دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

آزمایش سوم

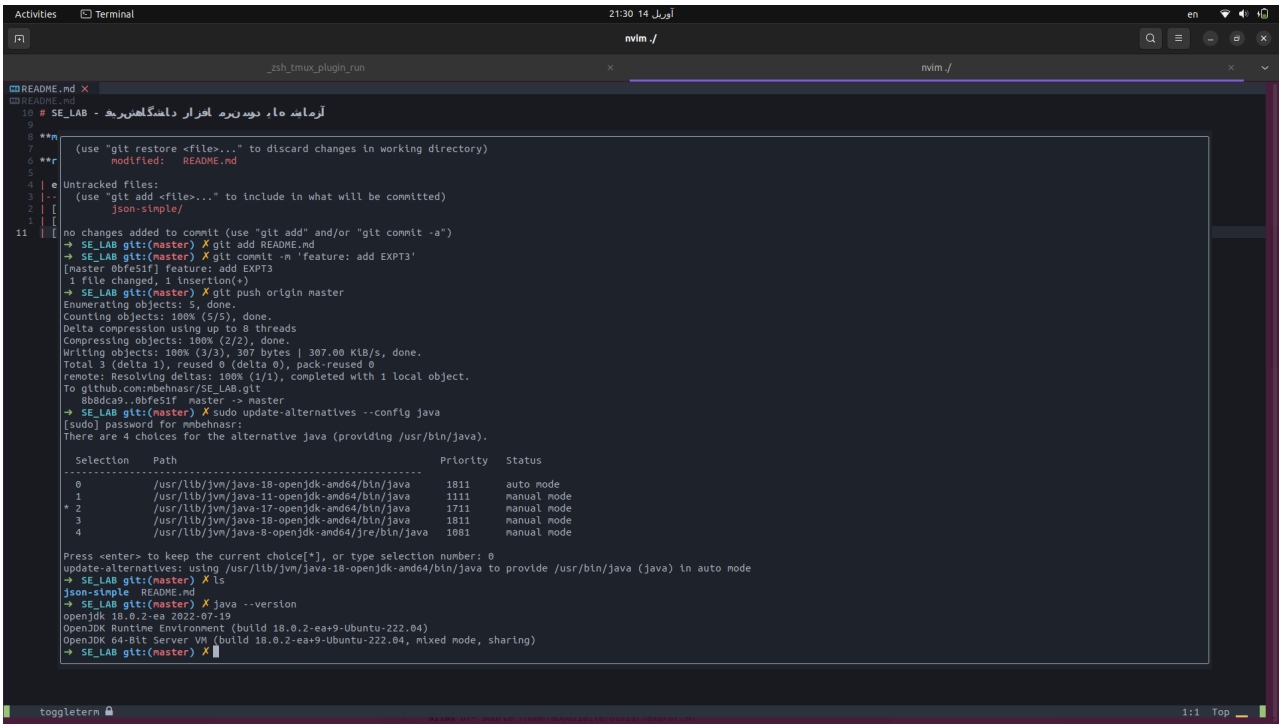
مهندسی نرم‌افزار

محمدرضا دولتی
۹۷۱۱۰۴۱۱

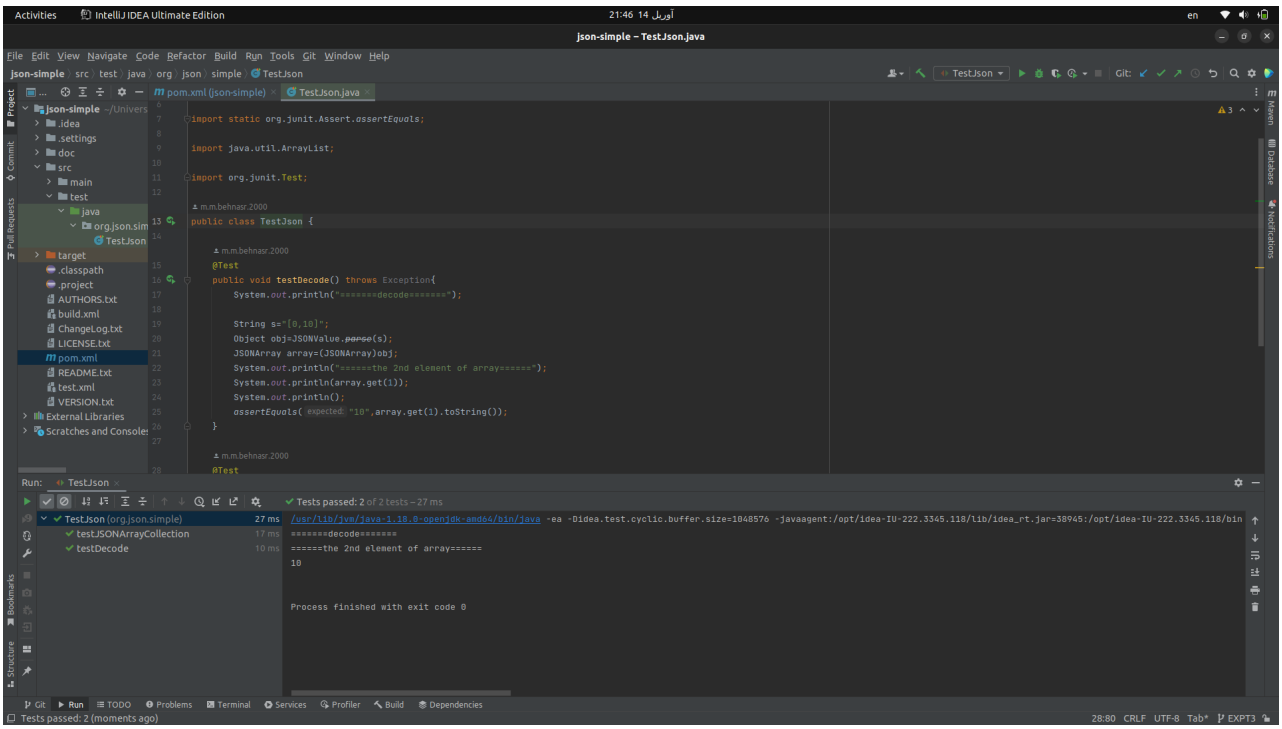
محمد مهدی به‌نصر
۹۷۱۰۵۷۹۳

فروردین ۱۴۰۲

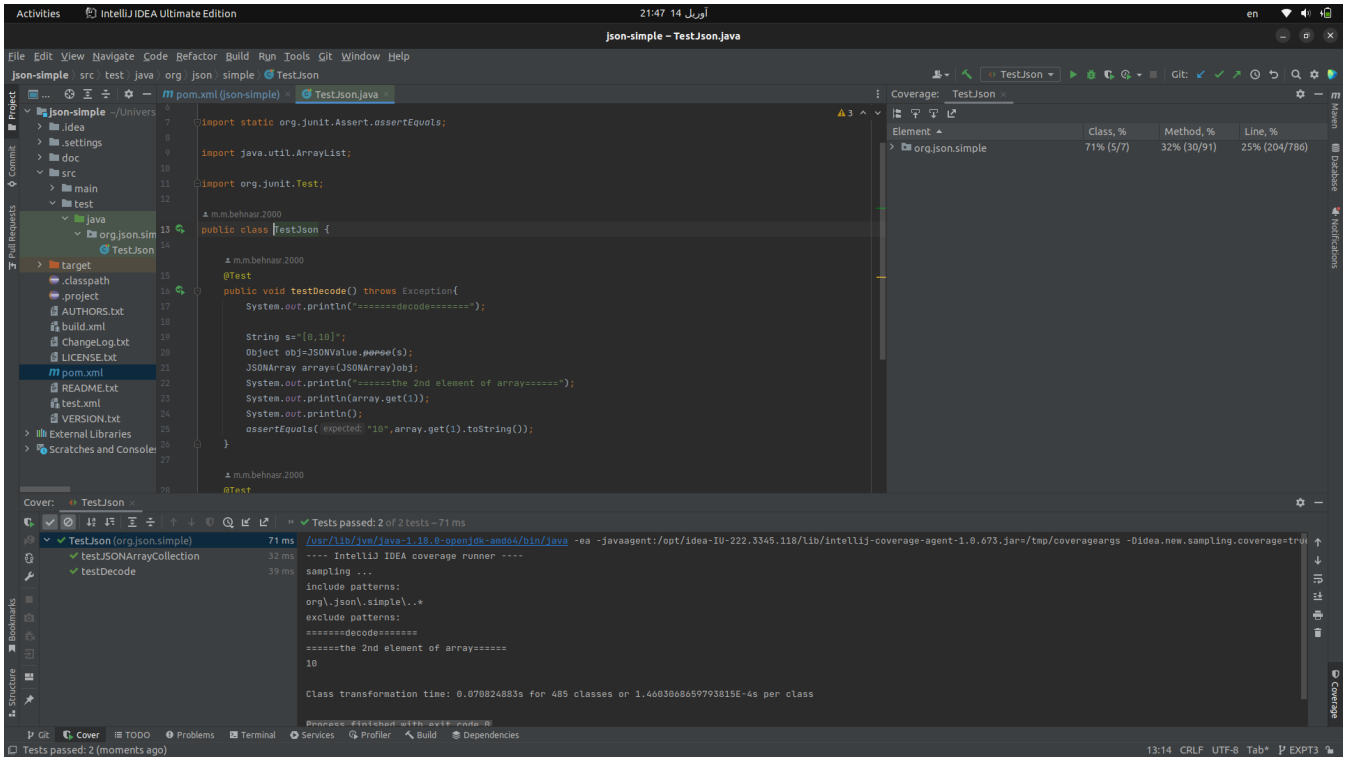
در اولین قدم پروژه را config کردیم و depeenency ها را initial کردیم و برای build کردن پروژه قسمت source و target فایل pom, ${java.version} را ست می‌کنیم:

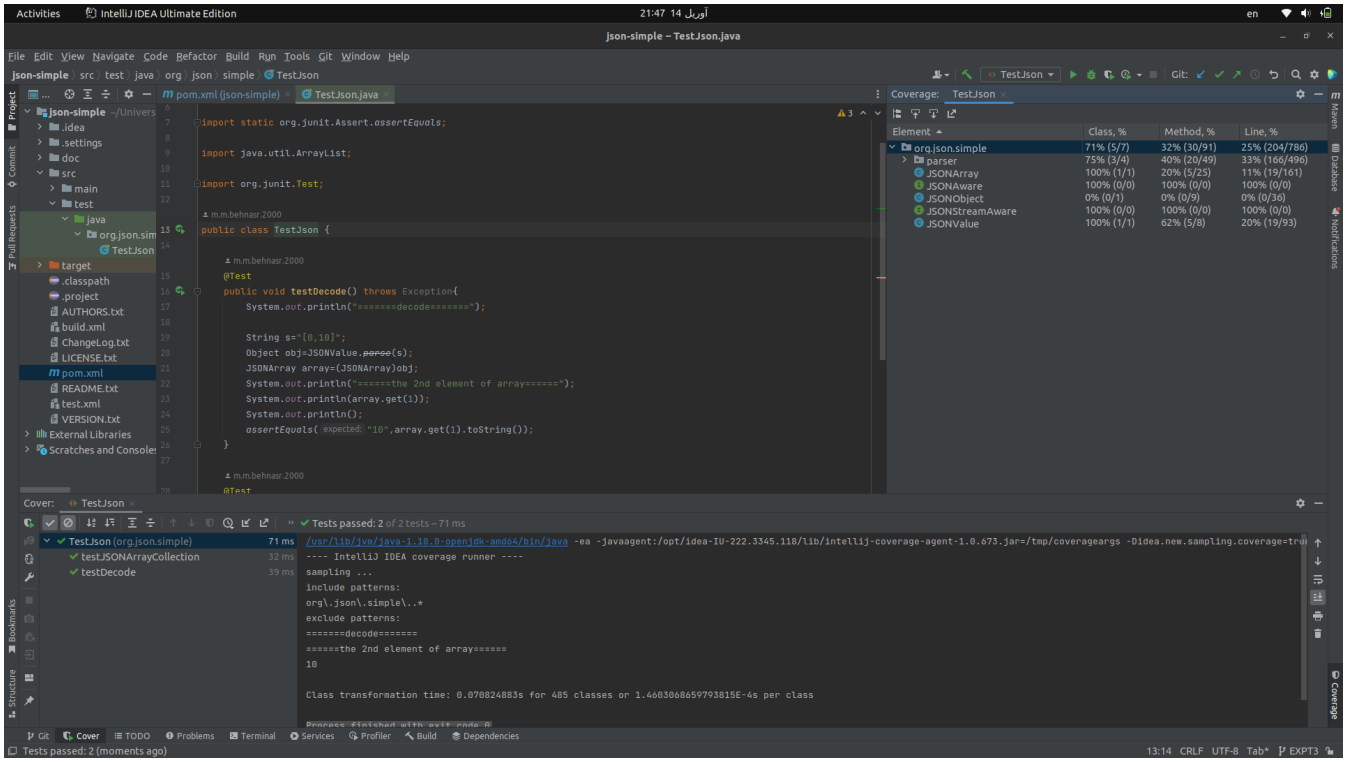


سپس تست‌های مربوطه را اجرا می‌کنیم:

