CSCI 5308
Advanced Software Development Concepts

ASSIGNMENT - 1

Banner ID: B00952865

Name: Aditya Maheshbhai Purohit

GitHub - My Forked Repo Link: https://github.com/adityap27/JSON-java

Table of Contents

Task 1: Choose a Java-based open-source repository.

I have select this repository https://github.com/stleary/JSON-java [1] and have added my entry in the excel sheet: Sheet Link

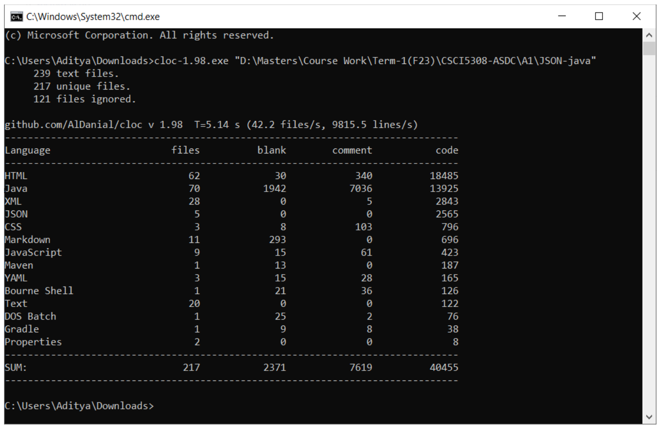
It follows all the mentioned assignment conditions (as of commit: 783d298f990c75fa2de4f458d1647cbf8e46a858)

Condition 1: It must be a maven or gradle-based project.

Answer: Yes, it used both maven and gradle. I will be using maven primarily.

Condition 2: It must have at least 10,000 lines of code

Answer: Yes, it has 13,925 Lines of java code as calculated by "cloc" tool [2].

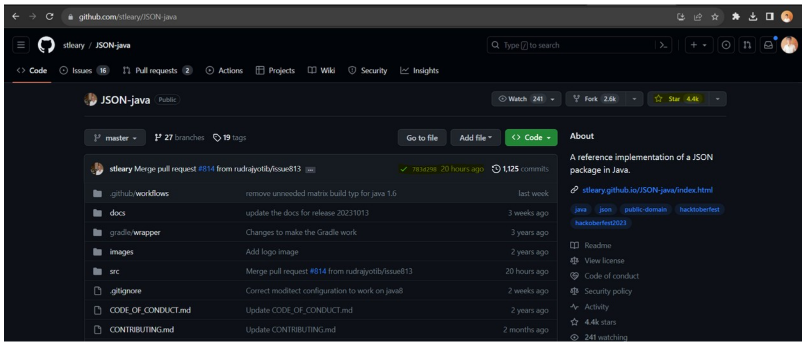


```
(c) Microsoft Corporation. All rights reserved.

C:\Users\Aditya\Downloads>cloc-1.98.exe "D:\Masters\Course Work\Term-1(F23)\CSCI5308-ASDC\A1\JSON-java"
     239 text files.
     217 unique files.
     121 files ignored.

github.com/AlDanial/cloc v 1.98  T=5.14 s (42.2 files/s, 9815.5 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
HTML                            62             30            340          18485
Java                            70           1942           7036          13925
XML                             28              0              5           2843
JSON                             5              0              0           2565
CSS                              3              8            103            796
Markdown                        11            293              0            696
JavaScript                       9             15             61            423
Maven                            1             13              0            187
YAML                             3             15             28            165
Bourne Shell                     1             21             36            126
Text                            20              0              0            122
DOS Batch                        1             25              2             76
Gradle                           1              9              8             38
Properties                       2              0              0              8
-------------------------------------------------------------------------------
SUM:                           217           2371           7619          40455
-------------------------------------------------------------------------------

C:\Users\Aditya\Downloads>
```

Figure 1: Lines of java code calculated using cloc tool [2].

Condition 3: It must have at least 50 stars.

Answer: Yes, it has 4.4k stars as shown in **figure 2**.



stleary Merge pull request #814 from rudrajyotib/issue813 ... ✓ 783d298 20 hours ago 1,125 commits

.github/workflows	remove unneeded matrix build typ for java 1.6	last week
docs	update the docs for release 20231013	3 weeks ago
gradle/wrapper	Changes to make the Gradle work	3 years ago
images	Add logo image	2 years ago
src	Merge pull request #814 from rudrajyotib/issue813	20 hours ago
.gitignore	Correct moditect configuration to work on java8	2 weeks ago
CODE_OF_CONDUCT.md	Update CODE_OF_CONDUCT.md	2 years ago
CONTRIBUTING.md	Update CONTRIBUTING.md	2 months ago

About

A reference implementation of a JSON package in Java.

stleary.github.io/JSON-java/index.html

java json public-domain hacktoberfest hackoberfest2023

Readme
View license
Code of conduct
Security policy
Activity
4.4k stars
241 watching

Figure 2: Stars of the repository and activeness [1].

Condition 4: It must have tests written using the JUnit framework

Answer: Yes, it uses Junit 4.13.2 and it is mentioned in the pom.xml.

Condition 5: It must not be a tutorial or example repository

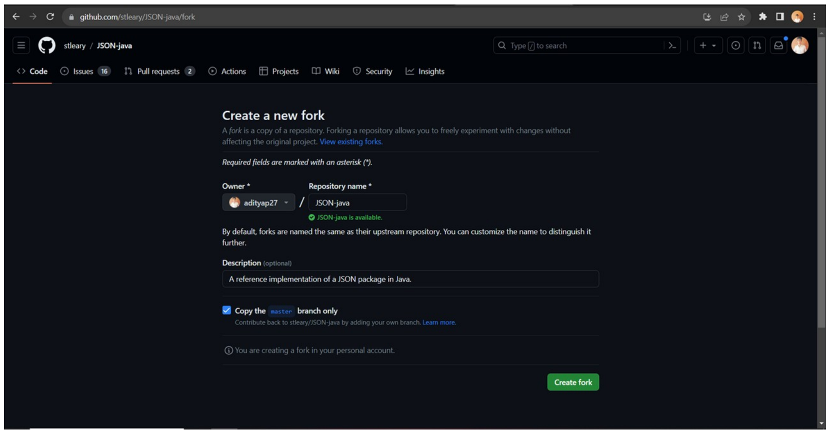
Answer: It is not a tutorial or example repository.

Condition 6: It must be active (at least one commit in the past one year)

Answer: Yes, it is active, as shown in **figure 2**.

Later steps:

1. I forked that repository in my GitHub account. Link: https://github.com/adityap27/JSON-java



github.com/stleary/JSON-java/fork

stleary / JSON-java

Code · Issues 16 · Pull requests 2 · Actions · Projects · Wiki · Security · Insights

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. View existing forks.

Required fields are marked with an asterisk (*).

Owner *: adityap27 / Repository name *: JSON-java

JSON-java is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional): A reference implementation of a JSON package in Java.

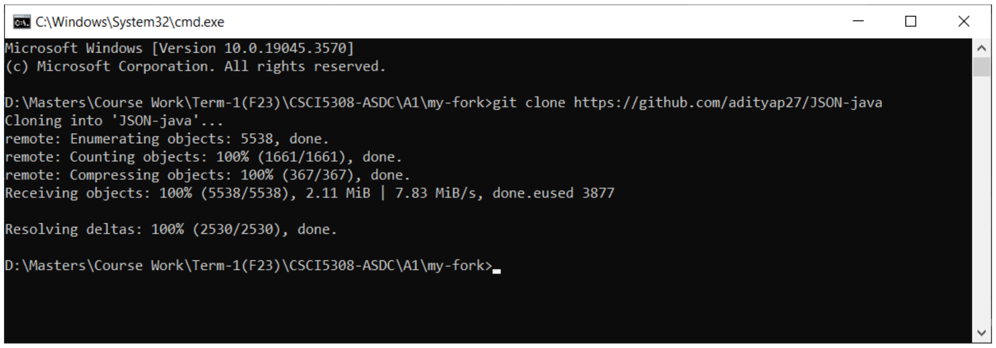
☑ Copy the `master` branch only

Contribute back to stleary/JSON-java by adding your own branch. Learn more.

