Ash Ezani
Jason Lim
Kimberly Tom
CS 362
3/15/19

Final Project Part B URLValidatorIncorrect

Onids:

Ash Ezani	wanahmaw
Jason Lim	limjas
Kimberly Tom	tomki

Total points: 80% of the total final project grade.
CS362-004
Final Project: Part-B
The final project is designed to check your cumulative understanding.
Notes:
• Use the URLValidatorInCorrect folder for Part-B.
• Submit one copy per group to Canvas (pdf) and the GitHub class repository (code).
• Add a comment in Canvas and give the URL of the student fork who submitted your project in GitHub (under final project).
• Write your group members details in the pdf files
• The code and documentation for the Apache Commons Validator is available in this link http://commons.apache.org/proper/commons-validator/
• The GitHub account is available at this link https://github.com/apache/commons-validator
Part-B, Testing URL Validator:
You are provided a buggy version of URLValidator and you will need to find as many bugs as you can. In Part-A, you were provided the current test framework that Apache commons team used to test URLValidator but you need to assume that those tests don't exist.
For this project, assume your team is a testing company and a client comes to you with a URLValidator implementation and asks for your help making it bug-free. You will only need to concentrate on isValid() method. Your task is to find the bugs, then find the failure causes and write up your findings.
Note: You don't need to fix any of the bugs. Developers will do it.
Use the various methodologies that you've learned to perform your testing:

Methodology:

1. Do manual testing. Call the isValid() method of URLValidator with different possible valid/invalid inputs. Document any failures that you find. (15 points)

Manual Testing:
To do the manual testing, we called isValid using different valid or invalid inputs.

Manual test batch 1:

URL	Expected Result	Actual Result
ww.google.com	false	false
http://google.com	true	true
gogle.c	false	False

Manual testing errors:

Manual test batch 2:
failures are in red

URL	Expected Result	Actual Result
http://255.255.255.255	true	true
http://255.255.255.255:65536	false	false
http://255.255.255.255:999999999999999999	false	false
http://255.255.255.255/test1	true	true
http://255.255.255.255/t123	true	true
http://255.255.255.255/..//file	false	false
http://255.255.255.255/t123?action=view	true	true
http://255.com	true	true
http/www.google.com	false	true

http:www.google.com	false	false
h3t://go.cc:65535/test1/file	true	true
http/go.cc:65535/test1/file	false	true
http://255.255.255.255:80	true	true
http://255.255.255.255:65535	true	true
http://255.255.255.255:0	true	true
http://go.1aa	false	false
http://aaa	false	false
h3t://www.google.com	true	true
http://1.2.3.4.	false	true

2. Do programming based testing. Write several unit test cases that implement various methods to test different URLs. Document any failures that you find. (20 points)

Programming Unit Testing:
We did unit testing and put the different portions such as scheme and query and did these small tests to make sure that they worked. We tested it as such:

Scheme	Expected	Result
http://	Pass	Fail
ftp://	Pass	Fail
mailto:	Pass	Pass
fake://	Fail	Pass
“” (blank)	Pass	Fail

Query	Expected	Result
?action=read	Pass	Pass
?name=ferret	Pass	Fail

?color=purple	Pass	Pass
“”	Fail	Fail
???Fakeaction	Fail	Fail

3. Do random testing. Come up with good input partitioning. Try to provide a varying set of inputs that partition the overall input set well. Document any failures that you find. (15 points)

Below are the failures out of 50 test cases. There are 19 failures out of 50 tests:

```
test case #1
3ht://255.255.255.255/happy3/?action=edit&mode=up
scheme is: 3ht://
scheme should be invalid
authority is: 255.255.255.255
authority should be valid
port is:
port should be valid
path is: /happy3/
path should be valid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED

test case #2
3ht://happy.asdf.doesntwork@$%%dontwork?action=view
scheme is: 3ht://
scheme should be invalid
authority is: happy.asdf.doesntwork
authority should be invalid
port is:
port should be valid
path is: @$%%dontwork
path should be invalid
query is: ?action=view
query should be valid
isValid returns that the url is true
TEST FAILED

test case #3
```

```
3ht://www.apple.comb1234/works5?action=edit&mode=up
scheme is: 3ht://
scheme should be invalid
authority is: www.apple.com
authority should be valid
port is: b1234
port should be invalid
path is: /works5
path should be valid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED

test case #12
3ht://invalid.url.httpbadport/hi//?action=edit&mode=up
scheme is: 3ht://
scheme should be invalid
authority is: invalid.url.http
authority should be invalid
port is: badport
port should be invalid
path is: /hi//
path should be invalid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED

test case #14
http/.bbb/$777?action=edit&mode=up
scheme is: http/
scheme should be invalid
authority is: .bbb
authority should be invalid
port is:
port should be valid
path is: /$777
path should be valid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED
```

```
test case #17
http/www.apple.combadport/works5?action=view
scheme is: http/
scheme should be invalid
authority is: www.apple.com
authority should be valid
port is: badport
port should be invalid
path is: /works5
path should be valid
query is: ?action=view
query should be valid
isValid returns that the url is true
TEST FAILED

test case #18
http/this.is.invalidbadport/happy3/
scheme is: http/
scheme should be invalid
authority is: this.is.invalid
authority should be invalid
port is: badport
port should be invalid
path is: /happy3/
path should be valid
query is:
query should be valid
isValid returns that the url is true
TEST FAILED

test case #25
3ht://www.apple.combadport/..?action=view
scheme is: 3ht://
scheme should be invalid
authority is: www.apple.com
authority should be valid
port is: badport
port should be invalid
path is: /..
path should be invalid
query is: ?action=view
query should be valid
isValid returns that the url is true
TEST FAILED


test case #28
http/com.asdf.www:0/works5?action=view
scheme is: http/
scheme should be invalid
authority is: com.asdf.www
authority should be invalid
port is: :0
port should be valid
path is: /works5
path should be valid
query is: ?action=view
query should be valid
isValid returns that the url is true
TEST FAILED


test case #30
http/com.asdf.www:0/$777?action=view
scheme is: http/
scheme should be invalid
authority is: com.asdf.www
authority should be invalid
port is: :0
port should be valid
path is: /$777
path should be valid
query is: ?action=view
query should be valid
isValid returns that the url is true
TEST FAILED


test case #36
3ht://com.asdf.wwwbadport/happy3/?action=edit&mode=up
scheme is: 3ht://
scheme should be invalid
authority is: com.asdf.www
authority should be invalid
```

```
port is: badport
port should be invalid
path is: /happy3/
path should be valid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED

test case #37
3ht://.bbbbadport/..?action=edit&mode=up
scheme is: 3ht://
scheme should be invalid
authority is: .bbb
authority should be invalid
port is: badport
port should be invalid
path is: /..
path should be invalid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED

test case #38
http/www.apple.comb1234/..?action=edit&mode=up
scheme is: http/
scheme should be invalid
authority is: www.apple.com
authority should be valid
port is: b1234
port should be invalid
path is: /..
path should be invalid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED

test case #39
http/this.is.invalid:0/..?action=view
scheme is: http/
scheme should be invalid
```

```
authority is: this.is.invalid
authority should be invalid
port is: :0
port should be valid
path is: /..
path should be invalid
query is: ?action=view
query should be valid
isValid returns that the url is true
TEST FAILED

test case #40
3ht://asdf.combadport/hi//?action=edit&mode=up
scheme is: 3ht://
scheme should be invalid
authority is: asdf.com
authority should be valid
port is: badport
port should be invalid
path is: /hi//
path should be invalid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED

test case #41
3ht://invalid.url.httpb1234/$777?action=view
scheme is: 3ht://
scheme should be invalid
authority is: invalid.url.http
authority should be invalid
port is: b1234
port should be invalid
path is: /$777
path should be valid
query is: ?action=view
query should be valid
isValid returns that the url is true
TEST FAILED

test case #42
http/invalid.url.httpbadport/hi//?action=edit&mode=up
```

```
scheme is: http/
scheme should be invalid
authority is: invalid.url.http
authority should be invalid
port is: badport
port should be invalid
path is: /hi//
path should be invalid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
TEST FAILED



test case #46
http/this.is.invalid:0/$777?action=view
scheme is: http/
scheme should be invalid
authority is: this.is.invalid
authority should be invalid
port is: :0
port should be valid
path is: /$777
path should be valid
query is: ?action=view
query should be valid
isValid returns that the url is true
TEST FAILED

test case #47
http/asdf.com/happy3/?action=edit&mode=up
scheme is: http/
scheme should be invalid
authority is: asdf.com
authority should be valid
port is:
port should be valid
path is: /happy3/
path should be valid
query is: ?action=edit&mode=up
query should be valid
isValid returns that the url is true
```

TEST FAILED

Random Testing/Partition Testing
4. Submit a report called ProjectPartB.pdf in Canvas that contains the following Sections: (40 points)
o Methodology Testing (15 points)

▪ Write in detail about each methodology you used for testing (manual, partition and programming based).
▪ Write in detail the test functions that implement each methodology.
▪ For manual testing, provide your URLs.
▪ For random testing, describe your partitions and list some of the URLs that represent each partition.

In our methodology we focused on each of the inputs for checking if a url is valid in URLValidator. For manual testing we focused on choosing valid and invalid inputs for the URLValidator and then determining what our expected result is and what the actual result was. If the actual result differed from our expected result we would consider examining the test further and determining if it is truly a bug. The manual tests would test a full url while our partitioned test would test each part or breakup a section such as the initial scheme or the body of the url and the ending. The partitioning would assist in trying to figure out which part of the URL validation was invalid. We would then use random testing to figure out the combinations of each of the partitions and whether or not a correct full URL worked when the partitions were combined together as one. Whenever we encountered a bug we would try further tests to examine if there could be other sections of the code that were buggy as well. We would try similar tests to determine the full extent of the bug.
The unit tests or programming based tests would take each of the url parts from the partitionings and then create an array for the full url. In the loop each of the components from the partitions would be combined to create a valid URL. Then this url would be tested with all the other combinations of the partitions and test the isValid() function. This allowed us to “divide and conquer” and isolate where the bug occurred as well as provide a case for debugging and seeing where the error occurred in the particular partition. Each partition for the random test would test things for example in scheme the https:// or ftp:// or mailto:

Overall, our methodology for for the URL validator testing came in the form of using full urls for manual testing, creating partitioned portions of the url in tests, and combining these partitions in programming based tests.

The following are some of the URLs for manual test:

```
 System.out.println("\nCorrect URL:");
     System.out.println("http://google.com");
```

```
System.out.println(urlVal.isValid("http://google.com"));
System.out.println("\nIncorrect URL:");
System.out.println("ww.google.com");
System.out.println(urlVal.isValid("ww.google.com"));
System.out.println("gogle.c");
System.out.println(urlVal.isValid("gogle.c"));

System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255");
System.out.println(urlVal.isValid("http://255.255.255.255"));


System.out.println("\nIncorrect URL:");
System.out.println("http://255.255.255.255:65536");
System.out.println(urlVal.isValid("http://255.255.255.255:65536"));


System.out.println("\nIncorrect URL:");
System.out.println("http://255.255.255.255:999999999999999999");
System.out.println(urlVal.isValid("http://255.255.255.255:999999999999999999"));


System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255/test1");
System.out.println(urlVal.isValid("http://255.255.255.255/test1"));

System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255/t123");
System.out.println(urlVal.isValid("http://255.255.255.255/t123"));

System.out.println("\nInorrect URL:");
System.out.println("http://255.255.255.255/..//file");
System.out.println(urlVal.isValid("http://255.255.255.255/..//file"));
System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255/t123?action=view");
System.out.println(urlVal.isValid("http://255.255.255.255/t123?action=view"));

System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255/?action=edit&mode=up");
System.out.println(urlVal.isValid("http://255.255.255.255/?action=edit&mode=up"));

System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255/");
System.out.println(urlVal.isValid("http://255.255.255.255/"));


System.out.println("\nCorrect URL:");
System.out.println("http://255.com");
System.out.println(urlVal.isValid("http://255.com"));

System.out.println("\nIncorrect URL:");
System.out.println("http/www.google.com");
System.out.println(urlVal.isValid("http/www.google.com"));

System.out.println("\nIncorrect URL:");
System.out.println("http:www.google.com");
System.out.println(urlVal.isValid("http:www.google.com"));

System.out.println("\nCorrect URL:");
System.out.println("h3t://go.cc:65535/test1/file");
System.out.println(urlVal.isValid("h3t://go.cc:65535/test1/file"));

System.out.println("\nIncorrect URL:");
System.out.println("http/go.cc:65535/test1/file");
System.out.println(urlVal.isValid("http/go.cc:65535/test1/file"));


System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255:80");
System.out.println(urlVal.isValid("http://255.255.255.255:80"));

System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255:65535");
System.out.println(urlVal.isValid("http://255.255.255.255:65535"));

System.out.println("\nCorrect URL:");
System.out.println("http://255.255.255.255:0");
System.out.println(urlVal.isValid("http://255.255.255.255:0"));


System.out.println("\nIncorrect URL:");
System.out.println("http://go.1aa");
System.out.println(urlVal.isValid("http://go.1aa"));


System.out.println("\nIncorrect URL:");
```

```
      System.out.println("http://aaa");
      System.out.println(urlVal.isValid("http://aaa"));

      System.out.println("\nCorrect URL:");
      System.out.println("h3t://www.google.com");
      System.out.println(urlVal.isValid("h3t://www.google.com"));

      System.out.println("\nIncorrect URL:");
      System.out.println("http://1.2.3.4.");
      System.out.println(urlVal.isValid("http://1.2.3.4."));
```

For the random test we would also take values from the url.txt file which is on github and use all the values from it as a partition. See github for the url.txt. Additionally, in some of the random tests, we created a combination from different strings of parts of a url, combined them together, and put them into one url to test.
See example below:

```
     StringBuilder urlBuilder = new StringBuilder();
       UrlValidator urlVal;
       String[] scheme = {"http://","ftp://",
                          "h3t://",
                          ""};

       String[] authority = {"www.google.com",
                             "go.com",
                             "go.au",
                             "123.com",
                             "255.255.255.255"};

       String[] port = {":10",
                        ":65535",
                        ":0",
                        ""};

       String[] path = {"/yay",
                        "/yay1",
                        "/1yay"
                         "/$17",
                        "/yay/",
                        "/yay/yatta"};

       String[] query = {"?cat=meow",
                         "?cat=meow&claws=out",
                         ""};
```

o Bug Report (10 points)
▪ Write a bug report for each of the bugs you found.
• Describe the failure.
• Describe how you found it terms of the test case that detected it.
• Describe the cause of the failure. Explain what part of the code is causing it and provide a screenshot of the faulty code.

Title: incorrect URL is showing as valid after calling isValid

Class: minor bug

Date: 3/15/19
Reported By: Kimberly Tom
Email: tomki@oregonstate.edu

Product: UrlValidator.java   Version: 2
Platform: Java   Version: jdk 1.8
Browser: IntelliJ   Version: 2018.3.4
URL:

Is it reproducible: Yes

Description
===========
http/www.google.com should be invalid url, but isValid returns true for this url in manual test. I looked at the other manual tests that included www.google.com and http/ and it appears there is an error with the http/ portion which is part of the scheme. http/ with a valid remaining url is coming back as true when it should be false.

In line 311, an invalid scheme is returning true, when it should return false.

Steps to Produce/Reproduce

I first ran isValid with http/www.google.com, which returns true. If I run isValid with scheme of http/ then it still returns true if the rest of the url is valid for multiple urls.

Expected Results

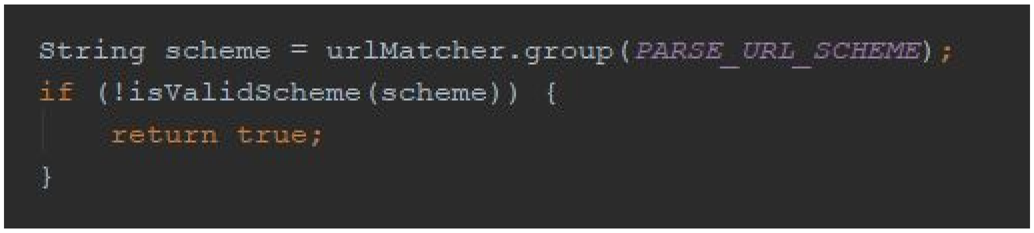
----
running isValid with http/www.google.com should return the url as false

Actual Results
----

running isValid with http/www.google.com returns the url as true

Workarounds
----
don’t call isValid with a scheme of http/ or 3ht://

Attachments


```
String scheme = urlMatcher.group(PARSE_URL_SCHEME);
if (!isValidScheme(scheme)) {
    return true;
}
```

Scheme	Expected	Result
http://	**Pass**	**Fail**
ftp://	**Pass**	**Fail**
mailto:	Pass	Pass
fake://	**Fail**	**Pass**
“” (blank)	**Pass**	**Fail**

Query	Expected	Result
?action=read	Pass	Pass
?name=ferret	**Pass**	**Fail**

?color=purple	Pass	Pass
“”	Fail	Fail
???Fakeaction	Fail	Fail

We bolded the ones which were bugs:

Bugs in Scheme test:
http://, ftp://, and fake:// were all bugs. The http:// and ftp:// were bugs since they were supposed to pass since they are valid schemes but are not coming as true in the output.
Noverse Bug Reporting Template
==============================

Title: Bug for Scheme

Class:
e.g. "Minor Bug"

Date: 3/15/19
Reported By: Jason Lim
Email: limjas@oregonstate.edu

Product: UrlValidator.java Version: 2
Platform: Java Version: jdk 1.8
Browser: Eclipse Version: Neon
URL: http://, ftp://, and fake://

Is it reproducible: **Yes** / Occasionally / One Time / No

Description
===========
Description: In isValidScheme line 310 is true but this is supposed to be false. I looked into it while in debug mode and found that perhaps the bugs that we introduced in lines 310 and 316 could be issues:

