Kamila Almurzayeva (almurzak), Dustin LaGrone (lagroned), John Williams (willjohn)
CS 362 Fall 2018
Final Part B

Methodology:

To create a valid testing scheme we will focus on the domain of inputs for the URL validator. Breaking up each section of the domain by its piece of the URL, if it is valid, and then using the scheme parameter that is used when the new URLValidator object is created.

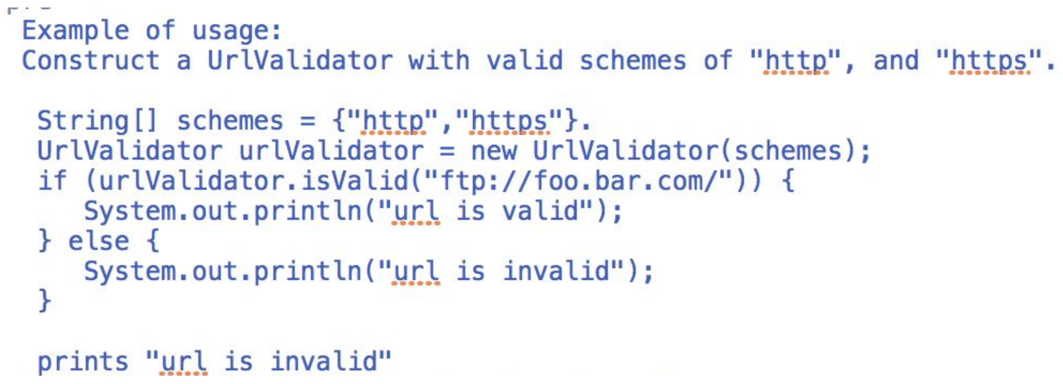
After determining the domain and we prepared our plan for partitioning, we will focus on the different testing methods that can be utilized to find bugs. If a bug is found, we can then run further tests to focus in on the section of code with the problems.

Manual testing should be able to locate the bugs in the overall URL. Partition tests would show us which part of the URL was failing the validation test. Then, random testing can test for combinations that we did not have time to manually implement. This makes sure each part of the URL, if correct, works when it is integrated with its other components.

Manual Testing:

First, I tested URL “http://www.google.com”, it returned url to be invalid. I tried to test them separately by testing first “http://” and then “www.google.com”, neither passed. Changing scheme or authority didn’t help either. By default http, https and ftp supposed to be valid schemes, but they didn’t pass validation test when I tested them, so I set UrlValidator variable I was using to ALLOW_ALL_SCHEMES, it should have allowed all the valid schemes pass the test. When I tested “http://”, it passed the validation test, but when I changed “http://” to “https://”, it showed RegEx expression error. The same happened with “ftp://”, “h3t://” and other valid schemes. Invalid schemes showed an error when tested just as expected. When tested “http://google.com” passed the validation test, different valid and invalid authorities with “http://” scheme passed the test as well. Invalid authorities which passed: “aaa.”, “1.2.3.4.5”, “.aaa”, “go.a” and so on. It seems UrlValidator validates url as long as the scheme is “http://”.

When trying to construct a UrlValidator with valid schemes of “http” and “https” own schemes that should pass validation test like in the UrlValidator code example shown below.



```
Example of usage:
Construct a UrlValidator with valid schemes of "http", and "https".

 String[] schemes = {"http","https"}.
 UrlValidator urlValidator = new UrlValidator(schemes);
 if (urlValidator.isValid("ftp://foo.bar.com/")) {
    System.out.println("url is valid");
 } else {
    System.out.println("url is invalid");
 }

 prints "url is invalid"
```

When testing schemes “http”, “ftp” and “https”, neither were valid.

When added ports, valid and invalid (“:80” (valid), “:-1” (invalid) , “:65535”(valid), “:0”(valid), and “:65536”(invalid)), to “http://google.com” (which has ALLOW_ALL_SCHEMES enabled) url didn’t pass the test. UrlValidator seems to not validate any ports.