You are creating a fork in your personal account.

Figure 3: Fork of JSON-java repository [1].

2. I cloned the forked repository in my local machine.



C:\Windows\System32\cmd.exe

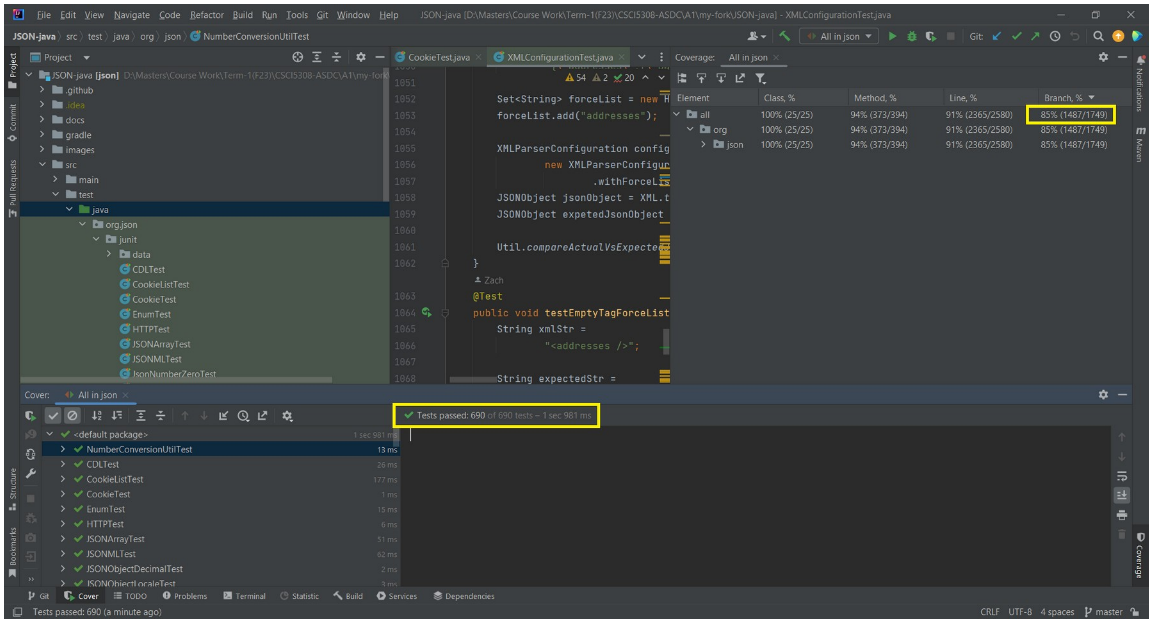
```
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserved.

D:\Masters\Course Work\Term-1(F23)\CSCI5308-ASDC\A1\my-fork>git clone https://github.com/adityap27/JSON-java
Cloning into 'JSON-java'...
remote: Enumerating objects: 5538, done.
remote: Counting objects: 100% (1661/1661), done.
remote: Compressing objects: 100% (367/367), done.
Receiving objects: 100% (5538/5538), 2.11 MiB | 7.83 MiB/s, done.eused 3877

Resolving deltas: 100% (2530/2530), done.

D:\Masters\Course Work\Term-1(F23)\CSCI5308-ASDC\A1\my-fork>
```

Figure 4: Clone forked repository into local machine.

Task 2: Quantitative measures of test implementation.

Total number of automated tests: 690, as shown in **figure 5**.

Code coverage (branch): 85, as shown in **figure 5**.



JSON-java [D:\Masters\Course Work\Term-1(F23)\CSCI5308-ASDC\A1\my-fork\JSON-java] - XMLConfigurationTest.java

```
        Set<String> forceList = new H
        forceList.add("addresses");

        XMLParserConfiguration config
                new XMLParserConfigur
                        .withForceLis
        JSONObject jsonObject = XML.t
        JSONObject expetedJsonObject

        Util.compareActualVsExpecte
    }
    @Test
    public void testEmptyTagForceList
        String xmlStr =
                "<addresses />";

        String expectedStr =
```

Coverage: All in json

Element	Class, %	Method, %	Line, %	Branch, %
all	100% (25/25)	94% (373/394)	91% (2365/2580)	85% (1487/1749)
org	100% (25/25)	94% (373/394)	91% (2365/2580)	85% (1487/1749)
json	100% (25/25)	94% (373/394)	91% (2365/2580)	85% (1487/1749)

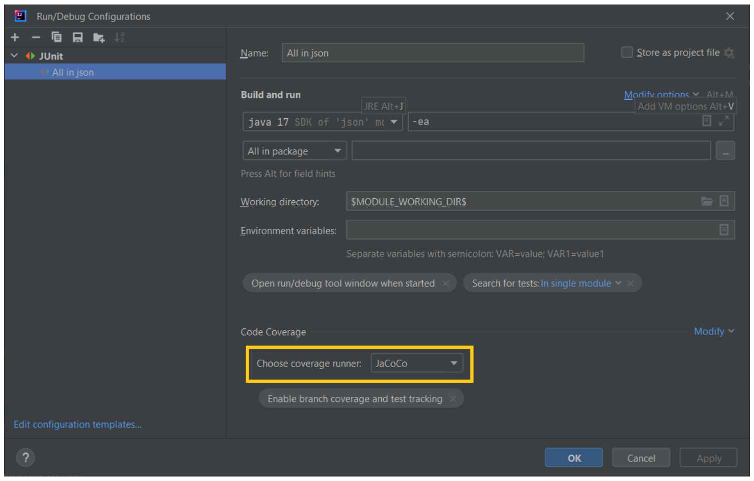
Tests passed: 690 of 690 tests – 1 sec 981 ms

```
<default package>            1 sec 981 ms
  NumberConversionUtilTest   13 ms
  CDLTest                    26 ms
  CookieListTest             177 ms
  CookieTest                 1 ms
  EnumTest                   15 ms
  HTTPTest                   6 ms
  JSONArrayTest              51 ms
  JSONMLTest                 62 ms
  JSONObjectDecimalTest      2 ms
  JSONObjectLocaleTest       3 ms
```

Tests passed: 690 (a minute ago)

Figure 5: Total number of test cases and test coverage [3] [4] [5] [6].

Note: I have used IntelliJ IDE along with JaCoCo code coverage runner, instead of the default IntelliJ code coverage runner as that default was interfering & failing 1 existing test case while finding the coverage.



