JSON-java Testing Report

Chase Gould

5/13/2020

Unit Testing with Sufficient Coverage

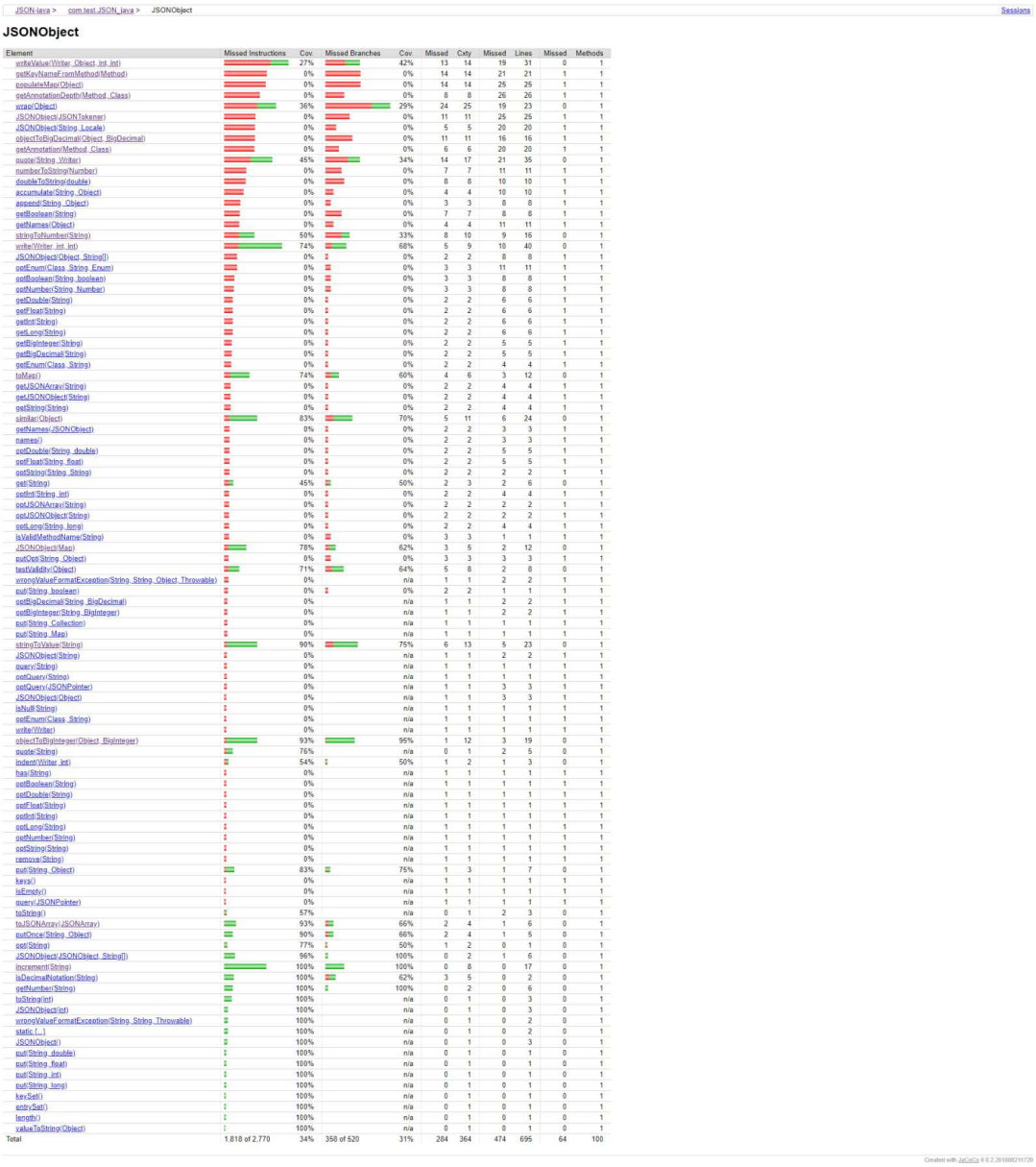
Here is a link to the test files that I created while working on this project:
https://github.com/csun-comp587-s20/JSON-java/tree/master/src/test/java/com/test/JSON_java

I wrote unit tests to test the following classes JSONArray, JSONObject and JSONTokener. The unit tests for these classes are contained in the files JSONArrayTest.java, JSONObjectTest.java and JSONTokenerTest.java.