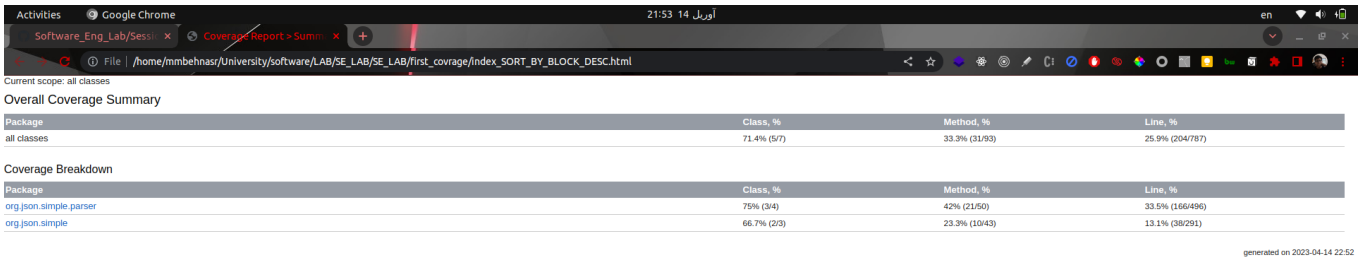


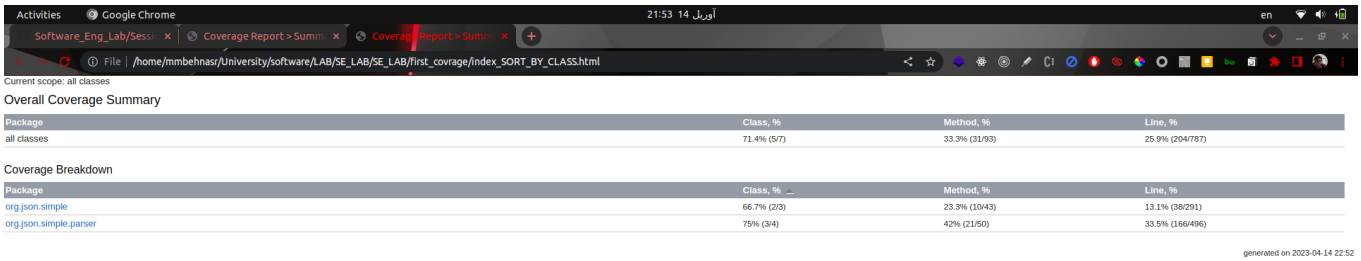
ابتدا از کد اولیه coverage می‌گیریم و نتیجه را مشاهده می‌کنیم:

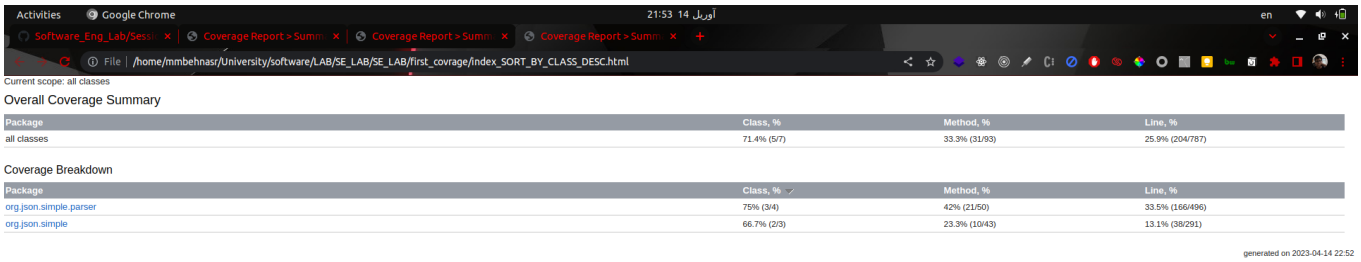


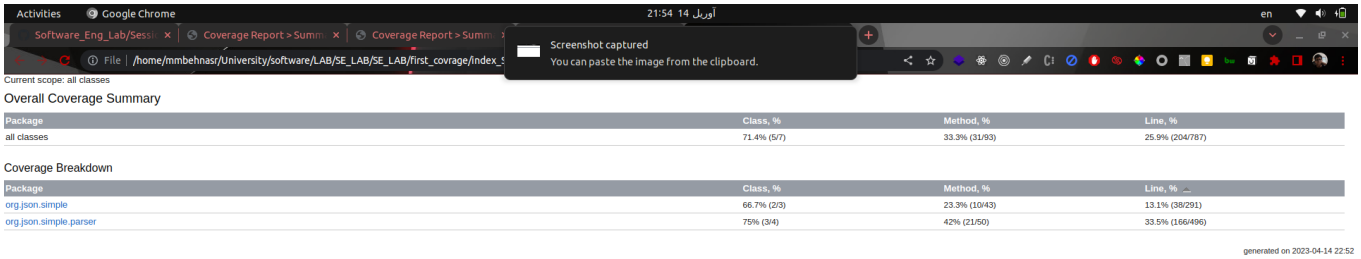


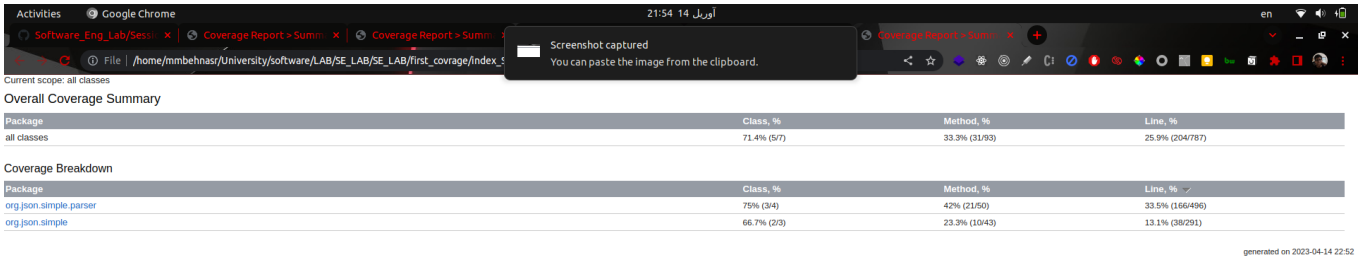
سپس، از نتیجه خروجی HTML نیز می‌گیریم:

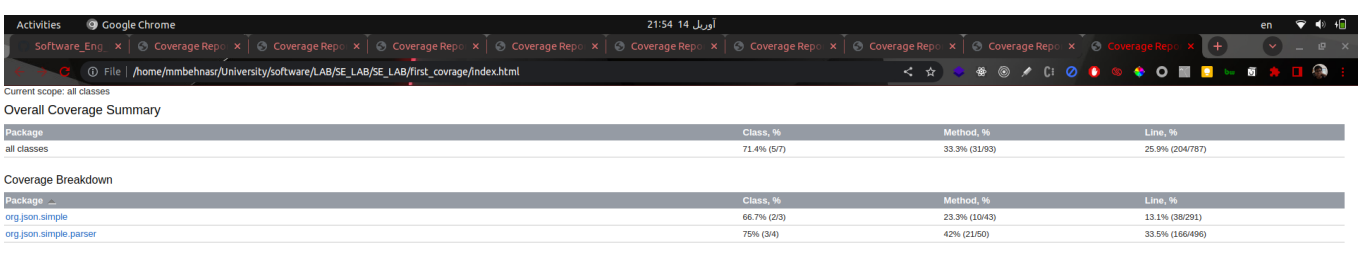






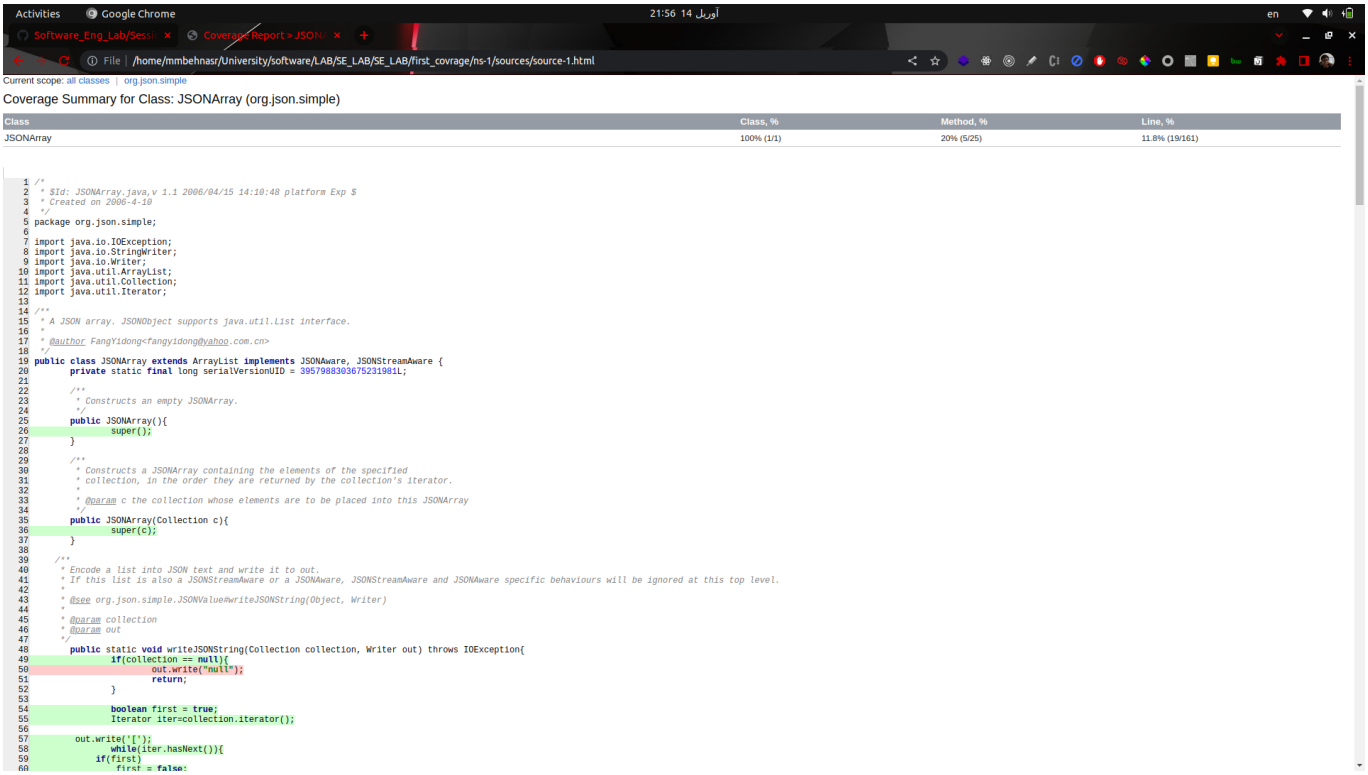




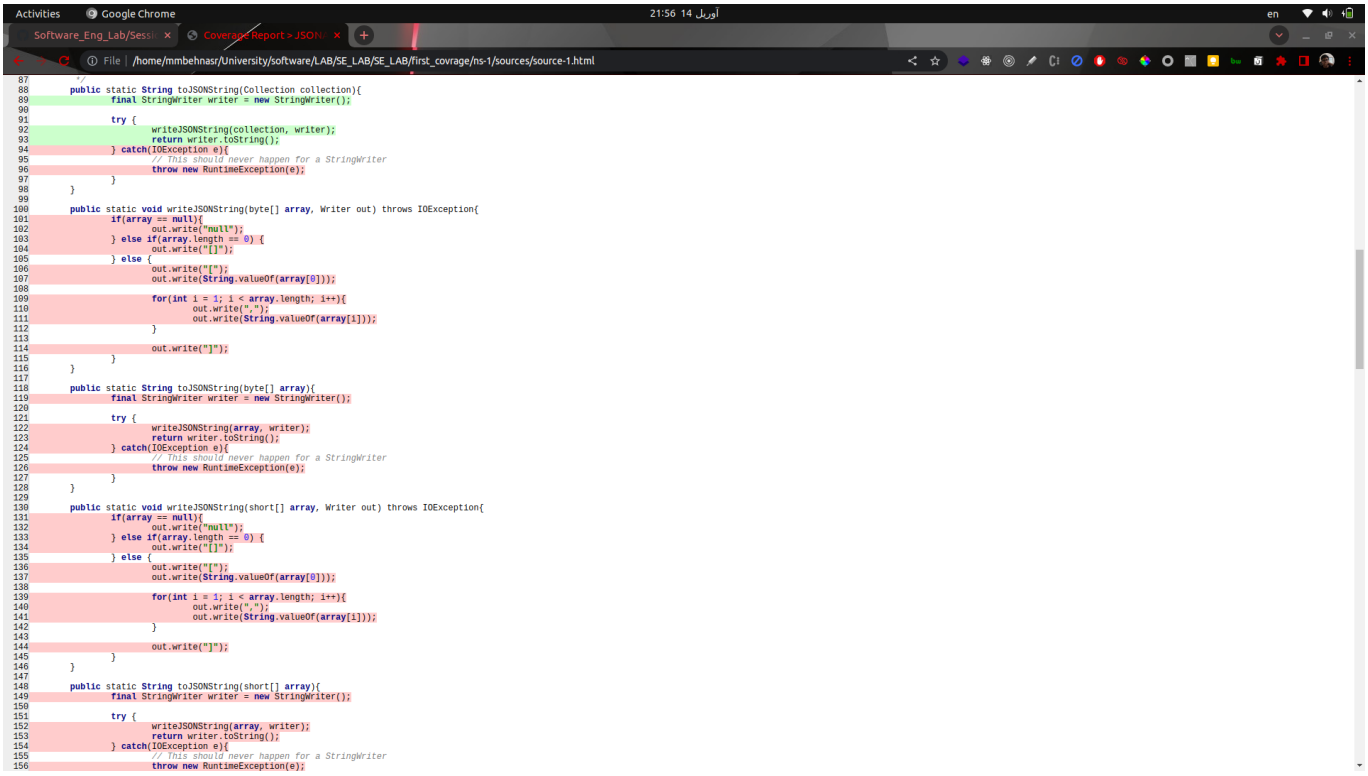


در صورت کلیک بر روی هر یک از لینک‌های فایل گزارش، خطوط پوشش داده شده هر تست به رنگ سبز و خطوط پوشش داده نشده به‌رنگ قرمز نشان داده می‌شود:

