FINAL PROJECT PART B
Jens Bodal, Emily Snyder, William George

Manual Testing

Manual tests are located in [projects/snyderem/finalProject/src/UrlValidatorManualTests.java]

URLs for manual testing were based on example URLs given in RFC 3986 (https://www.ietf.org/rfc/rfc3986.txt) and were validated against other online URL validators to ensure that they were constructed as desired. Both valid and invalid URLs were tested and the results of these tests revealed areas where the isValid() function does not work correctly.

- `testManualTest()`
  Runs through a variety of valid and invalid URLs to see if they are validated correctly by the isValid() method.

Valid URLs		
http://www.amazon.com	TRUE	
http://www.images.google.com	TRUE	
http://www.example.com/index.html	TRUE	
foo://example.com:8042/over/there?name=ferret#nose	FALSE	Bug Report 2
- http://www.example.com:8042	FALSE	
- http://www.example.com:804	TRUE	
http://www.ietf.org/rfc/rfc2396.txt	TRUE	
ftp://ftp.is.co.za/rfc/rfc1808.txt	FALSE	Bug Report 1
- ftp://ftp.is.com/rfc/rfc1808.txt	TRUE	
- http://www.google.co.za	FALSE	
telnet://192.0.2.16:80/	TRUE	
http://192.0.2.16:80/	TRUE	
http://www.example.com/index?test=false	FALSE	Bug Report 3
- http://www.example.com/index?2.3j=what	FALSE	
- http://www.example.com/index?test=false with spaces	FALSE	

Invalid URLs		
www.amazon.com	FALSE	
http://www.amazon	FALSE	
http://www.amazon:8042/	FALSE	
telnet://192.0.2.257:80/	TRUE	
http://www.example.com//index.html	FALSE	
telnet://1200::AB00:1234::2552:7777:1313	FALSE	
http://1200::AB00:1234::2552:7777:1313	FALSE	

Input Partitioning

Inputs were partitioned based on each component of the isValid() function, then further partitioned to test the most simple components of each partition. The isValid() function checks for validity in several parts of the parsed URL individually to determine whether the entire URL is valid. To best discover which portion of the isValid() function was failing we needed to check inputs which were either valid or invalid in the simplest form possible.

There were several inputs we could check in the input domain, a table of the possible inputs are below along with an explanation of the expected structure of each. Specifically we tested portions of the authority and query partitions for errors.

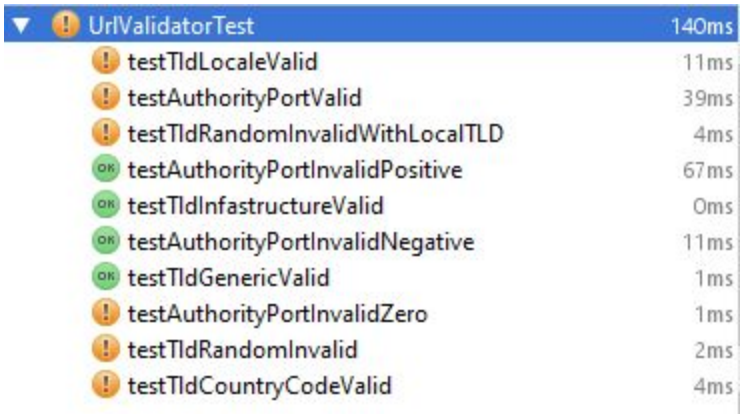
Testing authority was partitioned out to the various components of the authority which it includes. For example “port” is a part of the authority, and can fail the authority test by itself. This includes the user info and host. We know that ports over 1000 don’t work, but once that is fixed, we can’t guarantee that ports over 65535 will **not** work as those would be invalid ports.