When added path, some valid and some invalid (“/test1”(valid), “/t123” (valid), “$23”(valid), “/#” (invalid) , “/#/file”(invalid) , and “/#//file”(invalid)), to “http://google.com” (which has ALLOW_ALL_SCHEMES enabled) url passed the test. It didn’t pass test for paths: “/$23/file”(valid), “/../”(invalid) , “/..”(invalid) , “/test1/file”(valid), “/test1//file”(invalid) , “/test1/”(valid), and “/t123/file”(valid).

Queries: “?action=view”(valid), “?action=edit&mode=up”(valid), “?name=ferret”(valid), “?name=ferret&color=purple”(valid), “?#” (valid) when added to “http://google.com” (which has ALLOW_ALL_SCHEMES enabled) url passed the test. I couldn’t find any invalid queries to test against. So it’s hard to say if query affect validation process or not.

Input partitioning:

Partition testing produced results were limited to the scope of the URL validator ALLOW_ALL_SCHEMES as this is the only way http:// will pass and is the only scheme that passes this test. The other schemes tested were the default constructors ftp, http and https. Also schemes were generated and passed to the UrlValidator(schemes) and checked against failure cases all of these produced failed return values for any scheme tested. In order to continue testing partition cases results were tested with a catch statement which shows a bug initializing the DomainValidator class after testing with the various schemes the only passing value was http://. This scheme in conjunction with the other URL parts authority, port, path, a query was used for the remainder of the partition test. For the path isValid test the url validator constructor was set to both the allow all and the allow two slashes. With the allow two slashes mode enabled all test paths failed with two slashes in cases where this should have passed the test cases.

Scheme test for isValid():

Schemes	Valid	Invalid	Result (Default)	Result(ALLLOW_ALL)
http://	X		fail*	pass
https://	X		fail*	fail*
ftp://	X		fail*	fail*
4abc		X	fail	fail
“”		X	fail	fail
“+-”		X	fail	fail

Authority test for isValid() **results with http only:*

Authority	Valid	Invalid	Result (Default)	Result(ALLLOW_ALL)
www.google.com	X		fail	pass
0.0.0.0	X		fail	pass
OSU.edu	X		fail	pass
.~google.com		X	fail	pass*
1.2.3		X	fail	pass*
“”		X	fail	pass*

Port test for isValid() **results with http only:*

Port	Valid (true)	Invalid (false)	Result (ALLOW_ALL)
:80	X		fail
:21	X		fail
:443	X		fail
65537		X	fail
-15		X	fail
-010		X	pass*

Path test for isValid() **results with http only:*

Path	Valid	Invalid	Result(ALLOW_ALL)	Result(ALLOW_2_SLASHES)
/path	x		pass	*fail
/123	x		pass	*fail
/path/	x		fail*	*fail
/path/path	x		fail*	*fail
/@	x		pass	*fail
//path		x	fail	*fail
/path//path		x	fail	*fail
/*		x	pass*	fail
{a,b,c}		x	pass*	fail
[]		x	pass*	fail

Query test for isValid(): **results with http only and valid path:*

Query	Valid	Results
?	X	pass
?key=value	X	pass
#	X	pass

Programming Based Testing:

For the programming testing, we should begin by taking the url pieces we deemed valid in the partitions and make an array from each piece. Next we use a loop to select each url piece from the array, making sure they are in proper order, to combine them into a valid address for the URLValidator. Using this, test all combinations in the proper order. isValid() is used for these tests. Each time the isValid() function returns false, we can record this in an array of invalid url pieces. At the end of the tests, we can print out the addresses that caused errors when piped into isValid().

A good way to isolate the most salient bugs is to take the worst offenders (of urls) and test them manually to get a good idea of what is messing them up. Maybe the error is being thrown in the parsing function or in the validation test. I think this will provide extra information about the bugs and be more useful than simply running everything through the machine and seeing what we get.

Tests:

UrlValidator urlVal = new UrlValidator();
UrlValidator urlVal = new UrlValidator(UrlValidator.ALLOW_ALL_SCHEMES);
UrlValidator urlVal = new UrlValidator(UrlValidator.ALLOW_2_SLASHES);

Public void testManualTest() - Manual test using input we choose.

