April 6, 2016
APPP-514 Assignment 5

Group 14 members:
Babatunde Akinniyi - Cyrus Sassani

5.1

The name of the application we chose for the purposes of this assignment is called `tabula-java`. `tabula-java` is a library for extracting tables from PDF files — it is the table extraction engine that used to power Tabula(repo). You can use `tabula-java` as a command-line tool to programmatically extract tables from PDFs.

To learn how it works from the command line, enter:

```
$ java -jar ./target/tabula-0.9.0-jar-with-dependencies.jar --help
```

The URL of the forked repository is at https://github.com/tuneribaba/tabula-java or https://github.com/tuneribaba/tabula-java.git

5.2

The Unit testing framework used here is `JUnit` version 4.11. Generally, the technique used is basically data content comparison using `JUnit assertEquals()` since we want to ensure that table contents in the the PDF version is exactly the same after conversion to the new file format and no data is lost. (More details on the approach later). An `assertTableEquals()` method was implemented to make an easy and fast comparison of expected table contents with output content of tabula pdf table extractor.

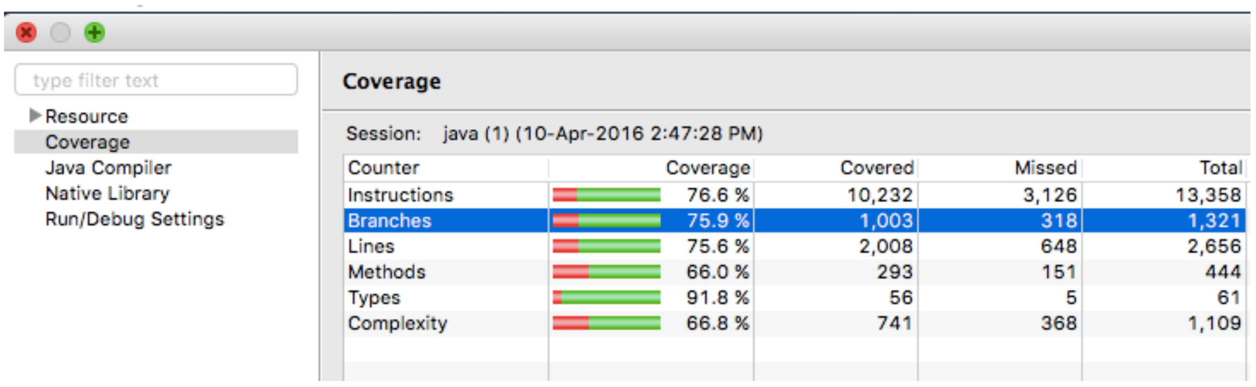
JUnit runners - `org.junit.runners.Parameterized` technique were employed in some test classes such as TestTableDetection.java where a large amount of varying test case values were required due to text localization from branding formats. There wasn't any mocking technique implemented in this test suite and this is satisfactory, as it is a standalone extraction tool that doesn't call external API services or require a database connection. However, if the tabula-java repository was ever to be consolidated with the tabula parent module, then integrated tests would be required probably utilizing some mocking techniques to allow for a better testing performance.

Console log output traces implemented in test classes like `TestTableDetection.java` was very helpful in analyzing and tracing test suite execution and expected results.

How it works (More details). The proprietary extraction algorithm accepts instances of the `tabula.Page` class (of the Java `rectangle2D` type) that has reference points on (x, y) coordinates of the page area, and the length and width dimensions of the page. Using an order of sorting contents and special `WHITE_SPACE_CHARACTERS (\n \t \s “”)` delimiters the algorithm deciphers table text values in the PDF file. It then extracts and stores the content in the `tabula.Table` (also of of the Java `rectangle2D` type, having similar properties of `tabula.Page` type) which can easily be compared with any 2D array string object, or can easily be converted to desired JSON, csv or excel formats.

What to improve. One of the things we noticed for improvement was that most test classes had no test fixture in setup() or teardown() methods. The overall unit test suite design could also be improved to have a proper test suite hierarchy where by child test classes inherit parent test classes behavior and core common functions, as well as making test utility helper functions more generic an exposed to test suite as a whole as opposed to each test class.
We extended the test suite by adding more test cases to increase code coverage and tried to cover corner cases and further input value scenarios as much as possible. Some setbacks we faced were with some classes that had a large number of private methods that could not be easily unit tested. Additionally, there are couple of code areas with "TODO" flags that have not being implemented yet bringing down the overall coverage.

