In "org.apache.commons.io", one particular class, CloseableURLConnection, only has 18% of coverage: missed 50 out of 62 lines and 41 out of 48 methods according to Figure 4. By diving deeply into this class, we can see there are a few lines of code that are not covered by the original test suite. Jacoco marked the uncovered lines to red color and covered ones to green, as you can see from the code snippet below. To increase the code coverage of CloseableURLConnection, we have added a few test cases which will be discussed in the next section.



serialization	100% (4/4)	100% (19/19)	96% (54/56)	100% (20/20)
ByteOrderMark	100% (1/1)	100% (9/9)	100% (45/45)	100% (24/24)
ByteOrderParser	100% (1/1)	100% (1/1)	100% (5/5)	100% (4/4)
Charsets	100% (1/1)	100% (4/4)	100% (17/17)	100% (4/4)
CloseableURLConnection	100% (1/1)	14% (7/48)	19% (12/62)	100% (0/0)
CopyUtils	100% (1/1)	84% (11/13)	80% (38/47)	100% (4/4)

Figure 4: Original Code Coverage for ClosableURLConnection From Jacoco, By Changhao Liu

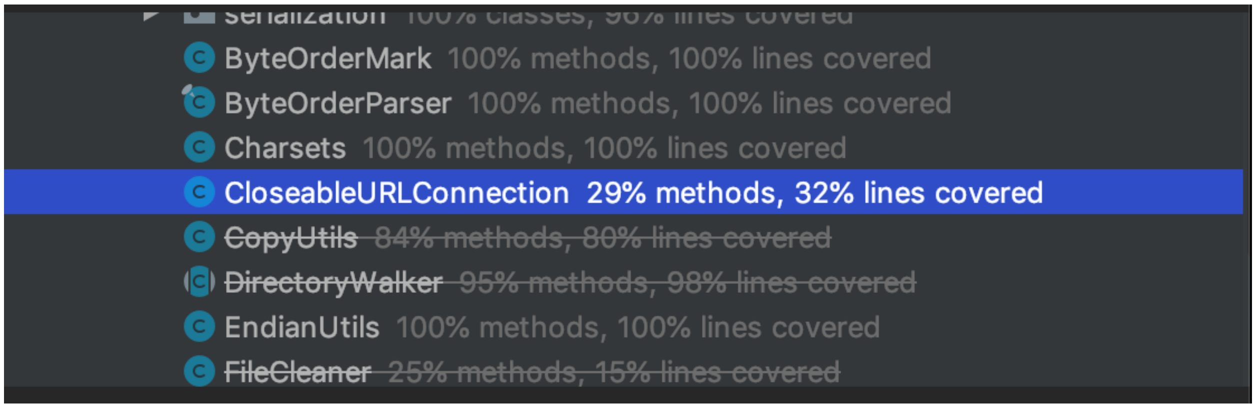
Here is one example that Jacoco marked unexecuted codes as red:

```
@Override
    public void addRequestProperty(final String key, final String value) {
        urlConnection.addRequestProperty(key, value);
    }
```

```java

    @Override
    public void close() {
        IOUtils.close(urlConnection);
    }

    @Override
    public void connect() throws IOException {
        urlConnection.connect();
    }

    @Override
    public boolean equals(final Object obj) {
        return urlConnection.equals(obj);
    }

    @Override
    public boolean getAllowUserInteraction() {
        return urlConnection.getAllowUserInteraction();
    }

    @Override
    public int getConnectTimeout() {
        return urlConnection.getConnectTimeout();
    }

    @Override
    public Object getContent() throws IOException {
        return urlConnection.getContent();
    }

    @Override
    public Object getContent(@SuppressWarnings("rawtypes") final Class[] classes) throws IOException {
        return urlConnection.getContent(classes);
    }

    @Override
    public String getContentEncoding() {
        return urlConnection.getContentEncoding();
    }

```

Newly Added Test Cases to improve coverage:

For the whole project, the coverage for classes, methods, line and branch are 94%, 88%, 87% and 84% separately. We chose to increase the coverage for ClosableURLConnection class, which is responsible for dealing issues with URI and URL resources, an essential feature in commons-io. Before we added more test cases, the original coverage was 14% methods and 19% lines covered. After adding new test cases, we successfully increased the coverage from 14% methods to 29 % methods, and from 19% lines to 32% lines covered as you can see from Figure 5.



```
serialization 100% classes, 96% lines covered
ByteOrderMark 100% methods, 100% lines covered
ByteOrderParser 100% methods, 100% lines covered
Charsets 100% methods, 100% lines covered
CloseableURLConnection 29% methods, 32% lines covered
```
~~CopyUtils 84% methods, 80% lines covered~~