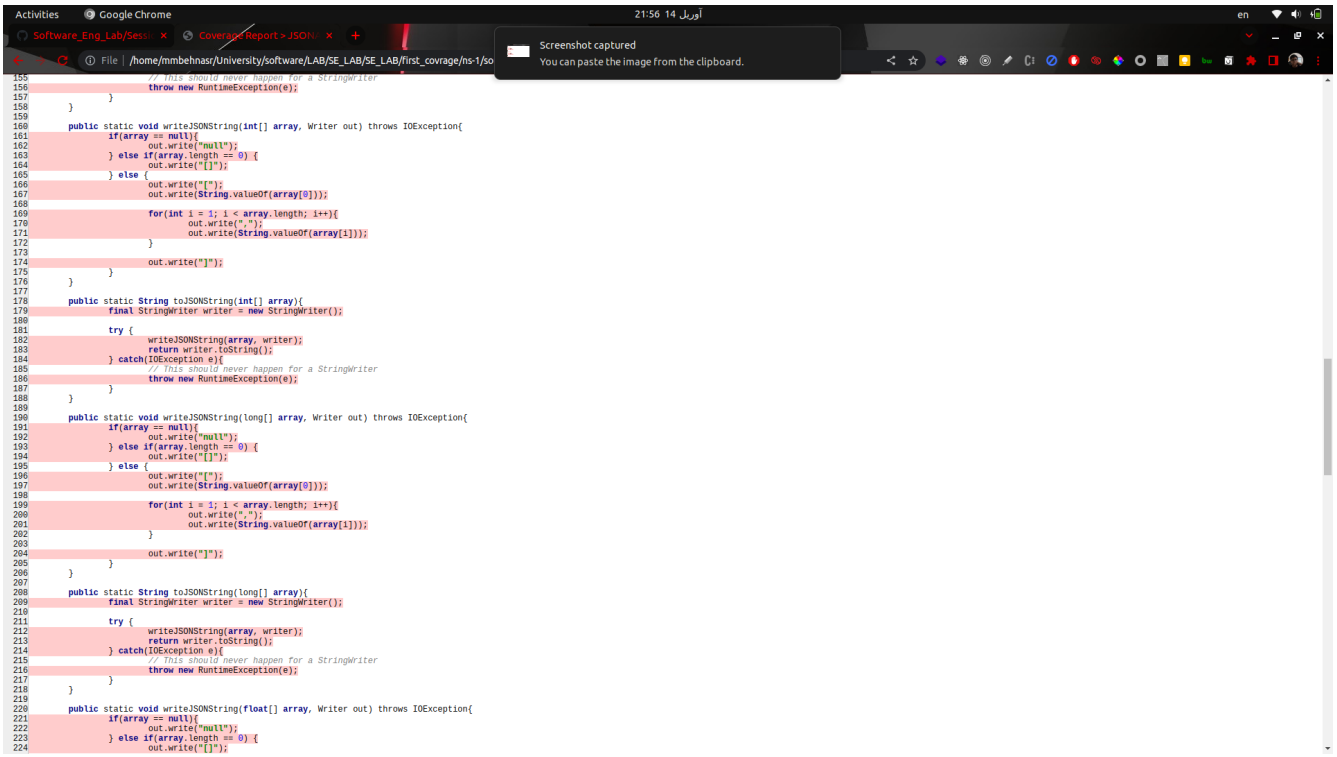
کلاس JSONArray:



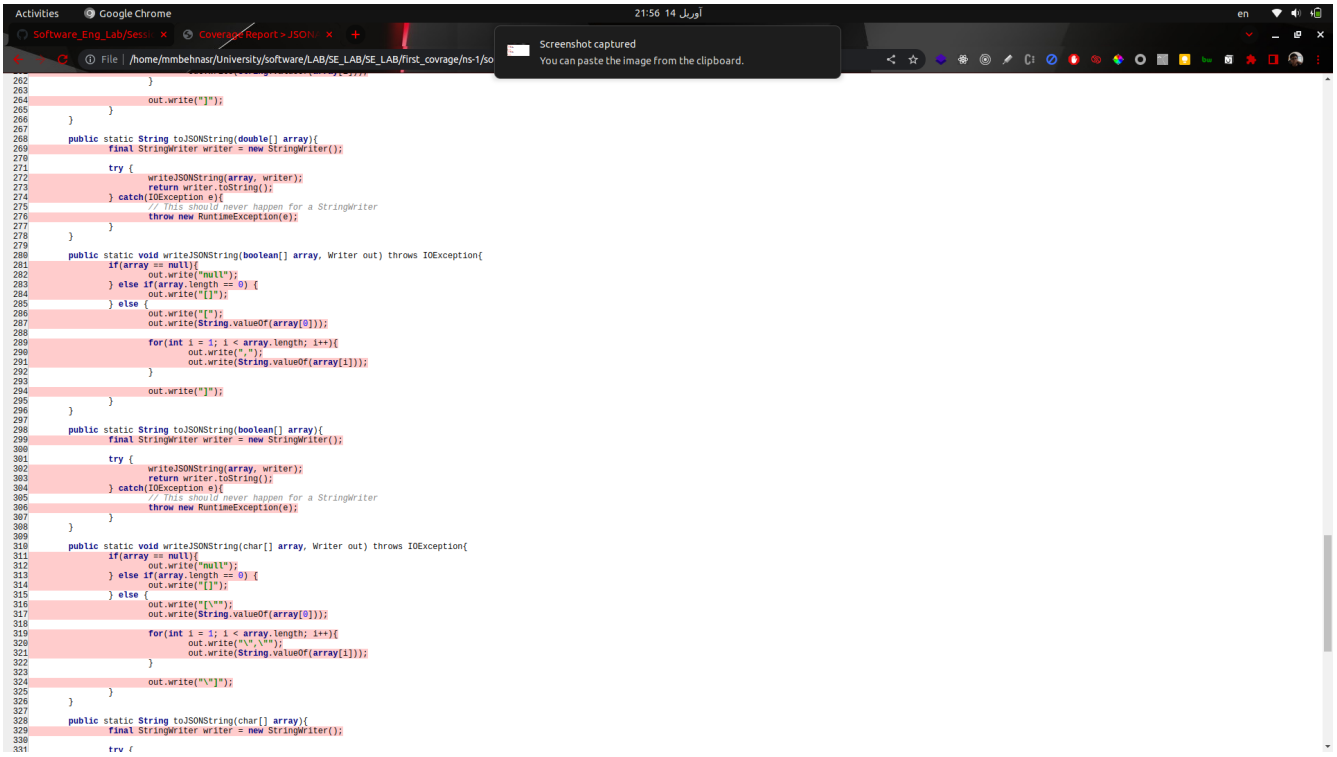

```
1  /*
2   * $Id: JSONArray.java,v 1.1 2006/04/15 14:10:48 platform Exp $
3   * Created on 2006-4-10
4   */
5  package org.json.simple;
6
7  import java.io.IOException;
8  import java.io.StringWriter;
9  import java.io.Writer;
10 import java.util.ArrayList;
11 import java.util.Collection;
12 import java.util.Iterator;
13
14 /**
15  * A JSON array. JSONObject supports java.util.List interface.
16  *
17  * @author FangYidong<fangyidong@yahoo.com.cn>
18  */
19 public class JSONArray extends ArrayList implements JSONAware, JSONStreamAware {
20     private static final long serialVersionUID = 3957988303675231981L;
21
22     /**
23      * Constructs an empty JSONArray.
24      */
25     public JSONArray(){
26         super();
27     }
28
29     /**
30      * Constructs a JSONArray containing the elements of the specified
31      * collection, in the order they are returned by the collection's iterator.
32      *
33      * @param c the collection whose elements are to be placed into this JSONArray
34      */
35     public JSONArray(Collection c){
36         super(c);
37     }
38
39    /**
40     * Encode a list into JSON text and write it to out.
41     * If this list is also a JSONStreamAware or a JSONAware, JSONStreamAware and JSONAware specific behaviours will be ignored at this top level.
42     *
43     * @see org.json.simple.JSONValue#writeJSONString(Object, Writer)
44     *
45     * @param collection
46     * @param out
47     */
48     public static void writeJSONString(Collection collection, Writer out) throws IOException{
49         if(collection == null){
50             out.write("null");
51             return;
52         }
53
54         boolean first = true;
55         Iterator iter=collection.iterator();
56
57     out.write('[');
58         while(iter.hasNext()){
59       if(first)
60         first = false;
```



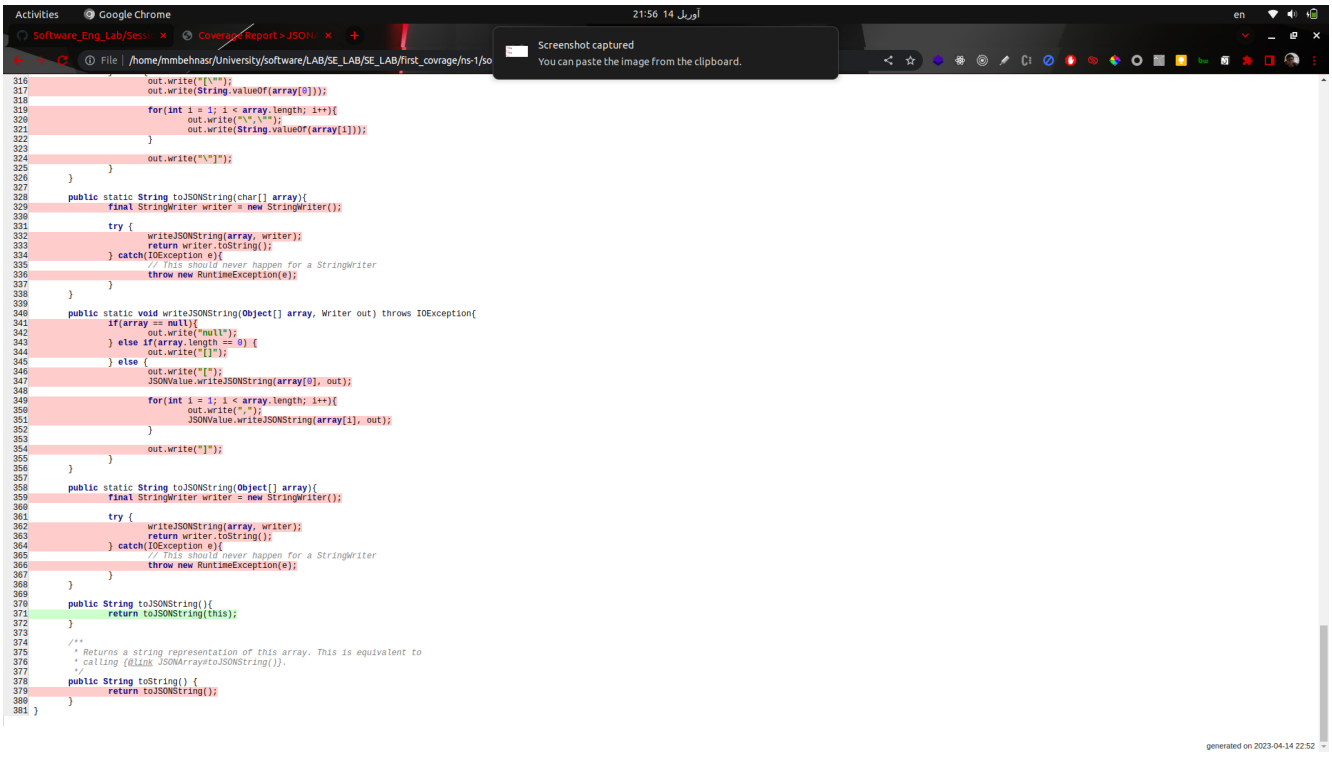

```
87      */
88     public static String toJSONString(Collection collection){
89         final StringWriter writer = new StringWriter();
90
91         try {
92             writeJSONString(collection, writer);
93             return writer.toString();
94         } catch(IOException e){
95             // This should never happen for a StringWriter
96             throw new RuntimeException(e);
97         }
98     }
99
100    public static void writeJSONString(byte[] array, Writer out) throws IOException{
101        if(array == null){
102            out.write("null");
103        } else if(array.length == 0) {
104            out.write("[]");
105        } else {
106            out.write("[");
107            out.write(String.valueOf(array[0]));
108
109            for(int i = 1; i < array.length; i++){
110                out.write(",");
111                out.write(String.valueOf(array[i]));
112            }
113
114            out.write("]");
115        }
116    }
117
118    public static String toJSONString(byte[] array){
119        final StringWriter writer = new StringWriter();
120
121        try {
122            writeJSONString(array, writer);
123            return writer.toString();
124        } catch(IOException e){
125            // This should never happen for a StringWriter
126            throw new RuntimeException(e);
127        }
128    }
129
130    public static void writeJSONString(short[] array, Writer out) throws IOException{
131        if(array == null){
132            out.write("null");
133        } else if(array.length == 0) {
134            out.write("[]");
135        } else {
136            out.write("[");
137            out.write(String.valueOf(array[0]));
138
139            for(int i = 1; i < array.length; i++){
140                out.write(",");
141                out.write(String.valueOf(array[i]));
142            }
143
144            out.write("]");
145        }
146    }
147
148    public static String toJSONString(short[] array){
149        final StringWriter writer = new StringWriter();
150
151        try {
152            writeJSONString(array, writer);
153            return writer.toString();
154        } catch(IOException e){
155            // This should never happen for a StringWriter
156            throw new RuntimeException(e);
```



