Software Testing and Maintenance

Project Report

Harris Nghiem – 1000572896

Damian Jimenez – 1000584863

May 5th, 2017

Table of Contents

CONTENTS

Project Summary, Discussion, and Lessons Learned

OVERVIEW

Summary:

This Printtokens2 project is designed to test the student's knowledge in the testing framework JUnit 4 and how to achieve full code coverage, which includes node and path coverage. The given code is approximately 200 lines without comment code and contains a total of 15 bugs. These bugs are to be found and fixed in order, and fully documented. The project also calls for ~90% code coverage, and full node and path coverage. This will be documented via JACOCO and its report. Lastly, a control graph for each method is to be provided.

Discussion:

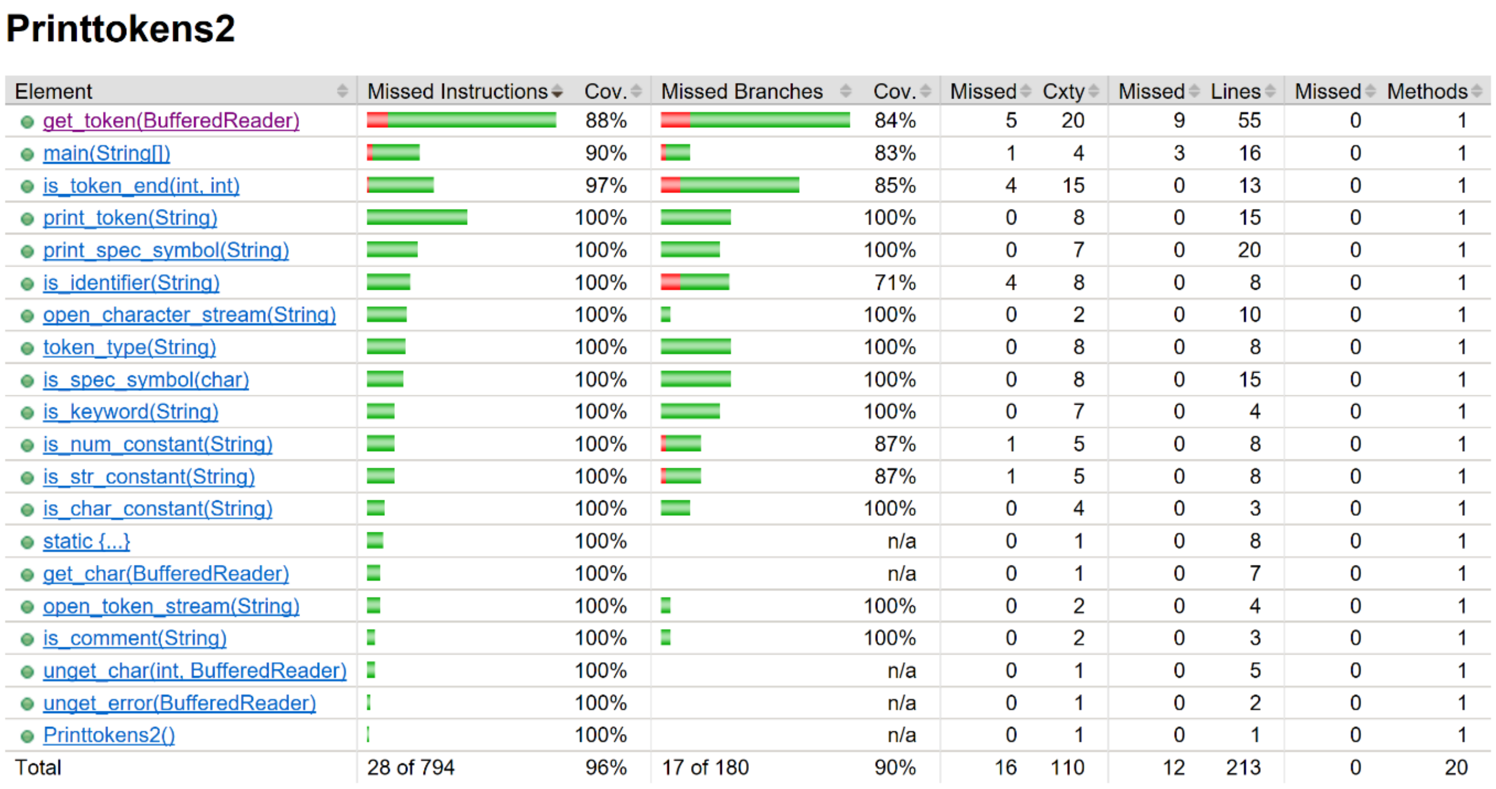
We decided to use IntelliJ's Idea IDE along and create a Maven project that allowed us to easily and quickly import any dependencies that were required to get our test cases successfully running. Achieving 90%+ code coverage was relatively easy, but finding the bugs turned out to be a bit more difficult than expected. The source code was formatted rather sloppily and it was unclear as to what it was supposed to be doing exactly. From looking at the code it appeared to be checking some for some form of assembly syntax. We worked on our project under this assumption, so some of our bugs considered so with that in mind. Path coverage was a little more tedious to achieve as it required implementing specific tests to cover paths that were unlikely to be used in regular use of the code. Some branches were seemingly impossible to cover without modifying the source code. For example, some branches were only accessible if an exception was thrown, but to throw that exception it sometimes required accessing method variables which I was unsure about how to go about doing so. We could have modified the code to make those variables class variables, but that felt like it would make the code unnecessarily more complex than it already was just for the sake of executing a couple of tests. In the end, we could achieve branch coverage of exactly 90%, which we were satisfied with given the circumstances stated above. We found 12 bugs total operating under the assumption that this code was written to check assembly syntax. One way to improve our results would be to go ahead and re-structure the code to make it more accessible to tests but doing so in a more thorough manner than just adding class variables. Also, figuring out exactly what the code is trying to accomplish would help in possibly identifying more bugs as well.

Lessons Learned:

Determining bugs and their respective fixes are much easier after full coverage. Determining unreachable code is also easier after running test cases, and running a coverage report immediately after. Determining bugs may be difficult before reading, understanding, and analyzing the code, thus a full comprehension of the methods and the flow of the program is necessary. Utilizing IntelliJ’s ability to run individual test methods makes the debugging of the respective method much easier, giving the method specific inputs and debugging until the correct output is given. Again, this requires a thorough understanding of each and every method.

Code Coverage Report

JACOCO OUTPUT



Printtokens2

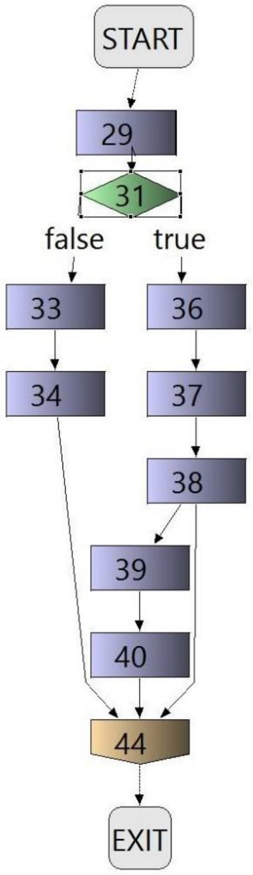
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
get_token(BufferedReader)		88%		84%	5	20	9	55	0	1
main(String[])		90%		83%	1	4	3	16	0	1
is_token_end(int, int)		97%		85%	4	15	0	13	0	1
print_token(String)		100%		100%	0	8	0	15	0	1
print_spec_symbol(String)		100%		100%	0	7	0	20	0	1
is_identifier(String)		100%		71%	4	8	0	8	0	1
open_character_stream(String)		100%		100%	0	2	0	10	0	1
token_type(String)		100%		100%	0	8	0	8	0	1
is_spec_symbol(char)		100%		100%	0	8	0	15	0	1
is_keyword(String)		100%		100%	0	7	0	4	0	1
is_num_constant(String)		100%		87%	1	5	0	8	0	1
is_str_constant(String)		100%		87%	1	5	0	8	0	1
is_char_constant(String)		100%		100%	0	4	0	3	0	1
static {...}		100%		n/a	0	1	0	8	0	1
get_char(BufferedReader)		100%		n/a	0	1	0	7	0	1
open_token_stream(String)		100%		100%	0	2	0	4	0	1
is_comment(String)		100%		100%	0	2	0	3	0	1
unget_char(int, BufferedReader)		100%		n/a	0	1	0	5	0	1
unget_error(BufferedReader)		100%		n/a	0	1	0	2	0	1
Printtokens2()		100%		n/a	0	1	0	1	0	1
Total	28 of 794	96%	17 of 180	90%	16	110	12	213	0	20

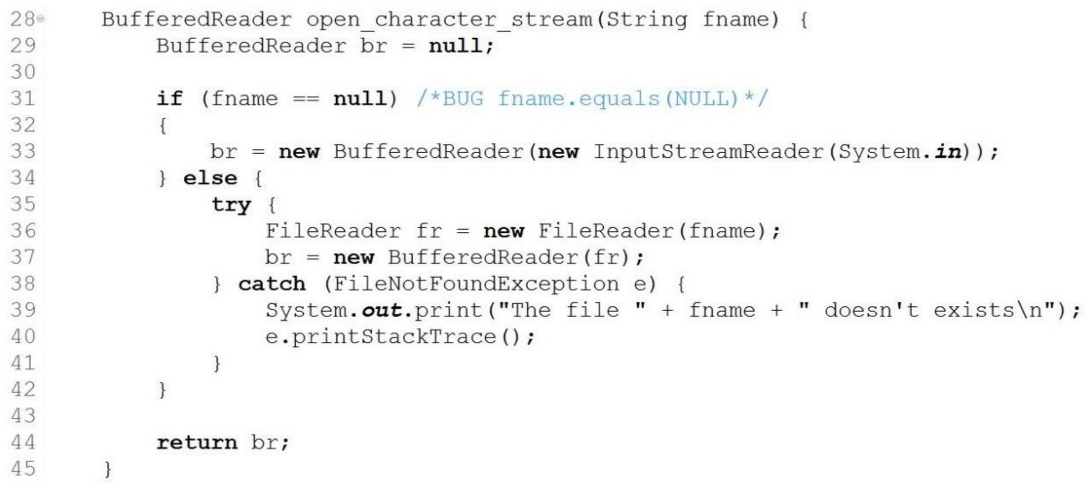
Testing Environment Variables

```
@Rule
public final ExpectedException exception = ExpectedException.none();
@Rule
public final ExpectedSystemExit exit = ExpectedSystemExit.none();
private final static String DEFAULT_TEST_FILE = "testFile0.txt";
private final static List<String> TEST_FILE_LIST = Arrays.asList("testFile0.txt", "testFile1.txt",
        "testFile2.txt", "testFile3.txt", "testFile4.txt", "testFile5.txt", "testFile6.txt",
"testFile7.txt",
        "testFile8.txt", "testFile9.txt", "testFile10.txt");
private final static List<String> TEST_CHAR_LIST = Arrays.asList("a", "b", "c", "1", "2", "A", "B", "C",
        "3", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "-", "_", "=", "+", "`", "~", "[", "{",
"]",
        "}", ";", ":", "'", ",", "<", ".", ">", "/", "?", "\\", "|", "\t", "\r", "\n", "\0", " ", "\b",
"\f",
        "and", "or", "if", "xor","lambda","=>","#a");
private Printtokens2 pt2;

@Before
public void setup() {
    pt2 = new Printtokens2();
}
```

METHOD 1 – OPEN_CHARACTER_STREAM





```
28    BufferedReader open_character_stream(String fname) {
29        BufferedReader br = null;
30
31        if (fname == null) /*BUG fname.equals(NULL)*/
32        {
33            br = new BufferedReader(new InputStreamReader(System.in));
34        } else {
35            try {
36                FileReader fr = new FileReader(fname);
37                br = new BufferedReader(fr);
38            } catch (FileNotFoundException e) {
39                System.out.print("The file " + fname + " doesn't exists\n");
40                e.printStackTrace();
41            }
42        }
43
44        return br;
45    }
```

Bugs and Test Cases

Bug(s)	Error(s)
`fname.equals(bool)`	`Uncaught NullPointerException when fname is null`

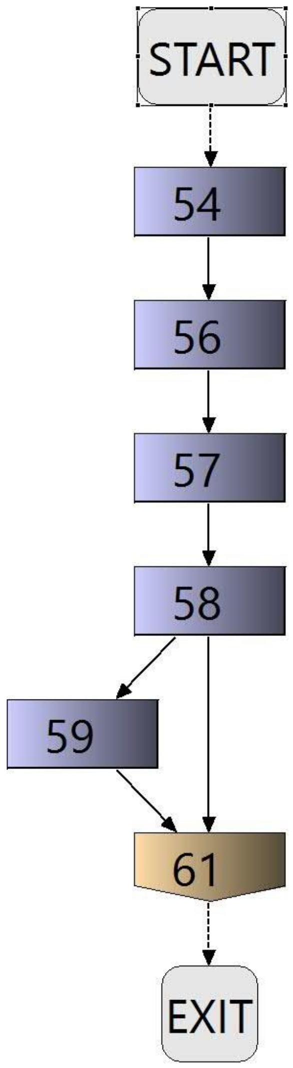
Test Methods

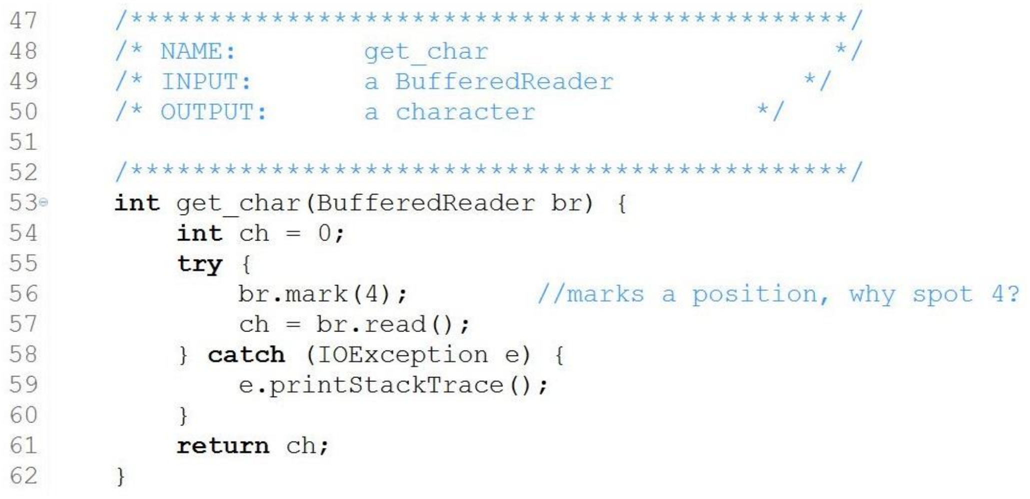
```java
@Test
public void test_open_character_stream_no_filename() {
    BufferedReader br = pt2.open_character_stream(null);
    assertTrue(br != null);
}
```

```java
@Test
public void test_open_character_stream_file_exists() {
    try {
        BufferedReader br = pt2.open_character_stream(DEFAULT_TEST_FILE);
        assertTrue(br.readLine().compareTo("Test File") == 0);
    } catch (IOException e) {
        System.out.println(e);
        fail();
    }
}
```

```java
@Test
public void test_open_character_stream_file_does_not_exist() {
    BufferedReader br = pt2.open_character_stream("nonexistant");
    assertNull(br);
}
```

METHOD 2 – GET_CHAR




```
47      /*********************************************/
48      /* NAME:         get_char                    */
49      /* INPUT:        a BufferedReader          */
50      /* OUTPUT:       a character            */
51
52      /*********************************************/
53      int get_char(BufferedReader br) {
54          int ch = 0;
55          try {
56              br.mark(4);         //marks a position, why spot 4?
57              ch = br.read();
58          } catch (IOException e) {
59              e.printStackTrace();
60          }
61          return ch;
62      }
```

Bugs and Test Cases

Bug(s)	Error(s)
None Found	None Found

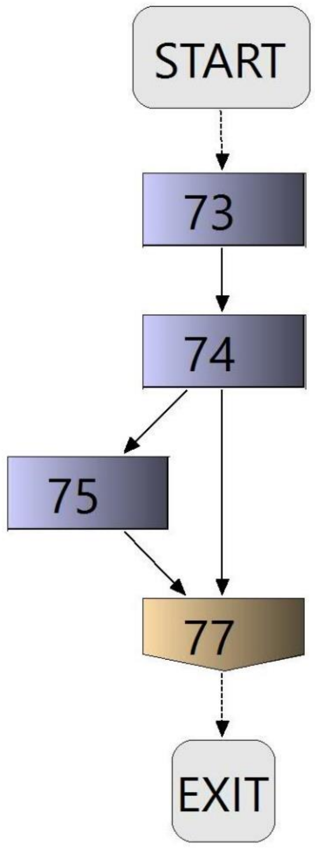
Test Methods

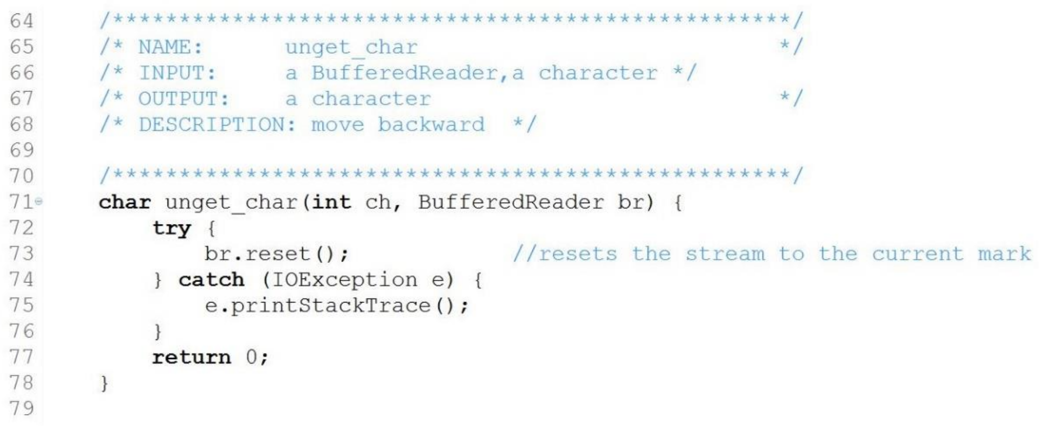
```
@Test
public void test_get_char() {
    byte [] bytes = {(byte) 'a'};
    BufferedReader br = new BufferedReader(new InputStreamReader(new ByteArrayInputStream(bytes)));
    assertEquals(pt2.get_char(br), (char)((byte)'a'));
}
```

```
@Test
public void test_get_char_io_exception() throws IOException {
    BufferedReader br = mock(BufferedReader.class);
    doThrow(new IOException()).when(br).read();
    pt2.get_char(br);
}
```

```
@Test
public void test_get_char_empty() {
    byte [] bytes = {};
    BufferedReader br = new BufferedReader(new InputStreamReader(new ByteArrayInputStream(bytes)));
    assertEquals(pt2.get_char(br), -1);
}
```

METHOD 3 – UNGET_CHAR




```
64      /*****************************************************/
65      /* NAME:       unget_char                            */
66      /* INPUT:      a BufferedReader,a character */
67      /* OUTPUT:     a character                           */
68      /* DESCRIPTION: move backward  */
69
70      /*****************************************************/
71      char unget_char(int ch, BufferedReader br) {
72          try {
73              br.reset();           //resets the stream to the current mark
74          } catch (IOException e) {
75              e.printStackTrace();
76          }
77          return 0;
78      }
79
```

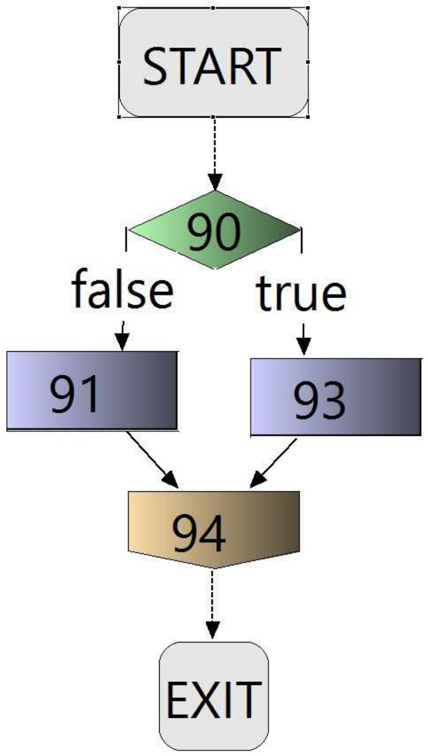
Bugs and Test Cases

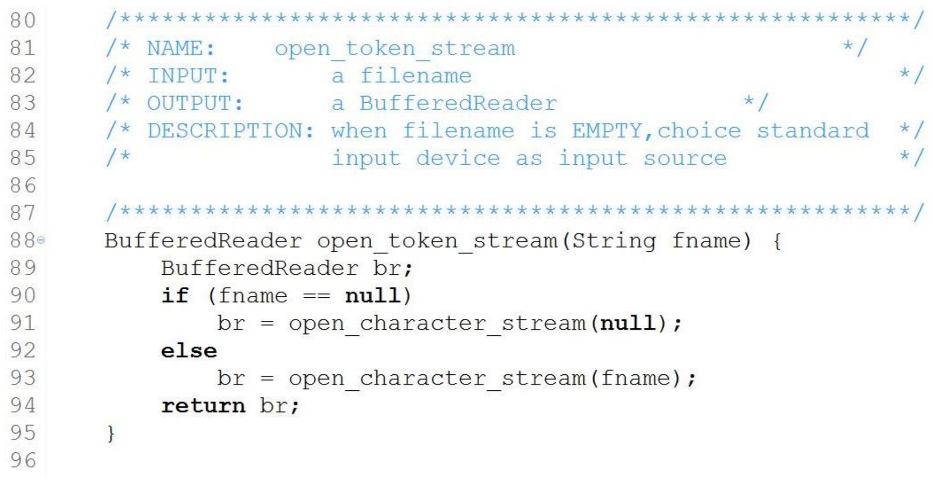
Bug(s)	Error(s)
None Found	None Found

Test Methods

```
char unget_char(int ch, BufferedReader br) {
    try {
        br.reset();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return 0;
}
```

METHOD 4 – OPEN_TOKEN_STREAM





```
80      /***************************************************************/
81      /* NAME:     open_token_stream                               */
82      /* INPUT:        a filename                                    */
83      /* OUTPUT:       a BufferedReader                    */
84      /* DESCRIPTION: when filename is EMPTY,choice standard  */
85      /*               input device as input source              */
86
87      /***************************************************************/
88      BufferedReader open_token_stream(String fname) {
89          BufferedReader br;
90          if (fname == null)
91              br = open_character_stream(null);
92          else
93              br = open_character_stream(fname);
94          return br;
95      }
96
```

Bugs and Test Cases

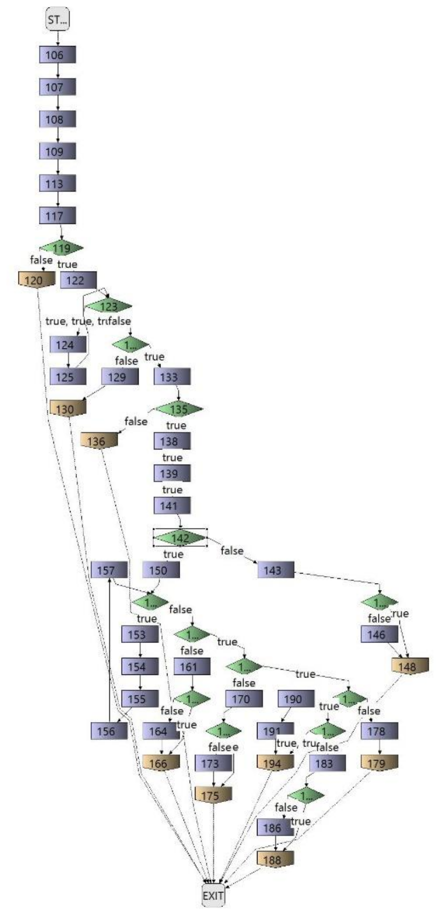
Bug(s)	Error(s)
None Found	None Found

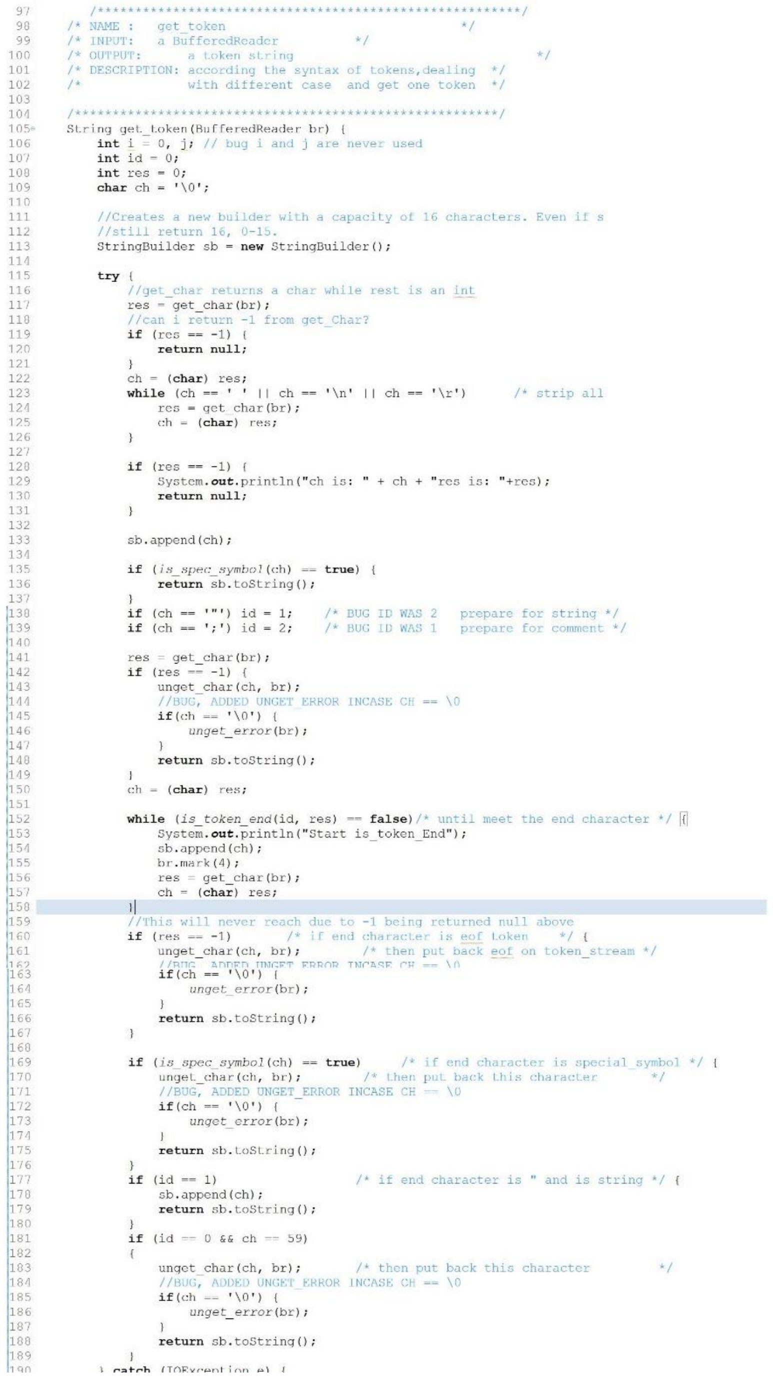
Test Methods

```
@Test
public void test_open_token_stream_null() {
    BufferedReader br = pt2.open_token_stream(null);
    assertTrue(br != null);
}
```

```
@Test
public void test_open_token_stream() {
    BufferedReader br = pt2.open_token_stream(DEFAULT_TEST_FILE);
    assertTrue(br != null);
}
```

METHOD 5 – GET_TOKEN





```
 97       /*******************************************************/
 98     /* NAME :   get_token                                  */
 99     /* INPUT:   a BufferedReader          */
100     /* OUTPUT:      a token string                                   */
101     /* DESCRIPTION: according the syntax of tokens,dealing  */
102     /*              with different case  and get one token  */
103
104     /*******************************************************/
105     String get_token(BufferedReader br) {
106         int i = 0, j; // bug i and j are never used
107         int id = 0;
108         int res = 0;
109         char ch = '\0';
110
111         //Creates a new builder with a capacity of 16 characters. Even if s
112         //still return 16, 0-15.
113         StringBuilder sb = new StringBuilder();
114
115         try {
116             //get_char returns a char while rest is an int
117             res = get_char(br);
118             //can i return -1 from get_Char?
119             if (res == -1) {
120                 return null;
121             }
122             ch = (char) res;
123             while (ch == ' ' || ch == '\n' || ch == '\r')      /* strip all
124                 res = get_char(br);
125                 ch = (char) res;
126             }
127
128             if (res == -1) {
129                 System.out.println("ch is: " + ch + "res is: "+res);
130                 return null;
131             }
132
133             sb.append(ch);
134
135             if (is_spec_symbol(ch) == true) {
136                 return sb.toString();
137             }
138             if (ch == '"') id = 1;    /* BUG ID WAS 2   prepare for string */
139             if (ch == ';') id = 2;    /* BUG ID WAS 1   prepare for comment */
140
141             res = get_char(br);
142             if (res == -1) {
143                 unget_char(ch, br);
144                 //BUG, ADDED UNGET_ERROR INCASE CH == \0
145                 if(ch == '\0') {
146                     unget_error(br);
147                 }
148                 return sb.toString();
149             }
150             ch = (char) res;
151
152             while (is_token_end(id, res) == false)/* until meet the end character */ {
153                 System.out.println("Start is_token_End");
154                 sb.append(ch);
155                 br.mark(4);
156                 res = get_char(br);
157                 ch = (char) res;
158             }
159             //This will never reach due to -1 being returned null above
160             if (res == -1)       /* if end character is eof token     */ {
161                 unget_char(ch, br);         /* then put back eof on token_stream */
162                 //BUG, ADDED UNGET_ERROR INCASE CH == \0
163                 if(ch == '\0') {
164                     unget_error(br);
165                 }
166                 return sb.toString();
167             }
168
169             if (is_spec_symbol(ch) == true)      /* if end character is special_symbol */ {
170                 unget_char(ch, br);         /* then put back this character       */
171                 //BUG, ADDED UNGET_ERROR INCASE CH == \0
172                 if(ch == '\0') {
173                     unget_error(br);
174                 }
175                 return sb.toString();
176             }
177             if (id == 1)                   /* if end character is " and is string */ {
178                 sb.append(ch);
179                 return sb.toString();
180             }
181             if (id == 0 && ch == 59)
182             {
183                 unget_char(ch, br);        /* then put back this character          */
184                 //BUG, ADDED UNGET_ERROR INCASE CH == \0
185                 if(ch == '\0') {
186                     unget_error(br);
187                 }
188                 return sb.toString();
189             }
190         } catch (IOException e) {
```

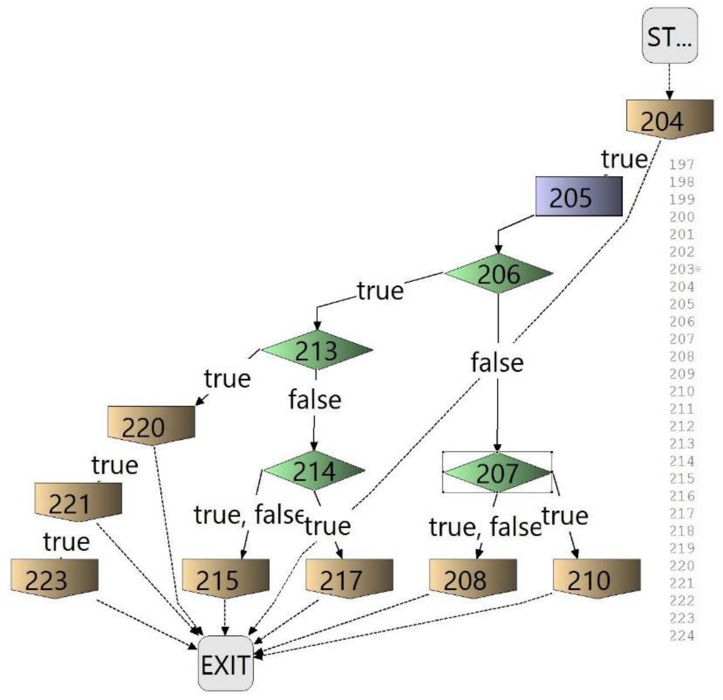
Bugs and Test Cases

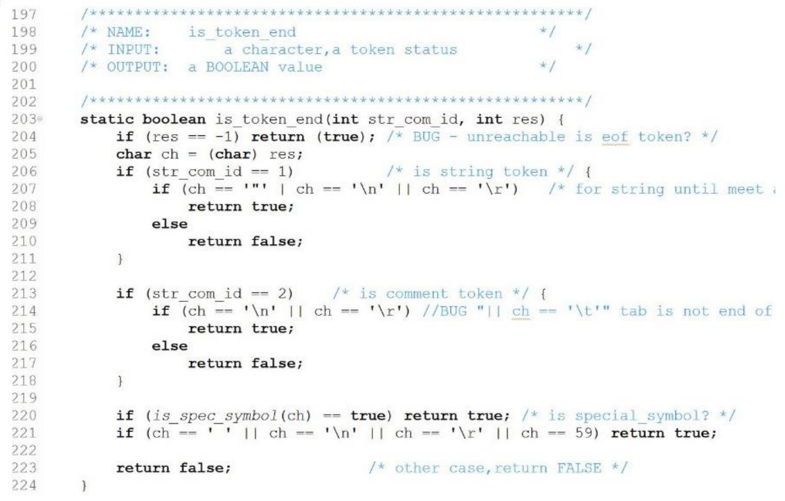
Bug(s)	Error(s)
`int i = 0, j;`	Variables are never used
`id = 2 when ch = '"'`	Misidentifies strings in the code
`id = 1 when ch = ';'`	Misidentifies comments in the code
`unget_error() isn't called after every unget_char()`	Improperly handles unget_char() process

Test Methods

```
@Test
public void test_string_get_token() {
    for(String testFile: TEST_FILE_LIST) {
        String result = pt2.get_token(pt2.open_token_stream(testFile));
        if(result != null) {
            assertFalse(result.equals(""));
        }
        else {
            assertNull(result);
        }
    }
}
```

METHOD 6 – IS_TOKEN_END




```
197  /********************************************************/
198  /* NAME:    is_token_end                               */
199  /* INPUT:       a character,a token status                */
200  /* OUTPUT:  a BOOLEAN value                            */
201
202  /********************************************************/
203  static boolean is_token_end(int str_com_id, int res) {
204      if (res == -1) return (true); /* BUG - unreachable is eof token? */
205      char ch = (char) res;
206      if (str_com_id == 1)           /* is string token */ {
207          if (ch == '"' | ch == '\n' || ch == '\r')   /* for string until meet 
208              return true;
209          else
210              return false;
211      }
212
213      if (str_com_id == 2)     /* is comment token */ {
214          if (ch == '\n' || ch == '\r') //BUG "|| ch == '\t'" tab is not end of
215              return true;
216          else
217              return false;
218      }
219
220      if (is_spec_symbol(ch) == true) return true; /* is special_symbol? */
221      if (ch == ' ' || ch == '\n' || ch == '\r' || ch == 59) return true;
222
223      return false;                /* other case,return FALSE */
224  }
```

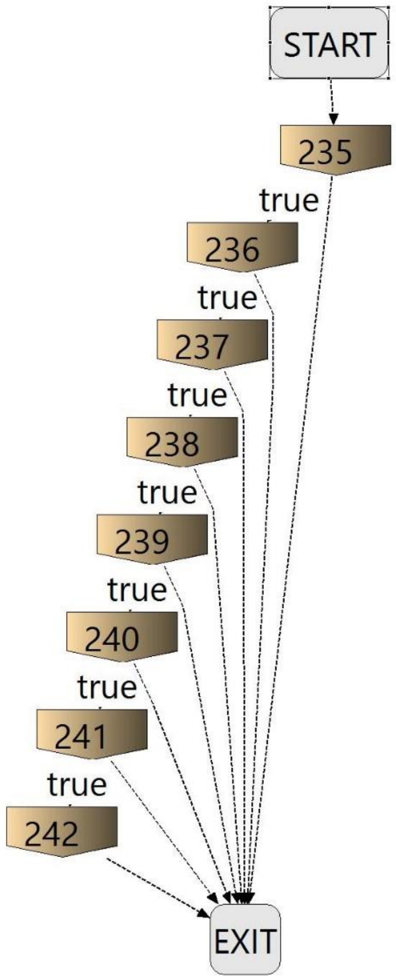
Bugs and Test Cases

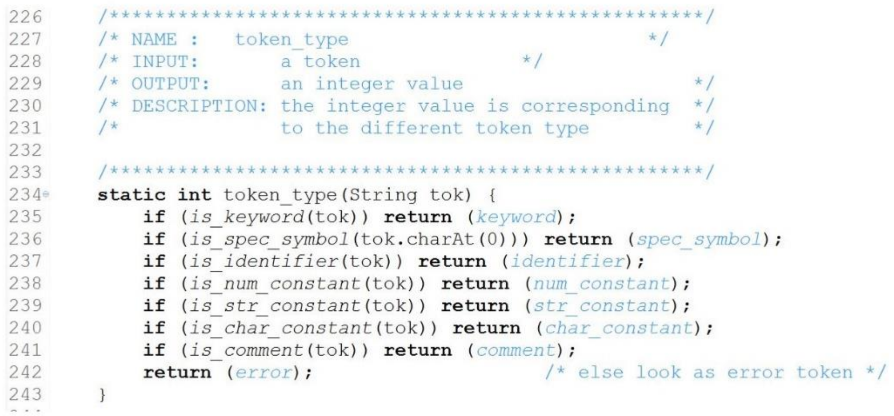
Bug(s)	Error(s)
Ch == ‘\t’	When str_com_id == 2, if condition checks for end of line. ‘\t’ is not end of line

Test Methods

```
@Test
public void test_is_token_end(){
    int com_id = 1;
    int res = -1;
    assertEquals(pt2.is_token_end(com_id,res), true);
}
```

METHOD 7 – TOKEN_TYPE





```
226      /*****************************************************/
227      /* NAME :   token_type                             */
228      /* INPUT:       a token             */
229      /* OUTPUT:      an integer value                     */
230      /* DESCRIPTION: the integer value is corresponding  */
231      /*              to the different token type         */
232
233      /*****************************************************/
234      static int token_type(String tok) {
235          if (is_keyword(tok)) return (keyword);
236          if (is_spec_symbol(tok.charAt(0))) return (spec_symbol);
237          if (is_identifier(tok)) return (identifier);
238          if (is_num_constant(tok)) return (num_constant);
239          if (is_str_constant(tok)) return (str_constant);
240          if (is_char_constant(tok)) return (char_constant);
241          if (is_comment(tok)) return (comment);
242          return (error);                    /* else look as error token */
243      }
```

