Final Project Part B

CS-362

Casey Balza
Nickolas Jurczak
Zane Salvatore

URLValidator Report

Testing Methodology

Manual Tests:

For manual testing many URLs were tested. The method for choosing which URL to test was to use common URLs such as www.google.com , www.amazon.com , www.test.com , etc. From there, parts of the URL were removed, or added to. The URL domain was tested by using random values, and by testing valid extensions, the same method was used with the URL's scheme. Null, empty, and valid inputs but in reverse order were also tested. Valid and invalid URLs that were proven correct from part a of the project were tested as well.

A few failures were found when performing the manual tests. The following URLs are invalid, but are being returned as valid.

http://wwwtest.com

This particular URL is missing the '.' after the www.

htp://www.test.com
sdfsffsdfsdfs://www.test.com

These two URLs have invalid schemes, infact it seems like as long as the scheme
doesn't have an integer in it, it is proven valid.

The following URLs should be valid but are returned as invalid

http://www.test.xxx
http://www.test.zw
http://www.test.je

These URL's have valid domain name extensions. I believe the reason why these are being returned invalid is because in DomainValidator.java, the extensions are not listed.

http://www.google.com:80/test1?action=view

This URL is being returned as invalid because of the '?'

www.google.com

A full URL missing the scheme is being returned as invalid.

Partitioning

The input for URLValidator can be partitioned based off the inclusion or exclusion of the 8 parts of the url. These parts include the protocol, subdomain, domain name, top level domain, file path, file extension, query, and anchor. The partitions would be all possible combinations of these parts. They are as follows:

Partitions:

1. domain_name **.** top_level_domain
2. sub_domain **.** domain_name **.** top_level_domain
3. domain_name **.** top_level_domain **/** path
4. protocol **://** domain_name **.** top_level_domain
5. protocol **://** sub_domain **.** domain_name **.** top_level_domain
6. sub_domain **.** domain_name **.** top_level_domain **/** path
7. protocol **://** domain_name **.** top_level_domain **/** path
8. domain_name **.** top_level_domain **/** path **.** file_extension
9. domain_name **.** top_level_domain **/** path **.** file_extension **?** query
10. domain_name **.** top_level_domain **/** path **.** file_extension **#** anchor
11. sub_domain **.** domain_name **.** top_level_domain **/** path **.** file_extension
12. protocol **://** domain_name **.** top_level_domain **/** path **.** file_extension
13. protocol **://** sub_domain **.** domain_name **.** top_level_domain **/** path
14. protocol **://** domain_name **.** top_level_domain **/** path **?** query
15. sub_domain **.** domain_name **.** top_level_domain **/** path **?** query
16. protocol **://** domain_name **.** top_level_domain **/** path **#** anchor
17. sub_domain **.** domain_name **.** top_level_domain **/** path **#** anchor
18. sub_domain **.** domain_name **.** top_level_domain **/** path **?** query **#** anchor
19. sub_domain **.** domain_name **.** top_level_domain **/** path **.** file_extension **?** query
20. sub_domain **.** domain_name **.** top_level_domain **/** path **.** file_extension **#** anchor
21. protocol **://** domain_name **.** top_level_domain **/** path **?** query **#** anchor
22. protocol **://** domain_name **.** top_level_domain **/** path **.** file_extension **?** query
23. protocol **://** domain_name **.** top_level_domain **/** path **.** file_extension **#** anchor
24. protocol **://** sub_domain **.** domain_name **.** top_level_domain **/** path **.** file_extension
25. protocol **://** sub_domain **.** domain_name **.** top_level_domain **/** path **#** anchor
26. protocol **://** sub_domain **.** domain_name **.** top_level_domain **/** path **?** query

27. protocol **://** sub_domain **.** domain_name **.** top_level_domain **/** path **?** query **#** anchor
28. protocol **://** sub_domain **.** domain_name **.** top_level_domain **/** path **.** file_extension**?** query
29. protocol **://** sub_domain **.** domain_name **.** top_level_domain **/** path **.** file_extension **#** anchor
30. protocol **://** sub_domain **.** domain_name **.** top_level_domain **/** path **.** file_extension**?** query **#** anchor

The test uses a pool of examples for each part, both valid and invalid examples. The examples cannot, however, contain the special characters that divide the parts next to them, or it could possibly overlap into another partition. While testing each partition, a random example for each part is selected. If any of the parts are taken from the pool of invalid examples, result from the valid() call is expected to be false. Conversely, if all parts are taken from the valid examples, the result form valid() is expected to be true.

