URLValidator.pdf: Final report for Final Project-Part B

Team Members
Juan Solis: <solisj@oregonstate.edu>
Myles Chatman <chatmanm@oregonstate.edu>
Todd Waddell <waddelto@oregonstate.edu>

Methodology of testing:

Our methodology of testing was to first create a document in which we brainstormed a variety of input for valid, invalid, and boundary cases. We than began with some manual testing in which we hardcoded known valid URLs and invalid URLs to be passed to the isValid() method.
We partitioned the different components of the input URL string to better narrow down the location of bugs. Finally, we built many URLs by combining different parts into a complete URL to input into isValid().

URLs used in manual testing:

Known valid URLS:

- http://www.google.com
- http://www.amazon.com
- https://www.google.com
- https://www.maps.google.com
- http://www.google.com:8000

Known invalid URLS:

- “ ” -Empty string
- http://www.amazon.123
- http://www.google.com:123456789
- https://www.maps google.com
- htt://www.amazon.com

Reason for choosing your partitions:

We partitioned the input space according to the components of a URL.

Thus, the first partition tests randomly generated URL schemes compared to the standard defined in RFC 2396. Of course, in order to do this, we've used the ALLOW_ALL_SCHEMES flag. We used "google.com" as a standard authority and domain suffix.

For our second partition slice, we looked at the authority component of the URL. First, we ran semi-random tests using randomized combinations of known valid inputs. Then we tested dotted decimal URLs. Then we tested randomly generated authorities with schemes and domains that were known to be valid. Then we ran randomized string lengths.

Provide names of tests:

1) testIsValid:
    a) testIsNull
    b) testASCII_PATTERN
    c) testURLMatcher
    d) testIsValidQuery
    e) testIsValidPath
    f) testIsValidFragment
    g) testIPv4Value
2) testManualTest
3) testAnyOtherUnitTest
4) testYourFirstPartition
5) testYourSecondPartition
6) testPortRange

Your role and work performed in the team:

Todd Waddell:

I wrote the following tests: testYourFirstPartition, testYourSecondPartition, testASCII_Pattern, testUrlMatcher, testIsNull, testIPv4Value. I also contributed to this report and the iIpSegment bug report.

Juan Solis:

I traced the isValid() method deeply and created a document breaking down each aspect of the method in terms of what it is supposed to do. This document was used as a way to divide the work amongst the team. I wrote the following tests: testIsValidQuery(), testIsValidPath(), testAnyOtherUnitTest(), and contributed to testManualTest(). I also contributed to this report and the query bug report # 3.

Myles Chatman:

I initially contributed to the testManualTest() to play around with the functionality and understand how things worked. I wrote the Partition Test testPortRange. I wrote the bug report #1 in this document.

How work was divided in your team:

Work was divided equally among team members. Each individual contributed code and tests of comparable length and complexity. Each team member contributed to this narrative and submitted one bug report.

How teammates collaborated:

The team met via Google Hangouts to discuss about the progress each team member made. This was very beneficial to be able to verify which tasks remained to be completed. Majority of our discussions were related to trying to understand the code so that we could brainstorm

different methods to create our tests. After dividing the work among the team, we continued to meet once a week. Feedback on a member's particular test was assured through push/pull through Github and running the tests in our own IDE. This allowed the group to determine if any improvements in any tests were required. We shared various resources that was related to the identification of an URL to help with the construction of our tests.

Bug reports:

Bug Report #1

==========

Title: URL Validator Port can't accept values that are 4 digits in length or greater.

Class: Serious Bug

Date: 3/1/2016

Reported By: Myles Chatman

Email: chatmanm@oregonstate.edu

Product: Domain Validator

Platform: Java

Is it reproducible: Yes

Description

==========

Testing a port number that is 4 digits in length or greater failed an assertion test returning false where it was expected to return true.

