Final Project

CS 362
Winter 2016

Cierra Shawe (shawec)

Philip Jones (jonesph)

Richard Ratliff (ratlifri)

Project Overview

Background

UrlValidator is a Java class in the "routines" package of Apache Commons, a collection of open source reusable Java components from the Apache community[1]. UrlValidator has routines used to validate URLs. The current version is in the *org.apache.commons.validator.routines* package. In particular, the UrlValidator class has a method isValid() that checks if the given input is a valid URL. There are other methods such as isValidPath() which checks a part of URL. We shall test isValid() and report results.

Collaboration

We have a small team of three people. We are located in Texas in the Dallas and Houston areas, but some travel outside Texas periodically during the semester. So we are definitely a geographically spread team. We held meetings using the Webex system by Cisco which allows desktop sharing and voice over IP, Google Chat for quick updates and statuses, and we used the Google Docs platform to collaborate in two main ways: a spreadsheet for project tracking and defining task ownership / division, and the report document (this document). Our Google Docs project directory also held supporting documents such as project requirements and notes. We divided the project into three main important areas, each having a Lead Owner: code (Richard), bug reports (Philip), and final report (Cierra). The Lead Owner was responsible for overseeing the completion of the area but by no means was expected to do the entire area by themselves. Each area was broken down into smaller tasks and each of those was assigned an owner. So we all owned and led a major area, and we all owned smaller tasks in each and every area.

Test Methodologies

Before getting into the test methodology of our code, let's review the test methodology of using a Java project in Eclipse with JUnit. We were given a Java project comprised of the UrlValidator package built with JUnit. The project does not seem to compile into an executable, but JUnit provides an interactive way to run tests within Eclipse. You can right click on the project and choose "Run As" and then JUnit. A JUnit window will come up beside Project Explorer that has a green "Run Again" button for future test runs. The results of each run come up in the console. We are just printing results for now to see all test results instead of asserting test values which would stop test execution on the first fail.

Testing of UrlValidator focused on tests of the isValid() method. Direct calls to isValid() were made by all of our test routines with valid, expected good URLs, and invalid, expected bad

URLs. But first we reviewed the UrlValidator code, in particular by answering some questions that challenged us to become familiar with the Java library. The main learning from this process was the understanding of URL components, such as scheme, authority, port, path, option and query as in the following examples:

- Scheme http://
- Authority www.amazon.com
- Port 80
- Path /path
- Option /path/option
- Query ?query=something

Three primary test methodologies were used to find bugs. Manual testing was used to get a quick overview and feel of the UrlValidator using valid and invalid URLs based on a commercial website. Then we partitioned the inputs based on the URL components themselves, coming up with sets of valid input partitions for each component as well as invalid partitions. A representative value from each partition was used to test isValid(). Finally, we tested with programming based applications of the input partitions to more thoroughly test isValid() with a larger range of values.

Manual Testing

Several manual tests were made using the Amazon commercial website. The manual tests exercised the component parts of sample Amazon expected good and expected bad URLs. No input partitioning was done for manual testing, just some quick checks of the various component parts. The manual testing is in the testManualTest() method. Below are examples, where true means expected good and was good (isValid), and false means expected bad and was bad:

Scheme

http://www.amazon.com = true
htp://www.amazon.com = false

Authority

http://54.239.25.208 = true (54.239.25.208 is the IP address of www.amazon.com)
http://www.amazon = false

Port

http://www.amazon.com:80 = true
http://www.amazon.com: = false

Path

http://www.amazon.com/valid = true
http://www.amazon.com/ invalid = false

Option

http://www.amazon.com/valid/option = true
http://www.amazon.com/invalid option = false

Query

http://www.amazon.com/b/?node=5 = false (see bugs found)
http://www.amazon.com/b/?node = 5 = false

Potential Bugs Found

The following indicate bugs were found during manual testing. They are all good URLs, and so the expectation was for isValid() to return 'true':

Under Scheme:	www.amazon.com = false	expected default use of http
Under Authority:	http://www.amazon.co.uk = false	Amazon home page in the UK
Under Ports:	http://www.amazon.com:8080 = false	standard Tomcat server port
Also Ports:	http://www.amazon.com:65535 = false	the max port number
Under Query:	http://www.amazon.com/b/?node=5 = false	no isValid() true

Indications are that we found five bugs with manual testing. However, some of these may not be bugs (scheme) and may be due to how the software operates requiring additional inputs. Others may be the same bug (ports). For the "bug" under scheme, a new object was created using the .ALLOW_ALL_SCHEMES field and a new manual test added but it failed too indicating the scheme bug may be real.