Run/Debug Configurations

JUnit
All in json

Name: All in json

Build and run

java 17 SDK of 'json' mc -ea

All in package

Working directory: $MODULE_WORKING_DIR$

Environment variables:

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started

Search for tests: In single module

Code Coverage

Choose coverage runner: JaCoCo

Enable branch coverage and test tracking

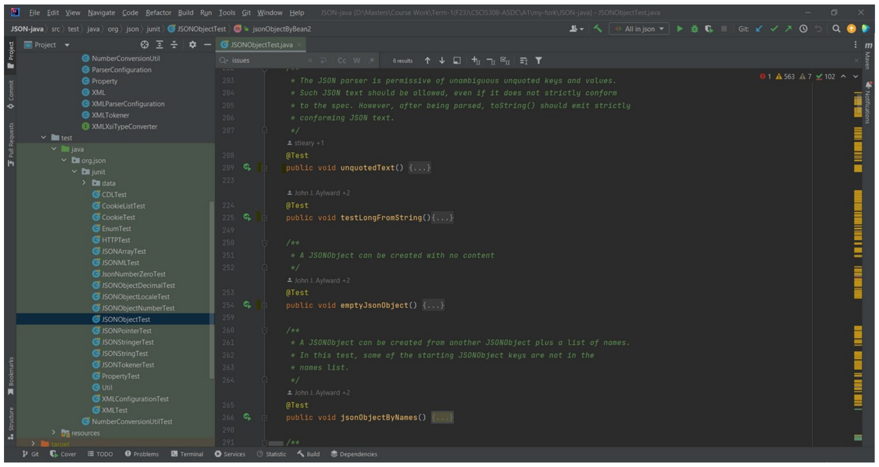
Figure 6: Choosing coverage runner in IntelliJ [3].

Task 3: Critique the test implementation.

Weak Aspects:

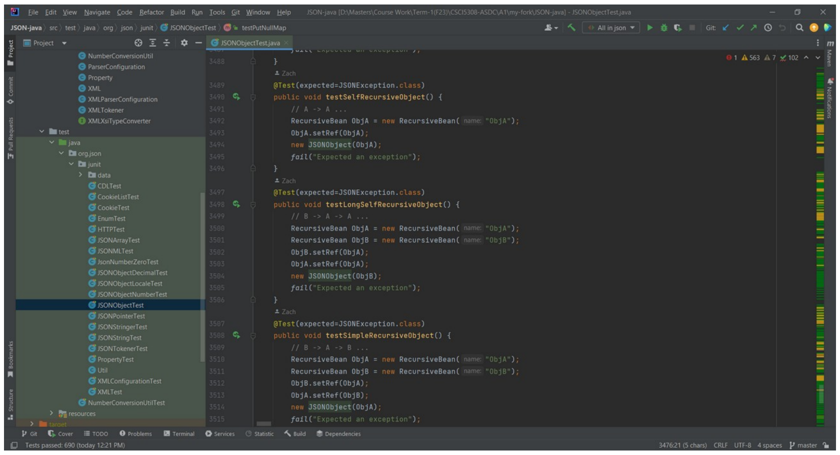
1. **Inconsistent Naming Convention of Test case methods:**
   The test case methods are having multiple naming conventions across the repository. (a) Some methods are named based on the type of input to verify. (b) Some are starting with the test keyword and then followed by the type of input to verify. (c) Some are starting with the “issue<issue_number>” keyword and then the type of input to verify and (d) some are starting with both “test” and “issue<issue_number>” keywords followed by the input type to verify.

   Example: The unquotedText() and emptyJsonObject() test cases in file JSONObjectTest.java are following the convention (a) as shown in figure 7. Here, unquotedText case has ‘str’ variable with unquoted keys and values in JSON format, which is the input type to be tested. Same file has test cases like testSelfRecursiveObject, testLongSelfRecursiveObject, etc., now with test keyword in starting as showing figure 8. Same file also has some test cases starting with issue keyword, with or without test keyword, such as issue654StackOverflowInput, issue654IncorrectNestingNoKey1, testIssue682SimilarityOfJSONString, etc. as shown in figure 9. Such variety of 4 naming conventions can be seen across the repository by the developer(s). Different files also have this inconsistency, such as JSONArrayTest.java file: issue654StackOverflowInputWellFormed and testIssue682SimilarityOfJSONString test cases.



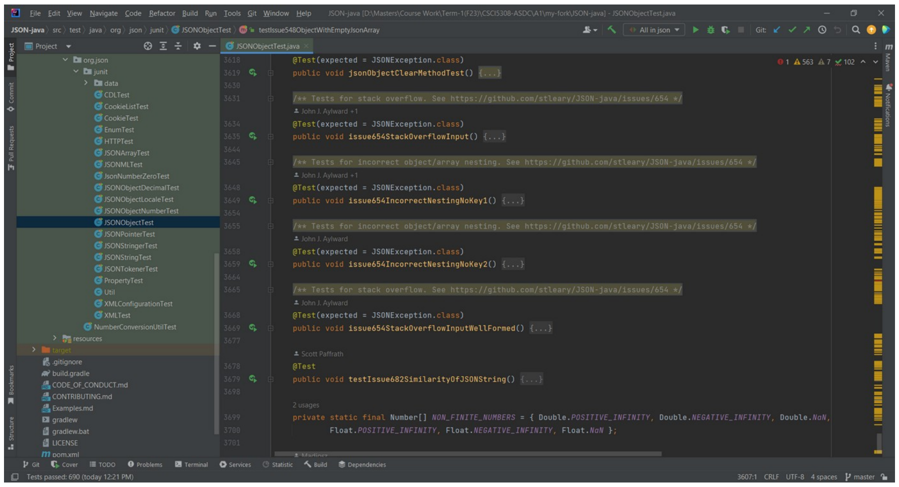
```
    /**
     * The JSON parser is permissive of unambiguous unquoted keys and values.
     * Such JSON text should be allowed, even if it does not strictly conform
     * to the spec. However, after being parsed, toString() should emit strictly
     * conforming JSON text.
     */
    @Test
    public void unquotedText() {...}

    @Test
    public void testLongFromString(){...}

    /**
     * A JSONObject can be created with no content
     */
    @Test
    public void emptyJsonObject() {...}

    /**
     * A JSONObject can be created from another JSONObject plus a list of names.
     * In this test, some of the starting JSONObject keys are not in the
     * names list.
     */
    @Test
    public void jsonObjectByNames() {...}

    /**
```

Figure 7: (a) method names based on the input type [3] [4] [5] [6].



```java
    }
    @Test(expected=JSONException.class)
    public void testSelfRecursiveObject() {
        // A -> A ...
        RecursiveBean ObjA = new RecursiveBean( name: "ObjA");
        ObjA.setRef(ObjA);
        new JSONObject(ObjA);
        fail("Expected an exception");
    }
    @Test(expected=JSONException.class)
    public void testLongSelfRecursiveObject() {
        // B -> A -> A ...
        RecursiveBean ObjA = new RecursiveBean( name: "ObjA");
        RecursiveBean ObjB = new RecursiveBean( name: "ObjB");
        ObjB.setRef(ObjA);
        ObjA.setRef(ObjA);
        new JSONObject(ObjB);
        fail("Expected an exception");
    }
    @Test(expected=JSONException.class)
    public void testSimpleRecursiveObject() {
        // B -> A -> B ...
        RecursiveBean ObjA = new RecursiveBean( name: "ObjA");
        RecursiveBean ObjB = new RecursiveBean( name: "ObjB");
        ObjB.setRef(ObjA);
        ObjA.setRef(ObjB);
        new JSONObject(ObjA);
        fail("Expected an exception");
```

Figure 8: (b) method names based on the input type + test keyword in starting [3] [4] [5] [6].