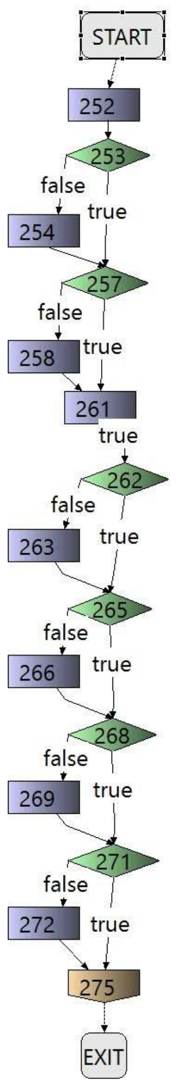
Bugs and Test Cases

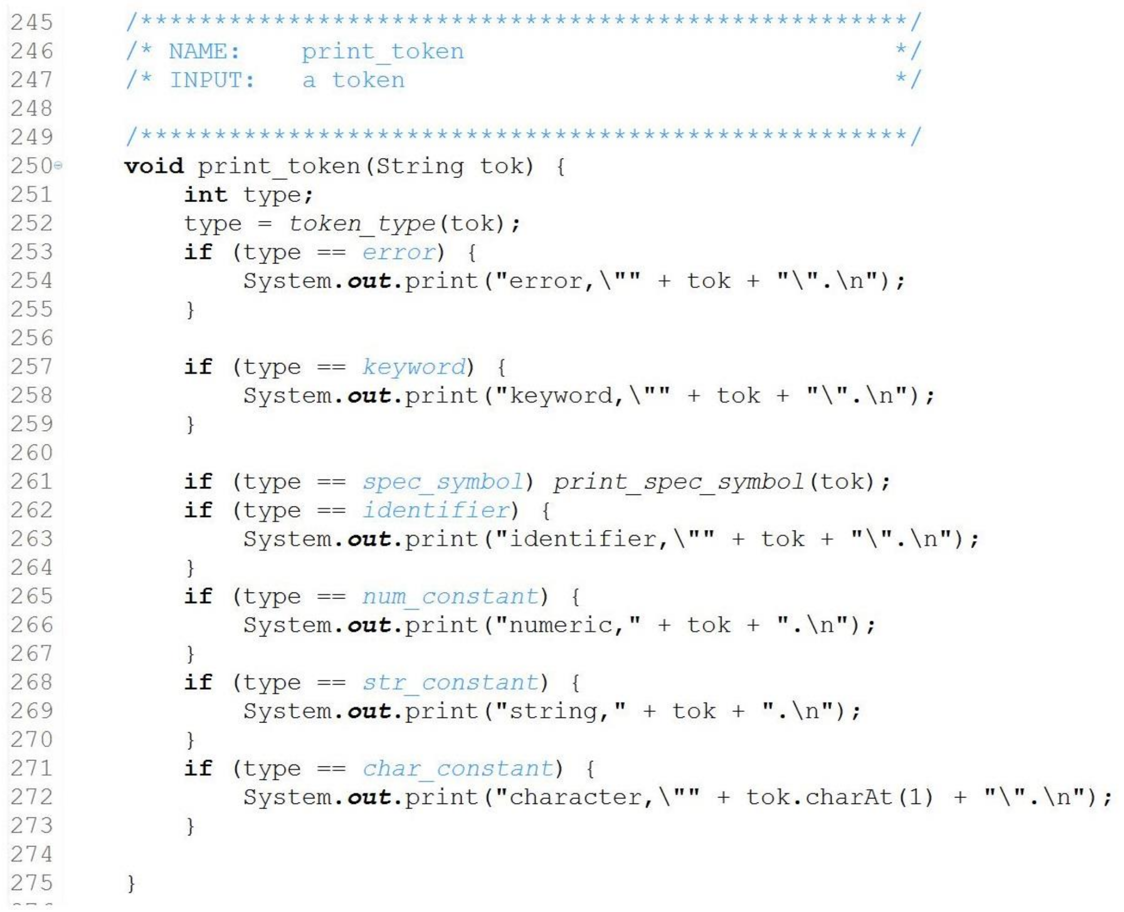
Bug(s)	Error(s)
None Found	None Found

Test Methods

```
@Test
public void test_token_type() {
    for(String testString: TEST_CHAR_LIST) {
        Integer result;
        result = pt2.token_type(testString);
        System.out.print(result);
     }
}
```

METHOD 8 – PRINT_TOKEN




```
245     /*************************************************/
246     /* NAME:    print_token                          */
247     /* INPUT:   a token                              */
248
249     /*************************************************/
250     void print_token(String tok) {
251         int type;
252         type = token_type(tok);
253         if (type == error) {
254             System.out.print("error,\"" + tok + "\".\n");
255         }
256
257         if (type == keyword) {
258             System.out.print("keyword,\"" + tok + "\".\n");
259         }
260
261         if (type == spec_symbol) print_spec_symbol(tok);
262         if (type == identifier) {
263             System.out.print("identifier,\"" + tok + "\".\n");
264         }
265         if (type == num_constant) {
266             System.out.print("numeric," + tok + ".\n");
267         }
268         if (type == str_constant) {
269             System.out.print("string," + tok + ".\n");
270         }
271         if (type == char_constant) {
272             System.out.print("character,\"" + tok.charAt(1) + "\".\n");
273         }
274
275     }
```

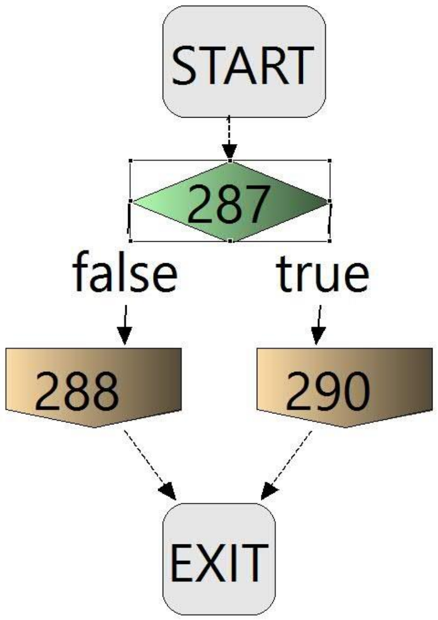
Bugs and Test Cases

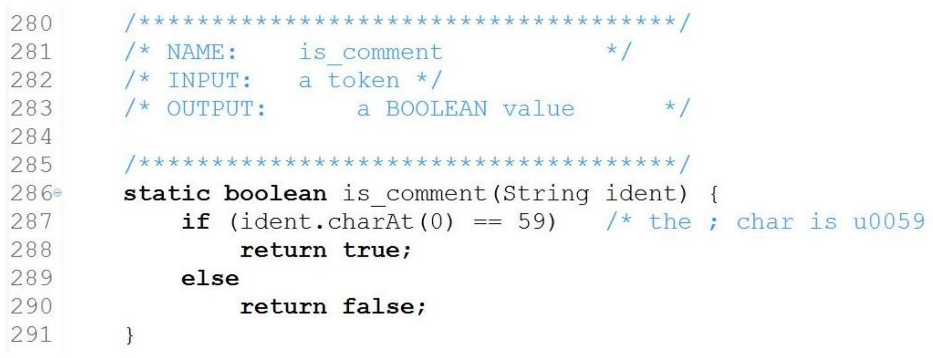
Bug(s)	Error(s)
None Found	None Found

Test Methods

```
@Test
public void test_print_token() {
    for(String testString: TEST_CHAR_LIST) {
        pt2.print_token(testString);
    }
}
```

METHOD 9 – IS_COMMENT





```
280    /***********************************/
281    /* NAME:    is_comment            */
282    /* INPUT:   a token */
283    /* OUTPUT:     a BOOLEAN value      */
284
285    /***********************************/
286    static boolean is_comment(String ident) {
287        if (ident.charAt(0) == 59)   /* the ; char is u0059
288            return true;
289        else
290            return false;
291    }
```

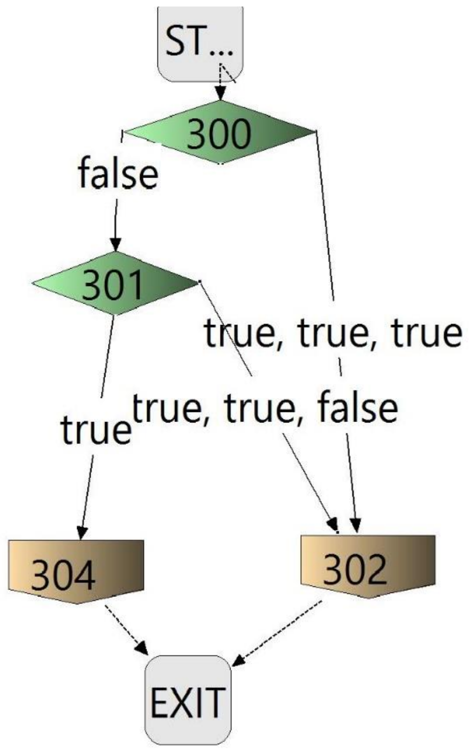
Bugs and Test Cases

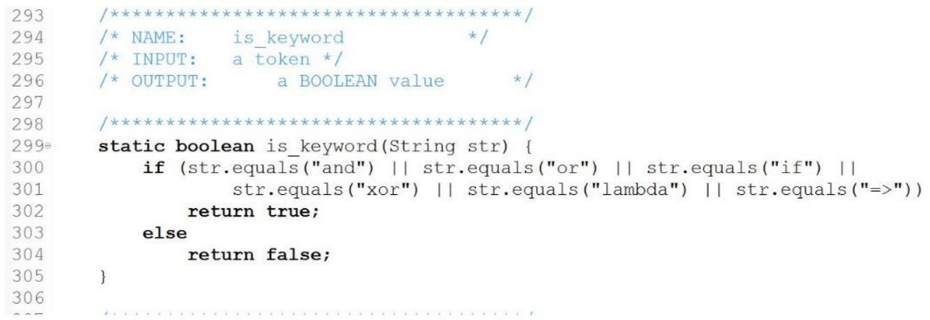
Bug(s)	Error(s)
None Found	None Found

Test Methods

```
@Test
public void test_true_is_comment(){
    String str = ";test";
    assertEquals(pt2.is_comment(str),true);
}
```