Partitioning

Methodology and Reasoning

Our input domain partitioning was based on looking at the component parts of a URL, which are scheme, authority, port, path, option and query. Next, we decided on sets of inputs which could represent each component in a test. Last, we used each of the testYour___Partition() functions to go through and test the various parts. In each test, five of the six elements of the URL would be “good” input as we defined it. The sixth section-- for each test a different one of the component parts above-- would be good, bad, and nonexistent (null). In this way, we could focus in the testing on a specific partitioned portion of the input and determine how the tester responded to various values.

Port is an easy one to illustrate the partition idea. Valid ports range in number from 0 to 65535 which is our good partition for ports used in testYourThirdPartition(). Since non-numerical characters are not part of value ports, we used letters to represent bad input. Finally, we placed blank input to test the URLs where no port was specified. Since ports aren’t usually required for URLs, we would expect the null value to return correct.

Scheme is another easy one to partition. Schemes are an enumerated type as there are only a limited set of valid schemes, or protocols, recognized on the internet. The default schemes supported by UrlValidator comprising the good partition are http, https, and ftp -- testYourFirstPartition() utilizes http. Bad input is created as partial http://-- it’s missing the colon and one of the slashes. Again, as with ports, a scheme is not necessary for a good URL-- a null value for the scheme should still result with a successful output. Other schemes, even invalid ones, can be supported if the value is passed into UrlValidator[1].

The rest of the partition tests are similar to the above. We fill all of the elements of the URL with given valid inputs, then switch out the one we are currently testing with valid, invalid, and null inputs. The only time we would expect for a null input to return invalid would be for the authority-- all other times, the component isn’t necessary, and thus the URL will be correct overall even if it’s missing.

Potential Bugs Found

There were four categories of outcomes for the partition tests-- valid URLs, invalid URLs, URLs that should be valid but were marked invalid, and URLs that should be invalid but were marked valid. Due to possible errors with query handling, only a single URL was correctly marked valid:
http://www.amazon.com:80/valid/option

A number of the URLs were correctly marked invalid:
http/www.amazon.com:80/valid/option?node=5
http://www.amazon:80/valid/option?node=5
http://www.amazon:80/valid/option?node=5
http://www.amazon.com:80abc/valid/option?node=5
http://www.amazon.com:80/valid/option/b/?node=
http://www.amazon.com:80/validinvalidOption?node=5
http://www.amazon.com:80invalid/option?node=5

Because of the issues with the query testing scheme, we didn't come across any that were accidentally valid (should have been invalid). We did, however, find a number that were incorrectly marked invalid:
http://www.amazon.com:80/valid/option?node=5
www.amazon.com:80/valid/option?node=5
http://www.amazon.com:80/valid/option?node=5
http://www.amazon.com:80/valid/option?node=5
http://www.amazon.com/valid/option?node=5
http://www.amazon.com:80/valid/option?node=5
http://www.amazon.com:80/option?node=5
http://www.amazon.com:80/valid/validOption?node=5
http://www.amazon.com:80/valid?node=5
http://www.amazon.com:80/valid/option?node=5

Overall, it appears there was an issue with the isValid function recognizing the query portion of the string as valid-- when it was left off, during the query partition, is the only time that it rang up as OK. Otherwise, there were a number of tests that should have been acceptable, but didn't end up being labeled so.

Programming Based

Methodology and Reasoning

For our programming based testing, we used a brute force method of testing various URLs. This involved using a very nested for loop, that was created using various held in string arrays inputs for scheme, authority, domain, port, path, option, and query concatenated into a string, and passed to URL validator.

What this showed, is that there were far more URL's that should be valid that weren't working, than were working. To test the function, we used JUnit to create the output and then noticed that it was overflowing what the output window was capable of showing.
Our manual and partition test results drove a focus for this test to use known good URLs to limit the messages going to the console and to better see or highlight those that fail. Everything that we tested, should have been valid, however it was not.

The results below show that the validator was only able to handle a scheme of 'http://' and 'https://', it could handle basic paths and options, and port 80. What is not represented below is the basic domains and country codes [ac-it]. The validator was not able to handle queries, ports>999 (4 or 5 digits), and empty schemes.