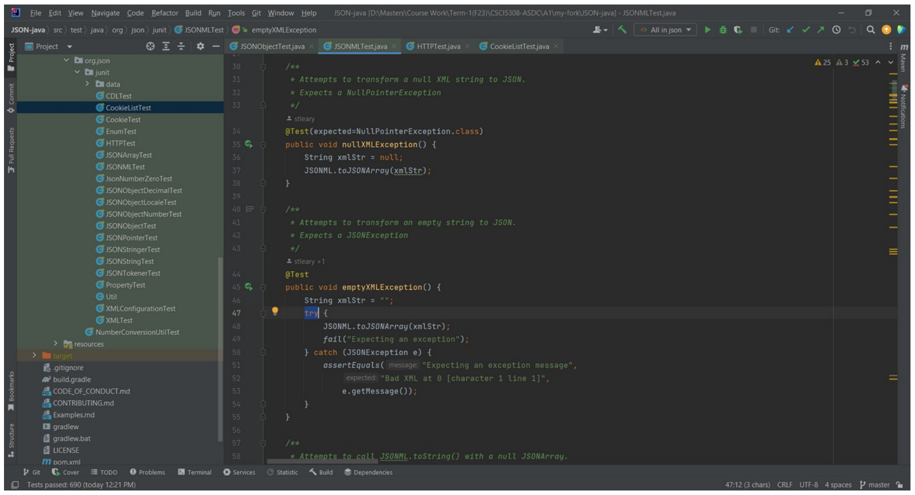
```java
    @Test(expected = JSONException.class)
    public void jsonObjectClearMethodTest() {...}

    /** Tests for stack overflow. See https://github.com/stleary/JSON-java/issues/654 */
    @Test(expected = JSONException.class)
    public void issue654StackOverflowInput() {...}

    /** Tests for incorrect object/array nesting. See https://github.com/stleary/JSON-java/issues/654 */
    @Test(expected = JSONException.class)
    public void issue654IncorrectNestingNoKey1() {...}

    /** Tests for incorrect object/array nesting. See https://github.com/stleary/JSON-java/issues/654 */
    @Test(expected = JSONException.class)
    public void issue654IncorrectNestingNoKey2() {...}

    /** Tests for stack overflow. See https://github.com/stleary/JSON-java/issues/654 */
    @Test(expected = JSONException.class)
    public void issue654StackOverflowInputWellFormed() {...}

    @Test
    public void testIssue682SimilarityOfJSONString() {...}

    private static final Number[] NON_FINITE_NUMBERS = { Double.POSITIVE_INFINITY, Double.NEGATIVE_INFINITY, Double.NaN,
            Float.POSITIVE_INFINITY, Float.NEGATIVE_INFINITY, Float.NaN };
```

Figure 9: (c) and (d), method names based on the input type + issue keyword in starting, with or without test keyword [3] [4] [5] [6].

2. **Mix of 'expected' attribute of junit and manual usage of Try-Catch for asserting exceptions:**

   There are several test cases in the repository which are testing for Exception. The developer(s) have used expected attribute of junit 4 for some tests but for many tests, manual try-catch blocks are written for exception assertions. **For example:** The nullXMLException test case in the JSONMLTest.java file uses the expected attribute to check for the NullPointerException, which will be thrown by the act statement at line number 37. However, the next test case emptyXMLException uses the try-catch block along with fail & assertEquals, to check for type of exception and message of exception as shown in figure 10. Such try-catches can be observed throughout the test folder of the repository: such as nonXMLException, emptyTagException, etc in JSONMLTest.java;

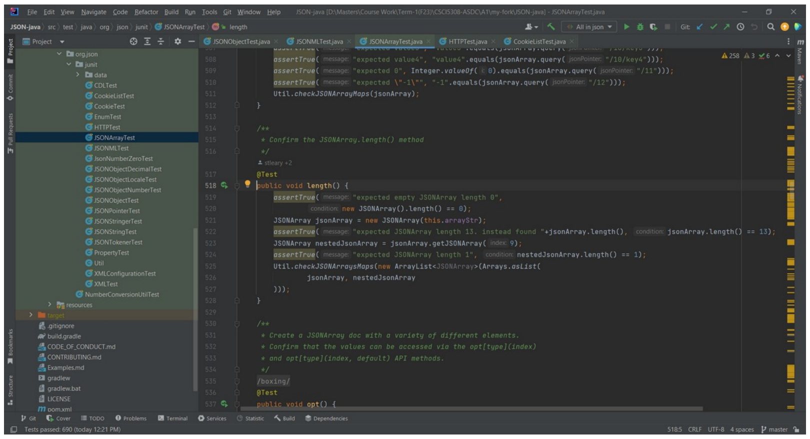
malFormedCookieListException in CookieListTest.java to name a few. A better approach would be to avoid the manual try-catch and instead use the expected attribute of junit 4 or the ExpectedException rule of junit 4.



```
/**
 * Attempts to transform a null XML string to JSON.
 * Expects a NullPointerException
 */
stleary
@Test(expected=NullPointerException.class)
public void nullXMLException() {
    String xmlStr = null;
    JSONML.toJSONArray(xmlStr);
}

/**
 * Attempts to transform an empty string to JSON.
 * Expects a JSONException
 */
stleary +1
@Test
public void emptyXMLException() {
    String xmlStr = "";
    try {
        JSONML.toJSONArray(xmlStr);
        fail("Expecting an exception");
    } catch (JSONException e) {
        assertEquals( message: "Expecting an exception message",
            expected: "Bad XML at 0 [character 1 line 1]",
            e.getMessage());
    }
}

/**
 * Attempts to call JSONML.toString() with a null JSONArray.
```

Figure 10: Usage of try-catch in test cases [3] [4] [5] [6].

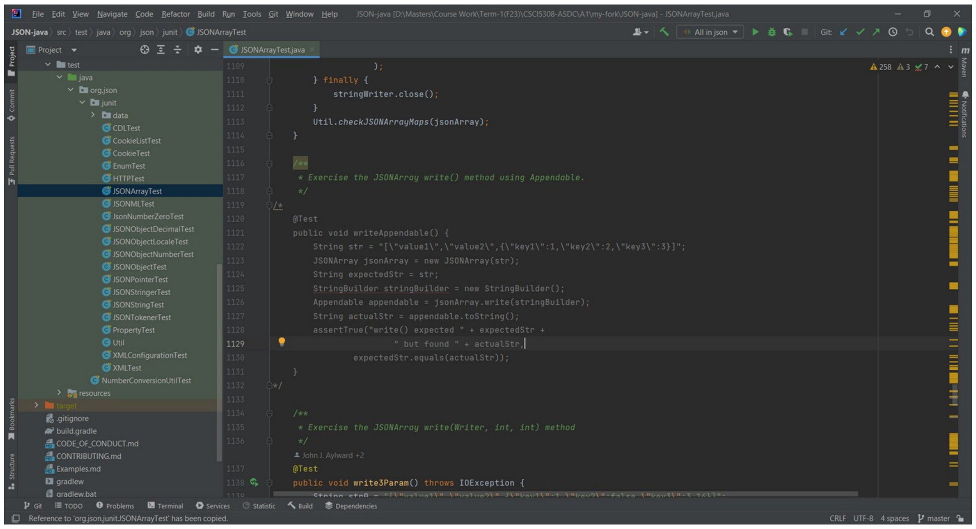
3. **Testing multiple things in a single test case:**
   There are several test cases in the repository where there are multiple tests inside a single test case method. These tests are independent and could have been written separately in-order to identify the failure of any test case better in future and also for better readability of test cases. **For example:** The "length" test case in the file JSONArrayTest has 3 assert statements as shown in figure 11. The assert at line 519, checks for 0 length array, assert at line 522 checks for some array that is having multiple elements and the assert at 524 line checks for array having single element. These 3 test cases could have been written separately in different methods. Moreover, such pattern is seen throughout the repository, such as: join, opt, optStringConversion, put method of JSONArrayTest.java file as well as unquotedText, stringToValueNumbersTest of JSONObjectTest.java to name a few.



```
508        assertTrue( message: "expected value4", "value4".equals(jsonArray.query( jsonPointer: "/10/key4")));
509        assertTrue( message: "expected 0", Integer.valueOf( i: 0).equals(jsonArray.query( jsonPointer: "/11")));
510        assertTrue( message: "expected \"-1\"", "-1".equals(jsonArray.query( jsonPointer: "/12")));
511        Util.checkJSONArrayMaps(jsonArray);
512    }
513
514    /**
515     * Confirm the JSONArray.length() method
516     */
       stleary +2
517    @Test
518    public void length() {
519        assertTrue( message: "expected empty JSONArray length 0",
520                condition: new JSONArray().length() == 0);
521        JSONArray jsonArray = new JSONArray(this.arrayStr);
522        assertTrue( message: "expected JSONArray length 13. instead found "+jsonArray.length(), condition: jsonArray.length() == 13);
523        JSONArray nestedJsonArray = jsonArray.getJSONArray( index: 9);
524        assertTrue( message: "expected JSONArray length 1", condition: nestedJsonArray.length() == 1);
525        Util.checkJSONArraysMaps(new ArrayList<JSONArray>(Arrays.asList(
526                jsonArray, nestedJsonArray
527        )));
528    }
529
530    /**
531     * Create a JSONArray doc with a variety of different elements.
532     * Confirm that the values can be accessed via the opt[type](index)
533     * and opt[type](index, default) API methods.
534     */
535    /boxing/
536    @Test
537    public void opt() {
```