Total coverage between the three components under test:
Lines of code covered: 1,705
Branches covered: 270
Number of unit tests: 91

The following pages break down the coverage of specific components

JSONObject – code coverage: 34%, branch coverage: 31%
Lines of code covered: 952
Branches covered: 162
Number of unit tests: 53



JSON-java > com.test.JSON_java > JSONObject Sessions

JSONObject

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
writeValue(Writer, Object, Int, Int)		27%		42%	13	14	19	31	0	1
getKeyNameFromMethod(Method)		0%		0%	14	14	21	21	1	1
populateMap(Object)		0%		0%	14	14	25	25	1	1
getAnnotationDepth(Method, Class)		0%		0%	8	8	26	26	1	1
wrap(Object)		36%		29%	24	25	19	23	0	1
JSONObject(JSONTokener)		0%		0%	11	11	25	25	1	1
JSONObject(String, Locale)		0%		0%	5	5	20	20	1	1
objectToBigDecimal(Object, BigDecimal)		0%		0%	11	11	16	16	1	1
getAnnotation(Method, Class)		0%		0%	6	6	20	20	1	1
quote(String, Writer)		45%		34%	14	17	21	35	0	1
numberToString(Number)		0%		0%	7	7	11	11	1	1
doubleToString(double)		0%		0%	8	8	10	10	1	1
accumulate(String, Object)		0%		0%	4	4	10	10	1	1
append(String, Object)		0%		0%	3	3	8	8	1	1
getBoolean(String)		0%		0%	7	7	8	8	1	1
getNames(Object)		0%		0%	4	4	11	11	1	1
stringToNumber(String)		50%		33%	8	10	9	16	0	1
write(Writer, Int, Int)		74%		68%	5	9	10	40	0	1
JSONObject(Object, String[])		0%		0%	2	2	8	8	1	1
getEnum(Class, String, Enum)		0%		0%	3	3	11	11	1	1
getBoolean(String, boolean)		0%		0%	3	3	8	8	1	1
getNumber(String, Number)		0%		0%	3	3	8	8	1	1
getDouble(String)		0%		0%	2	2	6	6	1	1
getFloat(String)		0%		0%	2	2	6	6	1	1
getInt(String)		0%		0%	2	2	6	6	1	1
getLong(String)		0%		0%	2	2	6	6	1	1
getBigInteger(String)		0%		0%	2	2	5	5	1	1
getBigDecimal(String)		0%		0%	2	2	5	5	1	1
getEnum(Class, String)		0%		0%	2	2	4	4	1	1
toMap()		74%		66%	4	6	3	12	0	1
getJSONArray(String)		0%		0%	2	2	4	4	1	1
getJSONObject(String)		0%		0%	2	2	4	4	1	1
getString(String)		0%		0%	2	2	4	4	1	1
similar(Object)		83%		70%	5	11	6	24	0	1
getNames(JSONObject)		0%		0%	2	2	3	3	1	1
names()		0%		0%	2	2	3	3	1	1
getDouble(String, double)		0%		0%	2	2	5	5	1	1
getFloat(String, float)		0%		0%	2	2	5	5	1	1
getString(String, String)		0%		0%	2	2	2	2	1	1
get(String)		45%		56%	2	3	2	6	0	1
getInt(String, Int)		0%		0%	2	2	4	4	1	1
getJSONArray(String)		0%		0%	2	2	2	2	1	1
getJSONObject(String)		0%		0%	2	2	2	2	1	1
getLong(String, long)		0%		0%	2	2	4	4	1	1