Potential Bugs Found

Here is the list of representative valid url's, using only the '.com' authority . Something that are repetitive such, as https://www.amazon.com:80/option and https://www.amazon.com:80/valid have been left out in lieu of the combined verson http://www.amazon.com:80/valid/option. It also only shows the scheme of https:// version, however all of the mentioned url's were also valid for the scheme http://.
https://www.amazon.com
https://www.amazon.com/valid/option
https://www.amazon.com:80
https://www.amazon.com:80/valid/option

Here is the list of invalid domains (changed by turning it from printing if valid, to printing if not valid), also based on the same method of selection above:
www.amazon.com
www.amazon.com/valid/option
www.amazon.com/valid/option/?node=5
www.amazon.com:80
www.amazon.com:80/valid/option/?node=5

www.amazon.com:8080
www.amazon.com:8080/valid/option/?node=5
www.amazon.com:65535
www.amazon.com:65535/valid/option/?node=5
https://www.amazon.com/?node=5
https://www.amazon.com/valid/option/?node=5
https://www.amazon.com:80/?node=5
https://www.amazon.com:80/valid/option/?node=5
https://www.amazon.com:8080
https://www.amazon.com:8080/?node=5
https://www.amazon.com:8080/valid/option/?node=5
https://www.amazon.com:65535
https://www.amazon.com:65535/valid/option/?node=5

Debug

Potential Bug 1 >> not a bug

Under Scheme: www.amazon.com = false expected default use of http

Agans Rules used:

- Understand the system - project part A work and reading comments during debug.
- Make it fail - the manual test made it fail.
- Quit thinking and look - used the debugger to move through the code.
- Keep an audit trail - see Steps below.

Steps:

- Set breakpoint at UrlValidatorTest.testManualTest() line: 55
- Debug As JUnit Test
- At breakpoint, Step Into isValid()
- Repeatedly using Step Into
- Eventually set a better breakpoint @ UrlValidator.isValid(String) line: 296
- Resume
- In Variables panel of the Debug perspective see the following Names and Values
    - this/UrlValidator (id=35)
    - value/"www.amazon.com" (id=79)
    - urlMatcher/Matcher (id=80)
    - scheme/null

- Possible bug fix - insert new line before line 296
    - if scheme == null then scheme = http
- Step Into isValidScheme and right off see at UrlValidator.isValidScheme(String) line: 336
    - if scheme == null return false
    - and in the comments above isValidScheme
    - "A null value is considered invalid."
- So this was a design decision and not a bug.

Potential Bug 2 >> bug found

Under Authority: http://www.amazon.co.uk = false Amazon home page in the UK

Agans Rules used:

- Understand the system - project part A work and reading comments during debug.
- Make it fail - the manual test made it fail.
- Quit thinking and look - used the debugger to move through the code and see values of variables.
- Keep an audit trail - see Steps below.

Steps:

- Set breakpoint at UrlValidatorTest.testManualTest() line: 75
- Debug As JUnit Test
- At breakpoint, Step Into isValid()
- Conveniently, prior breakpoint still enabled, stopped at UrlValidator.isValid(String) line: 296
- Scrolled down and set a better breakpoint at UrlValidator.isValid(String) line: 301
- Resume. Several prior breakpoints still enabled. Resume several more times.
- Finally, at breakpoint on line 301. Step Into.
- Now I'm off somewhere in String class. Step Return.
- Set a better breakpoint at UrlValidator.isValid(String) line: 305
- Resume. Step Into.
- In Variables panel of the Debug perspective see the following Names and Values
    - this/UrlValidator (id=30)
    - Authority/"www.amazon.co.uk" (id=72)
- Continue stepping into code and step returning from code outside the project.
- Eventually land on DomainValidator.isValidTld(String) line:154

- see interesting return code that included isValidCountryCodeTld(tld) on line 159 - set breakpoint
- step into isValidCountryCodeTld() and see return COUNTRY_CODE_TLD_LIST.contains(chompLeadingDot(ccTld.toLowerCase()));
- Hover over COUNTRY_CODE_TLD_LIST. It is an Arrays$ArrayList containing country codes. Here are the values shown in the debugger:
  [ac, ad, ae, af, ag, ai, al, am, an, ao, aq, ar, as, at, au, aw, ax, az, ba, bb, bd, be, bf, bg, bh, bi, bj, bm, bn, bo, br, bs, bt, bv, bw, by, bz, ca, cc, cd, cf, cg, ch, ci, ck, cl, cm, cn, co, cr, cu, cv, cx, cy, cz, de, dj, dk, dm, do, dz, ec, ee, eg, er, es, et, eu, fi, fj, fk, fm, fo, fr, ga, gb, gd, ge, gf, gg, gh, gi, gl, gm, gn, gp, gq, gr, gs, gt, gu, gw, gy, hk, hm, hn, hr, ht, hu, id, ie, il, im, in, io, iq, ir, is, it]
- The list is truncated. 'it' is Italy. No country after Italy is included.
- Added new manual test to check for Italy to verify this code works for this type of URL and it does.
- So the bug is truncated country list in file DomainValidator.java for COUNTRY_CODE_TLDS which starts on line 248:
  - private static final String[] COUNTRY_CODE_TLDS = new String[]