Figure 11: Multiple tests in same test case [3] [4] [5] [6].

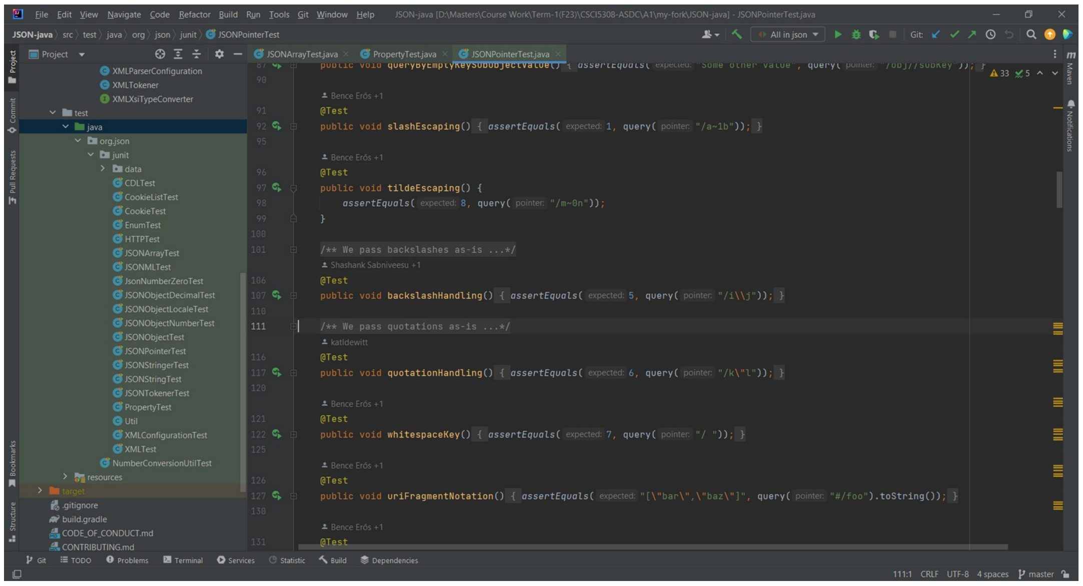
4. **Multiple commented test cases without any reasoning:**
   There are several various test cases commented in the repository. Also, the reason of disabling/commenting that test case is not mentioned anywhere. This might create a wrong impression that it is commented because it was failing and just to pas the build it has been commented to bypass the CI/CD. **For Example:**
   org.json.junit.JSONArrayTest.java on line 1120 and 1183.
   org.json.junit.JSONObjectTest.java on line 3018 and 3085.



```
1109                    );
1110            } finally {
1111                stringWriter.close();
1112            }
1113            Util.checkJSONArrayMaps(jsonArray);
1114        }
1115
1116        /**
1117         * Exercise the JSONArray write() method using Appendable.
1118         */
1119    /*
1120        @Test
1121        public void writeAppendable() {
1122            String str = "[\"value1\",\"value2\",{\"key1\":1,\"key2\":2,\"key3\":3}]";
1123            JSONArray jsonArray = new JSONArray(str);
1124            String expectedStr = str;
1125            StringBuilder stringBuilder = new StringBuilder();
1126            Appendable appendable = jsonArray.write(stringBuilder);
1127            String actualStr = appendable.toString();
1128            assertTrue("write() expected " + expectedStr +
1129                            " but found " + actualStr,
1130                    expectedStr.equals(actualStr));
1131        }
1132    */
1133
1134        /**
1135         * Exercise the JSONArray write(Writer, int, int) method
1136         */
            John J. Aylward +2
1137        @Test
1138        public void write3Param() throws IOException {
```

Figure 12: Commented test cases [3] [4] [5] [6].

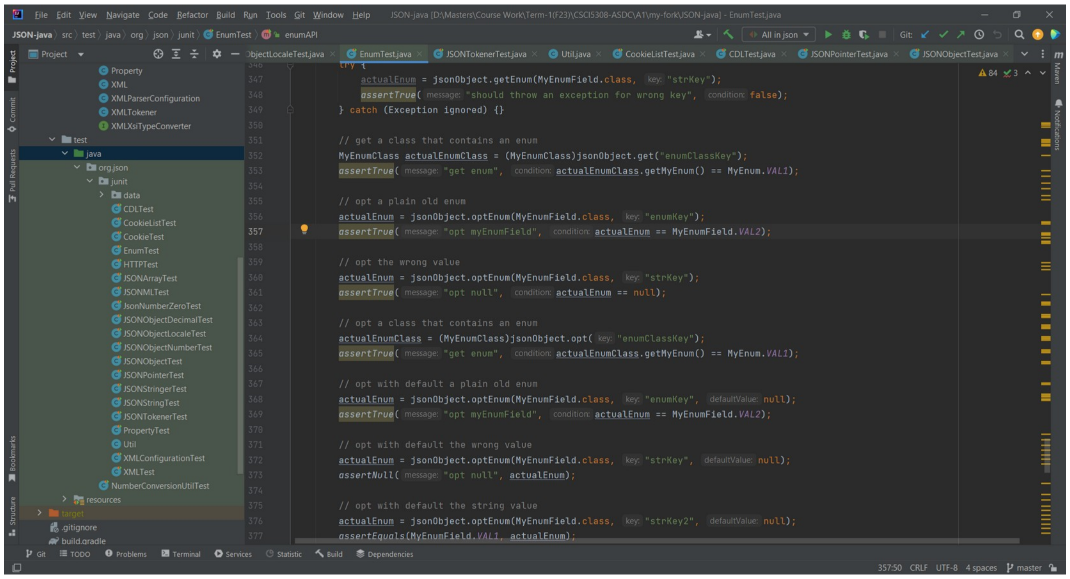
5. **Undescribed or Magic numbers in test cases:**
   There are several values that are not described via comments, variable name or any other way. This creates a confusion while some other developer is reading that test case. It may raise a question why a particular value is used. **For Example:**
   org.json.junit.JSONPointerTest.java has slashEscaping, tildeEscaping, backslashHandling, etc test cases with values such as 1, 8, 5 correspondingly. These values are not described by the developer(s).



```java
    public void queryByEmptyKeySubObjectValue() { assertEquals( expected: "Some other value", query( pointer: "/obj//subKey")); }

    Bence Erős +1
    @Test
    public void slashEscaping() { assertEquals( expected: 1, query( pointer: "/a~1b")); }

    Bence Erős +1
    @Test
    public void tildeEscaping() {
        assertEquals( expected: 8, query( pointer: "/m~0n"));
    }

    /** We pass backslashes as-is ...*/
    Shashank Sabniveesu +1
    @Test
    public void backslashHandling() { assertEquals( expected: 5, query( pointer: "/i\\j")); }

    /** We pass quotations as-is ...*/
    katldewitt
    @Test
    public void quotationHandling() { assertEquals( expected: 6, query( pointer: "/k\"l")); }

    Bence Erős +1
    @Test
    public void whitespaceKey() { assertEquals( expected: 7, query( pointer: "/ ")); }

    Bence Erős +1
    @Test
    public void uriFragmentNotation() { assertEquals( expected: "[\"bar\",\"baz\"]", query( pointer: "#/foo").toString()); }

    Bence Erős +1
    @Test
```