isValidMethodName(String)		0%		0%	3	3	1	1	1	1
JSONObject(Map)		78%		62%	3	5	2	12	0	1
putOpt(String, Object)		0%		0%	3	3	3	3	1	1
testValidity(Object)		71%		64%	5	8	2	8	0	1
wrongValueFormatException(String, String, Object, Throwable)		0%		n/a	1	1	2	2	1	1
put(String, boolean)		0%		0%	2	2	1	1	1	1
getBigDecimal(String, BigDecimal)		0%		n/a	1	1	2	2	1	1
getBigInteger(String, BigInteger)		0%		n/a	1	1	2	2	1	1
put(String, Collection)		0%		n/a	1	1	1	1	1	1
put(String, Map)		0%		n/a	1	1	1	1	1	1
stringToValue(String)		90%		75%	6	13	5	23	0	1
JSONObject(String)		0%		n/a	1	1	2	2	1	1
query(String)		0%		n/a	1	1	1	1	1	1
optQuery(String)		0%		n/a	1	1	1	1	1	1
getQuery(JSONPointer)		0%		n/a	1	1	3	3	1	1
JSONObject(Object)		0%		n/a	1	1	3	3	1	1
isNull(String)		0%		n/a	1	1	1	1	1	1
optEnum(Class, String)		0%		n/a	1	1	1	1	1	1
write(Writer)		0%		n/a	1	1	1	1	1	1
objectToBigInteger(Object, BigInteger)		93%		95%	1	12	3	19	0	1
quote(String)		76%		n/a	0	1	2	5	0	1
indent(Writer, Int)		54%		50%	1	2	1	3	0	1
has(String)		0%		n/a	1	1	1	1	1	1
optBoolean(String)		0%		n/a	1	1	1	1	1	1
optDouble(String)		0%		n/a	1	1	1	1	1	1
optFloat(String)		0%		n/a	1	1	1	1	1	1
optInt(String)		0%		n/a	1	1	1	1	1	1
optLong(String)		0%		n/a	1	1	1	1	1	1
optNumber(String)		0%		n/a	1	1	1	1	1	1
optString(String)		0%		n/a	1	1	1	1	1	1
remove(String)		0%		n/a	1	1	1	1	1	1
put(String, Object)		83%		75%	1	3	1	7	0	1
keys()		0%		n/a	1	1	1	1	1	1
isEmpty()		0%		n/a	1	1	1	1	1	1
query(JSONPointer)		0%		n/a	1	1	1	1	1	1
toString()		57%		n/a	0	1	2	3	0	1
toJSONArray(JSONArray)		93%		66%	2	4	1	6	0	1
putOnce(String, Object)		90%		66%	2	4	1	5	0	1
opt(String)		77%		50%	1	2	0	1	0	1
JSONObject(JSONObject, String[])		96%		100%	0	2	1	6	0	1
increment(String)		100%		100%	0	8	0	17	0	1
isDecimalNotation(String)		100%		62%	3	5	0	2	0	1
getNumber(String)		100%		100%	0	2	0	6	0	1
toString(int)		100%		n/a	0	1	0	3	0	1
JSONObject(int)		100%		n/a	0	1	0	3	0	1
wrongValueFormatException(String, String, Throwable)		100%		n/a	0	1	0	2	0	1
static {...}		100%		n/a	0	1	0	2	0	1
JSONObject()		100%		n/a	0	1	0	3	0	1
put(String, double)		100%		n/a	0	1	0	1	0	1
put(String, float)		100%		n/a	0	1	0	1	0	1
put(String, Int)		100%		n/a	0	1	0	1	0	1
put(String, long)		100%		n/a	0	1	0	1	0	1
keySet()		100%		n/a	0	1	0	1	0	1
entrySet()		100%		n/a	0	1	0	1	0	1
length()		100%		n/a	0	1	0	1	0	1
valueToString(Object)		100%		n/a	0	1	0	1	0	1
Total	1.818 of 2.770	34%	358 of 520	31%	284	364	474	695	64	100