Potential Bug 3 >> bug found

Under Ports: http://www.amazon.com:8080 = false standard Tomcat server port

Agans Rules used:

- Understand the system - project part A work and reading comments during debug.
- Make it fail - the manual test made it fail.
- Quit thinking and look - used the debugger to move through the code and see values of variables.
- Keep an audit trail - see Steps below.
- Check the Plug - restarted debugging when landed on "return false" unexpectedly

Steps:

- Set breakpoint at UrlValidatorTest.testManualTest() line: 95
- Debug As JUnit Test
- At breakpoint, Step Into isValid()
- Inconventiently, all the old breakpoints are stopping prematurely in scheme and authority so this time through started toggling them off.

- Fortunately used Step Into instead of Resume after toggling off breakpoint at UrlValidator.isValid(String) line: 305 because the step into resulted in hitting line 306, return false.
- Hovering over authority at this point see its value is www.amazon.com:8080
- So toggled breakpoint back on for at UrlValidator.isValid(String) line: 305
- Restarted debug by Terminate and Debug As JUnit Test
- Unfortunately, the breakpoint at 305 pauses for all authority checks, so hit Resume many times until stop at UrlValidatorTest.testManualTest() line: 95
- Resume and check Variables window for authority = “www.amazon.com:8080”
- ok so Step Into isValidAuthority() and continue stepping
- AUTHORITY_PATTERN.matcher which has value of ^([\p{Alnum}\-\.]*)(:\d*)?(.*)? passes ok
- Used Step Over for the section of code about hostname or IP Address
- When land on line 391 set breakpoint there in case I mess up and so can get back here faster. It is the line in isValid() saying: String port = authorityMatcher.group(PARSE_AUTHORITY_PORT);
- Step Into goes into Matcher.class so Step Return.
- Port now assigned the value :8080
- Now on line 393 that reads if (!PORT_PATTERN.matcher(port).matches()) {
- Hover over PORT_PATTERN and see ^:(\d{1,3})$
- Step Into goes to Pattern.class, code external to project, so Step Return
- Back on line 393. Step Into again goes to Matcher.class, again code external to project, so Step Return
- Back on line 393. Step Into again advances to line 394 - a return false.
- So the pattern matching for port 8080 is not working.
- The pattern is ^:(\d{1,3})$ so what does that mean?
    - ^ means beginning of line or pattern and $ means end of line or pattern
    - between begin and end must be a colon (explicitly the character :) and a grouping by parens ( )
    - The parentheses group digits by \d
    - the braces beside \d provide a range
    - in this case the range has been defined as 1 to 3 digits in the group
    - so any port integer greater than 999 will not match
- So the bug is in the pattern to verify the port number - it only allows ports up to 999 - and 8080 is a valid port especially observed with websites on servers using Tomcat.
- PORT_PATTERN is setup on line 158 of file UrlValidator.java as shown below

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

Potential Bug 4 >> same as above

Also Ports: http://www.amazon.com:65535 = false the max port number

Potential Bug 5 >> bug found

Under Query: http://www.amazon.com/b/?node=5 = false no isValid() true

Agans Rules used:

- Understand the system - project part A work and reading comments during debug.
- Make it fail - the manual test made it fail.
- Quit thinking and look - used the debugger to move through the code and see values of variables and inspect expressions.
- Keep an audit trail - see Steps below.

Steps:

- Started with Debug As JUnit Test
- Toggled off older breakpoints as they came across until the one at line 95 for Port debug above
- Toggled it off but paged down and Set breakpoint at UrlValidatorTest.testManualTest() line: 148
- Step Into and get into PrintStream class outside of project so Step Return
- Put breakpoint on wrong line, should have been UrlValidatorTest.testManualTest() line: 149
- Step Into isValid() and Step Over the parts of code for validating scheme, authority, ports, path
- On line 314 that reads: if (!isValidQuery(urlMatcher.group(PARSE_URL_QUERY))) {
  set a breakpoint in case need to get back here fast
- At this point on line 314 hovering over parts of the line shows the following of note may need for later:
  java.util.regex.Matcher[pattern=^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(\?([^#]*))?(#(.*))?
  region=0,29 lastmatch=http://www.amazon.com/?node=5]
- StepInto goes into Matcher.class outside of project so Step Return
- back on line 314, Step Into goes to another part of the file, line 442, of isValidQuery()