Public void testYourFirstPartition() - Tests Scheme
Public void testYourSecondPartition() - Tests Authority
Public void testYourThirdPartition() - Test Port
Public void testYourFourthParition() - Test Path
Public void testYourFifthPartition() - Test Query

Public void testIsValid() - test isValid() with a loop and test all combinations while keeping a log of failed combinations.

Bug Reports:

Individual bug details:

- The isValid() method makes the mistake of determining URLs with a port length of 4 or 5 are invalid.

- A NULL scheme is rejected by URLValidator. A NULL scheme should be invalid, but is considered valid in error.

- Whitespace is not being rejected by URLValidator. An example: “http://www.google.com /” is valid, but should not be.

- Path with double slash enabled during constructor setup doesn’t pass when path //path.

- Default scheme fails when isValid ftp,https are passed as schemes.
- Invalid authority passes .~google.com, 1.2.3, “”
- Negative port value passes (-010) with ALLOW_ALL_SCHEME
- Valid paths fail /path and /path/path
- Invalid paths pass /*, {a,b,c}, []

Created: 11/28/18
Reported by: Dustin LaGrone
Email: lagroned@oregonstate.edu

Type: Bug

Status: Open

Priority: Major

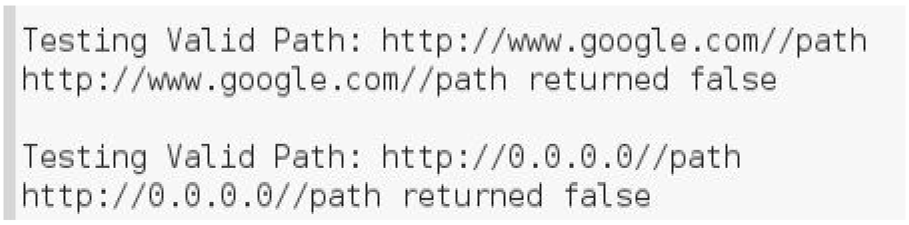
Resolution: Not Resolved

Affected Version: 1.4

File name: UrlValidator.java

Environment: Linux Ubuntu x64, Netbeans 8.2, Java 1.7

Description: The isValid() method with constructor ALLOW_2_SLASHES returns false for paths that include //path or /path//path. Partition testing shows the following output when calling isValid().



```
Testing Valid Path: http://www.google.com//path
http://www.google.com//path returned false

Testing Valid Path: http://0.0.0.0//path
http://0.0.0.0//path returned false
```

Code Causing bug:
protected boolean isValidPath(String path)
UrlValidator.java line 467

Created: 11/30/18
Reported by: Kamila Almurzayeva
Email: almurzak@oregonstate.edu