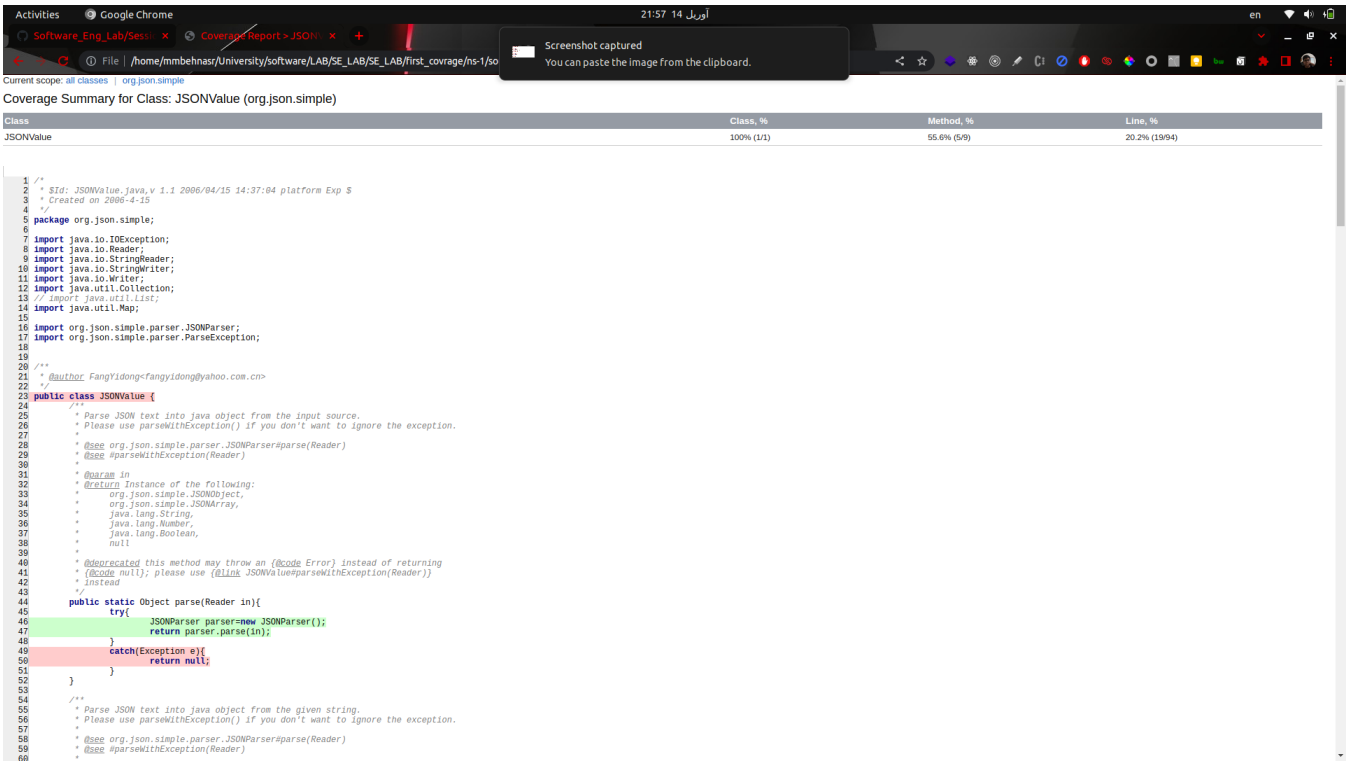

```
                // This should never happen for a StringWriter
                throw new RuntimeException(e);
            }
        }

        public static void writeJSONString(int[] array, Writer out) throws IOException{
            if(array == null){
                out.write("null");
            } else if(array.length == 0) {
                out.write("[]");
            } else {
                out.write("[");
                out.write(String.valueOf(array[0]));

                for(int i = 1; i < array.length; i++){
                    out.write(",");
                    out.write(String.valueOf(array[i]));
                }

                out.write("]");
            }
        }

        public static String toJSONString(int[] array){
            final StringWriter writer = new StringWriter();

            try {
                writeJSONString(array, writer);
                return writer.toString();
            } catch(IOException e){
                // This should never happen for a StringWriter
                throw new RuntimeException(e);
            }
        }

        public static void writeJSONString(long[] array, Writer out) throws IOException{
            if(array == null){
                out.write("null");
            } else if(array.length == 0) {
                out.write("[]");
            } else {
                out.write("[");
                out.write(String.valueOf(array[0]));

                for(int i = 1; i < array.length; i++){
                    out.write(",");
                    out.write(String.valueOf(array[i]));
                }

                out.write("]");
            }
        }

        public static String toJSONString(long[] array){
            final StringWriter writer = new StringWriter();

            try {
                writeJSONString(array, writer);
                return writer.toString();
            } catch(IOException e){
                // This should never happen for a StringWriter
                throw new RuntimeException(e);
            }
        }

        public static void writeJSONString(float[] array, Writer out) throws IOException{
            if(array == null){
                out.write("null");
            } else if(array.length == 0) {
                out.write("[]");
```

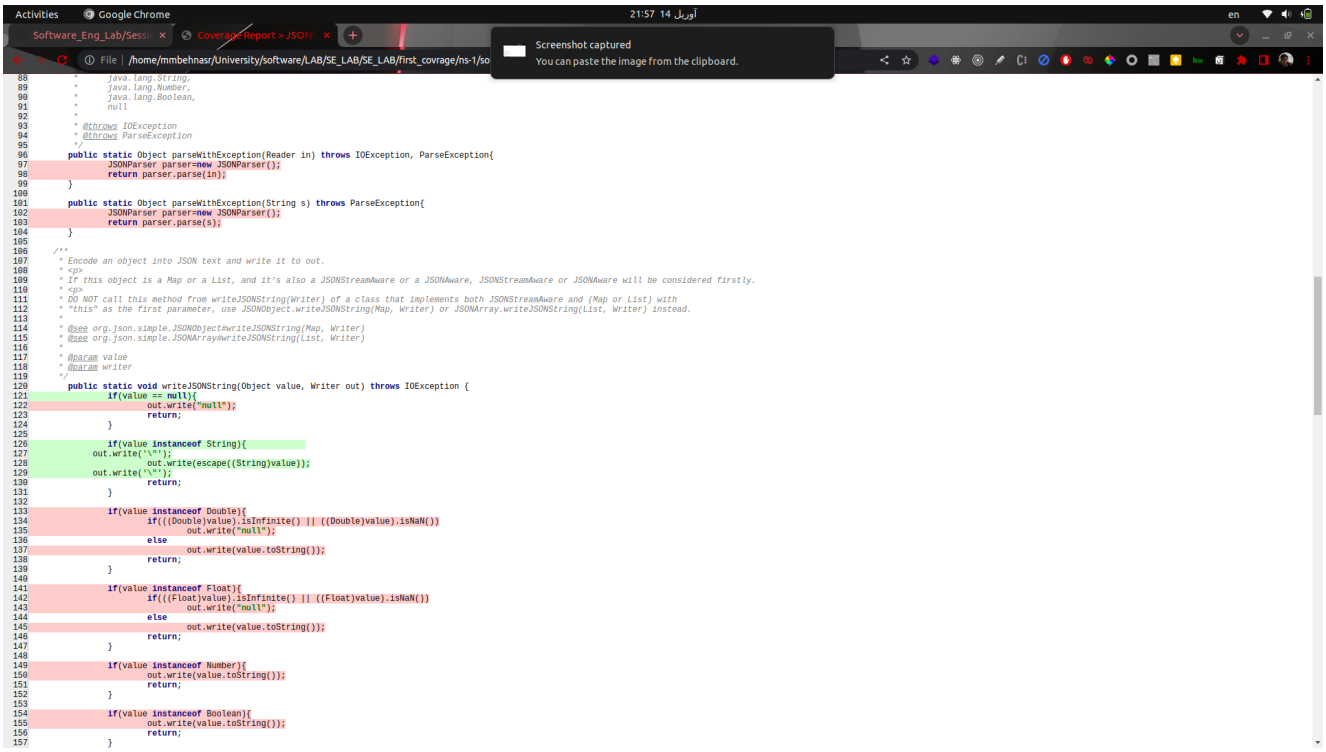



```
                }

                out.write("]");
            }
        }

        public static String toJSONString(double[] array){
            final StringWriter writer = new StringWriter();

            try {
                writeJSONString(array, writer);
                return writer.toString();
            } catch(IOException e){
                // This should never happen for a StringWriter
                throw new RuntimeException(e);
            }
        }

        public static void writeJSONString(boolean[] array, Writer out) throws IOException{
            if(array == null){
                out.write("null");
            } else if(array.length == 0) {
                out.write("[]");
            } else {
                out.write("[");
                out.write(String.valueOf(array[0]));

                for(int i = 1; i < array.length; i++){
                    out.write(",");
                    out.write(String.valueOf(array[i]));
                }

                out.write("]");
            }
        }

        public static String toJSONString(boolean[] array){
            final StringWriter writer = new StringWriter();

            try {
                writeJSONString(array, writer);
                return writer.toString();
            } catch(IOException e){
                // This should never happen for a StringWriter
                throw new RuntimeException(e);
            }
        }

        public static void writeJSONString(char[] array, Writer out) throws IOException{
            if(array == null){
                out.write("null");
            } else if(array.length == 0) {
                out.write("[]");
            } else {
                out.write("[\"");
                out.write(String.valueOf(array[0]));

                for(int i = 1; i < array.length; i++){
                    out.write("\",\"");
                    out.write(String.valueOf(array[i]));
                }

                out.write("\"]");
            }
        }

        public static String toJSONString(char[] array){
            final StringWriter writer = new StringWriter();

            try {
```



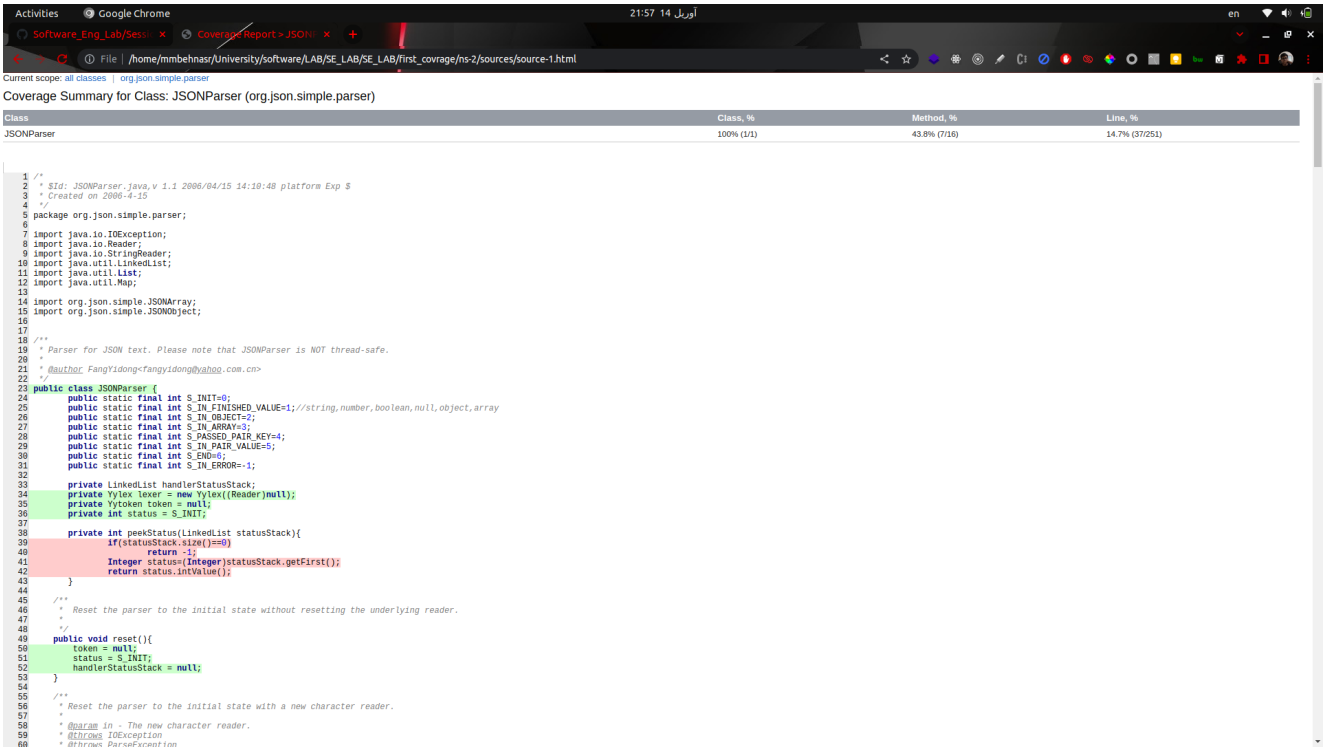
کلاس JSONValue:






```
88          *       java.lang.String,
89          *       java.lang.Number,
90          *       java.lang.Boolean,
91          *       null
92          *
93          * @throws IOException
94          * @throws ParseException
95          */
96         public static Object parseWithException(Reader in) throws IOException, ParseException{
97                 JSONParser parser=new JSONParser();
98                 return parser.parse(in);
99         }
100
101        public static Object parseWithException(String s) throws ParseException{
102                JSONParser parser=new JSONParser();
103                return parser.parse(s);
104        }
105
106     /**
107      * Encode an object into JSON text and write it to out.
108      * <p>
109      * If this object is a Map or a List, and it's also a JSONStreamAware or a JSONAware, JSONStreamAware or JSONAware will be considered firstly.
110      * <p>
111      * DO NOT call this method from writeJSONString(Writer) of a class that implements both JSONStreamAware and (Map or List) with
112      * "this" as the first parameter, use JSONObject.writeJSONString(Map, Writer) or JSONArray.writeJSONString(List, Writer) instead.
113      *
114      * @see org.json.simple.JSONObject#writeJSONString(Map, Writer)
115      * @see org.json.simple.JSONArray#writeJSONString(List, Writer)
116      *
117      * @param value
118      * @param writer
119      */
120        public static void writeJSONString(Object value, Writer out) throws IOException {
121                if(value == null){
122                        out.write("null");
123                        return;
124                }
125
126                if(value instanceof String){
127                out.write('\"');
128                        out.write(escape((String)value));
129                out.write('\"');
130                        return;
131                }
132
133                if(value instanceof Double){
134                        if(((Double)value).isInfinite() || ((Double)value).isNaN())
135                                out.write("null");
136                        else
137                                out.write(value.toString());
138                        return;
139                }
140
141                if(value instanceof Float){
142                        if(((Float)value).isInfinite() || ((Float)value).isNaN())
143                                out.write("null");
144                        else
145                                out.write(value.toString());
146                        return;
147                }
148
149                if(value instanceof Number){
150                        out.write(value.toString());
151                        return;
152                }
153
154                if(value instanceof Boolean){
155                        out.write(value.toString());
156                        return;
157                }
```

کلاس JSONParser:

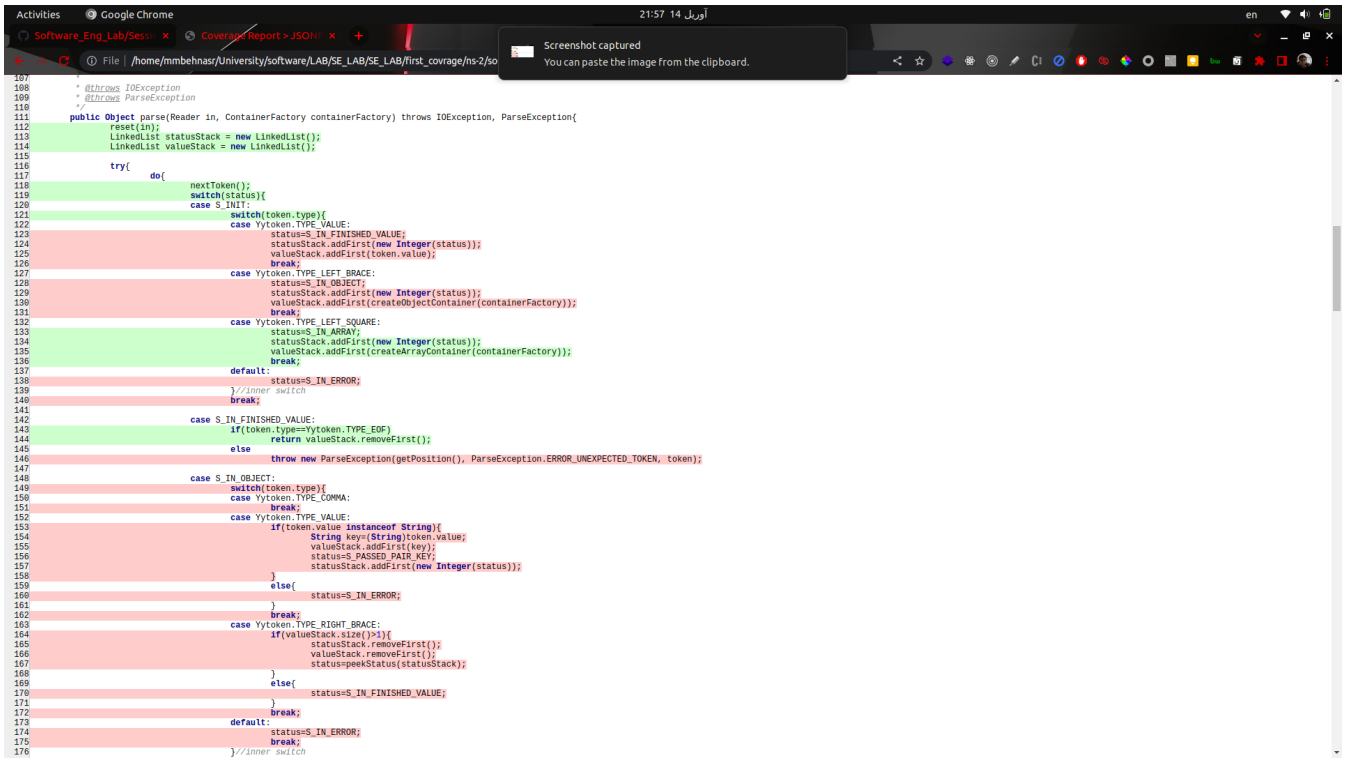


Current scope: all classes | org.json.simple.parser

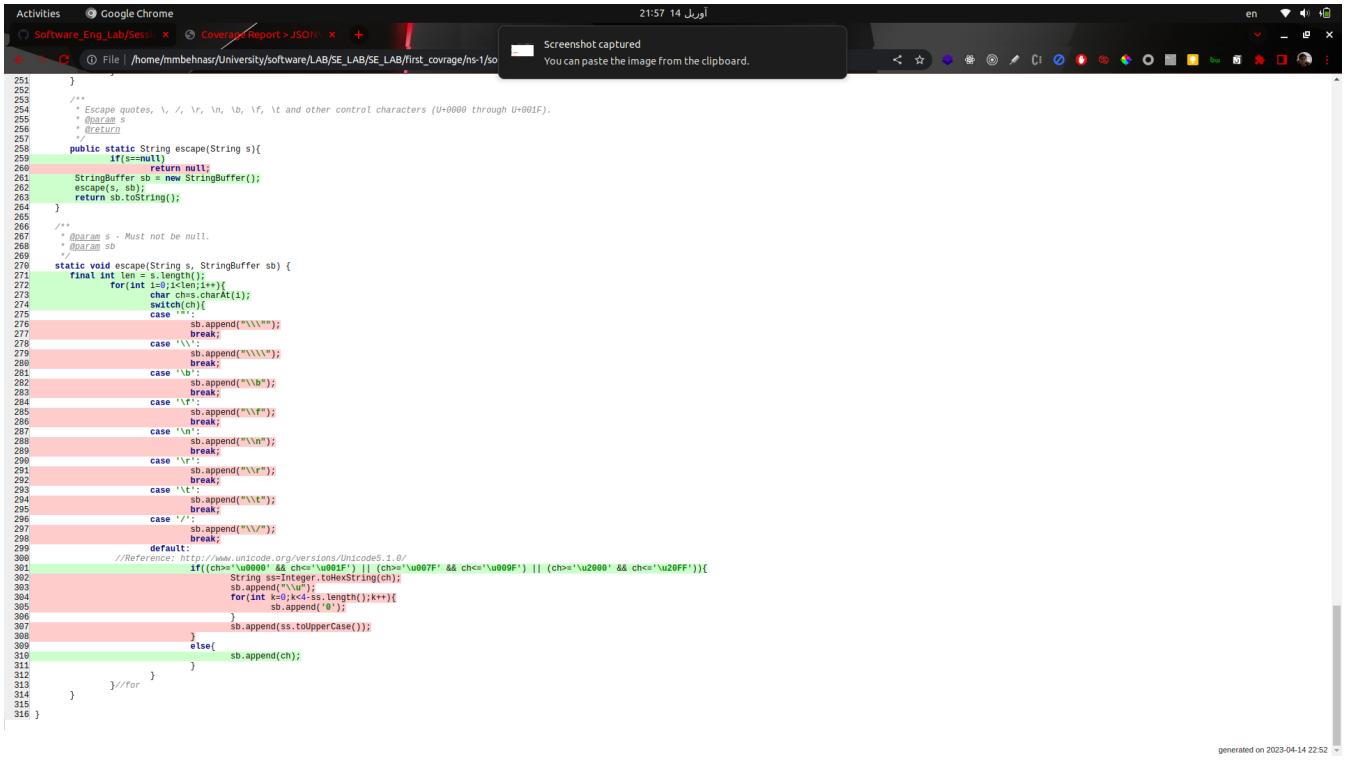
Coverage Summary for Class: JSONParser (org.json.simple.parser)

Class	Class, %	Method, %	Line, %
JSONParser	100% (1/1)	43.8% (7/16)	14.7% (37/251)

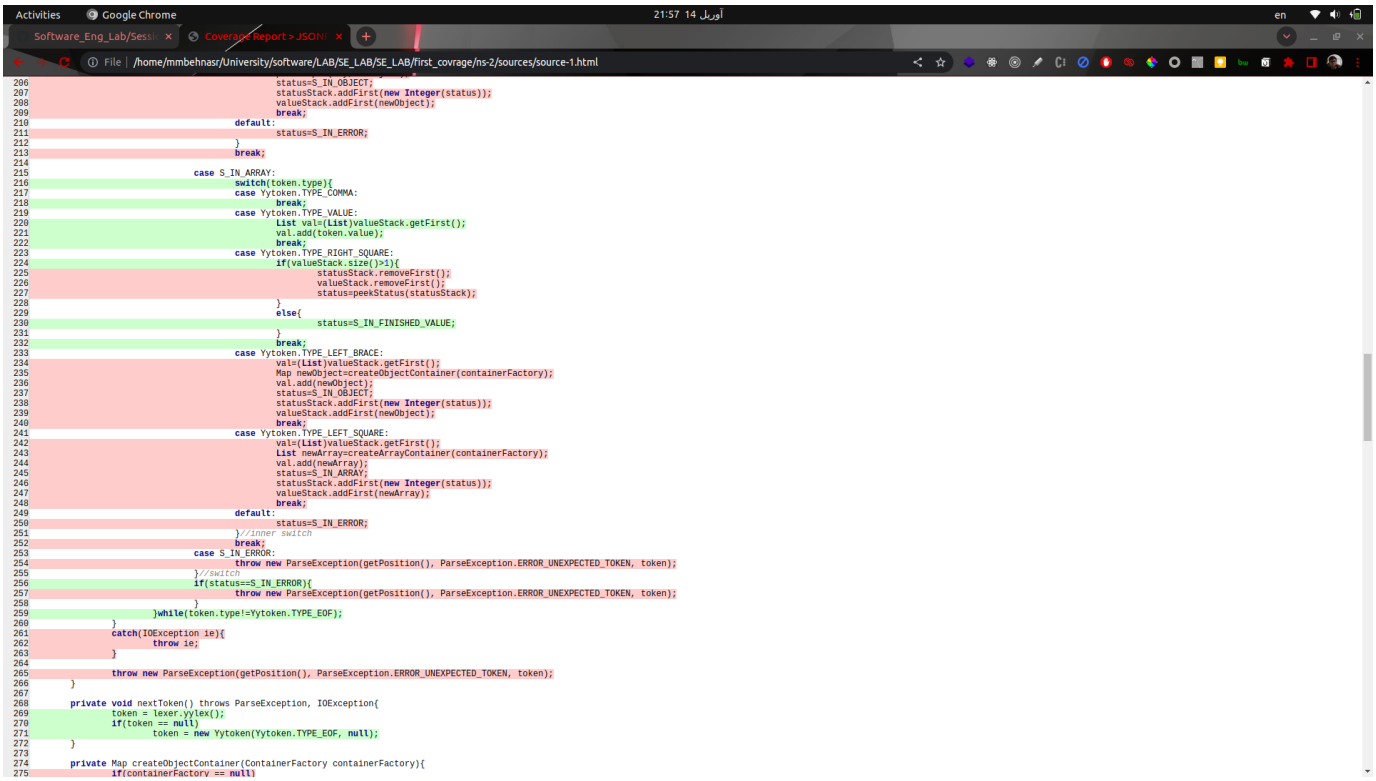
```
1  /*
2   * $Id: JSONParser.java,v 1.1 2006/04/15 14:10:48 platform Exp $
3   * Created on 2006-4-15
4   */
5  package org.json.simple.parser;
6
7  import java.io.IOException;
8  import java.io.Reader;
9  import java.io.StringReader;
10 import java.util.LinkedList;
11 import java.util.List;
12 import java.util.Map;
13
14 import org.json.simple.JSONArray;
15 import org.json.simple.JSONObject;
16
17
18 /**
19  * Parser for JSON text. Please note that JSONParser is NOT thread-safe.
20  *
21  * @author FangYidong<fangyidong@yahoo.com.cn>
22  */
23 public class JSONParser {
24         public static final int S_INIT=0;
25         public static final int S_IN_FINISHED_VALUE=1;//string,number,boolean,null,object,array
26         public static final int S_IN_OBJECT=2;
27         public static final int S_IN_ARRAY=3;
28         public static final int S_PASSED_PAIR_KEY=4;
29         public static final int S_IN_PAIR_VALUE=5;
30         public static final int S_END=6;
31         public static final int S_IN_ERROR=-1;
32
33         private LinkedList handlerStatusStack;
34         private Yylex lexer = new Yylex((Reader)null);
35         private Yytoken token = null;
36         private int status = S_INIT;
37
38         private int peekStatus(LinkedList statusStack){
39                 if(statusStack.size()==0)
40                         return -1;
41                 Integer status=(Integer)statusStack.getFirst();
42                 return status.intValue();
43         }
44
45     /**
46      *  Reset the parser to the initial state without resetting the underlying reader.
47      *
48      */
49     public void reset(){
50         token = null;
51         status = S_INIT;
52         handlerStatusStack = null;
53     }
54
55     /**
56      * Reset the parser to the initial state with a new character reader.
57      *
58      * @param in - The new character reader.
59      * @throws IOException
60      * @throws ParseException
```




```
107
108         * @throws IOException
109         * @throws ParseException
110         */
111        public Object parse(Reader in, ContainerFactory containerFactory) throws IOException, ParseException{
112                reset(in);
113                LinkedList statusStack = new LinkedList();
114                LinkedList valueStack = new LinkedList();
115
116                try{
117                        do{
118                                nextToken();
119                                switch(status){
120                                case S_INIT:
121                                        switch(token.type){
122                                        case Yytoken.TYPE_VALUE:
123                                                status=S_IN_FINISHED_VALUE;
124                                                statusStack.addFirst(new Integer(status));
125                                                valueStack.addFirst(token.value);
126                                                break;
127                                        case Yytoken.TYPE_LEFT_BRACE:
128                                                status=S_IN_OBJECT;
129                                                statusStack.addFirst(new Integer(status));
130                                                valueStack.addFirst(createObjectContainer(containerFactory));
131                                                break;
132                                        case Yytoken.TYPE_LEFT_SQUARE:
133                                                status=S_IN_ARRAY;
134                                                statusStack.addFirst(new Integer(status));
135                                                valueStack.addFirst(createArrayContainer(containerFactory));
136                                                break;
137                                        default:
138                                                status=S_IN_ERROR;
139                                        }//inner switch
140                                        break;
141
142                                case S_IN_FINISHED_VALUE:
143                                        if(token.type==Yytoken.TYPE_EOF)
144                                                return valueStack.removeFirst();
145                                        else
146                                                throw new ParseException(getPosition(), ParseException.ERROR_UNEXPECTED_TOKEN, token);
147
148                                case S_IN_OBJECT:
149                                        switch(token.type){
150                                        case Yytoken.TYPE_COMMA:
151                                                break;
152                                        case Yytoken.TYPE_VALUE:
153                                                if(token.value instanceof String){
154                                                        String key=(String)token.value;
155                                                        valueStack.addFirst(key);
156                                                        status=S_PASSED_PAIR_KEY;
157                                                        statusStack.addFirst(new Integer(status));
158                                                }
159                                                else{
160                                                        status=S_IN_ERROR;
161                                                }
162                                                break;
163                                        case Yytoken.TYPE_RIGHT_BRACE:
164                                                if(valueStack.size()>1){
165                                                        statusStack.removeFirst();
166                                                        valueStack.removeFirst();
167                                                        status=peekStatus(statusStack);
168                                                }
169                                                else{
170                                                        status=S_IN_FINISHED_VALUE;
171                                                }
172                                                break;
173                                        default:
174                                                status=S_IN_ERROR;
175                                                break;
176                                        }//inner switch
```




```
251        }
252
253        /**
254         * Escape quotes, \, /, \r, \n, \b, \f, \t and other control characters (U+0000 through U+001F).
255         * @param s
256         * @return
257         */
258        public static String escape(String s){
259                if(s==null)
260                        return null;
261                StringBuffer sb = new StringBuffer();
262                escape(s, sb);
263                return sb.toString();
264        }
265
266        /**
267         * @param s - Must not be null.
268         * @param sb
269         */
270        static void escape(String s, StringBuffer sb) {
271                final int len = s.length();
272                        for(int i=0;i<len;i++){
273                                char ch=s.charAt(i);
274                                switch(ch){
275                                case '"':
276                                        sb.append("\\\"");
277                                        break;
278                                case '\\':
279                                        sb.append("\\\\");
280                                        break;
281                                case '\b':
282                                        sb.append("\\b");
283                                        break;
284                                case '\f':
285                                        sb.append("\\f");
286                                        break;
287                                case '\n':
288                                        sb.append("\\n");
289                                        break;
290                                case '\r':
291                                        sb.append("\\r");
292                                        break;
293                                case '\t':
294                                        sb.append("\\t");
295                                        break;
296                                case '/':
297                                        sb.append("\\/");
298                                        break;
299                                default:
300                                //Reference: http://www.unicode.org/versions/Unicode5.1.0/
301                                        if((ch>='\u0000' && ch<='\u001F') || (ch>='\u007F' && ch<='\u009F') || (ch>='\u2000' && ch<='\u20FF')){
302                                                String ss=Integer.toHexString(ch);
303                                                sb.append("\\u");
304                                                for(int k=0;k<4-ss.length();k++){
305                                                        sb.append('0');
306                                                }
307                                                sb.append(ss.toUpperCase());
308                                        }
309                                        else{
310                                                sb.append(ch);
311                                        }
312                                }
313                        }//for
314        }
315
316 }
```