~~DirectoryWalker 95% methods, 98% lines covered~~

EndianUtils 100% methods, 100% lines covered

~~FileCleaner 25% methods, 15% lines covered~~

Figure 5: New Code Coverage for ClosableURLConnection From Jacoco, By Changhao Liu

Test Case Details:

1.Test case for checking whether it can open a CloseableURLConnection with a URI resource

```
@Test
void open() throws IOException {
   final File testFile = TestUtils.newFile(tempDirFile, "testFile.txt");
   CloseableURLConnection uriConnection =
CloseableURLConnection.open(testFile.toURI());
   assertNotNull(uriConnection);
   uriConnection.close();
}
```

2.Test case for checking whether it can open a CloseableURLConnection with a URL resource

```
@Test
void testOpen() throws IOException {
   final File testFile = TestUtils.newFile(tempDirFile, "testFile.txt");
   CloseableURLConnection urlConnection =
CloseableURLConnection.open(testFile.toURI().toURL());
   assertNotNull(urlConnection);
   urlConnection.close();
}
```

3. Test case for checking whether it can add requestProperty to CloseableURLConnection

```java
@Test
void addRequestProperty() throws IOException {
   final CloseableURLConnection connection = CloseableURLConnection.open(new
URL("https://www.google.com/"));
   final String key = "phone";
   final String value = "123456";
   connection.addRequestProperty(key, value);
   final String res = connection.getRequestProperty("phone");
   assertEquals(res, value);
}
```

4. Test case for checking whether the CloseableURLConnection can be closed as expected

```java
@Test
void close() throws IOException {
   File testFile = TestUtils.newFile(tempDirFile, "testFile.txt");
   CloseableURLConnection connection =
CloseableURLConnection.open(testFile.toURI().toURL());
   connection.close();
   assertThrows(FileNotFoundException.class, () -> connection.getContent());
}
```

5.Test case for checking whether two ClosableURLConnection are equal or not

```java
@Test
void testEquals() throws IOException {
   File testFile = TestUtils.newFile(tempDirFile, "testFile.txt");
   CloseableURLConnection connection1 =
CloseableURLConnection.open(testFile.toURI().toURL());
   CloseableURLConnection connection2 =
CloseableURLConnection.open(testFile.toURI().toURL());
   assertFalse(connection1.equals(connection2));
}
```

6.Test case for checking whether it can get connectTimeout or not

```java
@Test
void getConnectTimeout() throws Exception {
   CloseableURLConnection connection = CloseableURLConnection.open(new
URL("https://www.google.com/"));
   connection.setConnectTimeout(5);
   assertEquals(5, connection.getConnectTimeout());
}
```

7. Task case for checking whether it can get connection's content or not

```
@Test
void getContent() throws IOException {
   // write data to test file
   File testFile = TestUtils.newFile(tempDirFile, "testFile.txt");
   final String content = "Hello World!";
   FileUtils.write(testFile, content, Charset.defaultCharset());
   // open as connection
   final CloseableURLConnection connection =
CloseableURLConnection.open(testFile.toURI().toURL());
   // read connection content
   final InputStream is = (InputStream) connection.getContent();
   final BufferedReader br = new BufferedReader(new InputStreamReader(is));
   String res = "";
   String currentLine = "";
   while ((currentLine = br.readLine()) != null) {
       res += currentLine;
   }
   // compare expect result and real result
   assertEquals(content, res);
   // delete test file and close connection
   FileUtils.delete(testFile);
   connection.close();
}
```

8. Test case for checking whether it can get correct content length or not

```
@Test
void getContentLength() throws IOException {
   final File testFile = TestUtils.newFile(tempDirFile, "testFile.txt");
   final String content = "Hello World!";
   FileUtils.write(testFile, content, Charset.defaultCharset());
   CloseableURLConnection connection =
CloseableURLConnection.open(testFile.toURI().toURL());
   assertEquals(content.length(), connection.getContentLength());
}
```

9.Test case for checking whether it can get the expected content type. When writing String to file and create connection, we expect its type to be "text/plain".

```
@Test
void getContentType() throws IOException {
   final File testFile = TestUtils.newFile(tempDirFile,
"testFile.txt");
   final String content = "Hello World!";
   FileUtils.write(testFile, content, Charset.defaultCharset());
   CloseableURLConnection connection =
CloseableURLConnection.open(testFile.toURI().toURL());
   assertEquals("text/plain", connection.getContentType());
}
```

Reference

Pezzè Mauro, and Michal Young. *Software Testing and Analysis: Process, Principles, and Techniques*. Wiley, 2008.

Structural Testing Tutorial - What Is Structural Testing. (2022, February 3). Software Testing Help. https://www.softwaretestinghelp.com/structural-testing-tutorial/

Hamilton, T. (2021, December 18). What is WHITE Box Testing? Techniques, Example & Types. Guru99. https://www.guru99.com/white-box-testing.html