Unit/Random Tests

testRandomScheme() randomizes the url scheme and checks to see if the url is valid. Here are the specific details of this random tester.

Generate random values for the scheme in 3 different parts, for example http:// would be broken into three parts; (http), (:), and (//) Each of these three parts will be randomized using numbers 0-9 and random characters and symbols.

Phase 1, only randomize part 1, phase 2, only randomize part 2, phase 3 only randomize part 3, phase 4, randomize all three parts.

For phase 1 there were failures found, for example; “b”, “WK”, “V6F”, “m+”, “F.”, and “W1” were considered valid when used instead of ‘http’ for the protocol portion of the url. It appears whenever the protocol starts with a letter it is proven to be valid.

Phase 2 only considered ‘:’ valid so no failures found here, same for phase 3, only “//” was considered valid.

Phase 4 had no failures as well, it did put out some odd URLs but they are considered valid. For example “M://Yawww.google.com” and “J://bwww.google.com” are valid because anything after the :// is the subdomain which is not necessary to have in the URL so that portion is not needed to be validated.

testRandomSeparator() randomizes the two separators in the url. The Separators that I am referring to are the "." in the url. For example, http://www.google.com has two "." separators.

No Failures were found using this random tester. It does not matter if the first separator value is something else than a “.” because the www portion of the url is not

required. The last separator does need to be a "." which the validator makes certain already. Here are the valid outputs from this test; "http://www2google.com", "http://wwwJgoogle.com", and "http://wwwVgoogle.com".

testRandomPath() This test randomizes the path symbol and path content. What I mean by path symbol is the symbol in the Url right after the .com portion. and the path content is the path string that is after the path symbol. For example "http://www.google.com/path" the last "/" is the path symbol and the "path" is the path content or path string. No failures were found with this random tester.

testIsValid() performs combinatorial testing using a small set of URL strings. It combines valid schemes, authorities, ports, paths, queries, and fragments to test whether they are handled correctly by the validator.

testAnyOtherUnitTest() checks that domains of the maximum length (253 characters) and labels of the maximum length (63 characters) are accepted by the validator.

Teamwork

How we worked as a team -
We worked in a team by going through our tests together, explaining what they do, and going through the implementation of URLValidator to understand how it works. On occasion if one of the members in our team needed help, we all worked together to explain an issue to clarify a requirement or a design in a test implementation.

How we divided our work -
We began by going through the requirements and assigning tasks to each teammate. After we chose what we wanted to do we then divided the remaining tasks evenly by judging how long something would take to do, in combination of all our chosen tasks. We each contributed to the report in the section corresponding to the test we wrote.

How we collaborated -
We collaborated via email and google hangout video chat. We shared our work using Google Docs for the report and the class GitHub repo for our actual tests.

Bug Reports

Missing Domain Name Extensions

Bug #1
Type: Bug
Status: Open

Priority: Major
Resolution: Unresolved

Date: 11/18/2015
Reported By: Casey Balza
Email: balzac@oregonstate.edu

Description
Using domain name extensions from the following resource http://www.computerhope.com/jargon/num/domains.htm extensions beginning with the letters J-Z are reporting as invalid when a valid URL is being input into isValid().

This problem was found when performing manual tests checking for valid and invalid domain name extensions.

The cause of the failure seems to stem from DomainValidator.java. Starting from lines 225 to 364 are the listed acceptable domain name extensions for URL Validator. This file is missing the domain name extensions beginning with the letters J-Z.

Invalid URL Schemes Returning Valid

Bug #2
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved

Date: 11/18/2015
Reported By: Casey Balza
Email: balzac@oregonstate.edu

Description
Using invalid schemes such as “htp”, “jflsdjfdsfldsjf”, “b”, “WK”, “V6F”, “m+”, “F.”, and “W1” result in a valid return from isValid() in UrlValidator.java.