Steps to Produce/Reproduce

----

1. Validate an existing URL i.e.: http://www.google.com . Assert that the URL returns true.

2. Create a class object and assign a port value with four digits or greater

ResultPair[] testUrlPort = ResultPair(":8000", true)

3. Print expected and actual results to console and Assert the results.

Expected Results

Any value from 0 to 65535 is expected to return true.

Actual Results

Values returned true until the port exceeds digit length greater than or equal to 4.

Other Information

File: URL Validator

Line: 158

private static final String PORT_REGEX = "^:(\\d{1,3})$";

Change PORT_REGEX = “^:(\\d{1,5})$

Did you use any of Agan’s principles in debugging? When debugging, I tend to overthink the problem way too much and fell into this issue when trying to discover what the issue was. After discovering the bug, I had no idea what was causing the issue but had many different assumptions of what it could be. I dropped all the assumptions and revisited the code to understand the system better.

_____

Bug Report #2

==========

Title: UrlValidator IPv4 Dotted Decimal Value Check Error

Class: Serious Bug

Date: 3/12/2016

Reported By: Todd Waddell

Email: waddelto@oregonstate.edu

Product: Domain Validator **Version:**

Platform: Java **Version:**

Browser: **Version:**

URL:

Is it reproducible: yes

Description

==========

When given a dotted decimal URL, UrlValidator incorrectly accepts values greater than 255.

Steps to Produce/Reproduce

Submit a test string such as “http://256.256.256.256”

Expected Results

UrlValidator should return false/invalid.

Actual Results

UrlValidator returns true/valid.

Workarounds

None.

Other Information

Bug is in InetAddressValidator.java:94, isValidInet4Address.

Did you use any of Agan's principles in debugging? You could say that I used the "Check the plug rule." I had previously written a randomized tester for dotted decimal addresses. But I had left out a really basic check: testing values greater than 255.

Bug report #3

==========

Title: URL validator incorrectly validating URLS with queries.

Class: Serious Bug

Date: 3/1/2016

Reported By: Juan Solis

Email: solisj@oregonstate.edu

Product: URLValidator.java

Platform: Java

URL:

Is it reproducible: Yes

Description

==========

When given a URL with a query that is not a null string, isValid() is returning the opposite boolean value than what is expected.

Steps to Produce/Reproduce

----

1) First assert isValid() will return true for http://www.google.com or any other known valid URL to be used for the following test queries.
2) Run isValid with previous valid URL + the query “?action=delete”.
3) Run isValid with previous valid URL + the query “?=foo&bar=foo", false”.

Expected Results

----

http://www.google.com

Expected: true

http://www.google.com?action=delete

Expected: true

http://www.google.com?foo=bar&bar=foo

Expected: true

http://www.google.com?=foo&bar=foo

Expected: false

Actual Results

First testing that http://www.google.com

Result: true

http://www.google.com?action=delete

Expected: true

Actual: false

http://www.google.com?foo=bar&bar=foo

Expected: true

Actual: false

http://www.google.com?=foo&bar=foo

Expected: false

Actual: false

Other Information

This error was discovered by the testIsValidQuery() method in UrlValidatorTest.java.

By utilizing the eclipse debugger, the error has been located.

Failure location:

Filename: UrlValidator.java

Method name: isValidQuery()

Line: 446

Code containing error: return !QUERY_PATTERN.matcher(query).matches();

The method isValidQuery() is incorrectly returning the opposite of the expected boolean values due to the '!' operator in the return statement. The code should be as follows in order to resolve the error:

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

Did you use any of Agan's principles in debugging?

Agan's principle, "Understand the System", was used in debugging. I went through the entire isValid() method to better understand it and created a document breaking down what each part was trying to do. It was during this process that I first suspected there could be a bug in the isValidQuery() method called by isValid().

Agan's principle, "Divide and Conquer", was also used in debugging. The unit test I wrote put many different combinations of URL parts into a complete URL to input into isValid(). I than began to test specific URL parts on their own with a known valid URL such as "http://www.google.com". Based on my suspicions from following Agan's principle, "Understand the System", as mentioned earlier, I began with testing queries.