Team Members:
Andrew Calhoun
Jonathan Lagrew
Nancy Chan
CS 362 - Winter 2016

Final Project Part B Report

Bug #1 Report:

1 . What is the failure?

Authority - The highest numbered IPv4 address is represented as 255.255.255.255. Each four decimal number part ranges from 0 to 255 so any parts exceeding 255 such as 256 should return false instead of true.

2. How did you find it?

testIsValid()
Testing INVALID URL: http://**256.256.256.256**
- TEST FAILED, expected INVALID but returned VALID

3. What is the cause of that failure? Explain what part of code is causing it?

This considers IP segments greater than 255 to still be valid:

```
if (iIpSegment > 255) {

        return true;

}
```

Filename where bug manifested itself: InetAddressValidator.java
Line number bug manifested itself: 96
Debugging details: Replacing “true” with “**false**” fixes the bug.

Bug #2 Report:

1 . What is the failure?

Port - Anything over 3 digits returns false though this is not necessarily invalid as port numbers range from 0 to 65535.

2. How did you find it?
testIsValid()
Testing VALID URL: http://127.0.0.1:**8080**
- TEST FAILED, expected VALID but returned INVALID

3. What is the cause of that failure? Explain what part of code is causing it?
This regular expression only matches 1 to 3 consecutive occurrences of any digits 0-9 which does not account for valid port numbers consisting of 4 or more consecutive occurrences such as “8080”:
private static final String PORT_REGEX = "^:(\\d{**1,3**})$";

Filename where bug manifested itself: UrlValidator.java
Line number bug manifested itself: line 158
Debugging details: Changing “1,3” to “**1,5”** fixes the bug.

Bug #3 Report:
1 . What is the failure?
Query - Properly formatted field-value pairs are returned as false instead of true.

2. How did you find it?
testIsValid()
Testing VALID URL: http://127.0.0.1:80/path**?lvalue=rvalue**
- TEST FAILED, expected VALID but returned INVALID

3. What is the cause of that failure? Explain what part of code is causing it?
When a query string is determined to be valid, the expression: “QUERY_PATTERN.matcher(query).matches()” evaluates to true. However, the code below always returns the opposite result because of the negation operator:

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

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

Filename where bug manifested itself: UrlValidator.java
Line number bug manifested itself: 446
Debugging details: Removing the **negation operator (!)** fixes the bug.

Bug #4 Report:

1 . What is the failure?

The list of country code top-level domains is incomplete.
Expected results: Any valid country code should return a valid output
Actual results: Country codes after Italy return invalid

2. How did you find it?

testManualTest()
Testing VALID url: http://www.amazon.co.**jp**/
- TEST FAILED, expected VALID but returned INVALID

3. What is the cause of that failure? Explain what part of code is causing it?

Manual testing of the domains revealed that any country code after Italy was missing from the **COUNTRY_CODE_TLD** array.

Filename where bug manifested itself: DomainValidator.java
Line number bug manifested itself: 358
Debugging details: How to reproduce the bug:
Call isValidAuthority("www.google.**<countryCode>**");

Did you use any of Agan's principle in debugging URLValidator?

We looked at the Apache Commons documentation to gain a familiarity with the URLValidator, as well as reviewing the source code that was provided. We made ourselves familiar with the source code and then went from there.

Following that, we used #2, and created unit tests to make the system fail. We noted several of the aforementioned bugs right away. After we found the failures, we jumped into #3, and we looked through the code thoroughly and debugged, looking for points of failure or errors in the code that may be causing unexpected behavior.

We would then change one thing at a time, make a note of what worked and what did not, and would record what worked.

Frequent discussions on Google Chat allowed us to keep an audit trail (#6) as well, so if one of us got stumped, we could pick up where the others left off and try to find a solution.

Clearly mention your methodology of testing.
For manual testing, we each came up with 10 different urls for a total of 30 urls. We used a variety of schemes, authorities, ports, paths, and queries, selecting/constructing simple urls as well as more complex urls that are comprised of all valid components, all invalid components, or a mix of both valid and invalid components.

For input partitioning, we partitioned by url component (scheme, authority, port, path, and query) and ran several tests for each partition.