Figure 13: unknown numbers in test cases [3] [4] [5] [6].

6. **assertTrue being misused instead of assertEquals:**
There are several test cases where the equality of two variables is tested using == operator manually and then the assertTrue is being used. However there are some test cases where assertEquals is being used. Ideally, equalities should be asserted using assertEquals everywhere where feasible for better readability. **For Example:** The enumAPI test case of EnumTest.java file has assertTrue being used several times instead of using assertEquals of the junit's built in method.

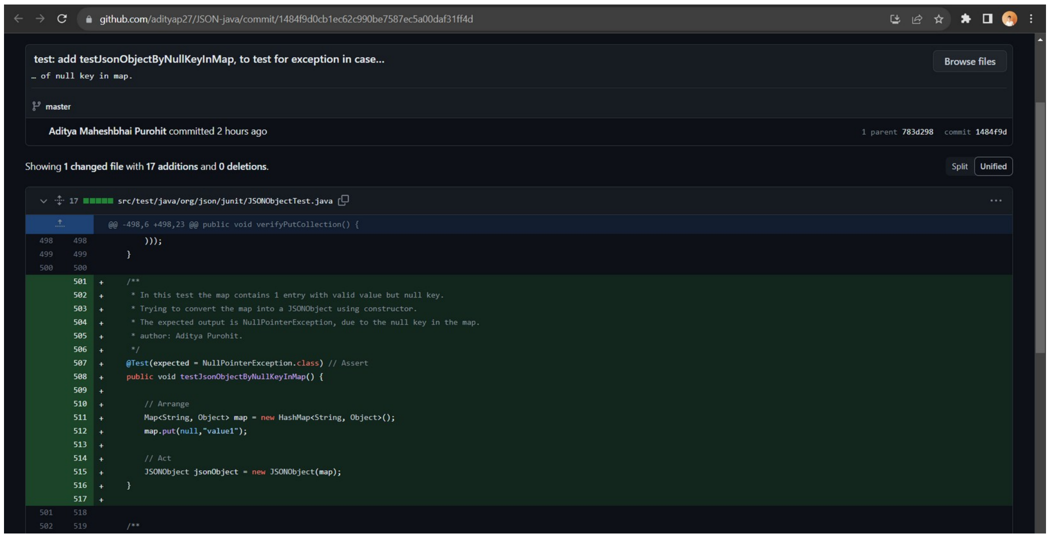


```java
        try {
            actualEnum = jsonObject.getEnum(MyEnumField.class, key: "strKey");
            assertTrue( message: "should throw an exception for wrong key", condition: false);
        } catch (Exception ignored) {}

        // get a class that contains an enum
        MyEnumClass actualEnumClass = (MyEnumClass)jsonObject.get("enumClassKey");
        assertTrue( message: "get enum", condition: actualEnumClass.getMyEnum() == MyEnum.VAL1);

        // opt a plain old enum
        actualEnum = jsonObject.optEnum(MyEnumField.class, key: "enumKey");
        assertTrue( message: "opt myEnumField", condition: actualEnum == MyEnumField.VAL2);

        // opt the wrong value
        actualEnum = jsonObject.optEnum(MyEnumField.class, key: "strKey");
        assertTrue( message: "opt null", condition: actualEnum == null);

        // opt a class that contains an enum
        actualEnumClass = (MyEnumClass)jsonObject.opt( key: "enumClassKey");
        assertTrue( message: "get enum", condition: actualEnumClass.getMyEnum() == MyEnum.VAL1);

        // opt with default a plain old enum
        actualEnum = jsonObject.optEnum(MyEnumField.class, key: "enumKey", defaultValue: null);
        assertTrue( message: "opt myEnumField", condition: actualEnum == MyEnumField.VAL2);

        // opt with default the wrong value
        actualEnum = jsonObject.optEnum(MyEnumField.class, key: "strKey", defaultValue: null);
        assertNull( message: "opt null", actualEnum);

        // opt with default the string value
        actualEnum = jsonObject.optEnum(MyEnumField.class, key: "strKey2", defaultValue: null);
        assertEquals(MyEnumField.VAL1, actualEnum);
```

Figure 14: assertTrue being used instead of assertEquals [3] [4] [5] [6].

Task 4: New tests for the repository.

Test Case 1: This is to check for an exception while converting from map to JSONObject while having a null key in the map. This increased 1 unit in the branch coverage.



github.com/adityap27/JSON-java/commit/1484f9d0cb1ec62c990be7587ec5a00daf31ff4d

test: add testJsonObjectByNullKeyInMap, to test for exception in case…
… of null key in map.

master

Aditya Maheshbhai Purohit committed 2 hours ago — 1 parent 783d298 — commit 1484f9d

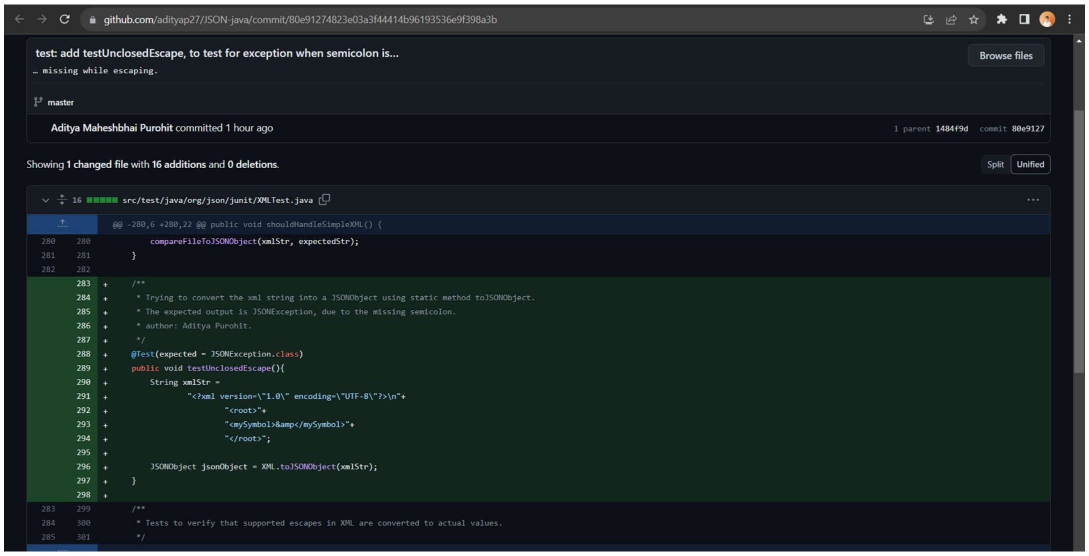
Showing 1 changed file with 17 additions and 0 deletions.