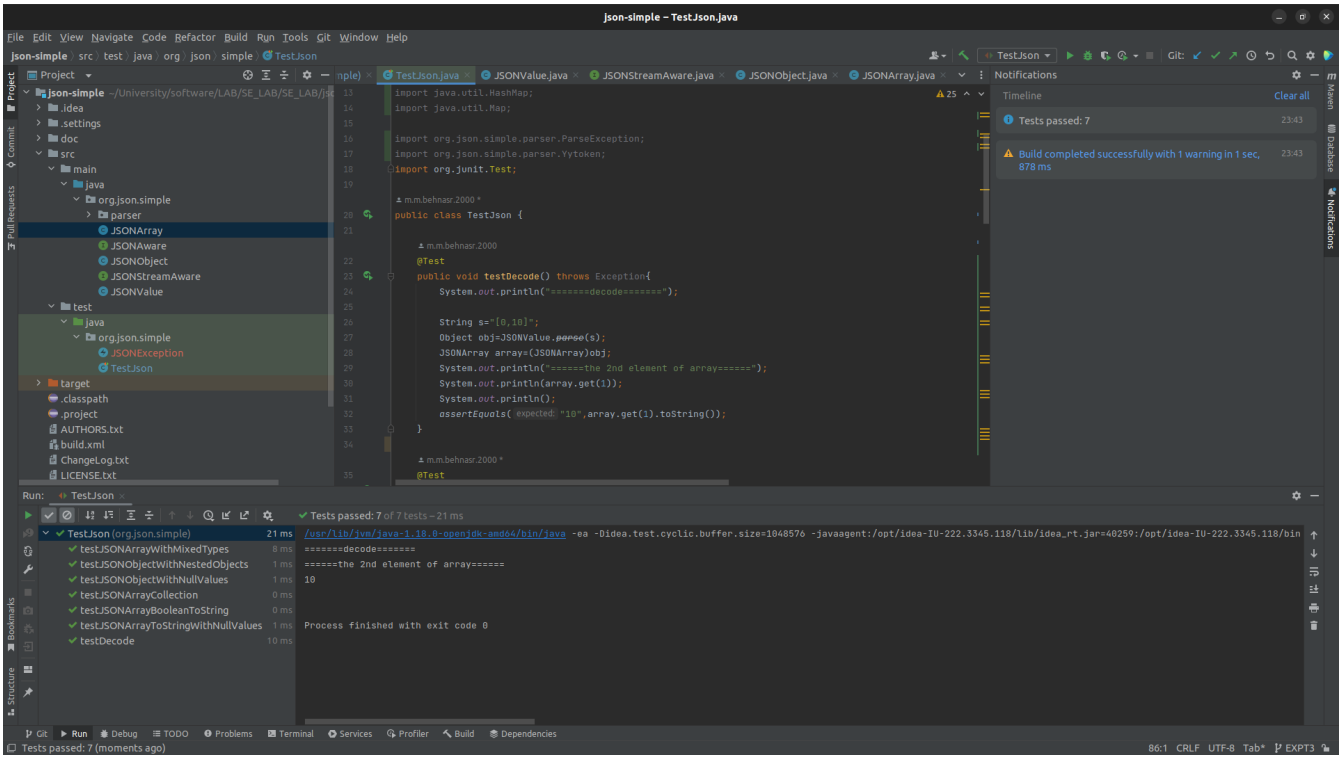

```
206                                     status=S_IN_OBJECT;
207                                     statusStack.addFirst(new Integer(status));
208                                     valueStack.addFirst(newObject);
209                                     break;
210                             default:
211                                     status=S_IN_ERROR;
212                             }
213                             break;
214
215                     case S_IN_ARRAY:
216                             switch(token.type){
217                             case Yytoken.TYPE_COMMA:
218                                     break;
219                             case Yytoken.TYPE_VALUE:
220                                     List vals=(List)valueStack.getFirst();
221                                     vals.add(token.value);
222                                     break;
223                             case Yytoken.TYPE_RIGHT_SQUARE:
224                                     if(valueStack.size()>1){
225                                             statusStack.removeFirst();
226                                             valueStack.removeFirst();
227                                             status=peekStatus(statusStack);
228                                     }
229                                     else{
230                                             status=S_IN_FINISHED_VALUE;
231                                     }
232                                     break;
233                             case Yytoken.TYPE_LEFT_BRACE:
234                                     vals=(List)valueStack.getFirst();
235                                     Map newObject=createObjectContainer(containerFactory);
236                                     vals.add(newObject);
237                                     status=S_IN_OBJECT;
238                                     statusStack.addFirst(new Integer(status));
239                                     valueStack.addFirst(newObject);
240                                     break;
241                             case Yytoken.TYPE_LEFT_SQUARE:
242                                     vals=(List)valueStack.getFirst();
243                                     List newArray=createArrayContainer(containerFactory);
244                                     vals.add(newArray);
245                                     status=S_IN_ARRAY;
246                                     statusStack.addFirst(new Integer(status));
247                                     valueStack.addFirst(newArray);
248                                     break;
249                             default:
250                                     status=S_IN_ERROR;
251                             }//inner switch
252                             break;
253                     case S_IN_ERROR:
254                             throw new ParseException(getPosition(), ParseException.ERROR_UNEXPECTED_TOKEN, token);
255                     }//switch
256                     if(status==S_IN_ERROR){
257                             throw new ParseException(getPosition(), ParseException.ERROR_UNEXPECTED_TOKEN, token);
258                     }
259             }while(token.type!=Yytoken.TYPE_EOF);
260         }
261         catch(IOException ie){
262                 throw ie;
263         }
264
265         throw new ParseException(getPosition(), ParseException.ERROR_UNEXPECTED_TOKEN, token);
266     }
267
268     private void nextToken() throws ParseException, IOException{
269             token = lexer.yylex();
270             if(token == null)
271                     token = new Yytoken(Yytoken.TYPE_EOF, null);
272     }
273
274     private Map createObjectContainer(ContainerFactory containerFactory){
275             if(containerFactory == null)
```

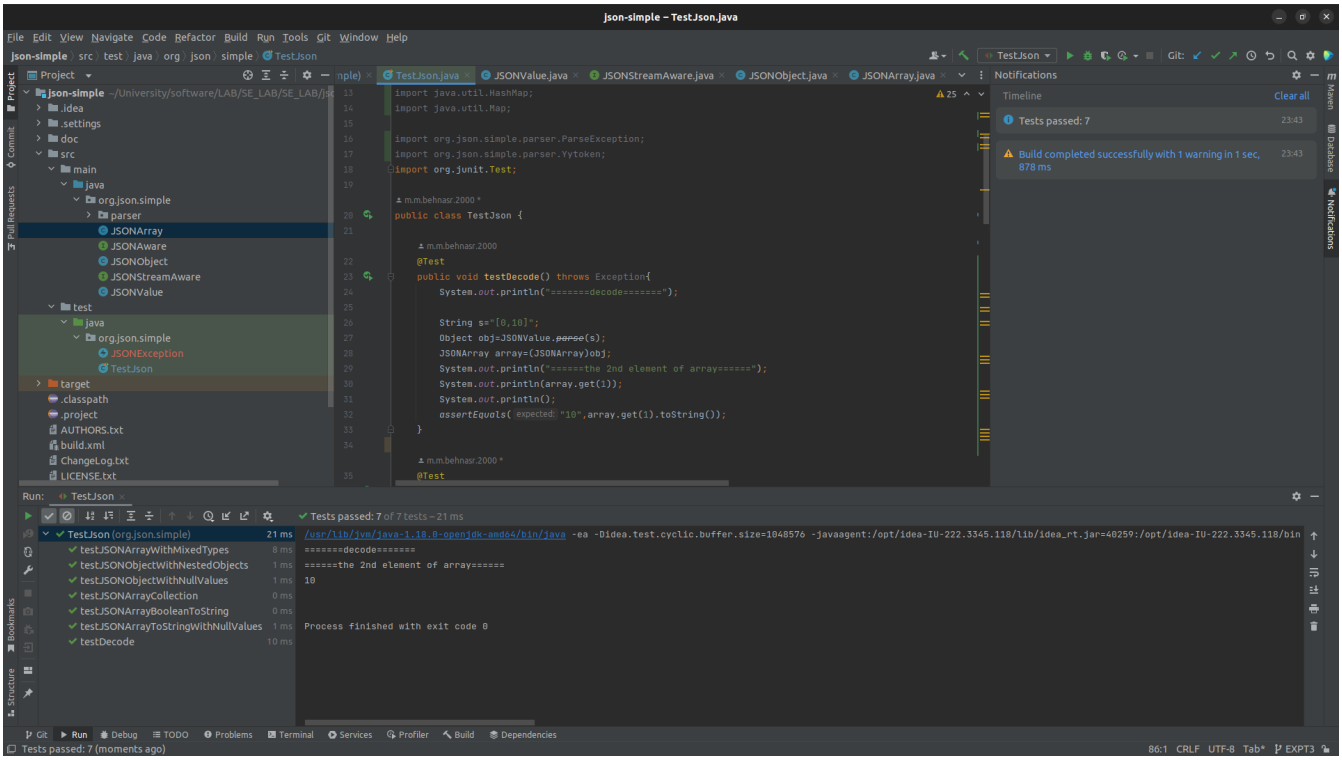
حال، با افزودن تست‌های جدید coverage هر کلاس را افزایش می‌دهیم:

در ادامه با اضافه کردن تست‌های جدید، coverage هر کلاس را رو به بالا بهبود می‌دهیم.

کلاس JSONArray:

با اضافه کردن تست زیر در کلاس TestJSON خواهیم داشت:

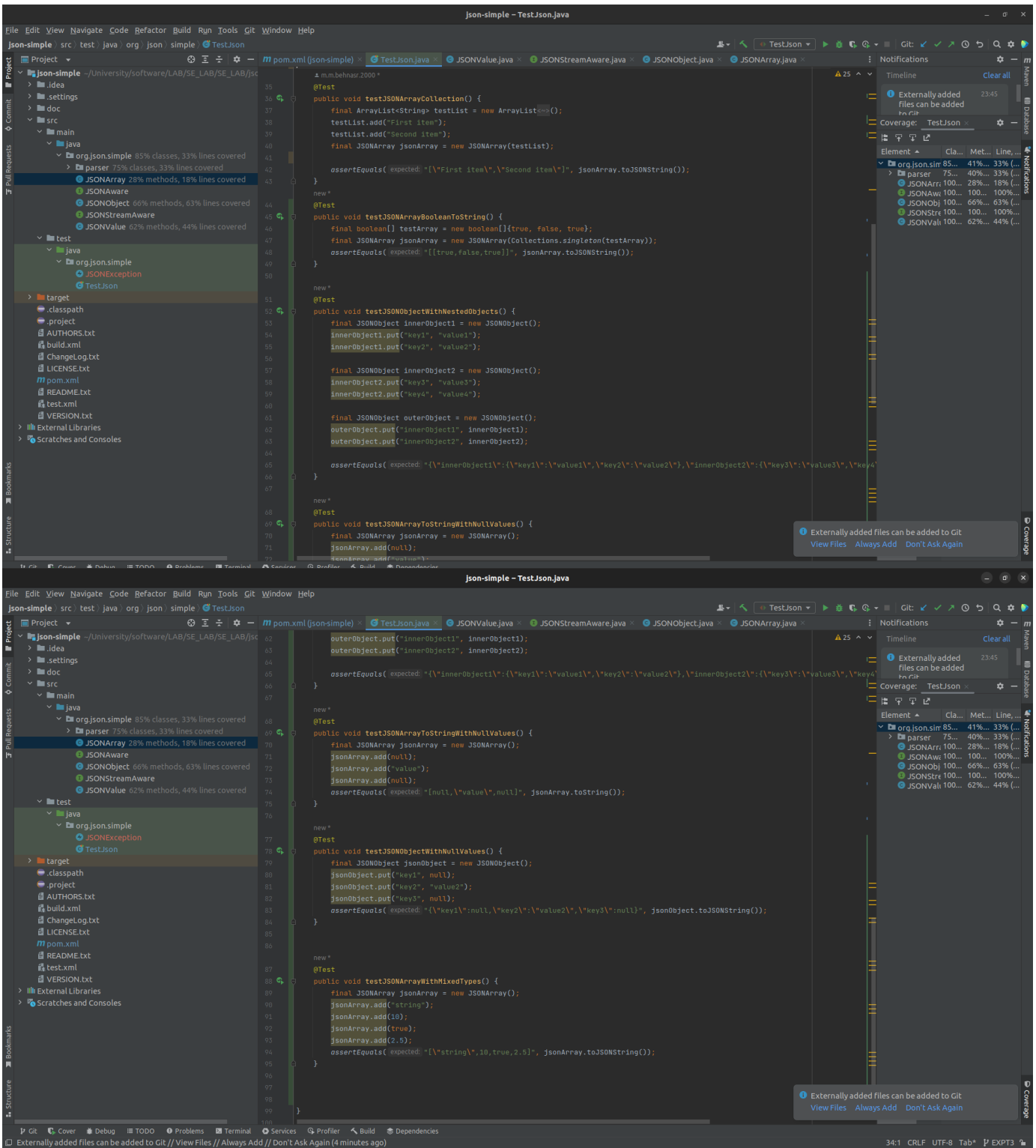




با افزودن این تست متد toJSONString که ورودی آن از نوع آرایه‌ای به‌صورت first item و second item می‌باشد فراخوانی می‌شود و این تست بررسی می‌کند که آیا این آرایه به درستی به فرمت JSON از نوع String تبدیل می‌شود یا خیر.

