URLValidator Testing Report

By Behnam Saeedi, Sarah Lemieux (ONID: 932300799), and Jonathan Harijanto (932126199)

Methodology of Testing

We initially used manual testing then we used input partition testing, and unit testing techniques in order to test the URL Validator. All of our tests are located in the file UrlValidatorTest.java. The manual tests are located in the function testManualTest(). The tests of our input partitions are located in testScheme(), testAuthority, testPort(), testPath(), testQuery(), testFragment(), and testOtherPartitions(). Our unit tests are located in testIsValid(). Each testing method is explained in more detail below.

Manual Tests

We used System.out.println() to print out the results of our manual tests. Here are some samples of the tests we performed:

- Tests that should print out “true”:
  - System.out.println(urlVal.isValid("http://www.amazon.com"));
  - System.out.println(urlVal.isValid("http://www.google.com"));
  - System.out.println(urlVal.isValid("http://go.au:80/test1?action=view true"));
  - System.out.println(urlVal.isValid("http://www.google.com:80/test1?action=view true"));
  - System.out.println(urlVal.isValid("www.google.com"));
  - System.out.println(urlVal.isValid("h3t://0.0.0.0/test1?action=view true"));

- Tests that should print out “false”:
  - System.out.println(urlVal.isValid("256.256.256.256:65636/test1?action=view"));
  - System.out.println(urlVal.isValid("http:1.2.3.4.5"));
  - System.out.println(urlVal.isValid("3ht://go.a:65a/test1//file/#"));
  - System.out.println(urlVal.isValid("://aaa.:-1/../.."));

Input Partition Testing

The input set was first divided based on the different URL components: scheme, authority, path, query, and fragment in this specific order. Then, for each component, the input set was further divided based on different properties of that component. For example, some partitions of the scheme component included the following: schemes that begin with a letter, schemes that contain capital letters, and schemes that contain invalid characters. Both valid and invalid partitions were tested, and each partition was tested at least once. These tests are documented in the functions testScheme(), testAuthority(), testPath(), testQuery(), and testFragment(), which are located in the file UrlValidatorTest.java. Additional cases not related to any specific component of the URL are located in the testOtherPartitions() function. The partitions are based off of the URL syntax specifications outlined in RFC 3896 (https://tools.ietf.org/html/rfc3986). Below is a list of input partitions that were used:

Component	**Valid Partitions**	**Invalid Partitions**
Scheme	· Schemes that begin with a letter · Schemes that contain letters, "+", "-", and "." · Schemes that contain capital letters	· Schemes that don't begin with a letter · Schemes that contain invalid characters
Authority	· Authority that begins with "//" · Authority with a port number between 0 and 65535 · Authority with the minimum port number (0) · Authority with the maximum port number (65535) · Authority containing a valid username	· Authority that doesn't begin with "//" · Authority that contains a colon but no port number · Empty authority
Path	· Path that is empty · Path that is not empty	· Authority is empty but path isn't

	· Path that contains path segments (“./” and “../”)	· Path contains spaces
Query	· Empty query · Single query · Single query with path · Multiple queries delimited by ampersand · Multiple queries delimited by semicolon · Query terminated by # · URL that contains a query and ends in a fragment · URL that ends in a question mark	· Query that contains spaces
Fragment	· Fragment contains uppercase letters · Fragment contains lowercase letters · URL terminated by # · Fragment contains numbers · Fragment contains number and special characters	· Fragment contains spaces
Other	· Empty URL	

Unit Testing

We used multiple for loops to test different combinations of valid and invalid URLs. We created several string arrays that contain lists of possible valid and possible invalid Scheme, Authority, Port, Path and Query components. To build a valid URL, we simply ran the for loops that take all the input from string arrays that contain valid values only. We applied this same process in order to build the invalid URLs. Furthermore, for each URL built from the for loops, we directly tested the URL by calling the isValid() function. These for loops are located in testIsValid() in UrlValidatorTest.java. Some errors occurred when testing the valid URLs. The bugs that caused these errors are documented in the Bug Report section of this report.

Teamwork

Our team coordination worked really well. We didn't have any problem in terms of communications because everyone checked their emails regularly. We were always on the same page; thus it wasn't difficult when we encountered non-working code along the way. In terms of workload, each of us wrote either a unittest or test partition. Furthermore, for the debugging session, we decided that each of us was responsible to find at least one bug since the instructor mentioned there are three bugs. All these collaboration processes happened on a personal GitHub repository that was shared with each group member so that we could share files as we worked on them. We worked both remotely and by meeting twice on the OSU campus to work together.

Agan's Principles

Several of Agan's debugging principles were implemented throughout the project. First, we applied Agan's principle #3 that says "Quit Thinking and Look". When we received numerous amount of Schemes, Authorities, Parts, Paths, and Queries, we didn't try to guess which links were valid based on our own reasoning. Instead, we directly created the unit tests so we could observe the output from isValid(). Second, we also applied Agan's principle #5 that says "Change One Thing at a Time". After doing manual testing, we knew that there were bugs in isValid() because it outputted "false" for many URLs that should have been considered valid. Thus, when we did input partition testing, we decided to isolate our input tests based on the different URL components (scheme, authority, etc.), and we only tested one partition at a time. After we found a partition that gave an unexpected result, we focused only on that partition in order to isolate factors that could have been causing the failure. Then, when debugging, we set a breakpoint on the line that that particular test was called in order to investigate what was causing the failure. Lastly, we applied Agan's principle #8 that says "Get a Fresh View". We met together several times, face-to-face, to discuss the problems that each of us encountered.

These group meetings helped us resolve many questions that we had, and our discussions helped us understand each of the bugs we found.

Bug Report

Summary

We found three bugs total in the URL Validator code. These bugs were found in the UrlValidator.java, InetAddressValidator.java, and the DomainValidator.java files. Here is a summary of the bugs we found and the changes that we propose in order to fix those bugs:

Bug number 1:

Line 446 (UrlValidator.java)
return !QUERY_PATTERN.matcher(query).matches();
becomes
return QUERY_PATTERN.matcher(query).matches();

Bug number 2:

Line 158 (UrlValidator.java)
private static final String PORT_REGEX = "^:(\\d{1,3})$";
becomes
private static final String PORT_REGEX = "^:(\\d{1,5})$";

Bug number 3:

Line 151 (URLValidator.java)
private static final String QUERY_REGEX = "^(.)$";
becomes
private static final String QUERY_REGEX = "^(.*)$";

Bug number 4:

Line 139 (DomainValidator.java)
if (!hostnameRegex.isValid(domain))
becomes
if (hostnameRegex.isValid(domain))

Line 205 (DomainValidator.java)
return LOCAL_TLD_LIST.contains(chompLeadingDot(iTld.toLowerCase()));
becomes

return !LOCAL_TLD_LIST.contains(chompLeadingDot(iTld.toLowerCase()));

Line 64 (InetAddressValidator.java)
return !isValidInet4Address(inetAddress);
becomes
return isValidInet4Address(inetAddress);

Name: Negation in Query Pattern Match Bug

Details:

- **Type:** Bug
- **Status:** CLOSED
- **Priority:** Major
- **Resolution:** VALID
- **Fix Version:** URLValidator & URLValidatorTest

Description:

- **What is the failure?**
  - isValid() returns false when given a valid query (“?action=view", "?action=edit&mode=up and “ “)
- **How did you find it? (Method)**
  - Build a url that is valid (http://www.google.com)
  - Started to added only the query part: http://www.google.com?action=view & http://www.google.com?action=edit&mode=up
  - Both of the test cause isValid() to returns false. This shouldn’t happened because both queries are valid
  - Set breakpoint at testIsValid() in *UrlValidatorTest.java* and isValid() in *UrlValidator.java*
  - Figured out when “step-into” isvalidQuery() during debugging that the return value is negated.
- **What is the cause of that failure? (Line)**
  - In line 449, the original code was “*return !QUERY_PATTERN.matcher(query).matches();”*
  - This cause the result value to be always false, even though the query pattern given already match. Why? Because “!” will always negate the original value, making it to always return (!true).
  - The solution is to remove the “!”, so it can return true whenever the query pattern is match.
  - Correction: *return QUERY_PATTERN.matcher(query).matches();*

People:

- **Reporter:** Jonathan Harijanto

Dates:

- **Created:** 5/19/2016
- **Resolved:** 6/2/2016

Name: Port Number Regular Expression Bug

Details:

- **Type:** Bug
- **Status:** CLOSED
- **Priority:** Major
- **Resolution:** VALID
- **Fix Version:** URLValidator

Description:

- **What is the failure?**
  - isValid() returns false when given port number more than three digits
- **How did you find it? (Method)**
  - Noticed that input partition tests in testAuthority() weren't working. isValid() returns true if the port number is 0 or 80, but not for 65535, which should be the maximum port number
  - Wrote additional tests in testPort() to figure out where the cutoff value was. Port numbers are only considered valid up to 999
  - Set breakpoint in testAuthority() at line 94 in UrlValidatorTest.java
  - Noticed that bug manifests itself on line 393 in UrlValidator.java. PORT_PATTERN has the incorrect pattern
  - Searched for the port regular expression, found bug on line 158. The pattern only matches ports with 1 to 3 digits
- **What is the cause of that failure? (Line)**
  - *private static final String PORT_REGEX = "^:(\\d{1,3})$";*
  - Line 159
  - Supposed to be *private static final String PORT_REGEX = "^:(\\d{1,5})$";*

People:

- **Reporter:** Sarah Lemieux

Dates:

- **Created:** 5/19/2016
- **Resolved:** 6/2/2016

Name: Regex domain fail

Details:

- **Type:** Bug
- **Status:** CLOSED
- **Priority:** Major
- **Resolution:** VALID
- **Fix Version:** URLValidator

Description:

- **What is the failure?**
  - Queries that should be valid are considered invalid. Ex: “?action=view”
- **How did you find it? (Method)**
  - Set breakpoint at line 150 in testQuery() in UrlValidatorTest.java, since that test wasn’t providing the correct result
  - Realized that query pattern on line 151 in UrlValidator.java may be wrong
- **What is the cause of that failure? (Line)**
  - *private static final String QUERY_REGEX = "^(.*)$";*
  - Line 151
  - Supposed to be *private static final String QUERY_REGEX = "^(.)$";*
  - This bug causes an empty query to be counted as a legitimate query
  - It may have been the “Negation in Query Pattern Match Bug” that caused this observed failure, but we noticed this bug while investigating that bug

People:

- **Reporter:** Behnam

Dates:

- **Created:** 5/19/2016
- **Resolved:** 6/2/2016

Name: Local Domain Failure

Details:

- **Type:** Bug
- **Status:** CLOSED
- **Priority:** Major
- **Resolution:** VALID
- **Fix Version:** URLValidator
- **Files involved:** DomainValidator.java, INETAddressValidator.java, URLValidator.java

Description:

- **What is the failure?**
  - Local host is supposed to be a valid input for URLValidator however it fails
  - Localhost is of local domain type and according to function description it supposed to be true
- **How did you find it? (Method)**
  - All of the URLs with localhost failed in our tests.
  - Further explanation available in our Bug Tracing map.
- **What is the cause of that failure? (Line)**
  - Logic errors:
    - The function returns the negation value Local domains.
    - In the method isValidAuthority (line 346)
      - domainValidator.isValid
        - isValid
          - Return (!hostname)
  - In order to fix it we removed the “!” from that and the logic test.

Bug tracing map:
"Localhost" and "localdomain" failed --> output error from isValid() ⟶ DEBUG process --> points at isValidAuthority (line 308, URLValidator) --> further reference to "protected boolean isValidAuthority" (line 364, URLValidator) --> points to "domainValidator.isValid = ... " (line 385, URLValidator)" --> indicates to "public boolean isValid" (134, domainValidator) --> figured out return (!hostname) is the problem.
"Localhost" and "localdomain" still failed --> DEBUG process again --> check the next step which is "public boolean isValidTld" (URLValidator) that reference to "isValidLocalTld()" (URLValidator) --> figured out that return (!LOCAL) is the problem.
"Localhost" and "localadmin" still print failed for the third time⟶ DEBUG process again --> check the next step which is inetAddressValidator (387, URLValidator) -- > reference to inetAddressValidator.isValid (389, URLValidator) --> it points to "public boolean isValid (62, InetAddress)" -- > figured out that change return (!isValidInet4Address) is the problem
"Localhost" and "localdomain" works.

People:

- **Reporter:** Behnam

Dates:

- **Created:** 5/19/2016
- **Resolved:** 6/2/2016