For programming based testing, Andrew's tests worked as a grand-scheme suite and tested many of the possibilities, they were largely manual and unit tests without randomized testing. The tests employ an approach that is similar to the testIsValid() in UrlValidatorCorrect but with Andrew's own logic and ideas.

testAnyOtherUnitTest() was created to fill in the gaps of scheme testing. In our manual testing, input partitioning, and programming based testing, we called the UrlValidator() constructor with ALLOW_ALL_SCHEMES. testAnyOtherUnitTest() has a separate set of test cases targeting different schemes by passing all other available options for the UrlValidator() constructor: default (http, https, ftp), ALLOW_2_SLASHES, ALLOW_LOCAL_URLS, and NO_FRAGMENTS.

For manual testing, provide some of your (not all) urls. Did you find any failures?
Some urls (12 listed from our total of 30):
h3tp:/\something.wlel.wakyyy
https:////127.0.0.1
hppt://www.kpcc.com
htt:/\www.yahoo.com
httpz://www.amazon.com
http://vww.amazon.com
http://www.amazon.notacountry
https://www.google.com/webhp?hl=en
ftp://ftp.funet.fi/pub/standards/RFC/rfc959.txt
https://piazza.com/class/iip31fywxe72b0?cid=135
https://secure.engr.oregonstate.edu:8000/teach.php?type=want_auth
http://www.amazon.co.jp/

Yes, we found several failures:
http://www.amazon.co.jp/ is valid but was returned as invalid due to the list of country code top-level domains being incomplete.

These appear to be false positives because we set it to ALLOW_ALL_SCHEMES:
Testing INVALID url: hppt://www.kpcc.com
- TEST FAILED, expected INVALID but returned VALID
Testing INVALID url: httpz://www.amazon.com
- TEST FAILED, expected INVALID but returned VALID

These are due to the query bug described in the Bug #3 report
Testing VALID url: https://www.google.com/webhp?hl=en
- TEST FAILED, expected VALID but returned INVALID
Testing VALID url: https://piazza.com/class/iip31fywxe72b0?cid=135
- TEST FAILED, expected VALID but returned INVALID

For partitioning, mention your partitions with reasons. Did you find any failures?
We partitioned by url component (scheme, authority, port, path, and query):

```
// Test Scheme
testYourFirstPartition()
// Test Authority
testYourSecondPartition()
// Test Port
testYourThirdPartition()
// Test Path
testYourFourthPartition()
// Test Query
testYourFifthPartition()
```

For each partition test, the same valid url was used. Then each url component is swapped out one by one with an invalid version and an empty version to construct new urls to be tested. The reasons for this partitioning are to only change one part at a time to better localize the cause of bugs and to choose values in which each value covers the behavior of a set so we have urls representative of a broad spectrum of urls.

Example (this is testYourThirdPartition() but the other partitions follow the same format for each respective url component):

":80" covers all valid ports
valid url: http://www.amazon.com:80/example?test=passed&result=true

":-100abc000~" covers all invalid ports
invalid url: http://www.amazon.com:-100abc000~/example?test=passed&result=true

"" covers all empty ports (which are also valid)
valid url: http://www.amazon.com/example?test=passed&result=true

Failures were found but unfortunately, there were false positives because of the query bug described in the Bug #3 report which caused all tested urls to be returned as invalid because of the “?test=passed&result=true” part that was present in every url tested.

For unit tests/random tests, submit your unit tests using svn under URLValidator folder. Did you find any failures?
Yes, the failures are described in the bug reports.

In report files mention the name of your tests.
// Manual testing
testManualTest()

// Input Partitioning
testYourFirstPartition()
testYourSecondPartition()
testYourThirdPartition()
testYourFourthPartition()
testYourFifthPartition()

// Programming Based Testing
testIsValid()
testAnyOtherUnitTest()

Also mention, how did you work in the team? How did you divide your work?
When we found a bug, we went to our unit tests and testing suites to test the failure to see if we could replicate the error in each of our own suites. We would also meet weekly and discuss the bugs that we found, potential solutions, and took a divide and conquer approach.

How did you collaborate?
We held weekly meetings on Google Hangouts to discuss our findings and offer solutions and ideas.