کلاس‌های JSONParser, JSONObject و Yylex:

با اضافه کردن تست زیر خواهیم داشت:

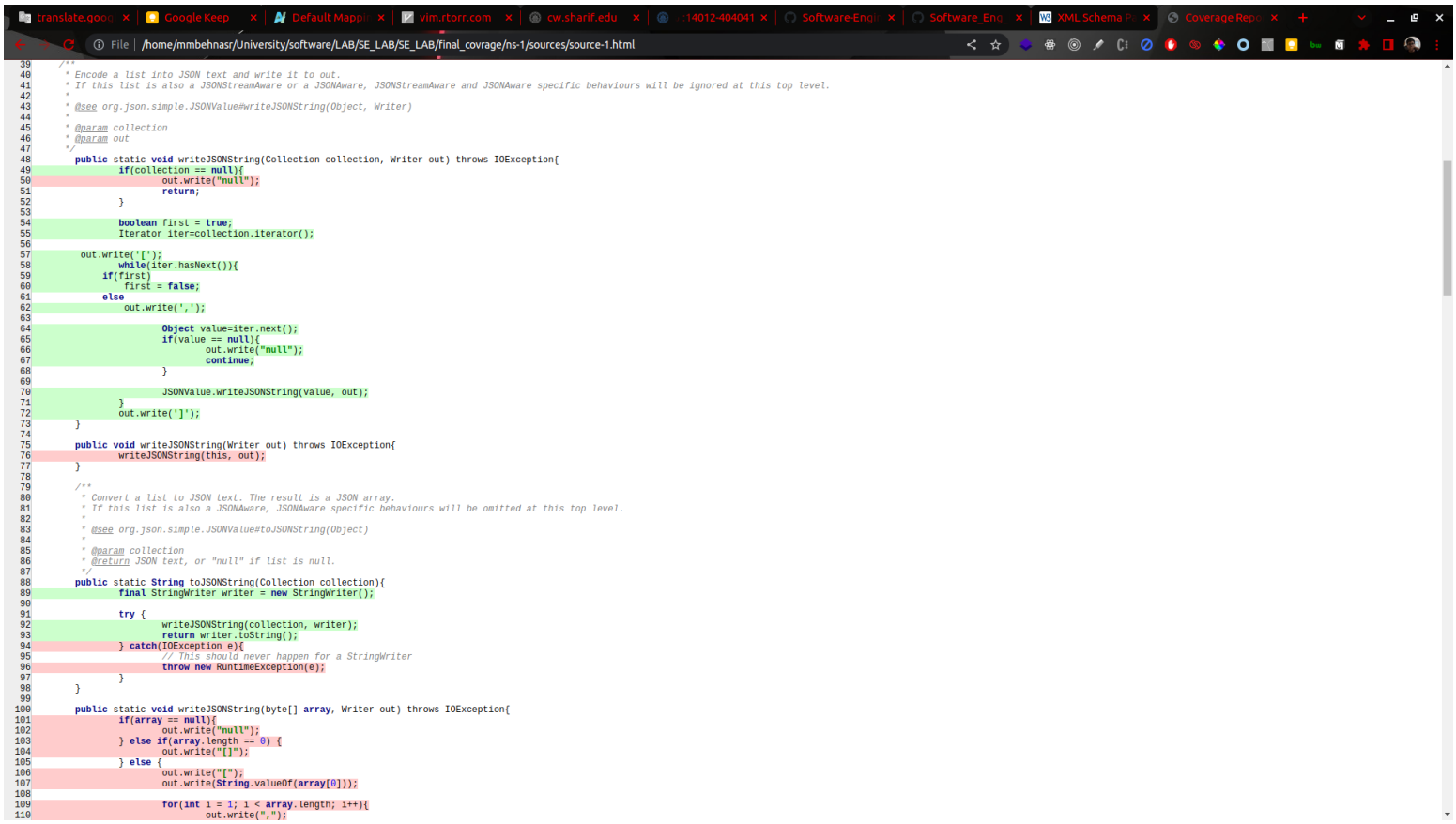


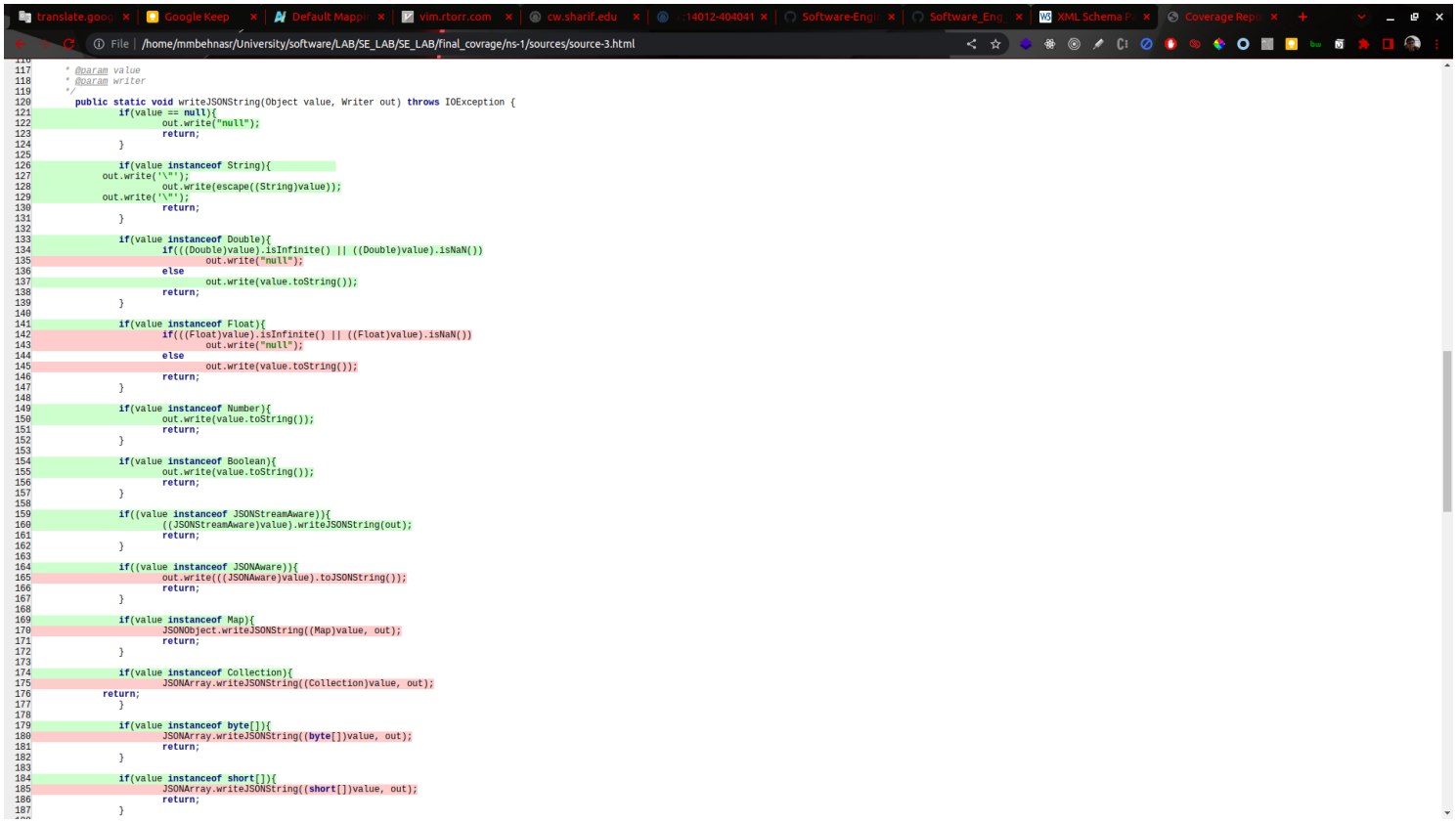
در این با فراخوانی متد parse از کلاس JSONValue، متد parse از کلاس JSONParser فراخوانی می‌شود و چون رشته‌ی ما شامل کاراکترهای } و { می‌باشد، در نتیجه توکن‌های جدیدی در متد yypushback ساخته می‌شود و کیس‌های جدیدی در متد parse در JSONParse اجرا می‌شوند و خطهایی اجرا می‌شود که در تست‌های قبلی اجرا نشده بود.

همچنین با توجه به اینکه رشته‌ای که در تست تعریف شده (s) یک آبجکت JSON می‌باشد، هنگام پارس‌کردن یک آبکت از نوع JSONParse ساخته می‌شود.

با اجرای این تست با توجه به اینکه مقادیر عددی یک آبجکت باید به رشته تبدیل شوند، خطوط جدیدی از متد writeJSONString و متدهای دیگر در کلاس JSONValue پیاده‌سازی می‌شوند که تصاویر همگی آنها و اثر coverage در کد کلاس‌ها و گزارش فایل آنها در ادامه آمده‌اند:

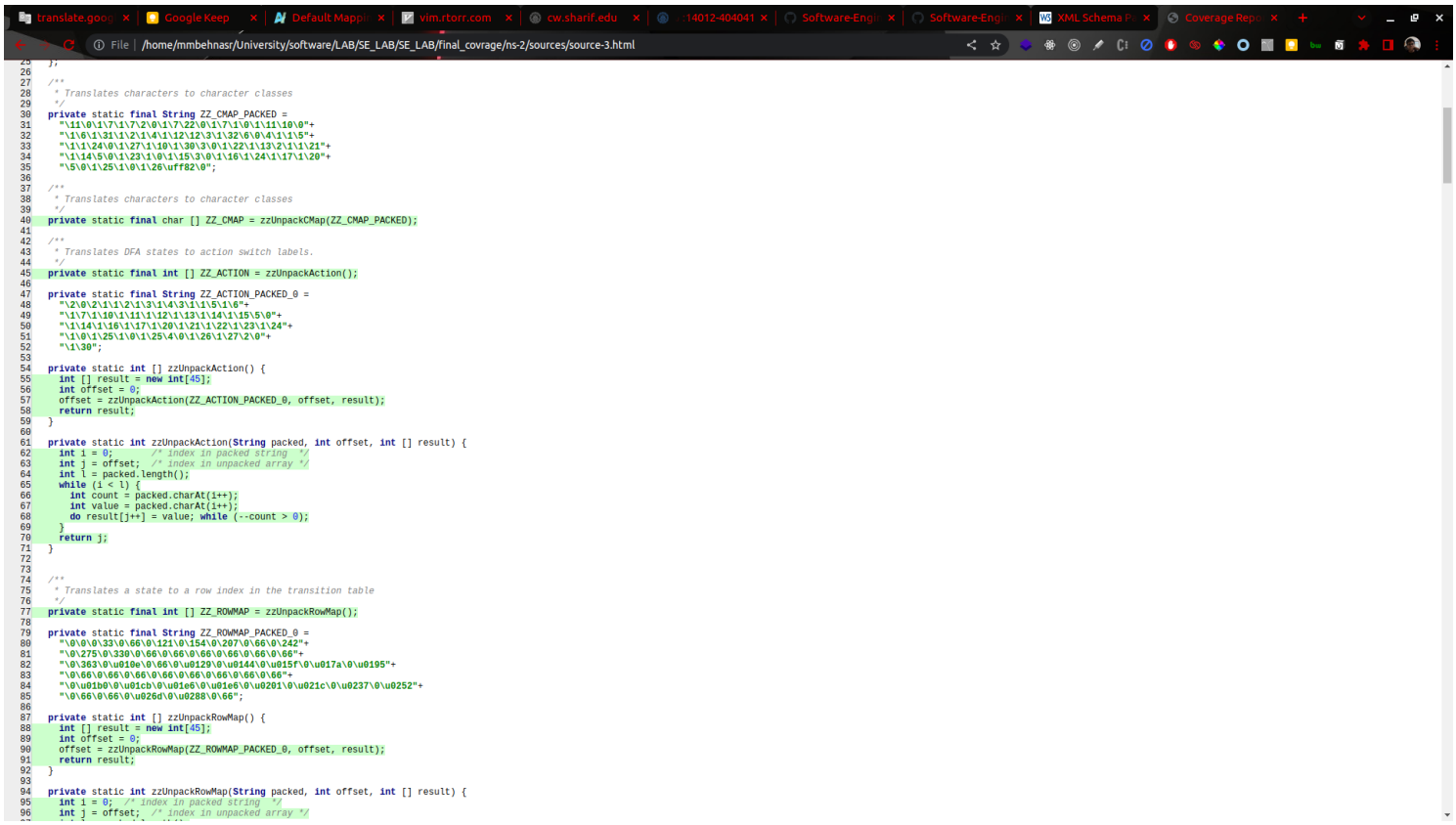
برای اولین تصاویر ns-1 را مشاهده می‌کنیم که در واقع coverage معلوم است:



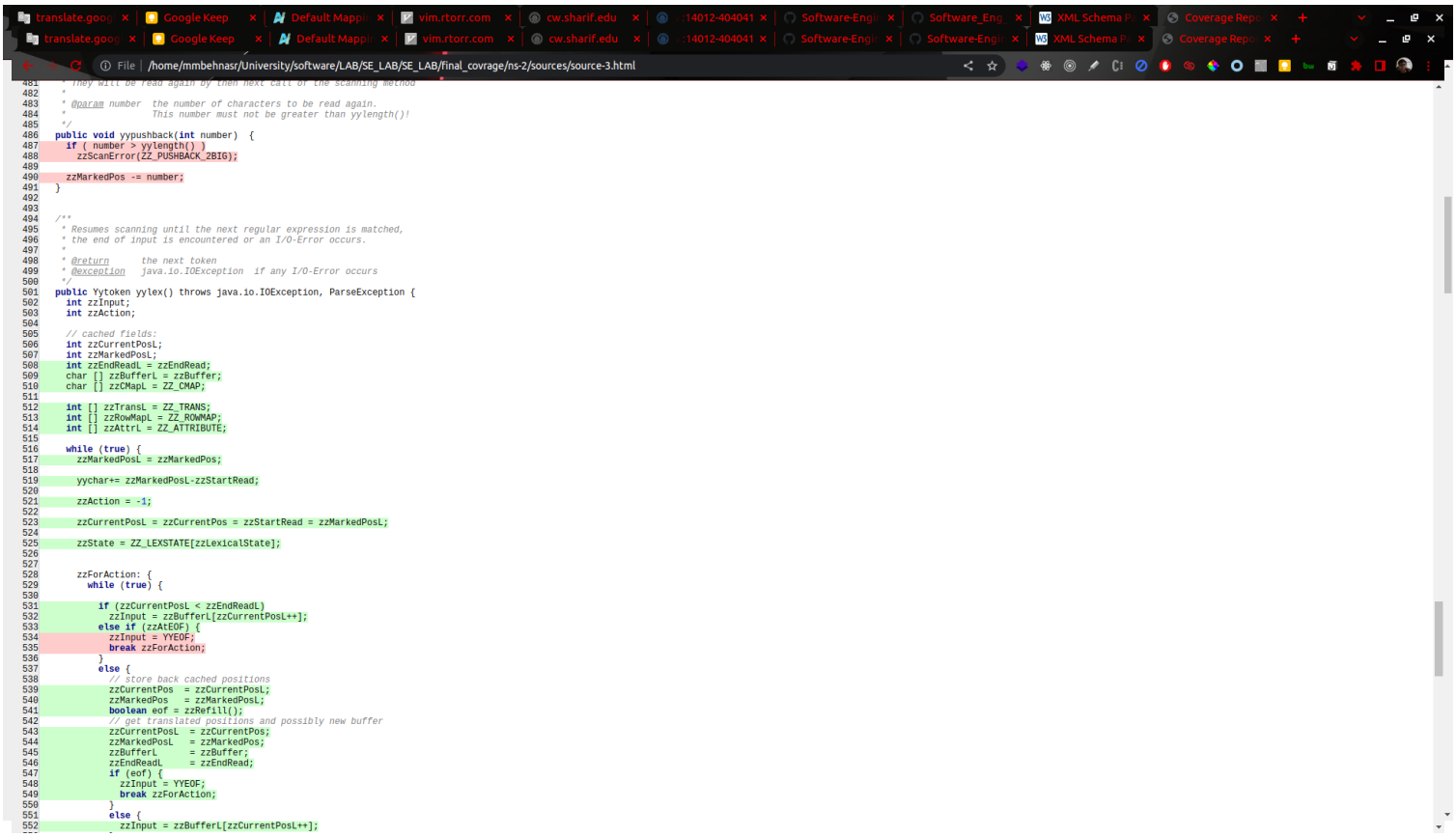



```
    * @param value
    * @param writer
    */
      public static void writeJSONString(Object value, Writer out) throws IOException {
            if(value == null){
                  out.write("null");
                  return;
            }

            if(value instanceof String){
          out.write('\"');
                  out.write(escape((String)value));
          out.write('\"');
                  return;
            }

            if(value instanceof Double){
                  if(((Double)value).isInfinite() || ((Double)value).isNaN())
                        out.write("null");
                  else
                        out.write(value.toString());
                  return;
            }

            if(value instanceof Float){
                  if(((Float)value).isInfinite() || ((Float)value).isNaN())
                        out.write("null");
                  else
                        out.write(value.toString());
                  return;
            }

            if(value instanceof Number){
                  out.write(value.toString());
                  return;
            }

            if(value instanceof Boolean){
                  out.write(value.toString());
                  return;
            }

            if((value instanceof JSONStreamAware)){
                  ((JSONStreamAware)value).writeJSONString(out);
                  return;
            }

            if((value instanceof JSONAware)){
                  out.write(((JSONAware)value).toJSONString());
                  return;
            }

            if(value instanceof Map){
                  JSONObject.writeJSONString((Map)value, out);
                  return;
            }

            if(value instanceof Collection){
                  JSONArray.writeJSONString((Collection)value, out);
          return;
            }

            if(value instanceof byte[]){
                  JSONArray.writeJSONString((byte[])value, out);
                  return;
            }

            if(value instanceof short[]){
                  JSONArray.writeJSONString((short[])value, out);
                  return;
            }
```

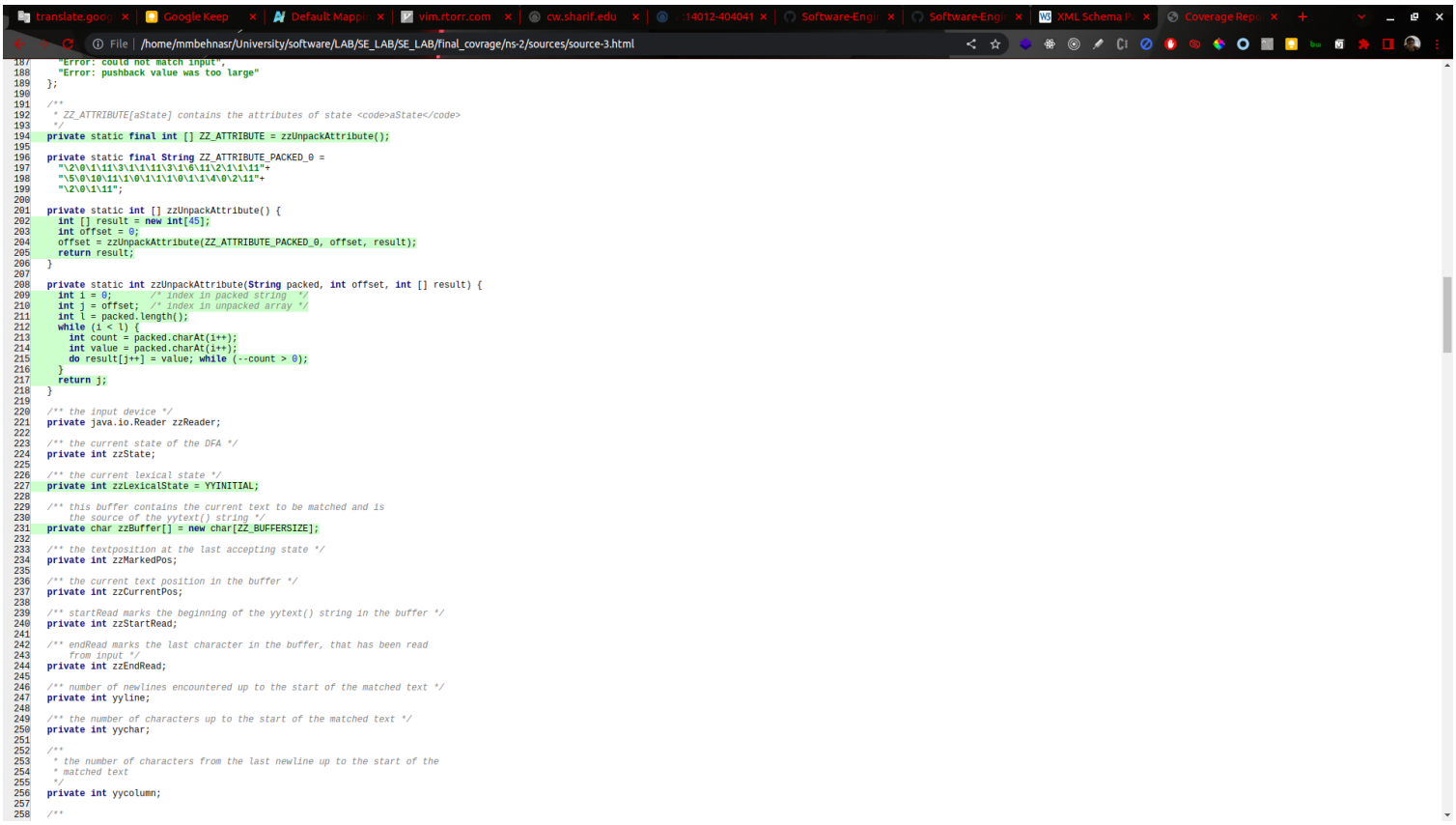
در تصاویر بعدی ns-2 را مشاهده می‌کنیم که در واقع coverage معلوم نیست:




```
};

/**
 * Translates characters to character classes
 */
private static final String ZZ_CMAP_PACKED =
  "\11\0\1\7\1\7\2\0\1\7\22\0\1\7\1\0\1\11\10\0"+
  "\1\6\1\31\1\2\1\4\1\12\3\1\32\6\0\4\1\1\5"+
  "\1\1\24\0\1\27\1\10\1\30\3\0\1\22\1\13\2\1\1\21"+
  "\1\14\5\0\1\23\1\0\1\15\3\0\1\16\1\24\1\17\1\20"+
  "\5\0\1\25\1\0\1\26\uff82\0";

/**
 * Translates characters to character classes
 */
private static final char [] ZZ_CMAP = zzUnpackCMap(ZZ_CMAP_PACKED);

/**
 * Translates DFA states to action switch labels.
 */
private static final int [] ZZ_ACTION = zzUnpackAction();

private static final String ZZ_ACTION_PACKED_0 =
  "\2\0\2\1\1\2\1\3\1\4\3\1\1\5\1\6"+
  "\1\7\1\10\1\11\1\12\1\13\1\14\1\15\5\0"+
  "\1\14\1\16\1\17\1\20\1\21\1\22\1\23\1\24"+
  "\1\0\1\25\1\0\1\25\4\0\1\26\1\27\2\0"+
  "\1\30";

private static int [] zzUnpackAction() {
  int [] result = new int[45];
  int offset = 0;
  offset = zzUnpackAction(ZZ_ACTION_PACKED_0, offset, result);
  return result;
}

private static int zzUnpackAction(String packed, int offset, int [] result) {
  int i = 0;       /* index in packed string  */
  int j = offset;  /* index in unpacked array */
  int l = packed.length();
  while (i < l) {
    int count = packed.charAt(i++);
    int value = packed.charAt(i++);
    do result[j++] = value; while (--count > 0);
  }
  return j;
}


/**
 * Translates a state to a row index in the transition table
 */
private static final int [] ZZ_ROWMAP = zzUnpackRowMap();

private static final String ZZ_ROWMAP_PACKED_0 =
  "\0\0\0\33\0\66\0\121\0\154\0\207\0\66\0\242"+
  "\0\275\0\330\0\66\0\66\0\66\0\66\0\66\0\66"+
  "\0\363\0\u010e\0\66\0\u0129\0\u0144\0\u015f\0\u017a\0\u0195"+
  "\0\66\0\66\0\66\0\66\0\66\0\66\0\66\0\66"+
  "\0\u01b0\0\u01cb\0\u01e6\0\u01e6\0\u0201\0\u021c\0\u0237\0\u0252"+
  "\0\66\0\66\0\u026d\0\u0288\0\66";

private static int [] zzUnpackRowMap() {
  int [] result = new int[45];
  int offset = 0;
  offset = zzUnpackRowMap(ZZ_ROWMAP_PACKED_0, offset, result);
  return result;
}

private static int zzUnpackRowMap(String packed, int offset, int [] result) {
  int i = 0;  /* index in packed string  */
  int j = offset;  /* index in unpacked array */
```



```
 * they will be read again by then next call of the scanning method
 *
 * @param number  the number of characters to be read again.
 *                This number must not be greater than yylength()!
 */
public void yypushback(int number)  {
  if ( number > yylength() )
    zzScanError(ZZ_PUSHBACK_2BIG);

  zzMarkedPos -= number;
}


/**
 * Resumes scanning until the next regular expression is matched,
 * the end of input is encountered or an I/O-Error occurs.
 *
 * @return      the next token
 * @exception   java.io.IOException  if any I/O-Error occurs
 */
public Yytoken yylex() throws java.io.IOException, ParseException {
  int zzInput;
  int zzAction;

  // cached fields:
  int zzCurrentPosL;
  int zzMarkedPosL;
  int zzEndReadL = zzEndRead;
  char [] zzBufferL = zzBuffer;
  char [] zzCMapL = ZZ_CMAP;

  int [] zzTransL = ZZ_TRANS;
  int [] zzRowMapL = ZZ_ROWMAP;
  int [] zzAttrL = ZZ_ATTRIBUTE;

  while (true) {
    zzMarkedPosL = zzMarkedPos;

    yychar+= zzMarkedPosL-zzStartRead;

    zzAction = -1;

    zzCurrentPosL = zzCurrentPos = zzStartRead = zzMarkedPosL;

    zzState = ZZ_LEXSTATE[zzLexicalState];


    zzForAction: {
      while (true) {

        if (zzCurrentPosL < zzEndReadL)
          zzInput = zzBufferL[zzCurrentPosL++];
        else if (zzAtEOF) {
          zzInput = YYEOF;
          break zzForAction;
        }
        else {
          // store back cached positions
          zzCurrentPos  = zzCurrentPosL;
          zzMarkedPos   = zzMarkedPosL;
          boolean eof = zzRefill();
          // get translated positions and possibly new buffer
          zzCurrentPosL  = zzCurrentPos;
          zzMarkedPosL   = zzMarkedPos;
          zzBufferL      = zzBuffer;
          zzEndReadL     = zzEndRead;
          if (eof) {
            zzInput = YYEOF;
            break zzForAction;
          }
          else {
            zzInput = zzBufferL[zzCurrentPosL++];
```



```
  "Error: could not match input",
  "Error: pushback value was too large"
};

/**
 * ZZ_ATTRIBUTE[aState] contains the attributes of state <code>aState</code>
 */
private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();

private static final String ZZ_ATTRIBUTE_PACKED_0 =
  "\2\0\1\11\3\1\1\11\3\1\6\11\2\1\1\11"+
  "\5\0\10\11\1\0\1\1\1\0\1\1\4\0\2\11"+
  "\2\0\1\11";

private static int [] zzUnpackAttribute() {
  int [] result = new int[45];
  int offset = 0;
  offset = zzUnpackAttribute(ZZ_ATTRIBUTE_PACKED_0, offset, result);
  return result;
}

private static int zzUnpackAttribute(String packed, int offset, int [] result) {
  int i = 0;       /* index in packed string  */
  int j = offset;  /* index in unpacked array */
  int l = packed.length();
  while (i < l) {
    int count = packed.charAt(i++);
    int value = packed.charAt(i++);
    do result[j++] = value; while (--count > 0);
  }
  return j;
}

/** the input device */
private java.io.Reader zzReader;

/** the current state of the DFA */
private int zzState;

/** the current lexical state */
private int zzLexicalState = YYINITIAL;

/** this buffer contains the current text to be matched and is
    the source of the yytext() string */
private char zzBuffer[] = new char[ZZ_BUFFERSIZE];

/** the textposition at the last accepting state */
private int zzMarkedPos;

/** the current text position in the buffer */
private int zzCurrentPos;

/** startRead marks the beginning of the yytext() string in the buffer */
private int zzStartRead;

/** endRead marks the last character in the buffer, that has been read
    from input */
private int zzEndRead;

/** number of newlines encountered up to the start of the matched text */
private int yyline;

/** the number of characters up to the start of the matched text */
private int yychar;

/**
 * the number of characters from the last newline up to the start of the
 * matched text
 */
private int yycolumn;

/**
```

درصدهای coverage نهایی نیز در تصویر زیر آمده که نشان می‌دهد با اضافه کردن تست‌های نوشته شده coverage بهبود یافته: