Software Testing and Maintenance

Project Report

Harris Nghiem – 1000572896

Damian Jimenez – 1000584863

May 5th, 2017

Table of Contents

Contents

[Overview 4](#_Toc481762891)

[jacoco Output 6](#_Toc481762892)

[Method 1 – OPEN\_CHARACTER\_STREAM 7](#_Toc481762893)

[Bugs and Test Cases 8](#_Toc481762894)

[Method 2 – get\_char 9](#_Toc481762895)

[Bugs and Test Cases 10](#_Toc481762896)

[Method 3 – unget\_char 11](#_Toc481762897)

[Bugs and Test Cases 12](#_Toc481762898)

[Method 4 – open\_token\_stream 13](#_Toc481762899)

[Bugs and Test Cases 14](#_Toc481762900)

[Method 5 – get\_token 15](#_Toc481762901)

[Bugs and Test Cases 16](#_Toc481762902)

[Method 6 – is\_token\_end 17](#_Toc481762903)

[Bugs and Test Cases 18](#_Toc481762904)

[Method 7 – token\_type 19](#_Toc481762905)

[Bugs and Test Cases 20](#_Toc481762906)

[Method 8 – print\_token 21](#_Toc481762907)

[Bugs and Test Cases 22](#_Toc481762908)

[Method 9 – is\_comment 23](#_Toc481762909)

[Bugs and Test Cases 24](#_Toc481762910)

[Method 10 - is\_keyword 25](#_Toc481762911)

[Bugs and Test Cases 26](#_Toc481762912)

[Method 11 - is\_char\_constant 27](#_Toc481762913)

[Bugs and Test Cases 28](#_Toc481762914)

[Method 12 - is\_num\_constant 29](#_Toc481762915)

[Bugs and Test Cases 30](#_Toc481762916)

[Method 13 - is\_str\_constant 31](#_Toc481762917)

[Bugs and Test Cases 32](#_Toc481762918)

[Method 14 - is\_identifier 33](#_Toc481762919)

[Bugs and test case 34](#_Toc481762920)

[Method 15 - unget\_error 35](#_Toc481762921)

[Bugs and Test Cases 36](#_Toc481762922)

[Method 16 – print\_spec\_symbol 37](#_Toc481762923)

[Bugs and test cases 38](#_Toc481762924)

[Method 17 - is\_spec\_symbol 39](#_Toc481762925)

[Bugs and Test Cases 40](#_Toc481762926)

[Method 18 - Main 41](#_Toc481762927)

[Bugs and test cases 42](#_Toc481762928)

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:**

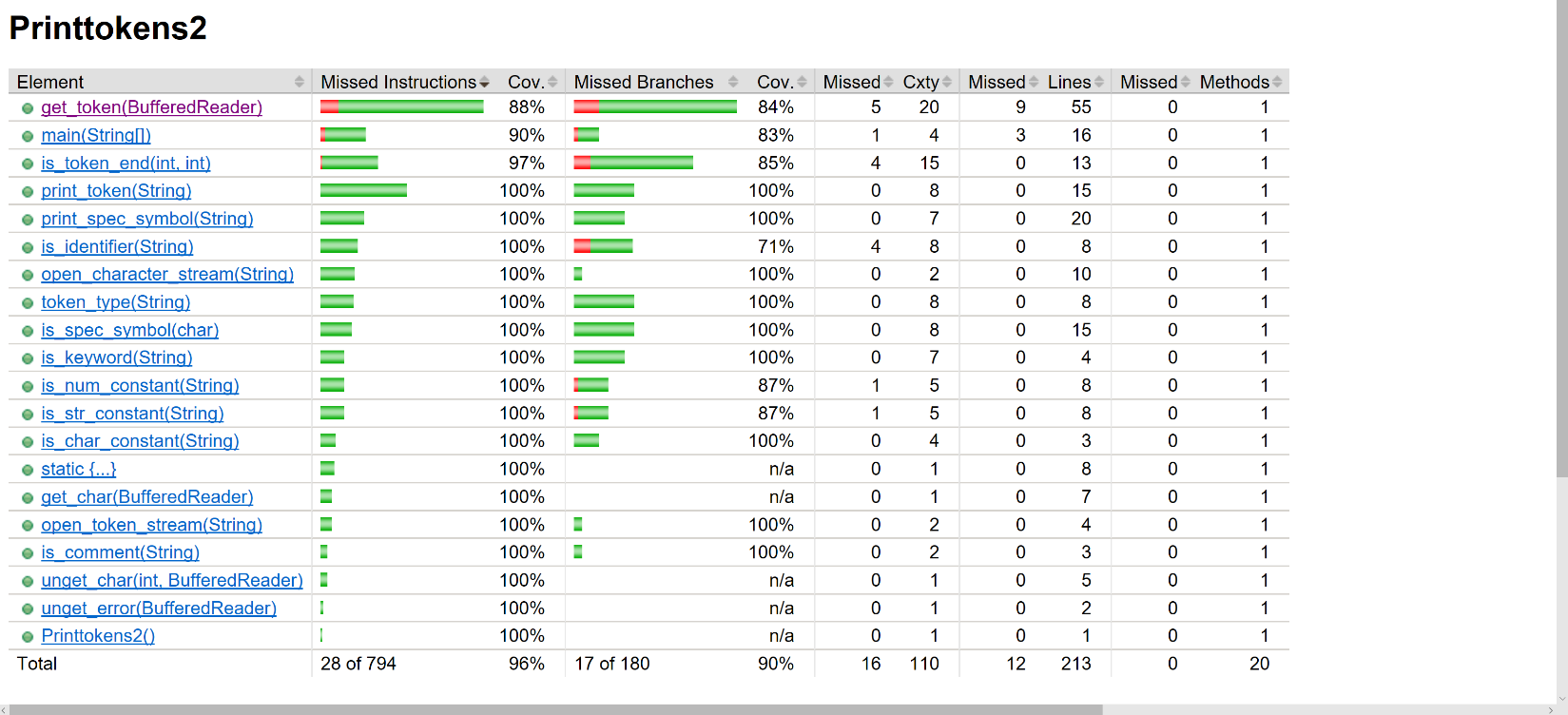
We decided to use IntelliJ’s Idea IDE along and create a Maven project that allowed us to easily and quickly import any dependencies that were required to get our test cases successfully running. Achieving 90%+ code coverage was relatively easy, but finding the bugs turned out to be a bit more difficult 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**



|  |
| --- |
| 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**

## ****Bugs and Test Cases****

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

|  |
| --- |
| Test Methods |
| @Test public void test\_open\_character\_stream\_no\_filename() {  BufferedReader br = pt2.open\_character\_stream(null);  *assertTrue*(br != null); } |
| @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*();  } } |
| @Test public void test\_open\_character\_stream\_file\_does\_not\_exist() {  BufferedReader br = pt2.open\_character\_stream("nonexistant");  *assertNull*(br); } |

# **Method 2 – get\_char**

## ****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**

## ****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**

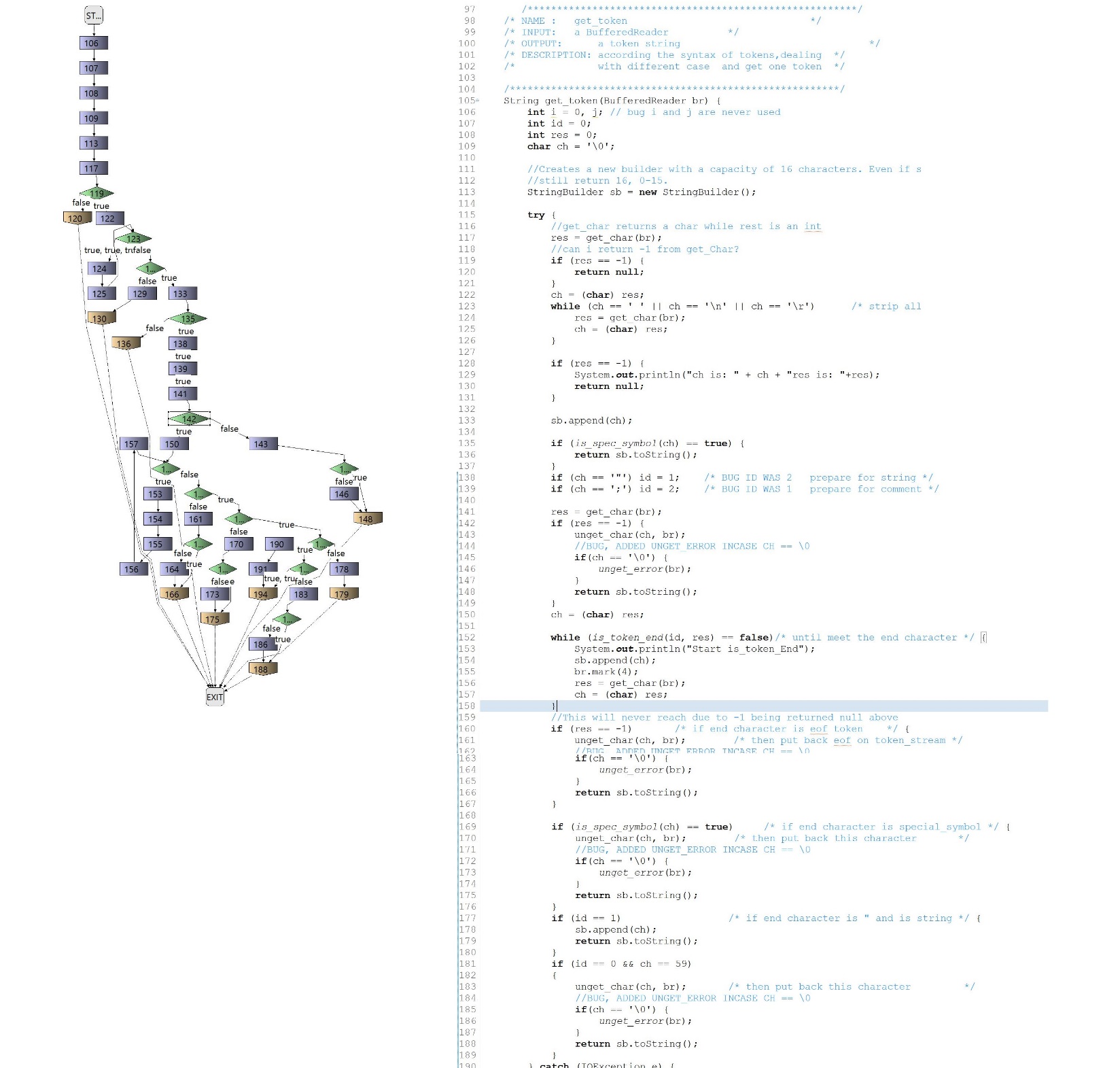


## ****Bugs and Test Cases****

|  |  |
| --- | --- |
| 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**

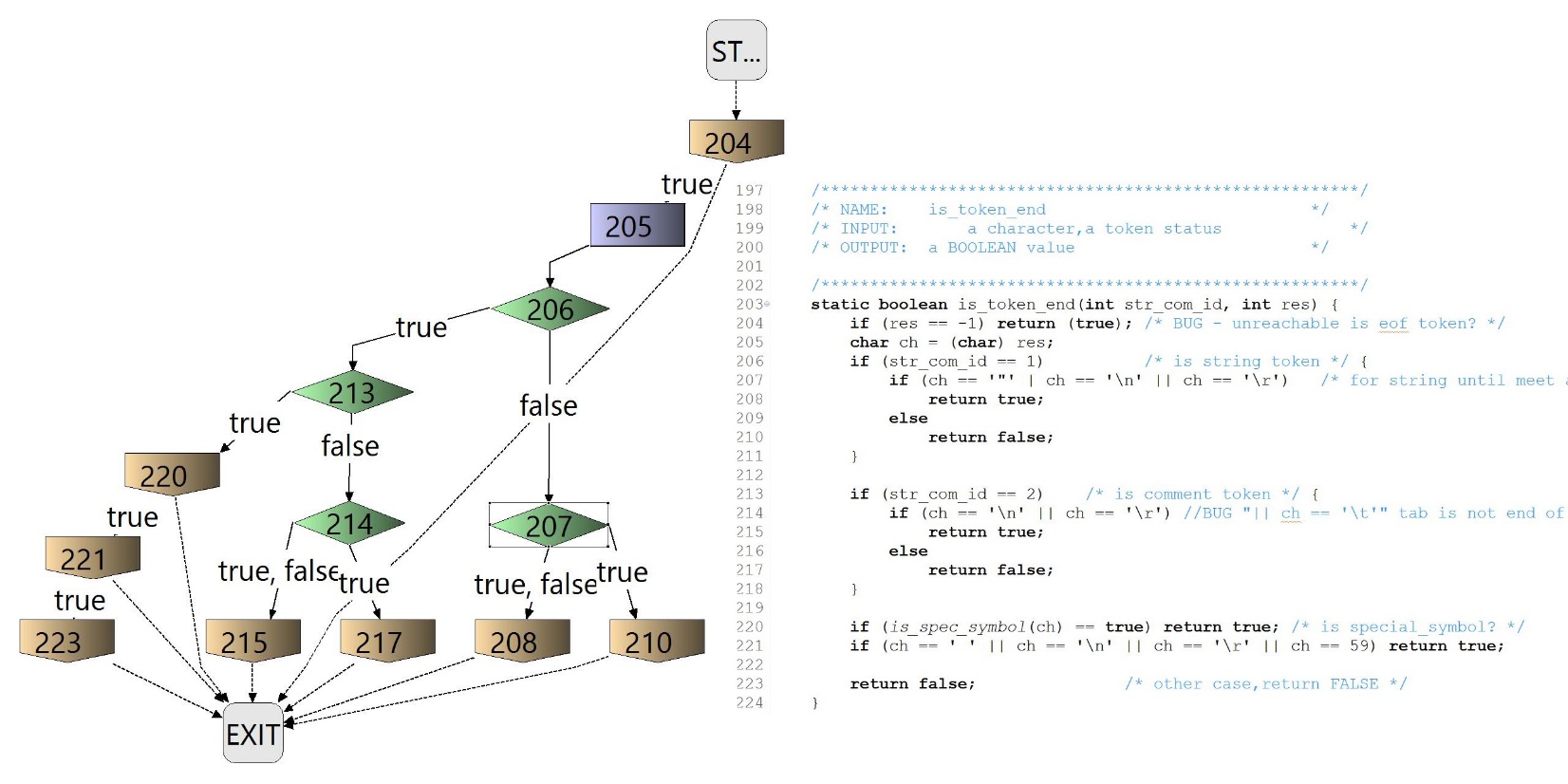


## ****Bugs and Test Cases****

|  |  |  |
| --- | --- | --- |
| 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**

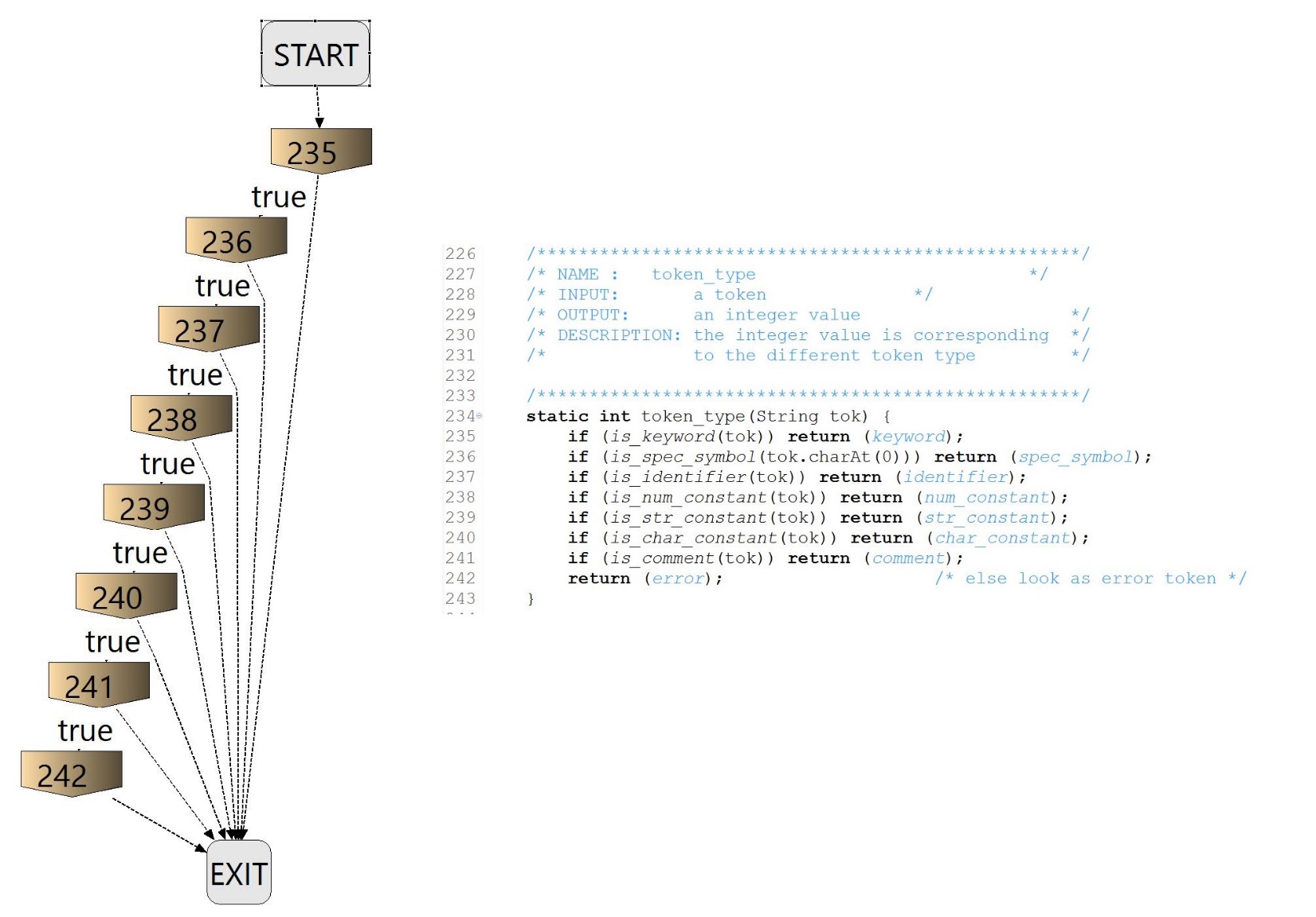


## ****Bugs and Test Cases****

|  |  |
| --- | --- |
| 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**



## ****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**

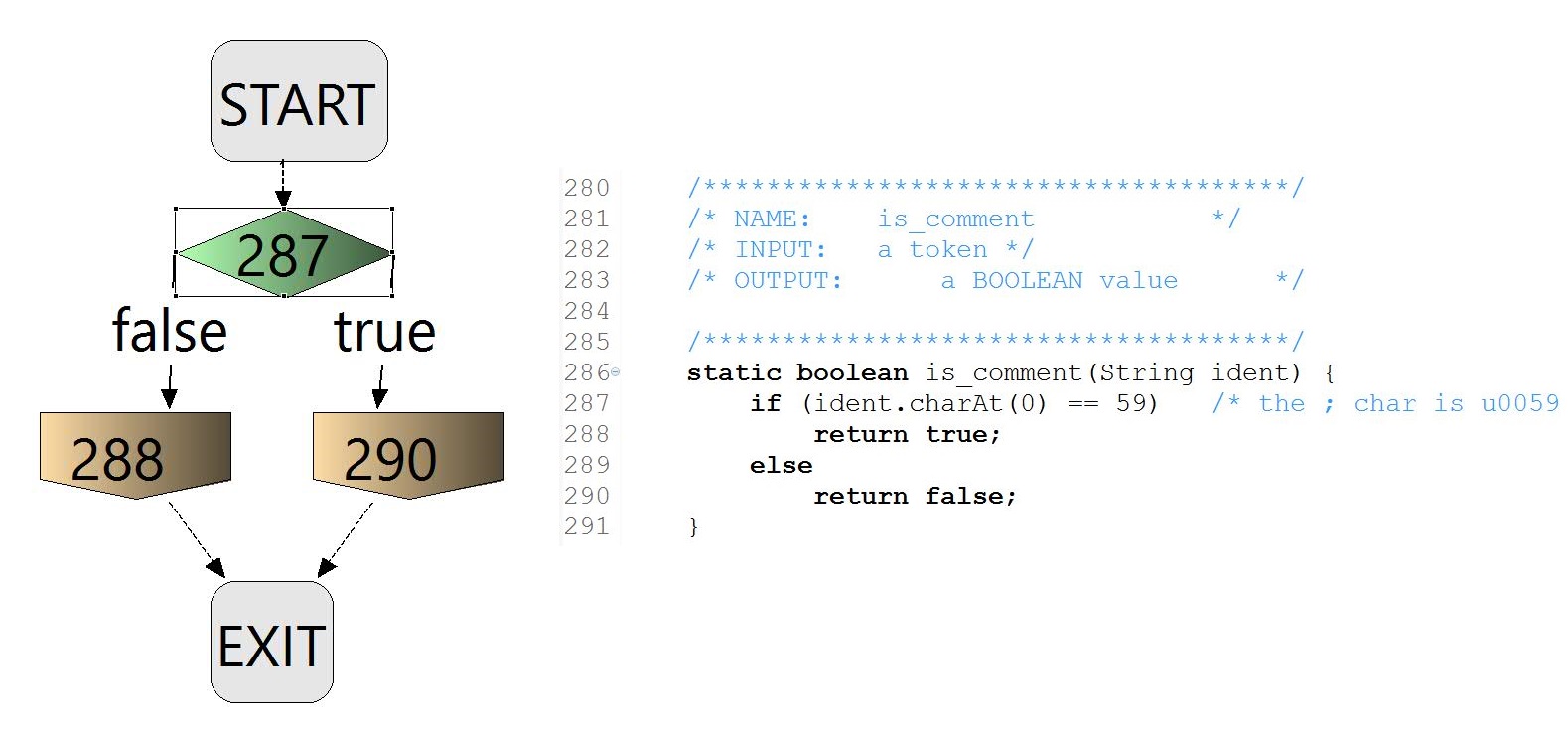


## ****Bugs and Test Cases****

|  |  |
| --- | --- |
| 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**



## ****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**



## **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**

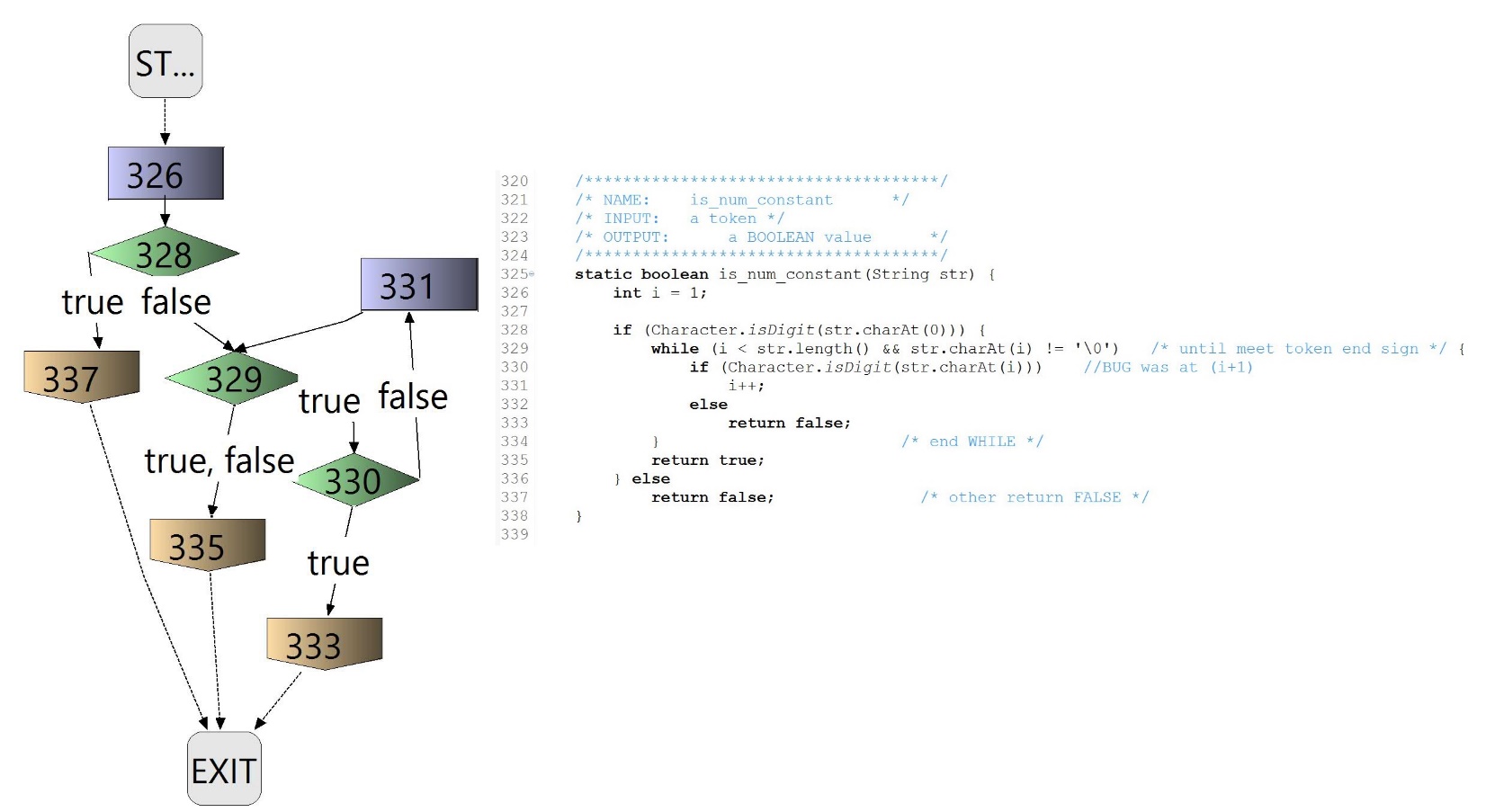


## **Bugs and Test Cases**

|  |  |
| --- | --- |
| 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 |
| @Test public void test\_is\_char\_constant(){  String str = "#h";  *assertTrue*(pt2.*is\_char\_constant*(str)); } |
| @Test public void test\_true\_is\_char\_constant(){  String str = "#A";  *assertEquals*(pt2.*is\_char\_constant*(str),true); } |
| @Test public void test\_false\_is\_char\_constant(){  String str = "$test";  *assertEquals*(pt2.*is\_char\_constant*(str),false); } |
| @Test public void test\_empty\_string\_is\_char\_constant(){  String str = "";  *assertEquals*(pt2.*is\_char\_constant*(str),false); } |

# **Method 12** **- is\_num\_constant**

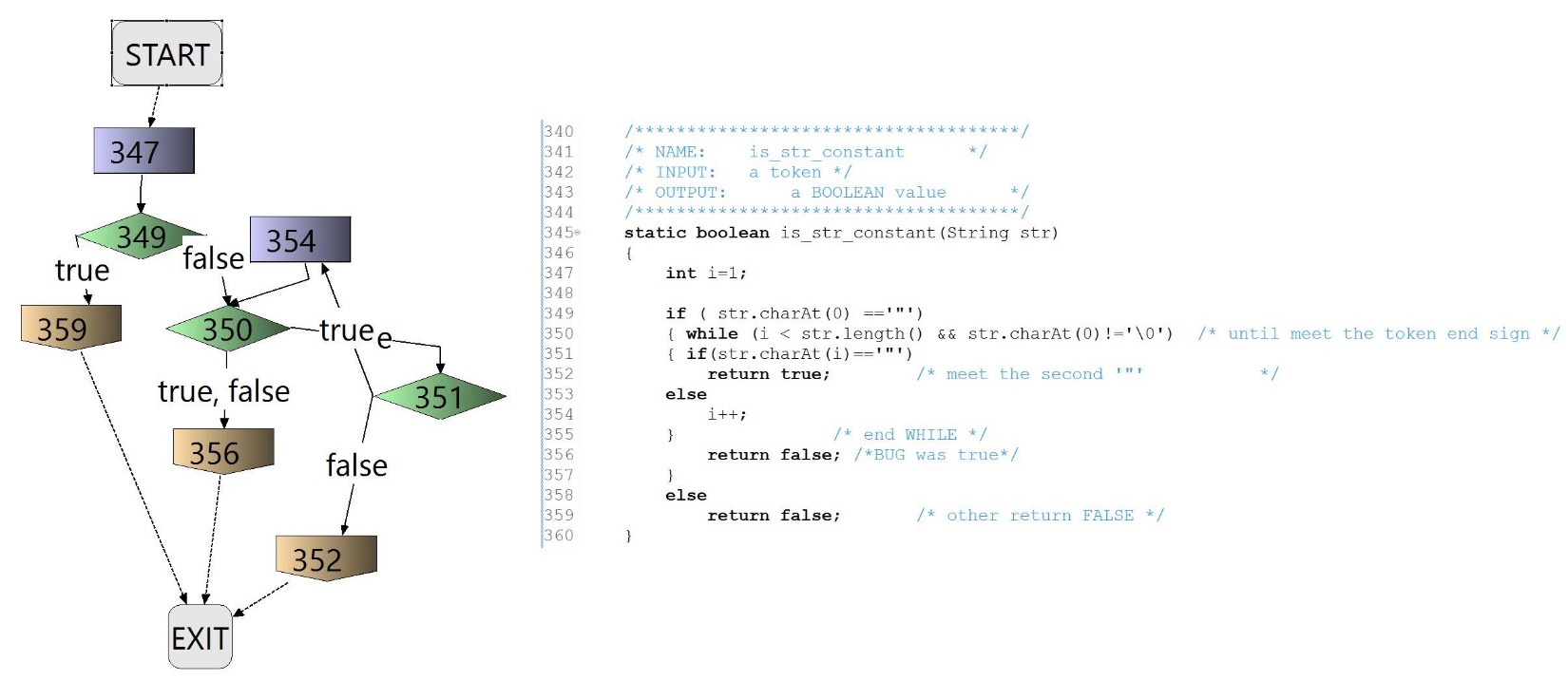


## **Bugs and Test Cases**

|  |  |
| --- | --- |
| 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**

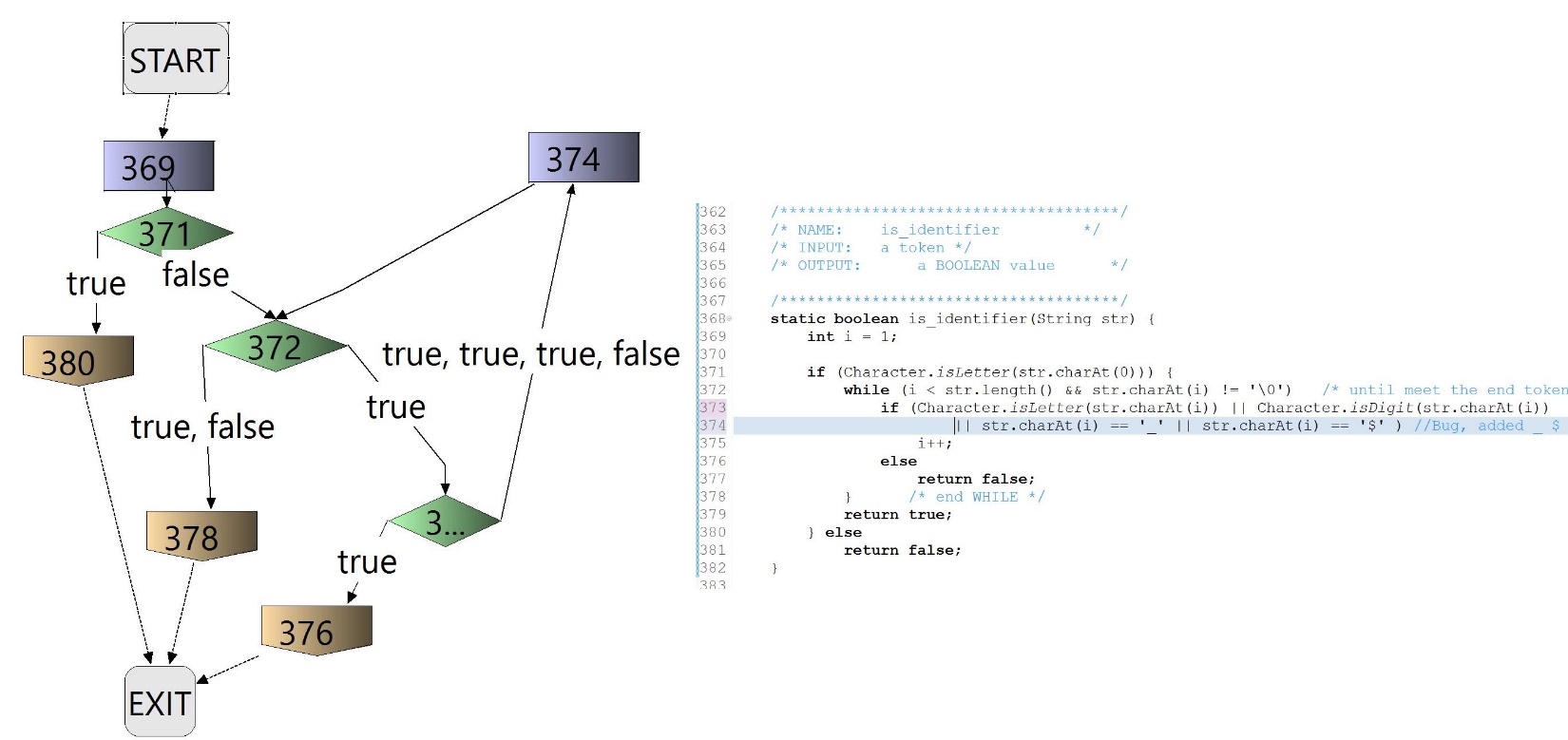


## **Bugs and Test Cases**

|  |  |
| --- | --- |
| 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**

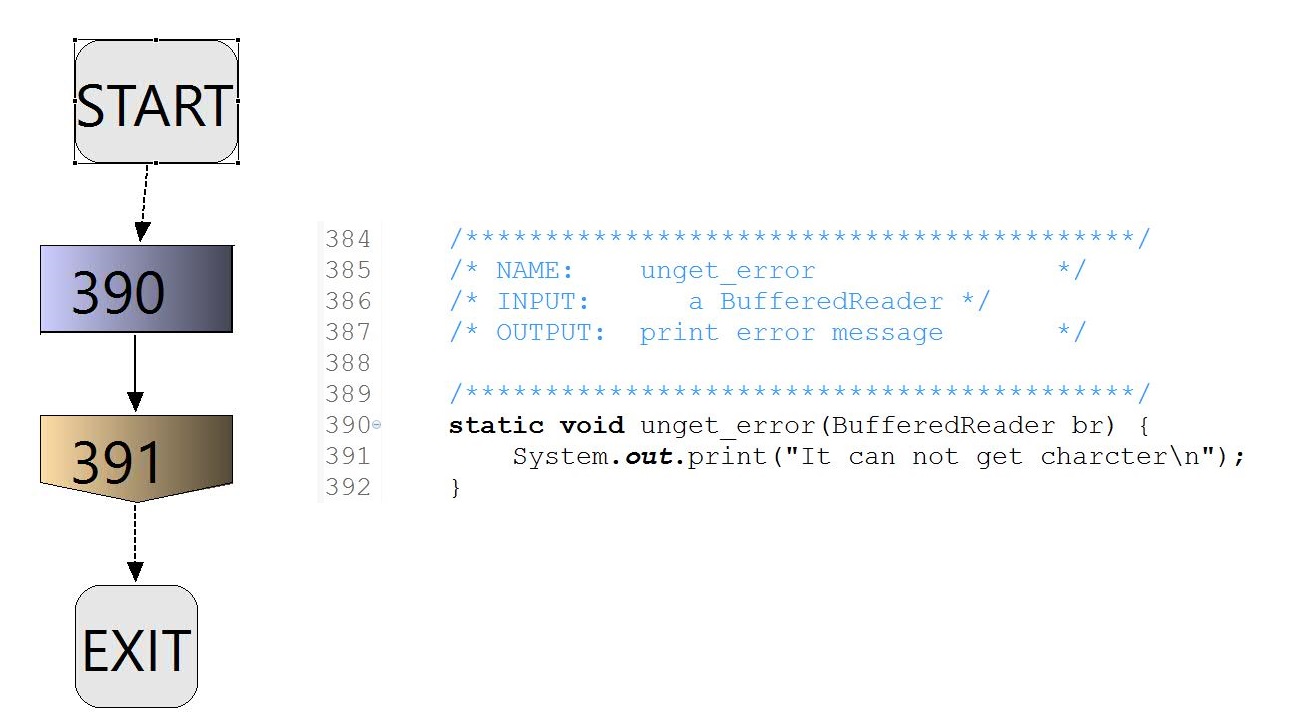