Using the following resource http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml as a guide for determining what is a valid scheme. Manual tests were then performed by inputting values that were not valid and values that are valid into a valid URL structure and the output from isValid() was observed for a true or false response.

The cause of the failure is believed to be from these two lines.
UrlValidator.java Line 81, enables all schemes to pass validation.
UrlValidator.java Line 180 are the allowable default schemes.

If line 81 is set to 0 then the invalid URL schemes return invalid, but valid schemes such as
"geo" are then returned invalid. Must add valid schemes to line 180 to solve this.

Valid Query Pattern using a "?" returns False

Bug #3
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved

Date: 11/29/2015
Reported By: Casey Balza
Email: balzac@oregonstate.edu

Description
When adding a query to the URL the URL is determined invalid when the "?" character is used.

A manual test of inputting "http://www.google.com:80/test1?action=view" into the isValid() returned false. I determined by removing the "?" in the URL that it was this character causing the url to return false.

The cause of the failure is from the "!" on line 446 in UrlValidator.java. If the "!" is removed
the URL is then deemed valid when input into isValid().

Ports Higher Than 999 Incorrectly Rejected

Bug #4
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved

Date: 12/5/2015
Reported By: Zane Salvatore

Email: balzac@oregonstate.edu

Description
URLs with ports greater than 999 are incorrectly rejected as invalid.

This is caused by the PORT_REGEX field of UrlValidator, which only permits port numbers between 1 and 3 digits in length.

Any URL not starting with a scheme returns False

Bug #5
Type: Bug
Status: Open
Priority: Major
Resolution: Unresolved

Date: 12/6/2015
Reported By: Nick Jurczak
Email: jurczakn@oregonstate.edu

Description
isValid always returns false if the url string does not start with a scheme, even if no schemes are set.

On line 336 of UrlValidator.java, inside isValidScheme, it checks if the scheme is null, if it is then isValidScheme returns false. This could be corrected by either removing it, or moving to be on line 346 inside the next if statement checking if all schemes are not allowed. That way if only certain schemes are allowed, it would make sure that one is included.

Debugging Details

The method used for debugging bug #1 was by looking through the project files and noticing the large list of domain name extensions. From here it was noticeable that there was a very large portion of domain name extensions missing, specifically extensions beginning with the letters J-Z.
I used Agan's principle number 1. "Understand the System" to find the location of this bug.

Bug #2 debugging method was performed by looking at the isValid() and noticing that a call to isValidScheme() was made. From there isValidScheme() was examined, this ended up in a

dead end. The next step was to search UrlValidator.java for any other mentions of “scheme”. This resulted in reading lines 34 and 35 which determined that all schemes will pass validation if ALLOW_ALL_SCHEMES was set to true. This in turn brought us to line 81 and testing the results of the manual tests by setting this value to 0. Afterwards it was set back to 1 and then “http” was searched for in the UrlValidator.java which led us to where the default schemes were listed, thus finding the problem. I used Agan’s principle number 3. “Quit thinking and look” and principle number 5. “Change one thing at a time” to find the location of this bug.

The method I used for debugging Bug #3 was searching UrlValidator.java for anything related to the keywords “query” when doing a search through the file. I at first tried adding a “?” to line 151, but that did not work. I then continued through the file using the “query” keyword and found that the logic on line 446 did not make sense so I removed the “!” and this fixed the issue. I then put the “!” back for the developers. The Agan principles used here are number 3 “Quit thinking and look” and number 5 “Change one thing at a time”.

For bug #4, I inspected the regular expression used to parse ports in the URL as a preliminary check. There, I discovered that the regex only allows port numbers up to 3 digits in length. I verified this by testing that the failing tests passed when the regex was adjusted to allow larger port numbers. I used Agan’s principle #1 to solve this problem.

When working on bug #5, I started by using Agan’s principle of getting a fresh view. I went ahead and asked my teammates about the bug. On there suggestion, I added more manual tests to better understand what was causing it. After going over the inputs for some time, I used Agan’s principle quit thinking and look. I followed the isValid function to where the scheme is checked, and then went to the isValidScheme function to find the bug.