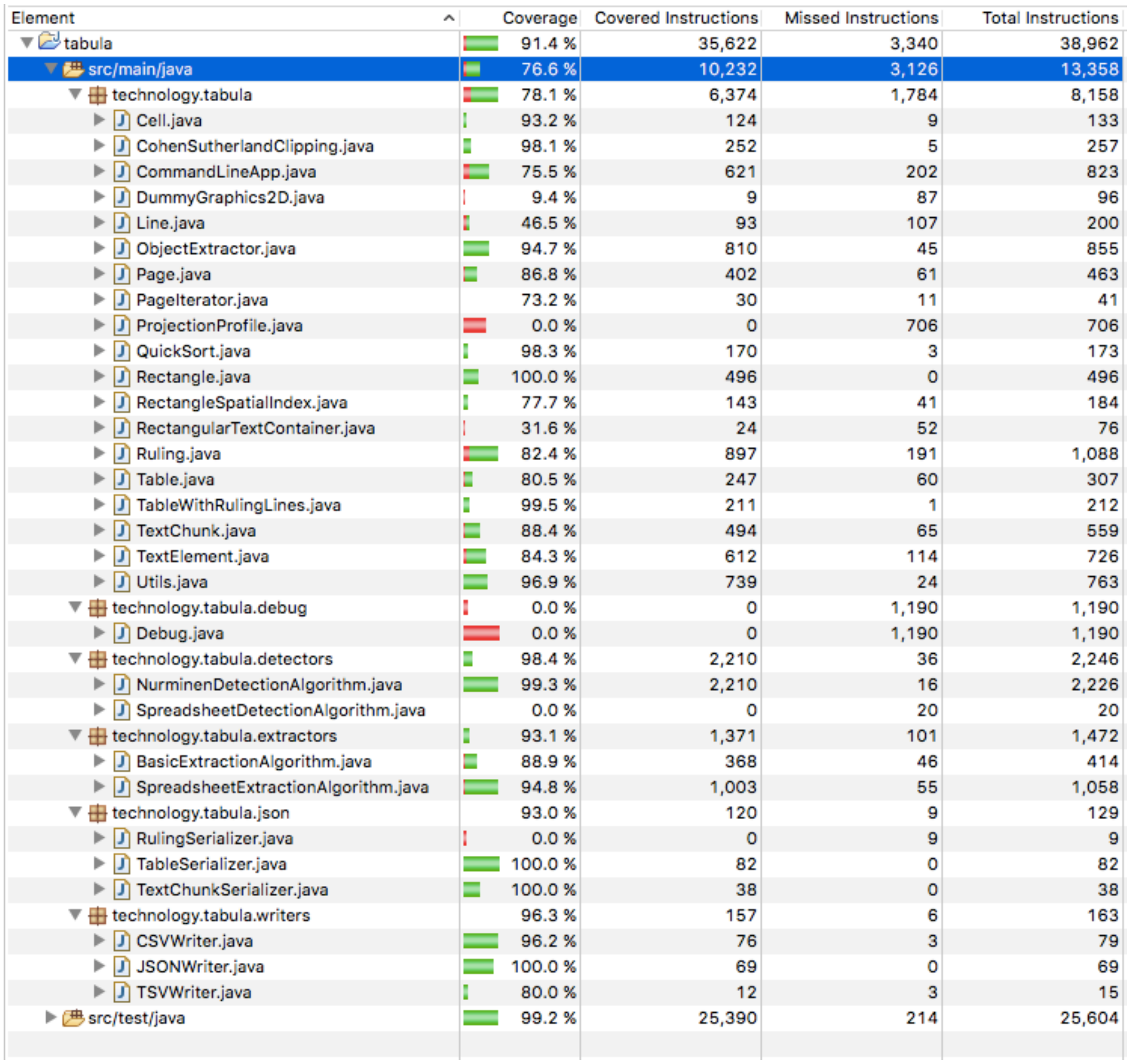
Coverage report. Currently, there are about 144 assertions in the test suite, in total. 76.6% of the tabula application code is covered from 10,232 instructions/13,358 total instructions, 1,003/1321 covered branches, 293/444 covered methods, and 56/61 types covered. The highest covered package is the `technology.tabula.detectors` having a high 98.4% code coverage, and the lowest covered package is the `technology.tabula.debug` package having a coverage of 0.0.0% (but this may be okay). There are some packages with insufficient code coverage like the `technology.tabula` package having a coverage of 78.1%. Additionally, there are some test classes of application features that have no coverage at all or very low coverage namely the `ProjectionProfile.java, SpreadSheetDetectionAlgorithm.java, RulingSerializer.java, and RectangularTextContainer.java` class. The diagram below shows the coverage picture better.