17 src/test/java/org/json/junit/JSONObjectTest.java

```
@@ -498,6 +498,23 @@ public void verifyPutCollection() {
            )));
        }

+    /**
+     * In this test the map contains 1 entry with valid value but null key.
+     * Trying to convert the map into a JSONObject using constructor.
+     * The expected output is NullPointerException, due to the null key in the map.
+     * author: Aditya Purohit.
+     */
+    @Test(expected = NullPointerException.class) // Assert
+    public void testJsonObjectByNullKeyInMap() {
+
+        // Arrange
+        Map<String, Object> map = new HashMap<String, Object>();
+        map.put(null,"value1");
+
+        // Act
+        JSONObject jsonObject = new JSONObject(map);
+    }
+

    /**
```

Figure 15: Test case 1.

Test Case 2: This is to check for an exception while converting from xml to JSONObject while having improper escaping. This again increased 1 unit in the branch coverage.



github.com/adityap27/JSON-java/commit/80e91274823e03a3f44414b96193536e9f398a3b

test: add testUnclosedEscape, to test for exception when semicolon is…
… missing while escaping.

master

Aditya Maheshbhai Purohit committed 1 hour ago — 1 parent 1484f9d — commit 80e9127

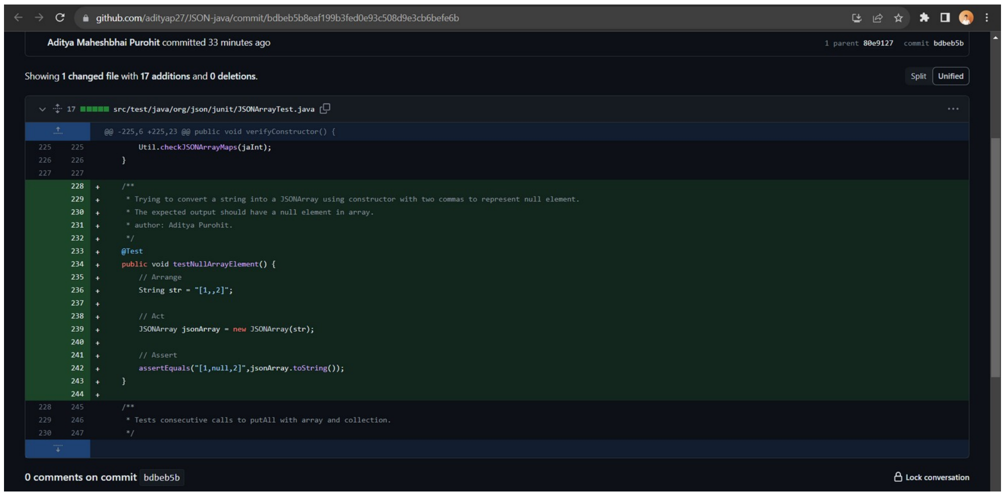
Showing 1 changed file with 16 additions and 0 deletions.

16 src/test/java/org/json/junit/XMLTest.java

```
@@ -280,6 +280,22 @@ public void shouldHandleSimpleXML() {
            compareFileToJSONObject(xmlStr, expectedStr);
        }

+    /**
+     * Trying to convert the xml string into a JSONObject using static method toJSONObject.
+     * The expected output is JSONException, due to the missing semicolon.
+     * author: Aditya Purohit.
+     */
+    @Test(expected = JSONException.class)
+    public void testUnclosedEscape(){
+        String xmlStr =
+                "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"+
+                        "<root>"+
+                        "<mySymbol>&amp</mySymbol>"+
+                        "</root>";
+
+        JSONObject jsonObject = XML.toJSONObject(xmlStr);
+    }
+
    /**
     * Tests to verify that supported escapes in XML are converted to actual values.
     */
```

Figure 16: Test case 2.

Test Case 3: This is to check for a null element generation while converting from the string to JSONArray object. This again increased 1 unit in the branch coverage.



Aditya Maheshbhai Purohit committed 33 minutes ago — 1 parent 80e9127 commit bdbeb5b

Showing 1 changed file with 17 additions and 0 deletions.

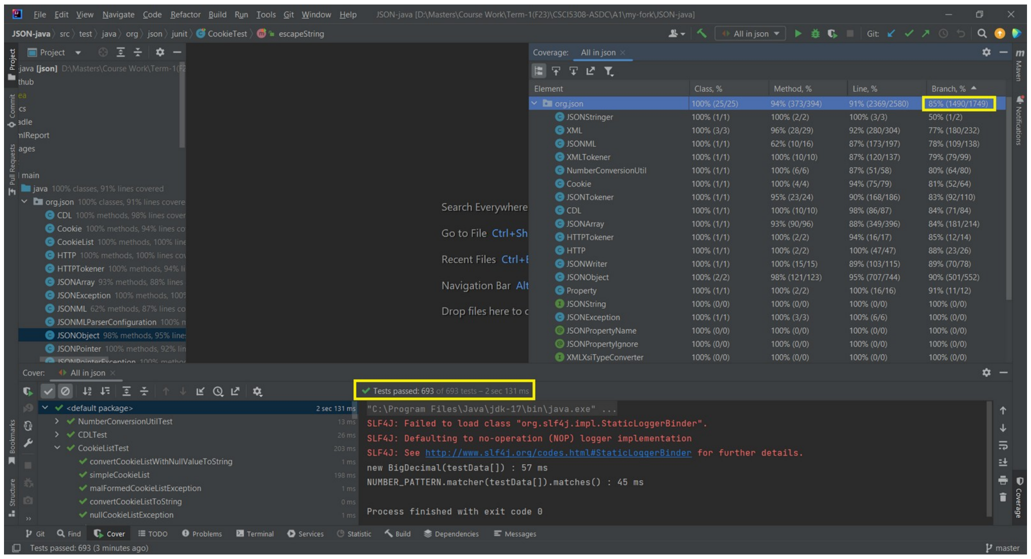
src/test/java/org/json/junit/JSONArrayTest.java

```
@@ -225,6 +225,23 @@ public void verifyConstructor() {
         Util.checkJSONArrayMaps(jaInt);
     }
 
+    /**
+     * Trying to convert a string into a JSONArray using constructor with two commas to represent null element.
+     * The expected output should have a null element in array.
+     * author: Aditya Purohit.
+     */
+    @Test
+    public void testNullArrayElement() {
+        // Arrange
+        String str = "[1,,2]";
+
+        // Act
+        JSONArray jsonArray = new JSONArray(str);
+
+        // Assert
+        assertEquals("[1,null,2]",jsonArray.toString());
+    }
+
     /**
      * Tests consecutive calls to putAll with array and collection.
      */
```

0 comments on commit bdbeb5b

Figure 17: Test case 3.

After adding 3 new test cases the branch coverage has increased from **85.02%** (1487/1749) to **85.19%** (1490/1749) as shown in figure below. Also, all the test cases are still passing and build is passing when executing mvn compile or mvn package.



Coverage: All in json

Element	Class, %	Method, %	Line, %	Branch, %
org.json	100% (25/25)	94% (373/394)	91% (2369/2580)	85% (1490/1749)
JSONStringer	100% (1/1)	100% (2/2)	100% (3/3)	50% (1/2)
XML	100% (3/3)	96% (28/29)	92% (280/304)	77% (180/232)
JSONML	100% (1/1)	62% (10/16)	87% (173/197)	78% (109/138)
XMLTokener	100% (1/1)	100% (10/10)	87% (120/137)	79% (79/99)
NumberConversionUtil	100% (1/1)	100% (6/6)	87% (51/58)	80% (64/80)
Cookie	100% (1/1)	100% (4/4)	94% (75/79)	81% (52/64)
JSONTokener	100% (1/1)	95% (23/24)	90% (168/186)	83% (92/110)
CDL	100% (1/1)	100% (10/10)	98% (86/87)	84% (71/84)
JSONArray	100% (1/1)	93% (90/96)	88% (349/396)	84% (181/214)
HTTPTokener	100% (1/1)	100% (2/2)	94% (16/17)	85% (12/14)
HTTP	100% (1/1)	100% (2/2)	100% (47/47)	88% (23/26)
JSONWriter	100% (1/1)	100% (15/15)	89% (103/115)	89% (70/78)
JSONObject	100% (2/2)	98% (121/123)	95% (707/744)	90% (501/552)
Property	100% (1/1)	100% (2/2)	100% (16/16)	91% (11/12)
JSONString	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
JSONException	100% (1/1)	100% (3/3)	100% (6/6)	100% (0/0)
JSONPropertyName	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
JSONPropertyIgnore	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
XMLXsiTypeConverter	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)

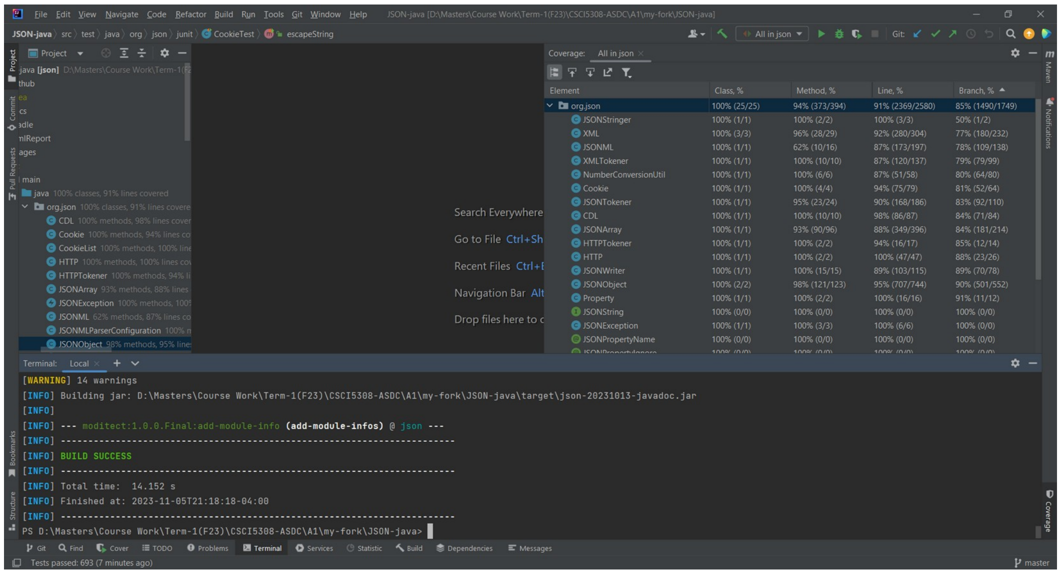
Tests passed: 693 of 693 tests – 2 sec 131 ms

```
<default package>                            2 sec 131 ms
  NumberConversionUtilTest                   13 ms
  CDLTest                                    26 ms
  CookieListTest                             203 ms
    convertCookieListWithNullValueToString   1 ms
    simpleCookieList                         198 ms
    malFormedCookieListException             1 ms
    convertCookieListToString                0 ms
    nullCookieListException                  1 ms
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
new BigDecimal(testData[]) : 57 ms
NUMBER_PATTERN.matcher(testData[]).matches() : 45 ms

Process finished with exit code 0
```

Tests passed: 693 (3 minutes ago)

Figure 18: Increased branch coverage [3] [4] [5] [6].



Element	Class, %	Method, %	Line, %	Branch, %
org.json	100% (25/25)	94% (373/394)	91% (2369/2580)	85% (1490/1749)
JSONStringer	100% (1/1)	100% (2/2)	100% (3/3)	50% (1/2)
XML	100% (3/3)	96% (28/29)	92% (280/304)	77% (180/232)
JSONML	100% (1/1)	62% (10/16)	87% (173/197)	78% (109/138)
XMLTokener	100% (1/1)	100% (10/10)	87% (120/137)	79% (79/99)
NumberConversionUtil	100% (1/1)	100% (6/6)	87% (51/58)	80% (64/80)
Cookie	100% (1/1)	100% (4/4)	94% (75/79)	81% (52/64)
JSONTokener	100% (1/1)	95% (23/24)	90% (168/186)	83% (92/110)
CDL	100% (1/1)	100% (10/10)	98% (86/87)	84% (71/84)
JSONArray	100% (1/1)	93% (90/96)	88% (349/396)	84% (181/214)
HTTPTokener	100% (1/1)	100% (2/2)	94% (16/17)	85% (12/14)
HTTP	100% (1/1)	100% (2/2)	100% (47/47)	88% (23/26)
JSONWriter	100% (1/1)	100% (15/15)	89% (103/115)	89% (70/78)
JSONObject	100% (2/2)	98% (121/123)	95% (707/744)	90% (501/552)
Property	100% (1/1)	100% (2/2)	100% (16/16)	91% (11/12)
JSONString	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
JSONException	100% (1/1)	100% (3/3)	100% (6/6)	100% (0/0)
JSONPropertyName	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)

```
[WARNING] 14 warnings
[INFO] Building jar: D:\Masters\Course Work\Term-1(F23)\CSCI5308-ASDC\A1\my-fork\JSON-java\target\json-20231013-javadoc.jar
[INFO]
[INFO] --- moditect:1.0.0.Final:add-module-info (add-module-infos) @ json ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  14.152 s
[INFO] Finished at: 2023-11-05T21:18:18-04:00
[INFO] ------------------------------------------------------------------------
PS D:\Masters\Course Work\Term-1(F23)\CSCI5308-ASDC\A1\my-fork\JSON-java>
```

Tests passed: 693 (7 minutes ago)

Figure 19: Build still passing after adding 3 new test cases [3] [4] [5] [6].

Forked repository link: https://github.com/adityap27/JSON-java/commits/master

References:

[1] Stleary, “Stleary/JSON-java: A reference implementation of a JSON package in Java.,” *GitHub*. [Online]. Available: https://github.com/stleary/JSON-java. [Accessed Nov. 5, 2023].

[2] AlDanial, “Aldanial/Cloc: Cloc Counts Blank Lines, comment lines, and physical lines of source code in many programming languages.,” *GitHub*. [Online]. Available: https://github.com/AlDanial/cloc. [Accessed Nov. 5, 2023].

[3] “IntelliJ IDEA – the leading Java and Kotlin IDE,” *JetBrains*. [Online]. Available: https://www.jetbrains.com/idea/. [Accessed: 05-Nov-2023].

[4] B. Porter, J. van Zyl, and O. Lamy, “Welcome to Apache maven,” *Apache.org*. [Online]. Available: https://maven.apache.org/. [Accessed: 05-Nov-2023].

[5] “Java Archive Downloads - Java SE 17,” *Oracle.com*. [Online]. Available: https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html. [Accessed: 05-Nov-2023].

[6] “JUnit – about,” *Junit.org*. [Online]. Available: https://junit.org/junit4/. [Accessed: 05-Nov-2023].