Type: Bug

Status: Open

Priority: Major

Resolution: Not Resolved

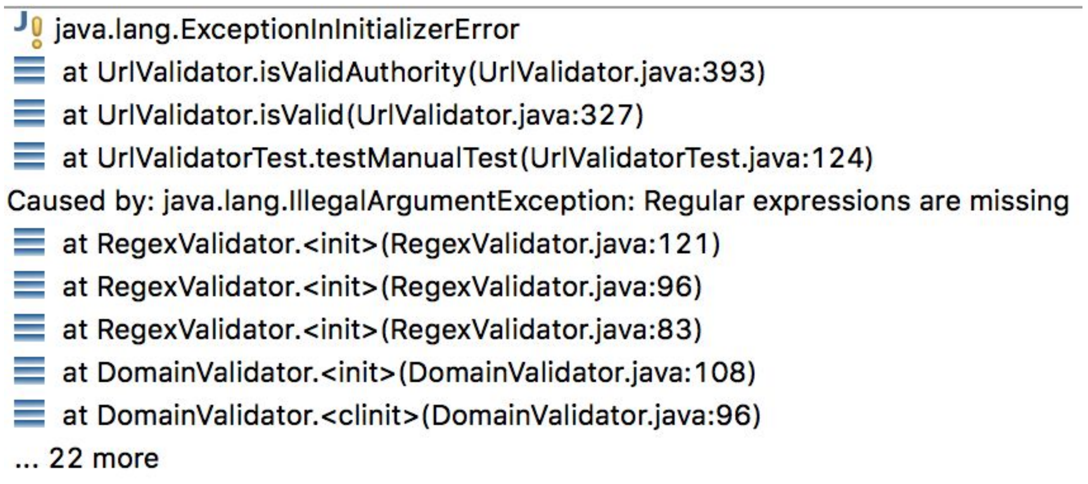
Affected Version: 1.4

File name: UrlValidator.java

Environment: mac OS High Sierra, Eclipse

Description: The isValid() method with constructor ALLOW_ALL_SCHEMES returns an error for some valid schemes, such as https:// and ftp://. It returns valid only for one valid scheme, http://, and returns invalid for invalid schemes.

Error:



```
java.lang.ExceptionInInitializerError
    at UrlValidator.isValidAuthority(UrlValidator.java:393)
    at UrlValidator.isValid(UrlValidator.java:327)
    at UrlValidatorTest.testManualTest(UrlValidatorTest.java:124)
Caused by: java.lang.IllegalArgumentException: Regular expressions are missing
    at RegexValidator.<init>(RegexValidator.java:121)
    at RegexValidator.<init>(RegexValidator.java:96)
    at RegexValidator.<init>(RegexValidator.java:83)
    at DomainValidator.<init>(DomainValidator.java:108)
    at DomainValidator.<clinit>(DomainValidator.java:96)
 ... 22 more
```

Created: 11/30/18

Reported by: John Williams
Email: willjohn@oregonstate.edu
Type: Bug
Status: Open
Priority: Major
Resolution: Not Resolved
Afflicted version: 1.4
File Name: URL Validator.java
Environment: Windows x64, Netbeans 8.2, Java 1.7
Description: The isValidAuthority() function removes trialing whitespaces, which allows invalid URLs to not be rejected. You can find this by submitting “http://www.google.com   /”.

Debugging:

An effective way to close in on the bugs we are seeking is to divide and conquer, localizing the error and manually finding it if needed. This allows us to determine which portion of the URL was failing the isValid() test.

Then, we could investigate closer on those specific areas that failed automatic tests. Checking the partitions for validity and setting breakpoints would be a common and fruitful method for finding the exact location of the bug, down to the line number.

Teamwork:

The team worked together by divvying up the work into thirds. We agreed to meet about once a week to discuss our progress, schedule and any questions that came up during testing. This was done in Google Hangouts since it is a quick and convenient way to communicate in real time.

All progress was recorded in a Google Document so that each member had access to up-to-date work being added to the project. Additionally the group made a Github repository to each work on and merge as a final master branch to submit. This way everyone knew what was completed, what needed to be completed and would be able to comment when there was a concern or questions.

Agan’s Principles:

Our use of Again’s principles was along the lines of:

- Make it Fail
    - In our tests, we made sure to also use input pieces that we knew were invalid to make sure it was handled correctly. Since we knew we would be given a buggy product, we knew that some correct inputs would cause failure and vice versa, so making it fail helped us understand where the issues were earlier.

- Change One Thing at a Time
    - Each time we tested a piece, we made sure the other pieces were valid and wouldn’t throw an error. That way, we could isolate what was erroneous and what was working more easily regarding the portion of the address.

- Understand the System
    - The first part of the final, part A, helped us understand the system at a lower level than just looking at it for the first time. By walking through the program and working on it, we gained knowledge about how the functions work and what their intended use was. This helped us better evaluate in final part B.

- Divide and Conquer
    - We tested the correct and incorrect pieces of the URL components using partitions and programming tests. Therefore, if a test failed, it would narrow down where the error was. We could use breakpoints in our debugger to find the most likely section of code that was causing the error.

- Keep an Audit Trail
    - All of our tests had plenty of print statements for our records to better analyze results and investigate the problem. This way other team members could understand what the rest of the team was working on without having to go through lengthy explanations. Keeping it mostly self-explanatory was useful.

Team member contribution:

- Kamila: manual
- Dustin: partitioning
- John: programing