Steps to Produce/Reproduce

Run program with the testIsValid in UrlValidatorTest.java

Expected Results

----
http:// - pass
ftp:// - pass
fake:// - fail

Actual Results
----
http:// - fail
ftp:// - fail
fake:// - pass

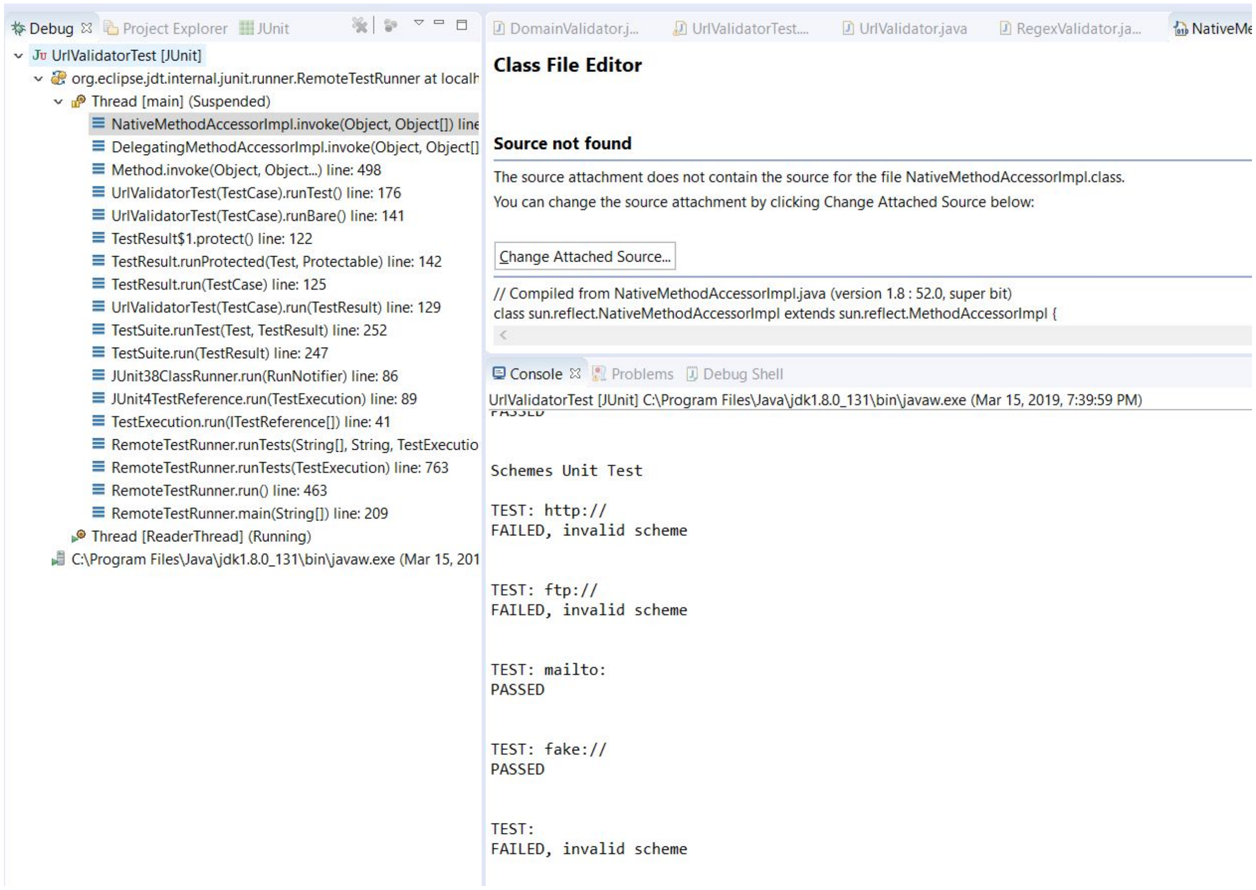
Workarounds

We can go to the actual lines in the debugger to see where this code was having problems.

We can also go to the function isValidScheme directly as below.

If we resolve line 310 to false and 366 the “:” instead of “::” this may work to cause the items to have no issues with the isValidScheme. I found that it was logical to look into this since these were the two bugs we deliberately introduced into the code and its related to scheme unit test so this could be a possible recommendation for a solution.

Attachments



```
UrlValidatorTest [JUnit]
  org.eclipse.jdt.internal.junit.runner.RemoteTestRunner at localh
    Thread [main] (Suspended)
      NativeMethodAccessorImpl.invoke(Object, Object[]) line
      DelegatingMethodAccessorImpl.invoke(Object, Object[]
      Method.invoke(Object, Object...) line: 498
      UrlValidatorTest(TestCase).runTest() line: 176
      UrlValidatorTest(TestCase).runBare() line: 141
      TestResult$1.protect() line: 122
      TestResult.runProtected(Test, Protectable) line: 142
      TestResult.run(TestCase) line: 125
      UrlValidatorTest(TestCase).run(TestResult) line: 129
      TestSuite.runTest(Test, TestResult) line: 252
      TestSuite.run(TestResult) line: 247
      JUnit38ClassRunner.run(RunNotifier) line: 86
      JUnit4TestReference.run(TestExecution) line: 89
      TestExecution.run(ITestReference[]) line: 41
      RemoteTestRunner.runTests(String[], String, TestExecutio
      RemoteTestRunner.runTests(TestExecution) line: 763
      RemoteTestRunner.run() line: 463
      RemoteTestRunner.main(String[]) line: 209
    Thread [ReaderThread] (Running)
  C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (Mar 15, 201
```

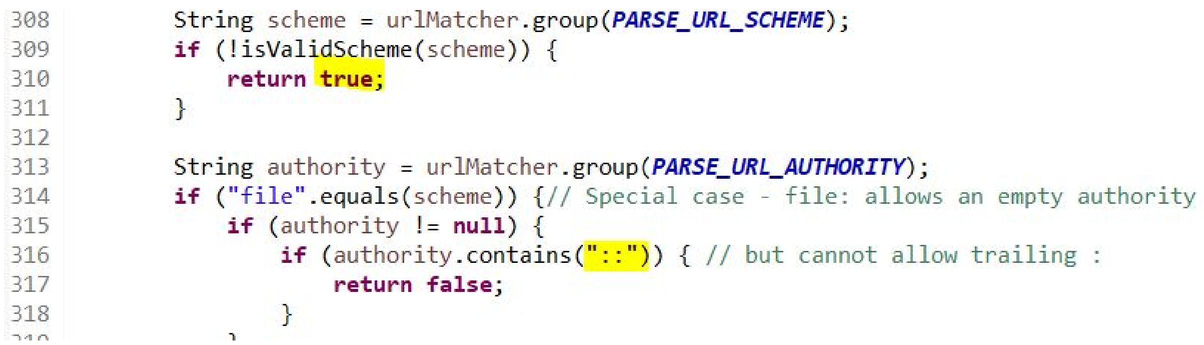
Class File Editor

Source not found

The source attachment does not contain the source for the file NativeMethodAccessorImpl.class.
You can change the source attachment by clicking Change Attached Source below:

```
// Compiled from NativeMethodAccessorImpl.java (version 1.8 : 52.0, super bit)
class sun.reflect.NativeMethodAccessorImpl extends sun.reflect.MethodAccessorImpl {
```

```
UrlValidatorTest [JUnit] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (Mar 15, 2019, 7:39:59 PM)
PASSED

Schemes Unit Test

TEST: http://
FAILED, invalid scheme


TEST: ftp://
FAILED, invalid scheme


TEST: mailto:
PASSED


TEST: fake://
PASSED


TEST: 
FAILED, invalid scheme
```



```
        String scheme = urlMatcher.group(PARSE_URL_SCHEME);
        if (!isValidScheme(scheme)) {
            return true;
        }

        String authority = urlMatcher.group(PARSE_URL_AUTHORITY);
        if ("file".equals(scheme)) {// Special case - file: allows an empty authority
            if (authority != null) {
                if (authority.contains("::")) { // but cannot allow trailing :
                    return false;
                }
```

Other Information

Title: Bug in authority
Class: “Minor Bug”

Date: 3/17/19
Reported By: Ash Ezani
Email: wanahmaw@oregonstate.edu

Product: UrlValidator.java Version: 2
Platform: Java Version: jdk 1.8
Browser: Eclipse Version: Neon
URL: http://www.google.com:

Is it reproducible: **Yes** / Occasionally / One Time / No

Description
===========

Description of bug: http://www.google.com: seems to be an an authority issue. We didn't introduce other URL parts to isolate the bug. According to URL syntax standards, we cannot leave a trailing colon symbol in the authority.

Steps to Produce/Reproduce

Run isolated testManualTest() function in UrlValidatorTest.java, picture below
1. Create urlVal with only ALLOW ALL SCHEMES turned on
2. Use JUnit assertFalse for urlVal.isValid("http://www.google.com:")

Expected Results

Calling isValid on http://www.google.com: returns FALSE

Actual Results

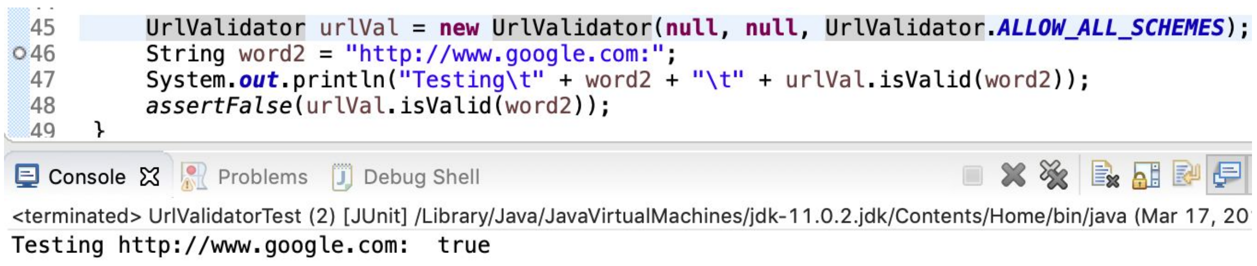
Calling isValid on http://www.google.com: returns TRUE

Workarounds

Set a breakpoint where isValid is run for this URL. Step into code and step over until you see the authority check on line 317 in UrlValidator.java. We see it checks for 2 trailing colons. It should just be 1 trailing colon. Replacing this fixes our bug. Pictures below.

Attachments

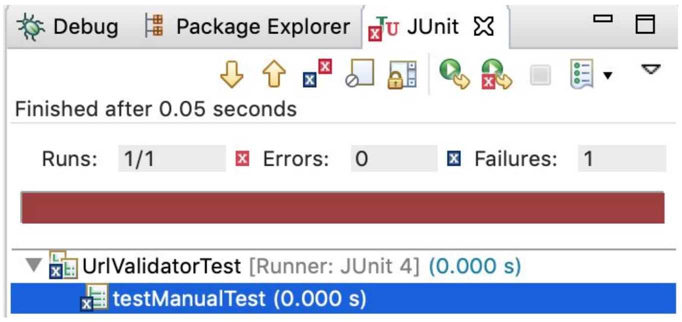
Code to reproduce error



```
    UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
    String word2 = "http://www.google.com:";
    System.out.println("Testing\t" + word2 + "\t" + urlVal.isValid(word2));
    assertFalse(urlVal.isValid(word2));
}
```

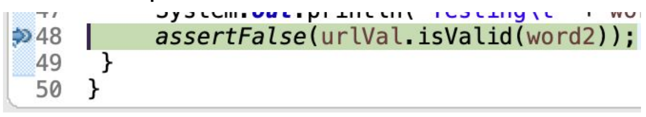
```
<terminated> UrlValidatorTest (2) [JUnit] /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/java (Mar 17, 20
Testing http://www.google.com:  true
```

JUnit caught failed assertion



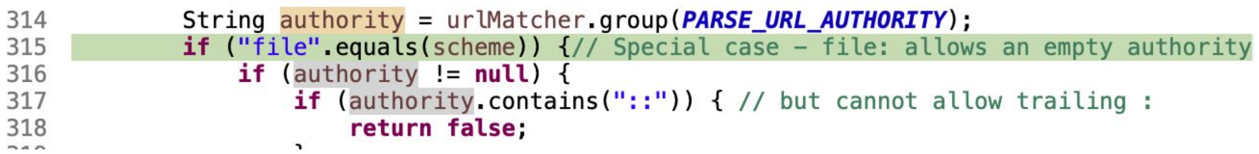
```
Finished after 0.05 seconds

Runs: 1/1    Errors: 0    Failures: 1

UrlValidatorTest [Runner: JUnit 4] (0.000 s)
    testManualTest (0.000 s)
```

Set a breakpoint



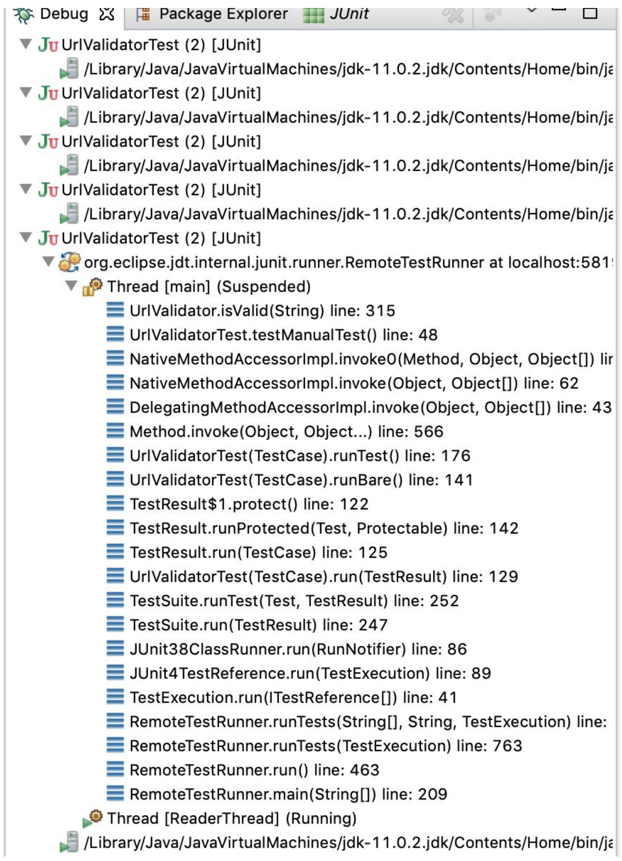
```
    assertFalse(urlVal.isValid(word2));
  }
}
```

Culprit on line 317. Should check for 1 single trailing colon instead of 2



```
314         String authority = urlMatcher.group(PARSE_URL_AUTHORITY);
315         if ("file".equals(scheme)) {// Special case - file: allows an empty authority
316             if (authority != null) {
317                 if (authority.contains("::")) { // but cannot allow trailing :
318                     return false;
```

Debug trace



```
Debug ⌧   Package Explorer   JUnit
▼ UrlValidatorTest (2) [JUnit]
    /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/ja
▼ UrlValidatorTest (2) [JUnit]
    /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/ja
▼ UrlValidatorTest (2) [JUnit]
    /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/ja
▼ UrlValidatorTest (2) [JUnit]
    /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/ja
▼ UrlValidatorTest (2) [JUnit]
  ▼ org.eclipse.jdt.internal.junit.runner.RemoteTestRunner at localhost:581
    ▼ Thread [main] (Suspended)
        UrlValidator.isValid(String) line: 315
        UrlValidatorTest.testManualTest() line: 48
        NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) lir
        NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
        DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
        Method.invoke(Object, Object...) line: 566
        UrlValidatorTest(TestCase).runTest() line: 176
        UrlValidatorTest(TestCase).runBare() line: 141
        TestResult$1.protect() line: 122
        TestResult.runProtected(Test, Protectable) line: 142
        TestResult.run(TestCase) line: 125
        UrlValidatorTest(TestCase).run(TestResult) line: 129
        TestSuite.runTest(Test, TestResult) line: 252
        TestSuite.run(TestResult) line: 247
        JUnit38ClassRunner.run(RunNotifier) line: 86
        JUnit4TestReference.run(TestExecution) line: 89
        TestExecution.run(ITestReference[]) line: 41
        RemoteTestRunner.runTests(String[], String, TestExecution) line:
        RemoteTestRunner.runTests(TestExecution) line: 763
        RemoteTestRunner.run() line: 463
        RemoteTestRunner.main(String[]) line: 209
      Thread [ReaderThread] (Running)
    /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/ja
```

==========================

Title: Query Bug in UrlValidatorTest
Class: “Minor Bug”

Date: 3/15/19
Reported By: Jason Lim
Email: limjas@oregonstate.edu

Product: UrlValidator.java Version: 2
Platform: Java Version: jdk 1.8
Browser: Eclipse Version: Neon
URL: ?name=ferret

Is it reproducible: **Yes** / Occasionally / One Time / No

Description
===========

Description of bug: ?name=ferret was not registering as pass value even though this is a valid query.

Steps to Produce/Reproduce

Run testIsValid2 function in UrlValidatorTest.java

Expected Results

?name=ferret to return pass

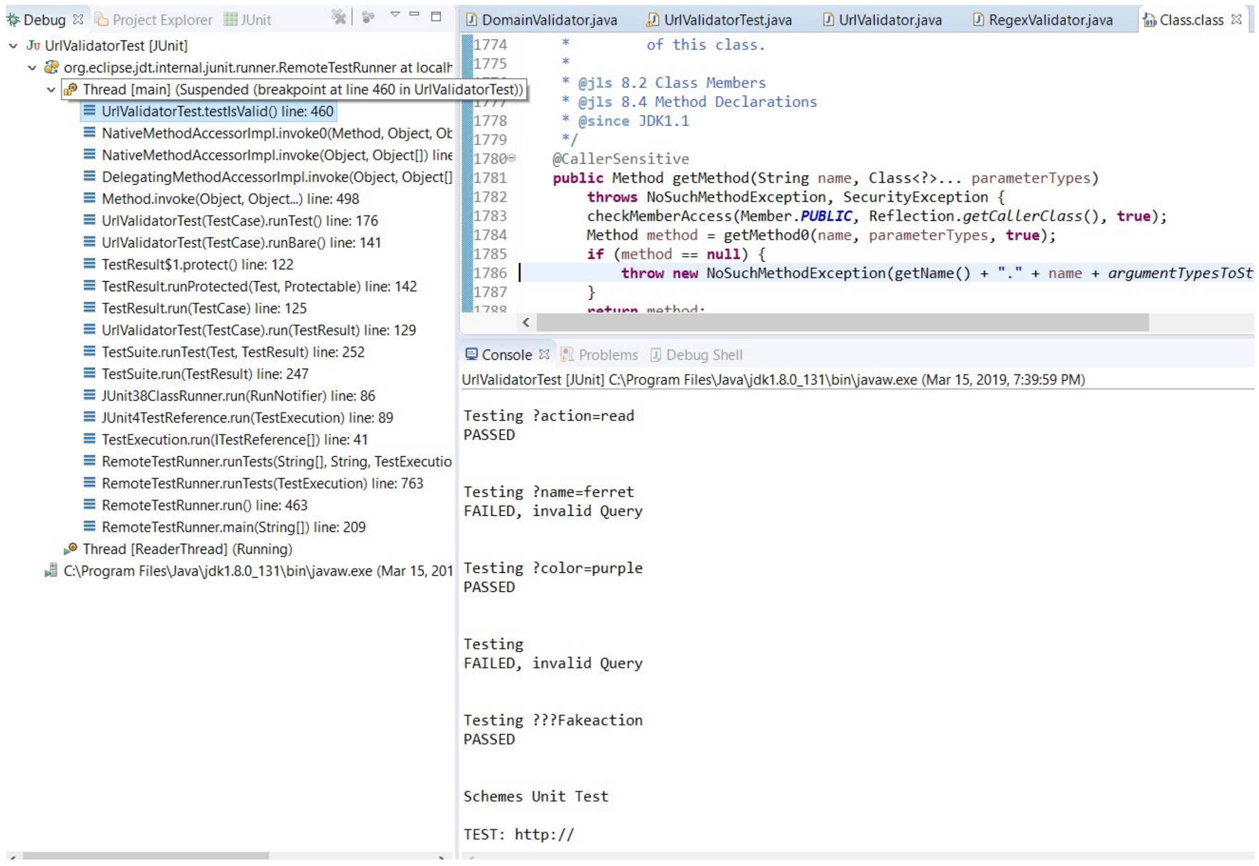
Actual Results

Returns fail

Workarounds

Tried to debug through this and the left hand side provided the list of functions to see to figure out where the bug is. Suggestions to fix this bug would be to look into isValidQuery and check to see if the boolean values in the source code for UrlValidator.java in lines 475-478 are correct.

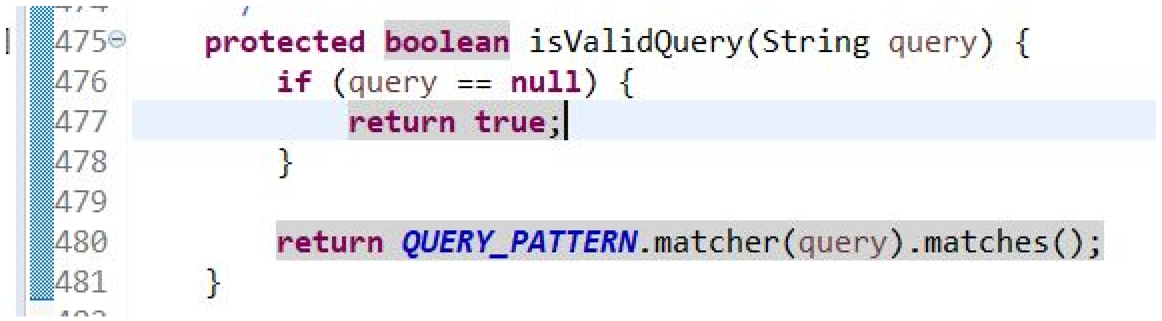
Attachments



```
Debug  Project Explorer  JUnit
UrlValidatorTest [JUnit]
  org.eclipse.jdt.internal.junit.runner.RemoteTestRunner at localh
    Thread [main] (Suspended (breakpoint at line 460 in UrlValidatorTest))
      UrlValidatorTest.testIsValid() line: 460
      NativeMethodAccessorImpl.invoke0(Method, Object, Ob
      NativeMethodAccessorImpl.invoke(Object, Object[]) line
      DelegatingMethodAccessorImpl.invoke(Object, Object[]
      Method.invoke(Object, Object...) line: 498
      UrlValidatorTest(TestCase).runTest() line: 176
      UrlValidatorTest(TestCase).runBare() line: 141
      TestResult$1.protect() line: 122
      TestResult.runProtected(Test, Protectable) line: 142
      TestResult.run(TestCase) line: 125
      UrlValidatorTest(TestCase).run(TestResult) line: 129
      TestSuite.runTest(Test, TestResult) line: 252
      TestSuite.run(TestResult) line: 247
      JUnit38ClassRunner.run(RunNotifier) line: 86
      JUnit4TestReference.run(TestExecution) line: 89
      TestExecution.run(ITestReference[]) line: 41
      RemoteTestRunner.runTests(String[], String, TestExecutio
      RemoteTestRunner.runTests(TestExecution) line: 763
      RemoteTestRunner.run() line: 463
      RemoteTestRunner.main(String[]) line: 209
    Thread [ReaderThread] (Running)
  C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (Mar 15, 201
```

```
DomainValidator.java  UrlValidatorTest.java  UrlValidator.java  RegexValidator.java  Class.class
     *         of this class.
     *
     * @jls 8.2 Class Members
     * @jls 8.4 Method Declarations
     * @since JDK1.1
     */
    @CallerSensitive
    public Method getMethod(String name, Class<?>... parameterTypes)
        throws NoSuchMethodException, SecurityException {
        checkMemberAccess(Member.PUBLIC, Reflection.getCallerClass(), true);
        Method method = getMethod0(name, parameterTypes, true);
        if (method == null) {
            throw new NoSuchMethodException(getName() + "." + name + argumentTypesToSt
        }
        return method;
```

```
Console  Problems  Debug Shell
UrlValidatorTest [JUnit] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (Mar 15, 2019, 7:39:59 PM)

Testing ?action=read
PASSED


Testing ?name=ferret
FAILED, invalid Query


Testing ?color=purple
PASSED


Testing
FAILED, invalid Query


Testing ???Fakeaction
PASSED


Schemes Unit Test

TEST: http://
```

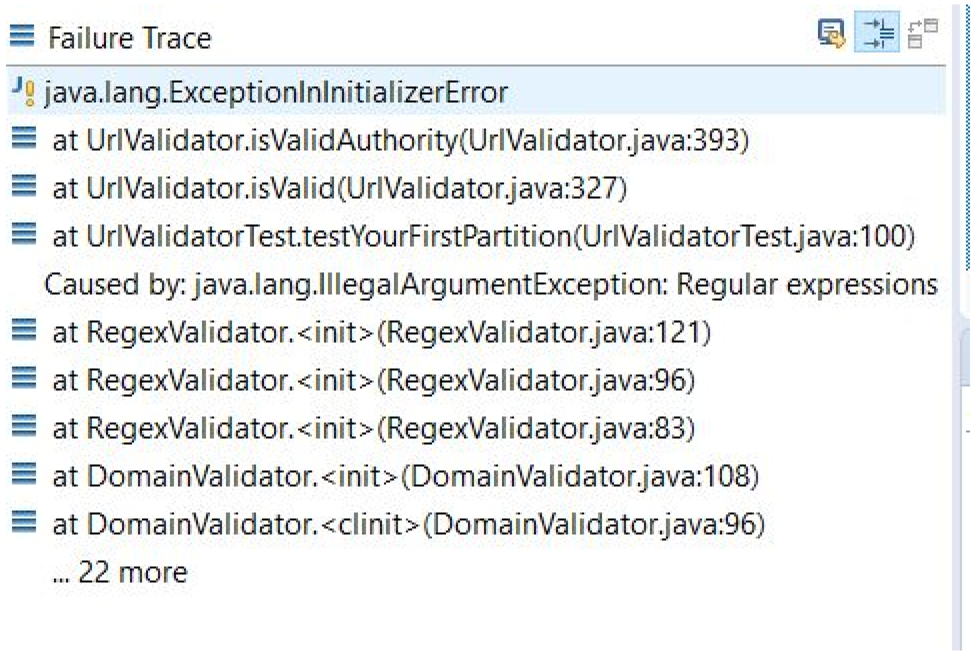


```
    protected boolean isValidQuery(String query) {
        if (query == null) {
            return true;
        }

        return QUERY_PATTERN.matcher(query).matches();
    }
```

Other Information
----

o Debugging (10 points)
▪ When you find a failure, debug it using Eclipse/IntelliJ debugger or any other tool and find its cause.
▪ Did you use any of Agan's principle in debugging URLValidator?
▪ Describe the failure and your debugging details for each bug. Provide the line numbers and file names where the failure manifested itself.

When we found failures, we debugged it using the Eclipse debugger to find its cause.
We analyzed the failure trace to see if there were any failures:

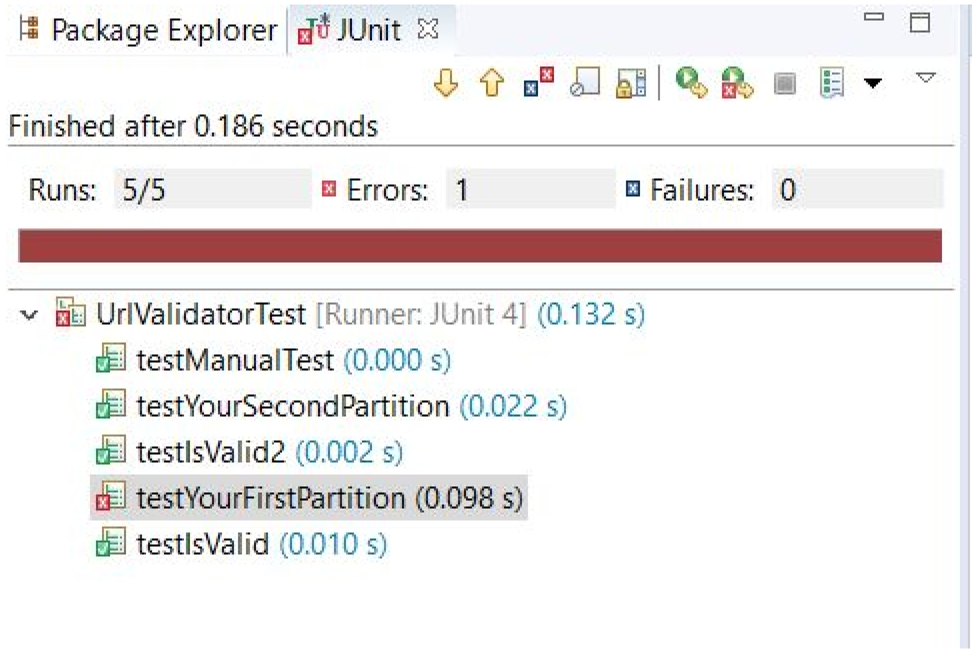


Failure Trace

```
java.lang.ExceptionInInitializerError
at UrlValidator.isValidAuthority(UrlValidator.java:393)
at UrlValidator.isValid(UrlValidator.java:327)
at UrlValidatorTest.testYourFirstPartition(UrlValidatorTest.java:100)
Caused by: java.lang.IllegalArgumentException: Regular expressions
at RegexValidator.<init>(RegexValidator.java:121)
at RegexValidator.<init>(RegexValidator.java:96)
at RegexValidator.<init>(RegexValidator.java:83)
at DomainValidator.<init>(DomainValidator.java:108)
at DomainValidator.<clinit>(DomainValidator.java:96)
... 22 more
```

Double clicking on each of these failure traces would let us go to which .java file had the issue and we would debug as such.

We would also make sure that our JUnit testing had no errors in the program files that were run:



Package Explorer | JUnit

```
Finished after 0.186 seconds

Runs: 5/5     Errors: 1     Failures: 0

UrlValidatorTest [Runner: JUnit 4] (0.132 s)
    testManualTest (0.000 s)
    testYourSecondPartition (0.022 s)
    testIsValid2 (0.002 s)
    testYourFirstPartition (0.098 s)
    testIsValid (0.010 s)
```

In this case there is something wrong with the first Partition so we would double click the testYourFirstPartition and see what is wrong.

Double clicking the function shows us that in line 100 of our code, this particular line:

```
                    if (!urlVal.isValid(finalUrl)) {
```

Seems to have something wrong with it when passing it the values of our first partition.

We had many problems with the files after introducing the bugs so we had to check each of the portions of the side bars with the Junit tests and also the console output. When we had an incorrect console output we would double click the error message which would take us to the line number which had the issue. We would use these facts to debug and fix up the code so that we had the values and output desired through our tests. Since some tests were initially created wrong, we used the debugger to provide a better test and improvements on our overall output. Many of these errors occured in the UrlValidatorTest.java file but some also occurred in the UrlValidator.java file after introducing the 2 bugs.

We also had issues with the data input output function test where we would use a .txt file in the same repository and check the file to make sure the url's are correct in it. One of our members debugged the issue of not placing the url.txt file in the correct repository through this method.

We also had some issues with the manual test where some of the items were not being printed out to the console. We ran some printf statements to help debug these issues.

Using Agan's Principle:

We read through David J. Agan's text in https://dwheeler.com/essays/debugging-agans.html and followed the following principles.

1. **Understand the system**: Read the manual, read everything in depth, know the fundamentals, know the road map, understand your tools, and look up the details.
2. **Make it fail**: Do it again, start at the beginning, stimulate the failure, don't simulate the failure, find the uncontrolled condition that makes it intermittent, record everything and find the signature of intermittent bugs, don't trust statistics too much, know that "that" *can* happen, and never throw away a debugging tool.
3. **Quit thinking and look** (get data first, don't just do complicated repairs based on guessing): See the failure, see the details, build instrumentation in, add instrumentation on, don't be afraid to dive in, watch out for Heisenberg, and guess only to focus the search.
4. **Divide and conquer**: Narrow the search with successive approximation, get the range, determine which side of the bug you're on, use easy-to-spot test patterns, start with the bad, fix the bugs you know about, and fix the noise first.
5. **Change one thing at a time**: Isolate the key factor, grab the brass bar with both hands (understand what's wrong before fixing), change one test at a time, compare it with a good one, and determine what you changed since the last time it worked.

6. **Keep an audit trail**: Write down what you did in what order and what happened as a result, understand that any detail could be the important one, correlate events, understand that audit trails for design are also good for testing, and write it down!
7. **Check the plug**: Question your assumptions, start at the beginning, and test the tool.
8. **Get a fresh view**: Ask for fresh insights (just explaining the problem to a mannequin may help!), tap expertise, listen to the voice of experience, know that help is all around you, don't be proud, report symptoms (not theories), and realize that you don't have to be sure.
9. **If you didn't fix it, it ain't fixed**: Check that it's really fixed, check that it's really your fix that fixed it, know that it never just goes away by itself, fix the cause, and fix the process

We used Agan's principle when debugging our code. For example, for manual testing, I noticed that when a scheme was invalid and the rest of the url was valid, it was showing as a valid url. I made it fail by trying different combinations of valid authority, port, path, and query with that same query, and it was still failing. This is the "Make it fail" portion of Agan's principles. For the random testing, I used Quit thinking and look principle. I used a random number generator to put together random parts of the url. I generated 50 urls and could clearly see that my tests were failing only when isValid is returning true. It was never failing when isValid was returning false.

We initially read the manual or guidelines of the project and made sure we understood our purpose. As a "testing company" we were hired on the spot to make sure that we execute great quality control of the software so that there are no bugs. This was our goal for the URL validator project B. We were given an incorrect or buggy file where we used unit testing, random/partitioned testing, as well as manual testing and had to find out if there were any bugs. If there were bugs then we would write them in a report which would be read by the developers and fixed. We recorded every test in our test cases and did multiple batches and made sure that our tests targeted ways to make the code fail. By keeping an audit trail or what we did in order in our report, we knew what we tested, what partitions we had, and how much coverage our functions would test based on our URLValidatorTest.java file. We also collaborated with one another to get a fresh view as to further input on how we should test our client's code thoroughly and effectively.

o Team Work (5 points)
▪ Write about how worked as a team? How did you divide your work? How did you collaborate? List the contributions of each of your team members.

We worked together as a team by fully communicating through messaging, google meeting, and scheduling time and deadlines to finish parts of Project B. We divided the work by looking at the easiest section which was manual testing and splitting that equally amongst us and the last three functions: test first partition, test second partition, and test is valid. We collaborated mostly through slack as well as through meetings and used Google Docs to simultaneously work on the project at once. For sharing code, we would use github as well as dropbox to share what code we had and updated. We also used slack to paste the code or functions into our group's slack channel. We would discuss significant issues through voice calls on google meets while minor issues would be worked on via messaging. We would also use github to commit, pull, and push our files.

Additionally in terms of our slack conversations, we analyzed and worked into how we should delve into the partitioning and unit testing piece and shared with each other many resources on what exactly is partition or unit testing. We were able to come away with expanded knowledge about partitioning and unit testing because of this.

Ash contributed by doing his part for the random function and wrote the function portion of the report as well as did the manual testing. Additionally, Ash helped Jason during the meeting by resolving an issue with Intellij and introducing how to use Eclipse for Project B so that Jason could get started on the project B for his part. Ash also contributed to the github with changes and suggested the 2 bugs for the code.

Jason contributed by hosting the meeting as well as making sure the project's progress was communicated across the team members and did the first partition, the project report for the partition, and submission of the assignment via github. Jason also wrote the methodology, team work, as well as the programming unit test portion. He also hosted the github repository and put the team members as collaborators to enable them to work on the project simultaneously.

Kimberly thoroughly checked each part of our project and double checked to make sure we were answering the question and following the assignment guidelines properly. She also did her part for her function for random test/partition and the project report as well as the manual testing.

Everyone worked on the parts of the report such as methodology and team work as well as communicated effectively across platforms to get things done.

5. The class GitHub repository (10 points)

o Submit your unit tests/random tests on GitHub under projects/youronid/ URLValidatorInCorrect/ folder.
o Create a new branch of your repository called "youronid-finalproject" that contains your final submission. This branch must be created before the due date to receive credit.

General Notes:
• Only one person needs to submit Part-b

- Write your names and onids (all group members) in the Canvas comment and the pdf files.
- Add a comment in Canvas and give the URL of the student fork who submitted your project in GitHub (under final project). (-10 for not submitting the URL)