Coverage

Session: java (1) (10-Apr-2016 2:47:28 PM)

Counter	Coverage	Covered	Missed	Total
Instructions	76.6 %	10,232	3,126	13,358
Branches	75.9 %	1,003	318	1,321
Lines	75.6 %	2,008	648	2,656
Methods	66.0 %	293	151	444
Types	91.8 %	56	5	61
Complexity	66.8 %	741	368	1,109



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
tabula	91.4 %	35,622	3,340	38,962
src/main/java	76.6 %	10,232	3,126	13,358
technology.tabula	78.1 %	6,374	1,784	8,158
Cell.java	93.2 %	124	9	133
CohenSutherlandClipping.java	98.1 %	252	5	257
CommandLineApp.java	75.5 %	621	202	823
DummyGraphics2D.java	9.4 %	9	87	96
Line.java	46.5 %	93	107	200
ObjectExtractor.java	94.7 %	810	45	855
Page.java	86.8 %	402	61	463
PageIterator.java	73.2 %	30	11	41
ProjectionProfile.java	0.0 %	0	706	706
QuickSort.java	98.3 %	170	3	173
Rectangle.java	100.0 %	496	0	496
RectangleSpatialIndex.java	77.7 %	143	41	184
RectangularTextContainer.java	31.6 %	24	52	76
Ruling.java	82.4 %	897	191	1,088
Table.java	80.5 %	247	60	307
TableWithRulingLines.java	99.5 %	211	1	212
TextChunk.java	88.4 %	494	65	559
TextElement.java	84.3 %	612	114	726
Utils.java	96.9 %	739	24	763
technology.tabula.debug	0.0 %	0	1,190	1,190
Debug.java	0.0 %	0	1,190	1,190
technology.tabula.detectors	98.4 %	2,210	36	2,246
NurminenDetectionAlgorithm.java	99.3 %	2,210	16	2,226
SpreadsheetDetectionAlgorithm.java	0.0 %	0	20	20
technology.tabula.extractors	93.1 %	1,371	101	1,472
BasicExtractionAlgorithm.java	88.9 %	368	46	414
SpreadsheetExtractionAlgorithm.java	94.8 %	1,003	55	1,058
technology.tabula.json	93.0 %	120	9	129
RulingSerializer.java	0.0 %	0	9	9
TableSerializer.java	100.0 %	82	0	82
TextChunkSerializer.java	100.0 %	38	0	38
technology.tabula.writers	96.3 %	157	6	163
CSVWriter.java	96.2 %	76	3	79
JSONWriter.java	100.0 %	69	0	69
TSVWriter.java	80.0 %	12	3	15
src/test/java	99.2 %	25,390	214	25,604

5.3

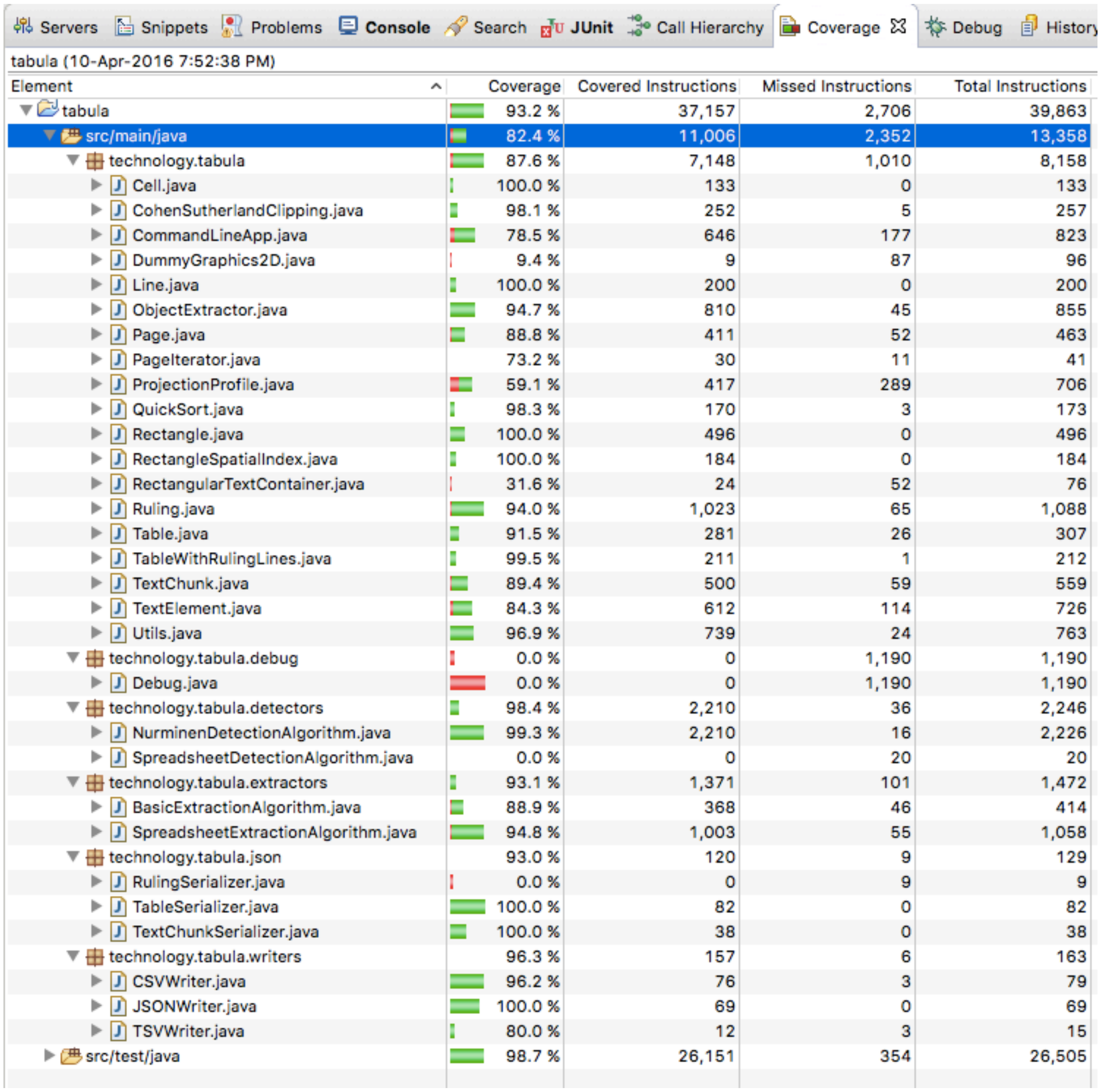
Most of our work was done under the `technology.tabula` package because it was the package with the lowest code coverage. The overall code coverage went up from 76.6% to 82.4% and reduced the number of missed instructions by about 789 instructions.