## **Bugs and test case**

|  |  |
| --- | --- |
| 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**

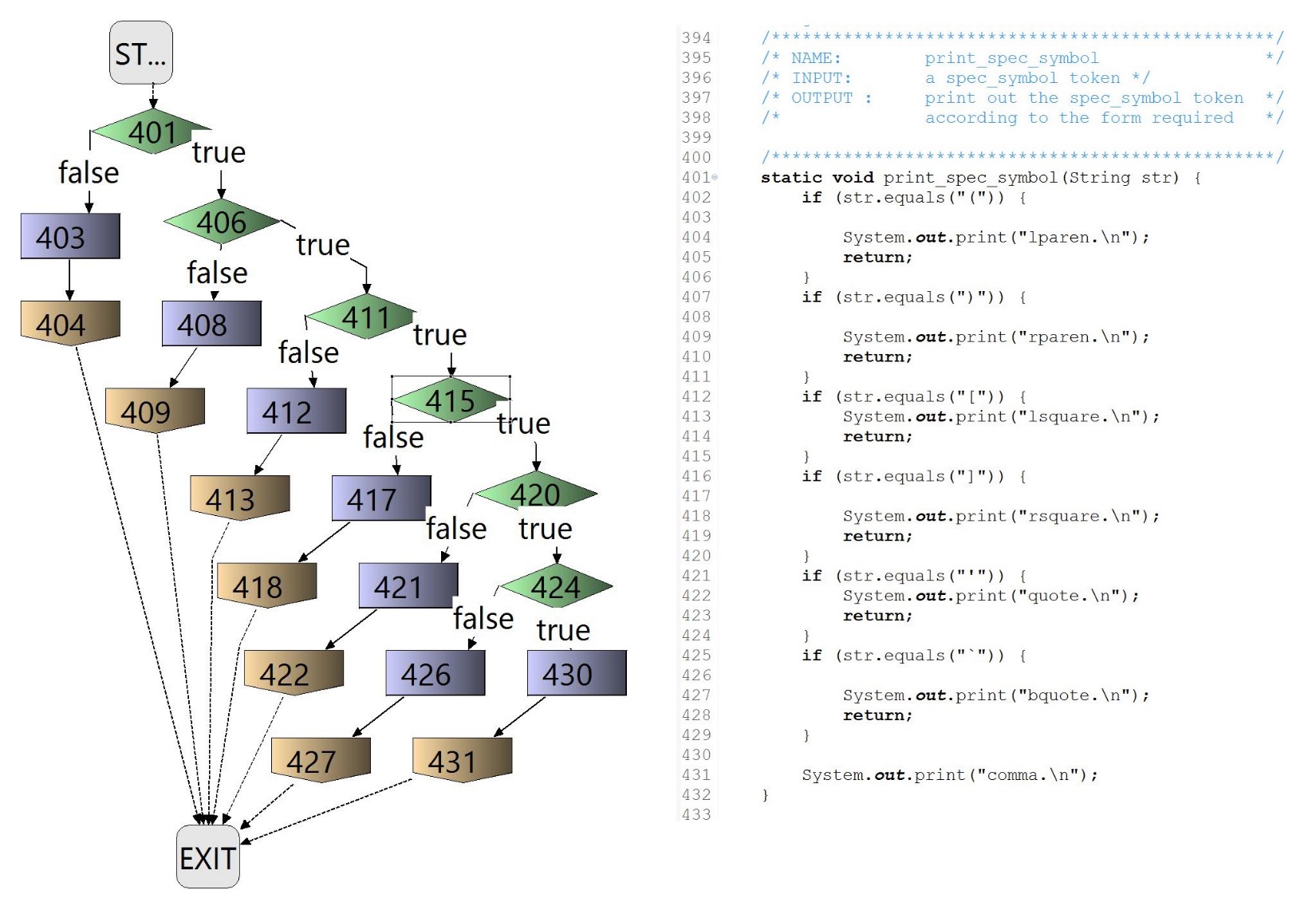


## **Bugs and Test Cases**

|  |  |
| --- | --- |
| 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**

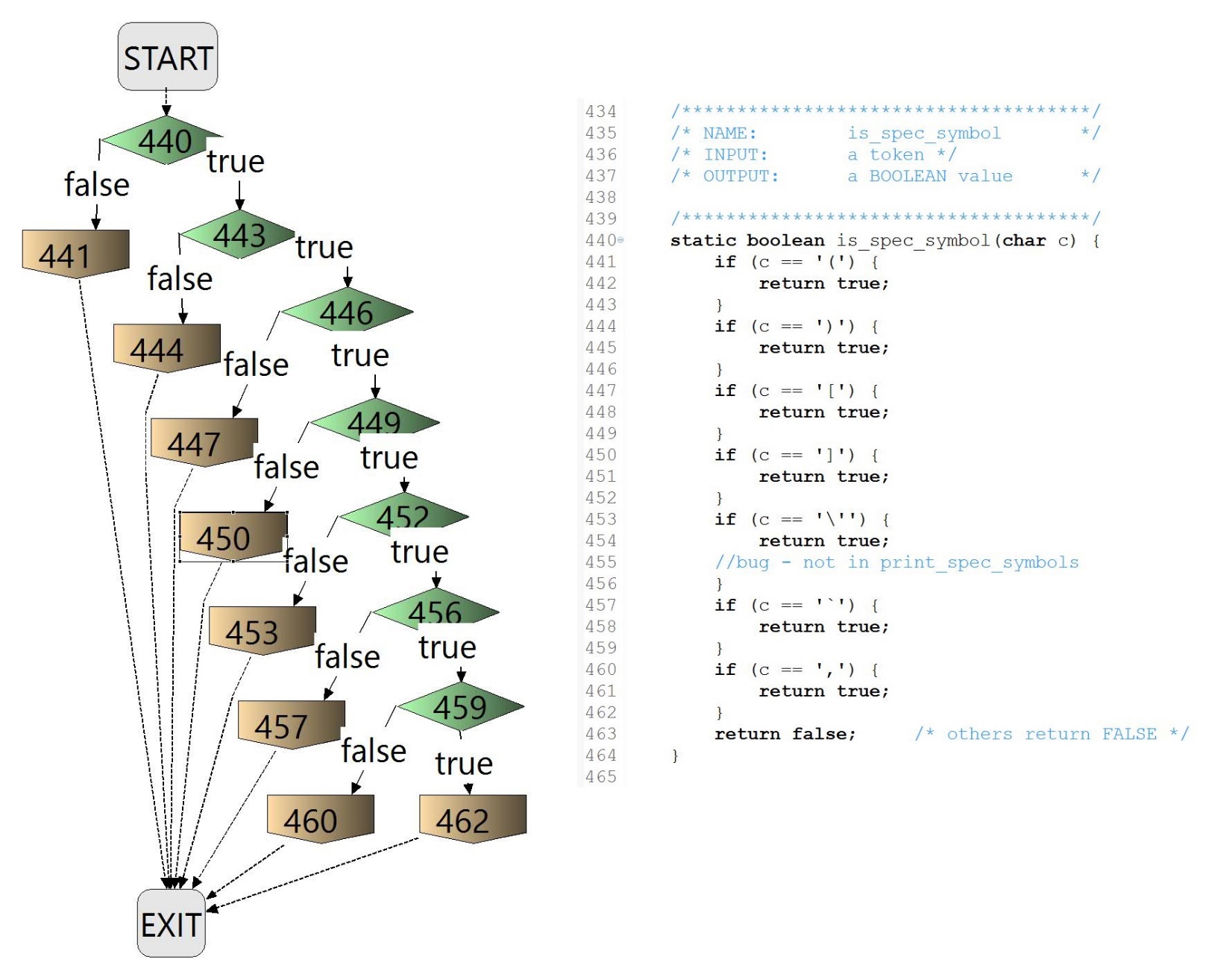


## **Bugs and test cases**

|  |  |
| --- | --- |
| 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**

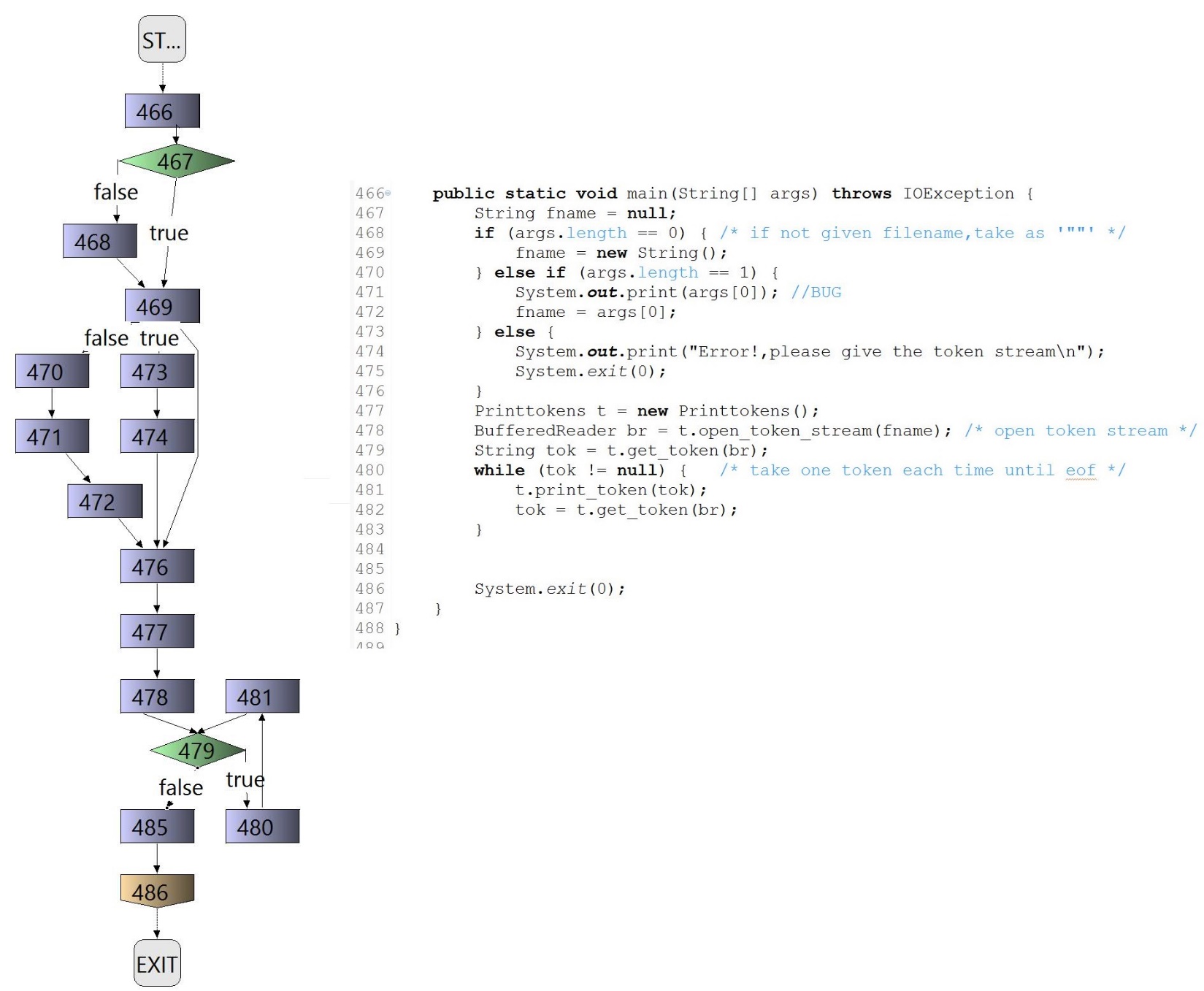


## Bugs and Test Cases

|  |  |
| --- | --- |
| 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



## **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 |
| @Test public void test\_main\_file\_as\_arg() throws IOException {  exit.expectSystemExitWithStatus(0);  for(String testFile: *TEST\_FILE\_LIST*) {  pt2.*main*(new String[]{testFile});  } } |
| @Test public void test\_main\_no\_args() throws IOException {  try {  pt2.*main*(new String[]{});  }  catch(NullPointerException e) {  // Expecting this to happen  } } |
| @Test public void test\_main\_too\_many\_args() throws IOException {  exit.expectSystemExitWithStatus(0);  pt2.*main*(new String[] {"testFile0.txt", "testFile0.txt", "testFile0.txt"}); } |