Possible Inputs for Input Partitioning	https://tools.ietf.org/html/rfc3986#section-1.1.1
Check null	
Check ascii	Test Non-ASCII characters:
Check structure of URL	foo://example.com:8042/over/there?name=ferret#nose _/ _____/_____/ _____/ __/ | | | | |

	scheme authority path query fragment | _____|_____ / \ / \ urn:example:animal:ferret:nose
Check scheme	scheme = ALPHA *(ALPHA / DIGIT / "+" / "-" / ".") Valid Schemes: Scheme names consist of a sequence of characters beginning with a lower case letter and followed by any combination of lower case letters, digits, plus ("+"), period ("."), or hyphen ("-"). Two slashes required by some schemes and not others Examples: http, ftp, mailto, file, data
Check Authority	The authority component is preceded by a double slash ("//") and is terminated by the next slash ("/"), question mark ("?"), or number sign ("#") character, or by the end of the URI. authority = [userinfo "@"] host [":" port] User info = user:password@ Host = can be a registered name including hostname, IPv4 address in dot decimal notation and IPv6 address which must be enclosed Port = is just a series of 1 to 3 digits delimited from host using : character
Check Path	Can start with / or is empty If no authority it cannot start with // If no scheme it cannot start with :
Check Query	query = *(pchar / "/" / "?") key=value pairs and are separated from the rest of the URL by a ? (question mark) character and are normally separated from each other by & (ampersand) characters. What you may not know is the fact that it is legal to separate them from each other by the ; (semi-colon) character as well.
Check fragment	fragment = *(pchar / "/" / "?") We usually see these used to link to a particular section of an html document. A fragment is separated from the rest of the URL with a # (hash) character.

Unit Tests

Unit tests are located in [projects/snyderem/finalProject/src/UrlValidatorTest.java]



- `testAuthorityPortInvalidNegative()`
  Testing for invalid ports which are negative. All tests passed.
- `testAuthorityPortInvalidZero()`
  Testing that reserved port 0 is invalid. This test failed.
- `testAuthorityPortValid()`
  Testing that all valid ports 1-65535 are valid. This test failed starting at port 1000.
- `testAuthorityPortInvalidNegative()`
  Testing for invalid ports which are positive (>65535). All tests passed.
- `testTldInfastructureValid()`
  Testing infrastructure TLDs. All tests passed.
- `testTldGenericValid()`
  Testing generic TLDs. All tests passed.
- `testTldCountryCodeValid()`
  Testing country code TLDs. This test failed starting at .je (jersey)
- `testTldLocaleValid()`
  Testing local URLs. This test failed.
- `testTldRandomInvalid()`
  Testing randomly generated TLDs that are not in one of the other four TLD lists. This test failed with TLD of .kr (Korea). Likely the non-random country TLD test needs to pass before using this test further as a baseline.
- `testTldRandomInvalidWithLocalTLD()`
  Testing randomly generated TLDs with the `UrlValidator.ALLOW_LOCAL_URLS` option which are not in one of the other four TLD lists. This test failed and shows that `http://www.google.omkztgia` was valid even though it should not be.

Agan's Rules

Divide and Conquer

Manual tests were run with more complex URLs. When a bug was found the URL was simplified down, removing parts until the simplest URL was found that still caused the test to fail. This made locating the bug much easier. For example, during manual testing it was noticed that foo://example.com:8042/over/there?name=ferret#nose failed. This URL was then simplified to http://www.example.com:8042/over/there?name=ferret#nose since http://www.example.com/index.html passed. When this still failed, removed the query to simplify further and ended up testing http://www.example.com:8042/. This lead to the discovery that some valid ports might not be working. We needed to create a base URL that was valid so that we could then append on the parts that we wanted to focus our tests on so that we could find out if those parts worked individually. Once we found a part that didn't work (or a part that worked that shouldn't), we had to then test the counterpart to that to start tracking down the cause.

Understand the System

We read documentation on w3.org and the RFC 3986 to understand how URLs must be created. Test URLs for manual testing were validated with different url validators, such as http://formvalidation.io/validators/uri/ to ensure test URLs were in fact valid or invalid and could interpret the results appropriately.

Change One Thing at a Time

Further to dividing and conquering, stripping a URL down to the bare minimum that would pass a valid test was important to then add on the parts to see which portions would and wouldn't work. It's important to note here that while testing each portion individually is important, it is also important to test combination of other portions to see if those work as a whole. There could be an overarching bug that causes a URL to be "invalid" with a combination of various valid portions of a URL.