Here is the list of test classes added or modified:

1. `TestLine.java` – added
2. `TestCell.java` – added
3. `TestCommandLineApp` – modified.
4. `TestRuling.java` – added
5. `TestRectangleSpatialIndex.java` – added

6. `TestCellPosition.java` – added
7. `TestProjectionProfile.java` - added

Here is the latest screenshot:



tabula (10-Apr-2016 7:52:38 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
tabula	93.2 %	37,157	2,706	39,863
src/main/java	82.4 %	11,006	2,352	13,358
technology.tabula	87.6 %	7,148	1,010	8,158
Cell.java	100.0 %	133	0	133
CohenSutherlandClipping.java	98.1 %	252	5	257
CommandLineApp.java	78.5 %	646	177	823
DummyGraphics2D.java	9.4 %	9	87	96
Line.java	100.0 %	200	0	200
ObjectExtractor.java	94.7 %	810	45	855
Page.java	88.8 %	411	52	463
PageIterator.java	73.2 %	30	11	41
ProjectionProfile.java	59.1 %	417	289	706
QuickSort.java	98.3 %	170	3	173
Rectangle.java	100.0 %	496	0	496
RectangleSpatialIndex.java	100.0 %	184	0	184
RectangularTextContainer.java	31.6 %	24	52	76
Ruling.java	94.0 %	1,023	65	1,088
Table.java	91.5 %	281	26	307
TableWithRulingLines.java	99.5 %	211	1	212
TextChunk.java	89.4 %	500	59	559
TextElement.java	84.3 %	612	114	726
Utils.java	96.9 %	739	24	763
technology.tabula.debug	0.0 %	0	1,190	1,190
Debug.java	0.0 %	0	1,190	1,190
technology.tabula.detectors	98.4 %	2,210	36	2,246
NurminenDetectionAlgorithm.java	99.3 %	2,210	16	2,226
SpreadsheetDetectionAlgorithm.java	0.0 %	0	20	20
technology.tabula.extractors	93.1 %	1,371	101	1,472
BasicExtractionAlgorithm.java	88.9 %	368	46	414
SpreadsheetExtractionAlgorithm.java	94.8 %	1,003	55	1,058
technology.tabula.json	93.0 %	120	9	129
RulingSerializer.java	0.0 %	0	9	9
TableSerializer.java	100.0 %	82	0	82
TextChunkSerializer.java	100.0 %	38	0	38
technology.tabula.writers	96.3 %	157	6	163
CSVWriter.java	96.2 %	76	3	79
JSONWriter.java	100.0 %	69	0	69
TSVWriter.java	80.0 %	12	3	15
src/test/java	98.7 %	26,151	354	26,505