Method 10 - is_keyword





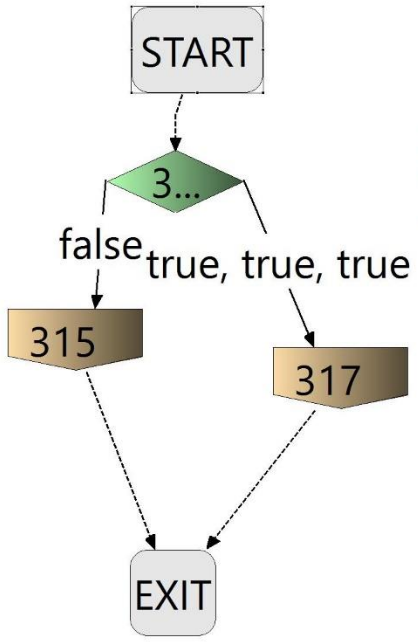
```
293     /*************************************/
294     /* NAME:    is_keyword           */
295     /* INPUT:   a token */
296     /* OUTPUT:      a BOOLEAN value      */
297
298     /*************************************/
299     static boolean is_keyword(String str) {
300         if (str.equals("and") || str.equals("or") || str.equals("if") ||
301                 str.equals("xor") || str.equals("lambda") || str.equals("=>"))
302             return true;
303         else
304             return false;
305     }
306
```

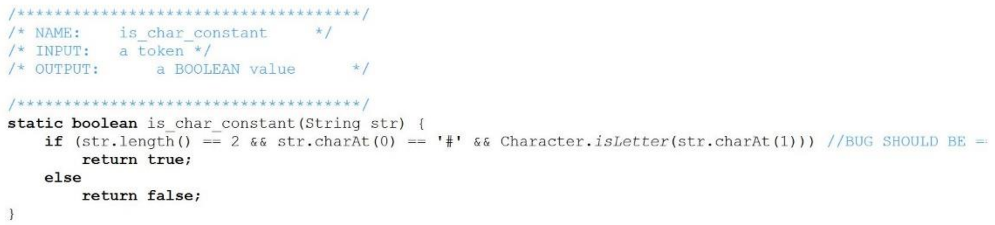
Bugs and Test Cases

Bug(s)	Error(s)
None Found	None Found

Test Methods
Covered by other tests and the code was relatively straightforward/simple.

METHOD 11 - IS_CHAR_CONSTANT





```
/**************************************/
/* NAME:    is_char_constant     */
/* INPUT:   a token */
/* OUTPUT:      a BOOLEAN value      */

/**************************************/
static boolean is_char_constant(String str) {
    if (str.length() == 2 && str.charAt(0) == '#' && Character.isLetter(str.charAt(1))) //BUG SHOULD BE =
        return true;
    else
        return false;
}
```

Bugs and Test Cases

Bug(s)	Error(s)
`str.length() >= 2`	Character constant should equal length of 2. Should reflect str.length() == 2, since it identifies the ‘#” plus the following character.

Test Methods

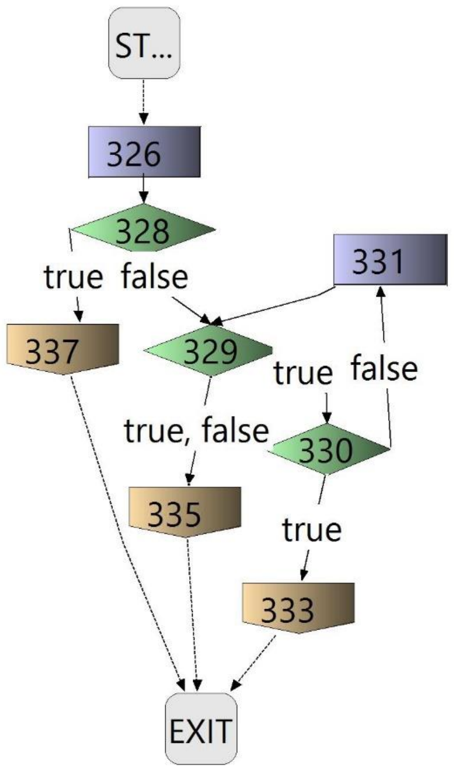
```java
@Test
public void test_is_char_constant(){
    String str = "#h";
    assertTrue(pt2.is_char_constant(str));
}
```

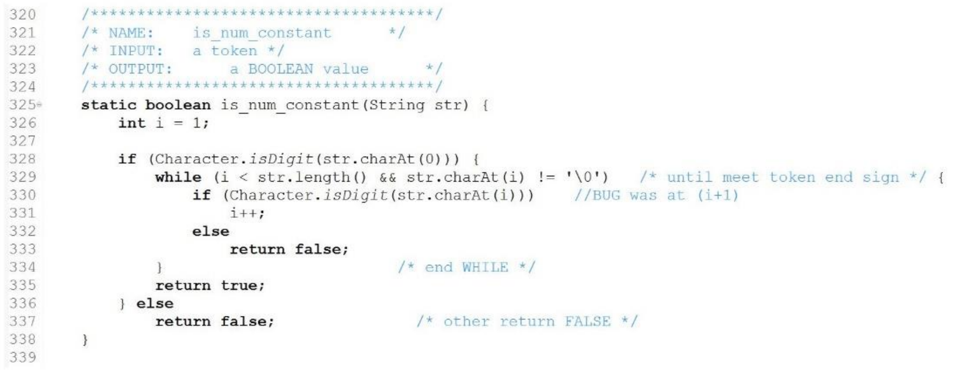
```java
@Test
public void test_true_is_char_constant(){
    String str = "#A";
    assertEquals(pt2.is_char_constant(str),true);
}
```

```java
@Test
public void test_false_is_char_constant(){
    String str = "$test";
    assertEquals(pt2.is_char_constant(str),false);
}
```

```java
@Test
public void test_empty_string_is_char_constant(){
    String str = "";
    assertEquals(pt2.is_char_constant(str),false);
}
```

METHOD 12 - IS_NUM_CONSTANT





```
320   /***************************************/
321   /* NAME:     is_num_constant       */
322   /* INPUT:    a token */
323   /* OUTPUT:       a BOOLEAN value      */
324   /***************************************/
325   static boolean is_num_constant(String str) {
326       int i = 1;
327
328       if (Character.isDigit(str.charAt(0))) {
329           while (i < str.length() && str.charAt(i) != '\0')   /* until meet token end sign */ {
330               if (Character.isDigit(str.charAt(i)))     //BUG was at (i+1)
331                   i++;
332               else
333                   return false;
334           }                          /* end WHILE */
335           return true;
336       } else
337           return false;                /* other return FALSE */
338   }
339
```

Bugs and Test Cases

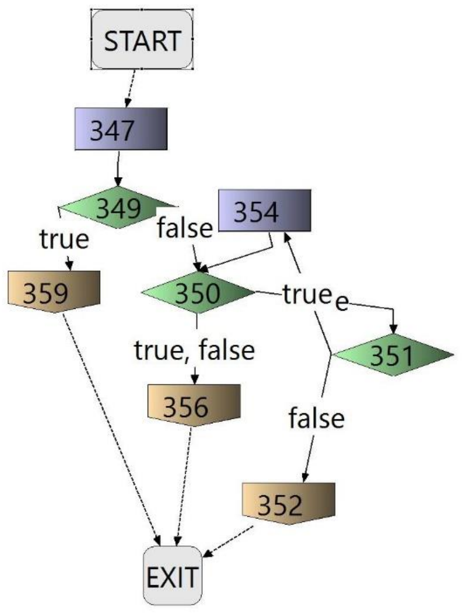
Bug(s)	Error(s)
str.charAt(i+1)	Parameter is incorrect. Needs to start initial check at 2nd index, or i when i = 1.

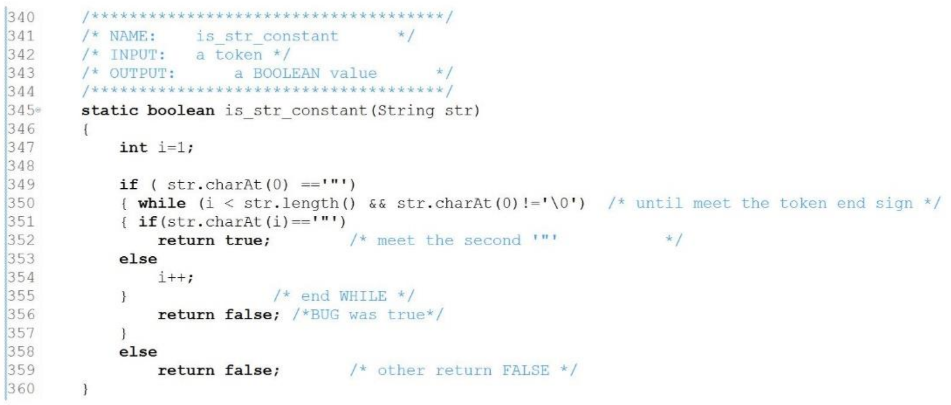
Test Methods

```
@Test
public void test_is_num_constant(){
    String str = "12345a";
    assertEquals(pt2.is_num_constant(str), false);
}
```

```
@Test
public void test1_is_num_constant(){
    String str = "12345";
    assertEquals(pt2.is_num_constant(str), true);
}
```

METHOD 13 - IS_STR_CONSTANT





```
340     /*************************************/
341     /* NAME:    is_str_constant      */
342     /* INPUT:   a token */
343     /* OUTPUT:       a BOOLEAN value      */
344     /*************************************/
345     static boolean is_str_constant(String str)
346     {
347         int i=1;
348
349         if ( str.charAt(0) =='"')
350         { while (i < str.length() && str.charAt(0)!='\0')  /* until meet the token end sign */
351         { if(str.charAt(i)=='"')
352             return true;         /* meet the second '"'            */
353         else
354             i++;
355         }                /* end WHILE */
356             return false; /*BUG was true*/
357         }
358         else
359             return false;        /* other return FALSE */
360     }
```

Bugs and Test Cases

Bug(s)	Error(s)
`else return true;`	Needs to return false if string does not end in quotations

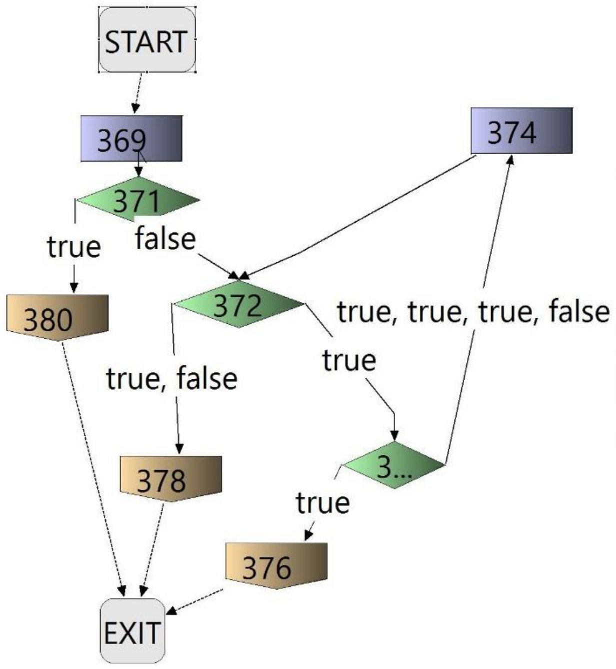
Test Methods

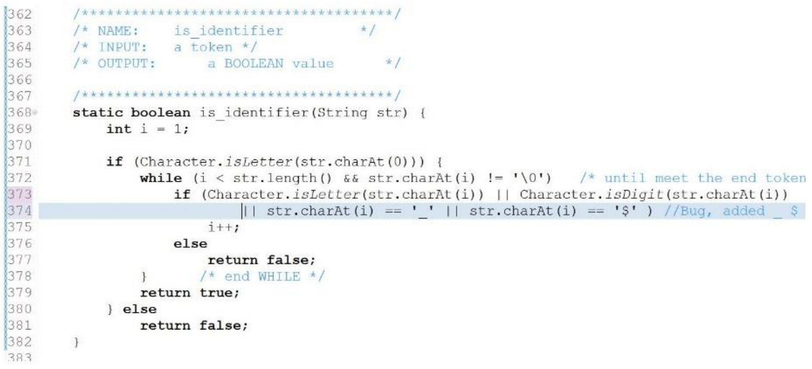
```
@Test
public void test_is_str_constant(){
    String str = "\"string\"";
    assertEquals(pt2.is_str_constant(str),true);
}
```

```
@Test
public void test_is_str_constant_alt(){
    String str = "\"string";
    assertEquals(pt2.is_str_constant(str),false);
}
```

```
@Test
public void test_false_is_str_constant(){
    String str = "S23ring";
    assertEquals(pt2.is_str_constant(str),false);
}
```

METHOD 14 - IS_IDENTIFIER





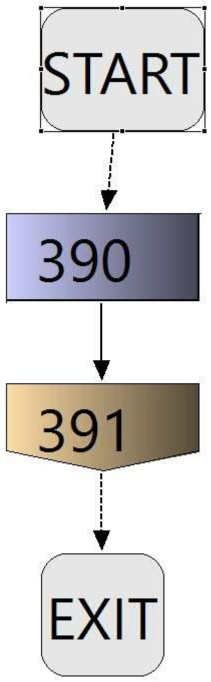
```
362     /*************************************/
363     /* NAME:    is_identifier          */
364     /* INPUT:   a token */
365     /* OUTPUT:      a BOOLEAN value      */
366 
367     /*************************************/
368     static boolean is_identifier(String str) {
369         int i = 1;
370 
371         if (Character.isLetter(str.charAt(0))) {
372             while (i < str.length() && str.charAt(i) != '\0')    /* until meet the end token
373                 if (Character.isLetter(str.charAt(i)) || Character.isDigit(str.charAt(i))
374                         || str.charAt(i) == '_' || str.charAt(i) == '$' ) //Bug, added _ $
375                     i++;
376                 else
377                     return false;
378             }       /* end WHILE */
379             return true;
380         } else
381             return false;
382     }
383
```

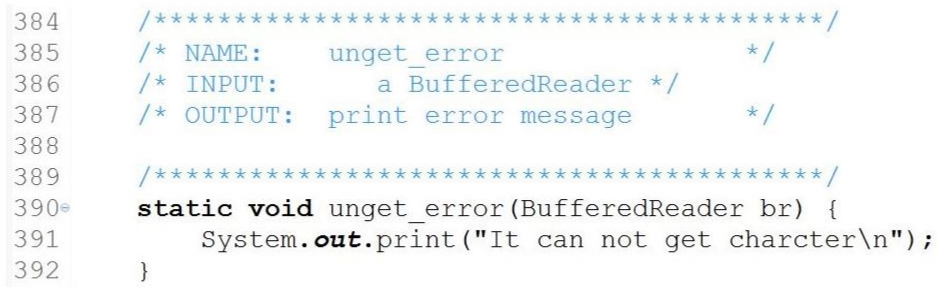
BUGS AND TEST CASE

Bug(s)	Error(s)
`Conditions for identifier`	Checks for only letters and digits. ‘_’ and ‘$’ are inclusive of identifiers

Test Methods
`Covered by other tests and the code was relatively straightforward/simple.`

METHOD 15 - UNGET_ERROR





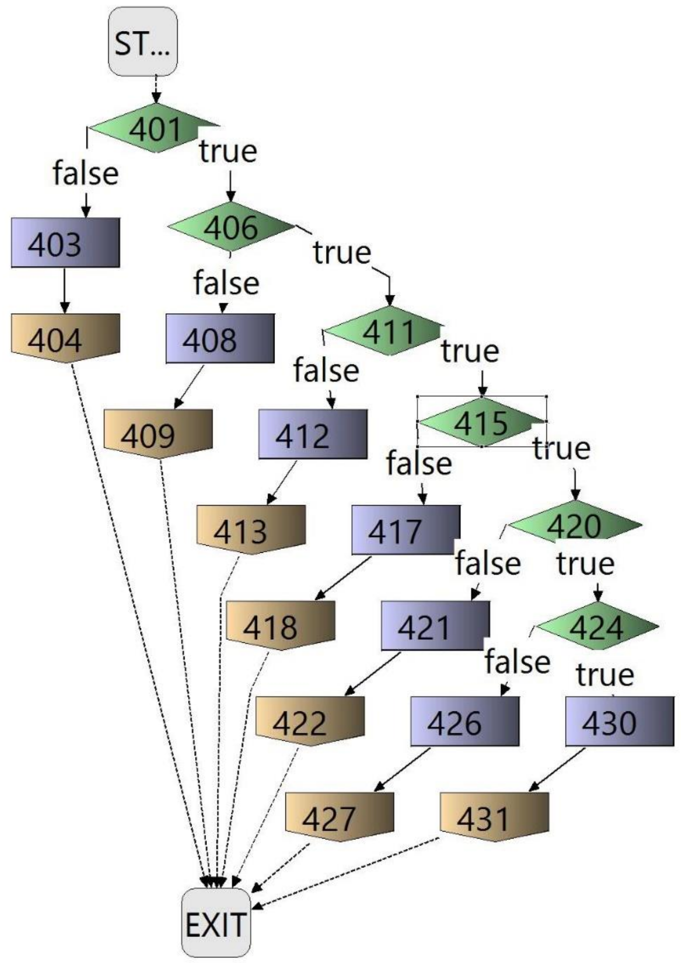
```
384     /*******************************************/
385     /* NAME:    unget_error                   */
386     /* INPUT:      a BufferedReader */
387     /* OUTPUT:  print error message         */
388
389     /*******************************************/
390     static void unget_error(BufferedReader br) {
391         System.out.print("It can not get charcter\n");
392     }
```

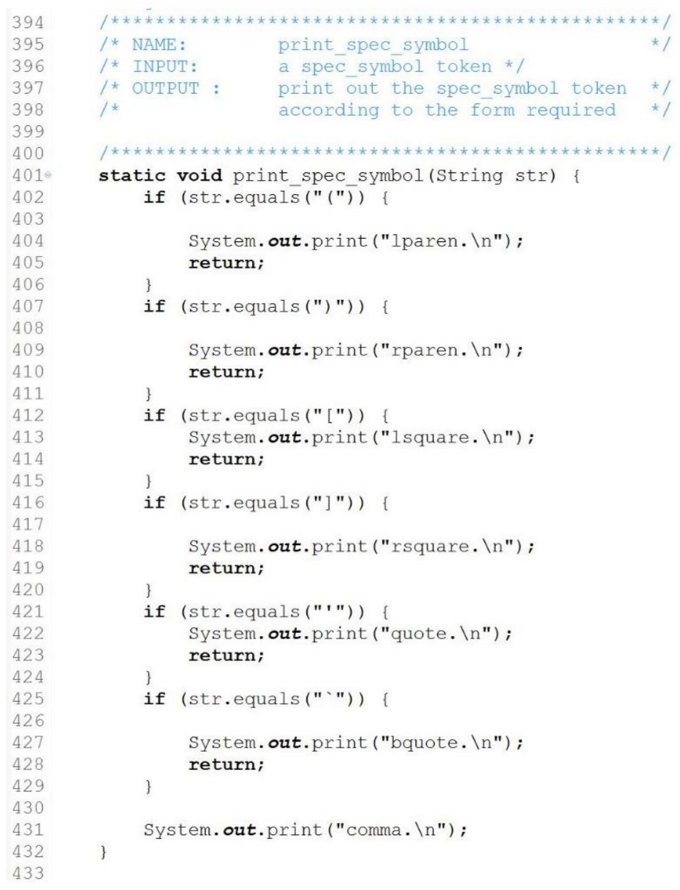
Bugs and Test Cases

Bug(s)	Error(s)
None Found	None Found

Test Methods
Covered by other tests and the code was relatively straightforward/simple.

METHOD 16 – PRINT_SPEC_SYMBOL



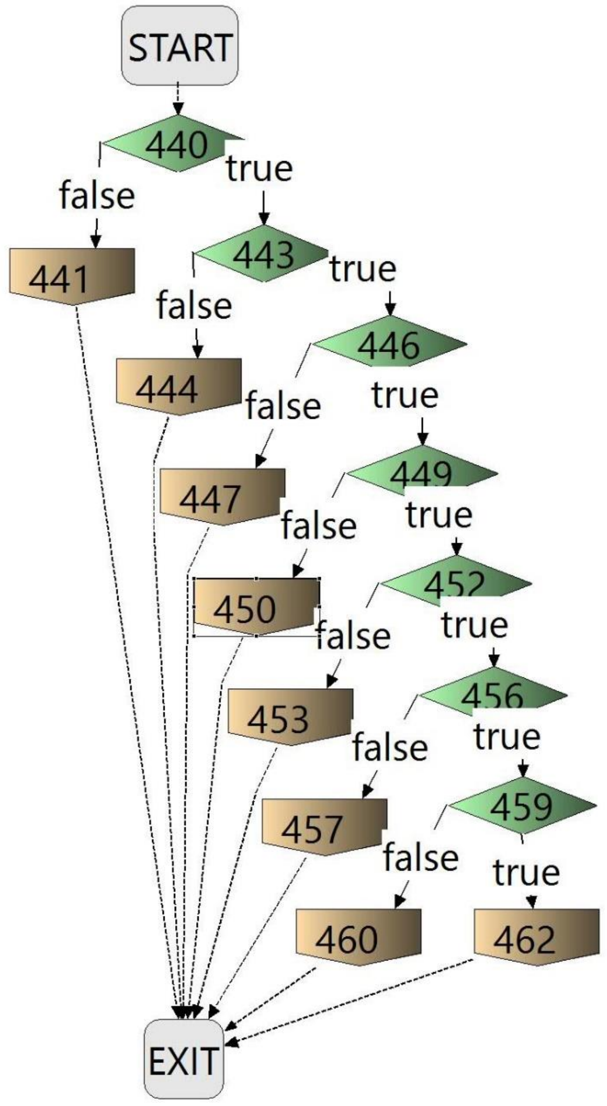

```
394  /**************************************************/
395  /* NAME:         print_spec_symbol                */
396  /* INPUT:        a spec_symbol token */
397  /* OUTPUT :      print out the spec_symbol token  */
398  /*               according to the form required   */
399
400  /**************************************************/
401  static void print_spec_symbol(String str) {
402      if (str.equals("(")) {
403
404          System.out.print("lparen.\n");
405          return;
406      }
407      if (str.equals(")")) {
408
409          System.out.print("rparen.\n");
410          return;
411      }
412      if (str.equals("[")) {
413          System.out.print("lsquare.\n");
414          return;
415      }
416      if (str.equals("]")) {
417
418          System.out.print("rsquare.\n");
419          return;
420      }
421      if (str.equals("'")) {
422          System.out.print("quote.\n");
423          return;
424      }
425      if (str.equals("`")) {
426
427          System.out.print("bquote.\n");
428          return;
429      }
430
431      System.out.print("comma.\n");
432  }
433
```

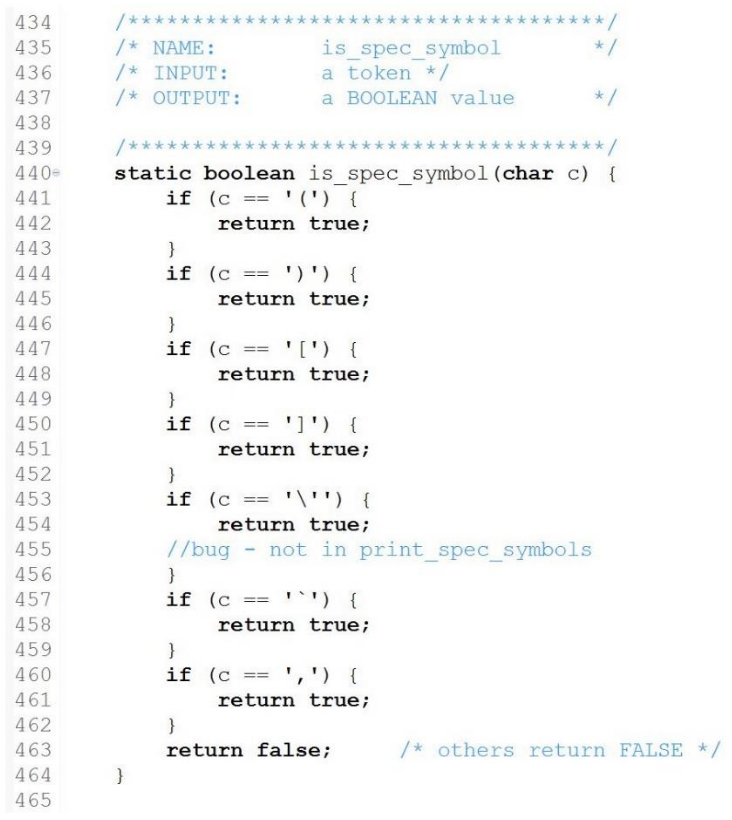
Bugs and Test Cases

Bug(s)	Error(s)
None Found	**None Found**

Test Methods
Covered by other tests and the code was relatively straightforward/simple.

METHOD 17 - IS_SPEC_SYMBOL



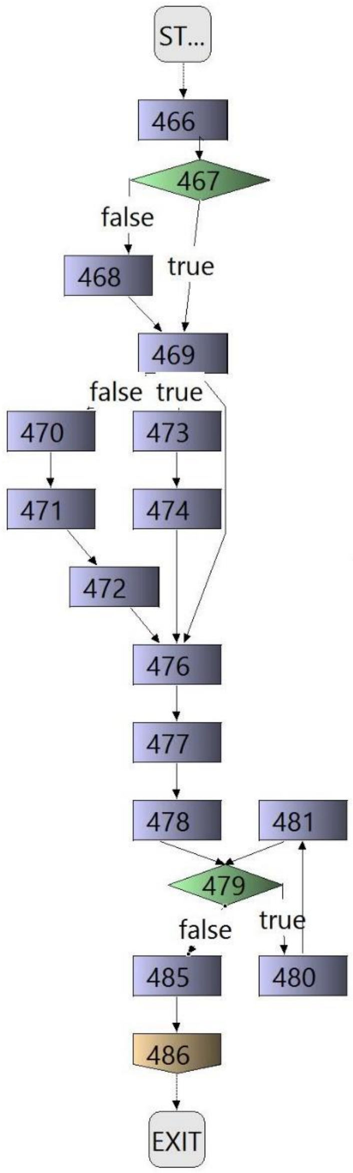

```
434    /***********************************/
435    /* NAME:        is_spec_symbol      */
436    /* INPUT:       a token */
437    /* OUTPUT:      a BOOLEAN value     */
438
439    /***********************************/
440    static boolean is_spec_symbol(char c) {
441        if (c == '(') {
442            return true;
443        }
444        if (c == ')') {
445            return true;
446        }
447        if (c == '[') {
448            return true;
449        }
450        if (c == ']') {
451            return true;
452        }
453        if (c == '\'') {
454            return true;
455        //bug - not in print_spec_symbols
456        }
457        if (c == '`') {
458            return true;
459        }
460        if (c == ',') {
461            return true;
462        }
463        return false;     /* others return FALSE */
464    }
465
```

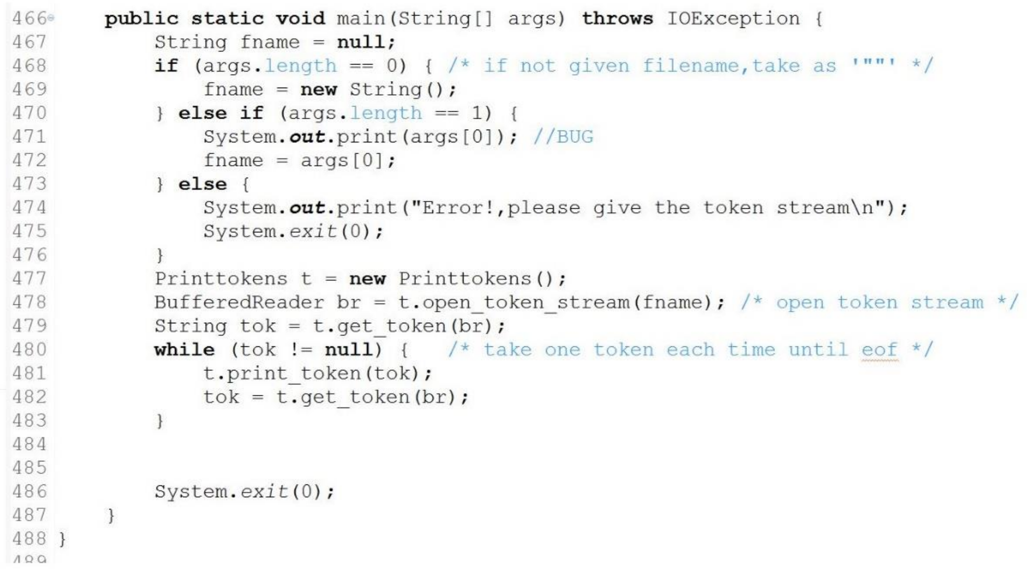
Bugs and Test Cases

Bug(s)	Error(s)
`if(c == '\'')`	Is not used in print_spec_symbols

Test Methods
Covered by other tests and the code was relatively straightforward/simple.

METHOD 18 - MAIN





```
466      public static void main(String[] args) throws IOException {
467          String fname = null;
468          if (args.length == 0) { /* if not given filename,take as '""' */
469              fname = new String();
470          } else if (args.length == 1) {
471              System.out.print(args[0]); //BUG
472              fname = args[0];
473          } else {
474              System.out.print("Error!,please give the token stream\n");
475              System.exit(0);
476          }
477          Printtokens t = new Printtokens();
478          BufferedReader br = t.open_token_stream(fname); /* open token stream */
479          String tok = t.get_token(br);
480          while (tok != null) {    /* take one token each time until eof */
481              t.print_token(tok);
482              tok = t.get_token(br);
483          }
484
485
486          System.exit(0);
487      }
488  }
```

BUGS AND TEST CASES

Bug(s)	Error(s)
fname = args[1] should call 0 element	Calls incorrect argument; should have element 1 or ‘0’ as parameter

Test Methods

```java
@Test
public void test_main_file_as_arg() throws IOException {
    exit.expectSystemExitWithStatus(0);
    for(String testFile: TEST_FILE_LIST) {
        pt2.main(new String[]{testFile});
    }
}
```

```java
@Test
public void test_main_no_args() throws IOException {
    try {
        pt2.main(new String[]{});
    }
    catch(NullPointerException e) {
        // Expecting this to happen
    }
}
```

```java
@Test
public void test_main_too_many_args() throws IOException {
    exit.expectSystemExitWithStatus(0);
    pt2.main(new String[] {"testFile0.txt", "testFile0.txt", "testFile0.txt"});
}
```