Created with JaCoCo 8.0.2.201808211720

JSONArray – code coverage: 35%, branch coverage: 37%
Lines of code covered: 423
Branches covered: 59
Number of Unit Tests: 21

JSON-java > com.test.JSON_java > JSONArray

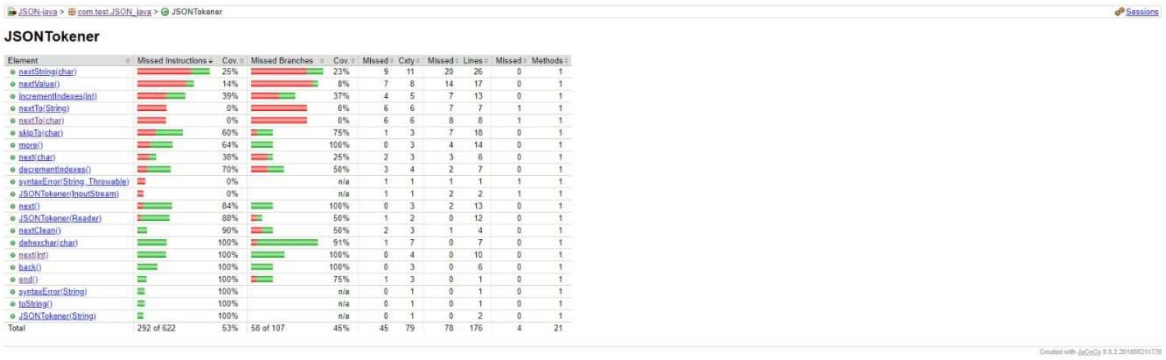
Sessions

JSONArray

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
JSONArray(JSONTokener)		43%		31%	9	10	14	25	0	1
toJSONObject(JSONArray)		0%		0%	5	5	6	6	1	1
JSONArray(Collection)		0%		0%	3	3	8	8	1	1
JSONArray(Object)		0%		0%	3	3	9	9	1	1
getEnum(Class, int, Enum)		0%		0%	3	3	11	11	1	1
similar(Object)		56%		35%	7	11	10	21	0	1
optNumber(int, Number)		0%		0%	4	4	10	10	1	1
write(Writer, int, int)		70%		75%	3	7	8	28	0	1
getFloat(int)		0%		0%	2	2	6	6	1	1
getInt(int)		0%		0%	2	2	6	6	1	1
getLong(int)		0%		0%	2	2	6	6	1	1
getNumber(int)		0%		0%	2	2	6	6	1	1
toList()		65%		60%	4	6	3	11	0	1
getBigDecimal(int)		0%		0%	2	2	5	5	1	1
getBigInteger(int)		0%		0%	2	2	5	5	1	1
getEnum(Class, int)		0%		0%	2	2	5	5	1	1
getJSONArray(int)		0%		0%	2	2	4	4	1	1
getJSONObject(int)		0%		0%	2	2	4	4	1	1
getDouble(int, double)		0%		0%	2	2	5	5	1	1
getFloat(int, float)		0%		0%	2	2	5	5	1	1
getString(int, String)		0%		0%	2	2	3	3	1	1
remove(int)		0%		0%	3	3	3	3	1	1
getInt(int, int)		0%		0%	2	2	4	4	1	1
getJSONArray(int)		0%		0%	2	2	2	2	1	1
getJSONObject(int)		0%		0%	2	2	2	2	1	1
getLong(int, long)		0%		0%	2	2	4	4	1	1
put(int, Map)		0%		n/a	1	1	2	2	1	1
wrongValueFormatException(int, String, Object, Throwable)		0%		n/a	1	1	1	1	1	1
optBigInteger(int, BigInteger)		0%		n/a	1	1	2	2	1	1
optBigDecimal(int, BigDecimal)		0%		n/a	1	1	2	2	1	1
put(boolean)		0%		0%	2	2	1	1	1	1
put(int, Collection)		0%		n/a	1	1	1	1	1	1
optBoolean(int, boolean)		0%		n/a	1	1	3	3	1	1
put(Collection)		0%		n/a	1	1	1	1	1	1
put(Map)		0%		n/a	1	1	1	1	1	1
query(String)		0%		n/a	1	1	1	1	1	1
optQuery(String)		0%		n/a	1	1	1	1	1	1
optQuery(JSONPointer)		0%		n/a	1	1	3	3	1	1
get(int)		57%		50%	1	2	1	4	0	1
isNull(int)		0%		n/a	1	1	1	1	1	1
optEnum(Class, int)		0%		n/a	1	1	1	1	1	1
put(int, double)		0%		n/a	1	1	1	1	1	1
put(int, float)		0%		n/a	1	1	1	1	1	1
put(int, long)		0%		n/a	1	1	1	1	1	1
write(Writer)		0%		n/a	1	1	1	1	1	1
getString(int)		66%		50%	1	2	1	4	0	1
getBoolean(int)		0%		n/a	1	1	1	1	1	1
optDouble(int)		0%		n/a	1	1	1	1	1	1
optFloat(int)		0%		n/a	1	1	1	1	1	1
optInt(int)		0%		n/a	1	1	1	1	1	1
optLong(int)		0%		n/a	1	1	1	1	1	1
optNumber(int)		0%		n/a	1	1	1	1	1	1
optString(int)		0%		n/a	1	1	1	1	1	1
put(double)		0%		n/a	1	1	1	1	1	1
put(float)		0%		n/a	1	1	1	1	1	1
put(int)		0%		n/a	1	1	1	1	1	1
put(long)		0%		n/a	1	1	1	1	1	1
iterator()		0%		n/a	1	1	1	1	1	1
query(JSONPointer)		0%		n/a	1	1	1	1	1	1
toString()		57%		n/a	0	1	2	3	0	1
opt(int)		84%		50%	2	3	0	2	0	1
put(int, Object)		100%		100%	0	5	0	12	0	1
getBoolean(int)		100%		100%	0	7	0	8	0	1
join(String)		100%		100%	0	3	0	9	0	1
getDouble(int)		100%		100%	0	2	0	6	0	1
toString(int)		100%		n/a	0	1	0	3	0	1
put(Object)		100%		n/a	0	1	0	3	0	1
put(int, boolean)		100%		100%	0	2	0	1	0	1
JSONArray()		100%		n/a	0	1	0	3	0	1
wrongValueFormatException(int, String, Throwable)		100%		n/a	0	1	0	1	0	1
JSONArray(String)		100%		n/a	0	1	0	2	0	1
put(int, int)		100%		n/a	0	1	0	1	0	1
length()		100%		n/a	0	1	0	1	0	1
isEmpty()		100%		n/a	0	1	0	1	0	1
Total	755 of 1,178	35%	99 of 158	37%	112	154	196	306	53	74

Created with JaCoCo

JSONTokener – Code coverage: 53%, Branch coverage: 45%
Lines of code covered: 330
Branches covered: 49
Number of unit tests: 17


JSON-java > com.test.JSON_java > JSONTokener

Sessions

JSONTokener

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
nextString(char)		25%		23%	9	11	20	26	0	1
nextValue()		14%		8%	7	8	14	17	0	1
incrementIndexes(int)		39%		37%	4	5	7	13	0	1
nextTo(String)		0%		0%	6	6	7	7	1	1
nextTo(char)		0%		0%	6	6	8	8	1	1
skipTo(char)		60%		75%	1	3	7	18	0	1
more()		64%		100%	0	3	4	14	0	1
next(char)		38%		25%	2	3	3	6	0	1
decrementIndexes()		70%		50%	3	4	2	7	0	1
syntaxError(String, Throwable)		0%		n/a	1	1	1	1	1	1
JSONTokener(InputStream)		0%		n/a	1	1	2	2	1	1
next()		84%		100%	0	3	2	13	0	1
JSONTokener(Reader)		88%		50%	1	2	0	12	0	1
nextClean()		90%		50%	2	3	1	4	0	1
dehexchar(char)		100%		91%	1	7	0	7	0	1
next(int)		100%		100%	0	4	0	10	0	1
back()		100%		100%	0	3	0	6	0	1
end()		100%		75%	1	3	0	1	0	1
syntaxError(String)		100%		n/a	0	1	0	1	0	1
toString()		100%		n/a	0	1	0	1	0	1
JSONTokener(String)		100%		n/a	0	1	0	2	0	1
Total	292 of 622	53%	58 of 107	45%	45	79	78	176	4	21

As you can see from the results my code coverage is not near the desired 90%. There are multiple reasons for this.

- Reason 1: I put a strong effort into covering corner cases. In my experience writing unit tests, often much of a function's code can be covered with just one unit-test. However, the remaining minority of the function's code may require multiple unit tests to cover the code. As part of my effort to cover corner cases I often wrote unit tests that cover as little as one line of code. I believe I could have achieved higher code coverage if I spent less time on corner cases. On the other hand, I do not believe that achieving a higher code coverage by avoiding testing for corner cases would have improved the quality of the testing accomplished in this project.
- Reason 2: Writing test that cover the same code. A unit test that covers a line of code does not guarantee that the code is free of bugs. Perhaps the same line of code may fail under a different input. One of my strategies for adequately testing a function is to write multiple unit tests using specific inputs that I would like to see tested. I then follow this with including a unit test which uses automated test input generation. Unit tests which utilize automated input generation are iterated over multiple times to get a wide array of random inputs. Following this strategy leads to writing multiple unit tests covering the same lines of code but improves the quality of the testing.
- Reason 3: There is a lot of code to test. The three components combined have a total of 4,570 lines of code.
- Reason 4: I spent a considerable amount of time learning how to implement JSON-java so that I could write useful unit tests.

While code coverage is a good benchmark for tracking progress when testing software, it does not guarantee high quality testing. I believe that the reasoning I have provided for my less than ideal code coverage ratio is sufficient in showing that I have put forth a strong effort in this project. Besides the low

coverage ratio, I believe the 1,705 lines of code covered is within the scope of what is expected in this project.

We did not discuss whether I would be emphasizing unit testing or automated testing. I have put a considerable amount of effort into both.

Automated Testing

In my project proposal I proposed developing an automatic testing capability, to automatically generate inputs for the unit tests of the JSON Object, JSON Array and JSON Tokener classes. I have accomplished this task by creating the classes JSONFormatStringGenerator and KeyandValuePairGenerator to generate test inputs. I also used the Java Standard Random class to generate less complex inputs. These two generators classes and the Random class are used throughout my unit tests to provide random inputs. I did not use any specific automated testing tools such as Junit's parameterized tests. Instead, I used my generators within unit tests. Therefore, I do not have specific coverage data for my automated tests, as the automated test coverage in included into my unit tests coverage. At least 24 of my unit tests incorporate automatically generated test inputs. When viewing my unit tests those tests incorporating automated test input generation have a comment above the test which states "auto generated inputs."

Lessons Learned

Knowing what I know I likely would have chosen a different software to test. It took me a couple weeks of playing with the components under test just to figure out how to implement their most basic functionalities. Knowing how to implement the components was essential to gaining insight on how to write unit tests for the components. It took weeks more to understand some more complex functionalities of the components. There are still several functions belonging to these components that I have yet to figure out how to implement. If I had more time, I would like to explore some other features of Junit such as parameterized testing. I enjoy using Junit and plan to continue building on my skill with this tool.