Keep an Audit Trail

Again in combination with changing one thing at a time, keeping track of which components you are testing is important so that we could then add on valid/invalid parts to know that what we are testing is what we want to test. For example if we found out that ports over 1000 caused the URL to be invalid, but then tried to test a query with a port over 1000, we would be getting bad test results. The test should be written but documented that the test is currently failing in part to the ports being flagged invalid when over 1000. We kept track of our audit trail by mentioning them in our Google Hangouts Chat and individually in comment documentation for our test cases.

Teamwork

1. How did you divide the work?

   Since there are three basic methods of testing, we split the work along those lines. Emily focused on manual testing, Will on input partitioning and Jens on unit tests. As bugs were found, the team member who found them started debugging and writing the bug report. If they needed to *Get a Fresh View*, they asked for the opinion of the other team members. Where multiple tests uncovered the same bugs, all team members contributed the details their tests found to have as thorough a report as possible.

2. How did you collaborate?

   Collaboration was done via Hangouts and Google Docs. When members of the team were working on the project at a time that the other members were not, we focused on sending questions on Hangouts, since typically at least one other team member was able to respond. We were able to write the project report collaboratively in Google Docs, either writing at the same time or leaving comments with suggestions of further information that was needed or ways to clarify what was written.

BUG REPORT 1

isValidCountryCodeTld() returns false if the country code comes after “it”

Details

Type:	Bug	Status:	Open
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

Description

The list of of accepted country codes in the COUNTRY_CODE_TLDS String[] in DomainValidator.java only lists countries with codes from “ac” - “it”. This means any domain with a country code TLD that comes after “it” will fail. The list of accepted country code are on lines 249-357 of DomainValidator.java, this is where it shows that the codes end at “it”. In turn this causes isValidTld() to return false on line 137 of DomainValidator.java if the domain consists of a country code after “it”.

```
248     private static final String[] COUNTRY_CODE_TLDS = new String[] {
249         "ac",                 // Ascension Island
250         "ad",                 // Andorra
...
357         "it",                 // Italy
358 
359     };
```

Example input

URL	Expected Result	Actual Result
http://www.google.co.za	True	False
http://www.google.com	True	True

BUG REPORT 2

isValidAuthority() returns false if contains a port longer than 3 digits and true if the port is 0

Details

Type:	Bug	Status:	Open
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

Description

The authority validator returns false if the authority contains a port more than 3 digits long but allows the invalid port number 0. Valid ports are from 1 - 65535.

PORT_REGEX on line 158 of UrlValidator.java only checks that ports are 1 - 3 digits long, this causes the PORT_PATTERN.matcher on line 393 of UrlValidator.java to return false if the port is more than 3 digits and true if the port is 0.

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

Example input

URL	Expected Result	Actual Result
http://www.example.com:0	False	True
http://www.example.com:8042	True	False
http://www.example.com:804	True	True

BUG REPORT 3

isValidQuery() returns as false for all URLs with a query

Details

Type:	Bug	Status:	Open
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

Description

On line 446 of UrlValidator.java the function isValidQuery() returns the boolean opposite of the QUERY_PATTERN.matcher instead of returning its result. This results in valid queries being returned as invalid and invalid queries returning as valid.

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

QUERY_REGEX on line 153 of UrlValidator.java matches any character at all including whitespace. This allows characters that need to be escaped such as whitespace to be considered valid.

```
153         private static final String QUERY_REGEX = "^(.*)$";
```

Example input

URL	Expected Result	Actual Result
http://www.example.com/index?test=false	True	False
http://www.example.com/index?2.3j=what	True	False
http://www.example.com/index?test=false with spaces	False	False ** FALSE POSITIVE **

BUG REPORT 4

isValid() returns opposite when using local url option

Details

Type:	Bug	Status:	Open
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

Description

When using the UrlValidator.ALLOW_ALL_SCHEMES option we are getting incorrect validations for local domains and for invalid non-local domains.

Example input

URL	Expected Result	Actual Result
http://www.google.zsrhe	False	True
http://localhost	True	False