- Here on line 442 the variable query has the value “node=5” which seems correct meaning the java regex matching pattern is probably working ok.
- The variable query is not null, so passes the null check, and the only other thing to do is line 446
  return !QUERY_PATTERN.matcher(query).matches();
- hovering over QUERY_PATTERN shows value ^(.*)$
- The pattern is ^(.*)$ what does that mean?
  - ^ means beginning of line or pattern and $ means end of line or pattern
  - between begin and end can be any character by the period
  - the asterisk modifies what comes before it to mean the range 0 or more
  - so zero or more characters from beginning to end of the line or pattern
  - parentheses show a group (of zero or more characters in this case)
- Step Into goes to Pattern.class and matcher() which says creates a matcher that will match the given input against this pattern. This pattern though is zero or more characters so should match anything, in our case, node=5. Returned, the debugger says is, Matcher m - java.util.regex.Pattern.matcher(CharSequence)
- Back on line 446 of UrlValidator.java. Hovering over the return line, matches() part shows javadoc:
  - **boolean java.util.regex.Matcher.matches()**
  - Attempts to match the entire region against the pattern.
  - If the match succeeds then more information can be obtained via the start, end, and group methods.
  - Returns:
  - true if, and only if, the entire region sequence matches this matcher's pattern
- I am used to a debugger having ‘watch’ or ‘evaluate’ for a variable or an expression. After some searching, this is under the Run menu, as Watch and Inspect. Since I am sitting at a breakpoint already I will use Inspect, or Ctrl-Shift-I.
- Putting the cursor on variable query and Ctrl-Shift-I shows node=5 so its working (same value as just hovering over variable query).
- Putting cursor on matches() and Ctrl-Shift-I says cannot resolve variable.
- Putting cursor on QUERY_PATTERN and Ctrl-Shift-I shows regex ^(.*)$
- Highlighting QUERY_PATTERN.matcher(query).matches() and Ctrl-Shift-I shows ‘true’ but looks like return will negate that to false see line: return !QUERY_PATTERN.matcher(query).matches();

- Back at line 314 in UrlValidator.java the if statement has a negation:
  if (!isValidQuery(urlMatcher.group(PARSE_URL_QUERY))) {
  so if the if evaluates true it will enter the braces where it has “return false;”.
- And in the end, false is printed, failing our test.
- So either the negation on isValidQuery in the if statement is a bug or the return value negation is a bug for the valid query node=5. Based on the structure of the other statements in isValid(), and the “speech” of saying if not isValidQuery then return false, I would say the if statement is ok.
- So the bug is the negation in the return statement on line 446 in isValidQuery() of UrlValidate.java as shown below:
  - return !QUERY_PATTERN.matcher(query).matches();

Bug Reports

We reviewed Apache Software Foundation information at these locations to guide our bug reporting:

- Bug Writing Guidelines[2]
- Commons Validator Issue Tracking[3]
- Commons Validator JIRA project page[4]

Bug 1

Overview Description:
The URL Validator is unable to accommodate country codes, past Italy.
The bug is a truncated country list in file DomainValidator.java for COUNTRY_CODE_TLDS which starts on line 248:

- private static final String[] COUNTRY_CODE_TLDS = new String[]

Steps to Reproduce:

- Set breakpoint at UrlValidatorTest.isValid(String) line: 305
- Set breakpoint at isValidCountryCodeTld(tld) on line 159
- Debug As JUnit Test
- Step Into.
- In Variables panel of the Debug perspective see the following Names and Values
  - this/UrlValidator (id=30)
  - Authority/”www.amazon.co.uk” (id=72)
- Resume to isValidCountryCodeTld(tld) on line 159, then Step Into isValidCountryCodeTld()
- See line return COUNTRY_CODE_TLD_LIST.contains(chompLeadingDot(ccTld.toLowerCase()));
- Hover over COUNTRY_CODE_TLD_LIST. It is an Arrays$ArrayList containing country codes. Here are the values shown in the debugger:
  [ac, ad, ae, af, ag, ai, al, am, an, ao, aq, ar, as, at, au, aw, ax, az, ba, bb, bd, be, bf, bg, bh, bi, bj, bm, bn, bo, br, bs, bt, bv, bw, by, bz, ca, cc, cd, cf, cg, ch, ci, ck, cl, cm, cn, co, cr, cu, cv, cx, cy, cz, de, dj, dk, dm, do, dz, ec, ee, eg, er, es, et, eu, fi, fj, fk, fm, fo, fr, ga, gb, gd, ge, gf, gg, gh, gi, gl, gm, gn, gp, gq, gr, gs, gt, gu, gw, gy, hk, hm, hn, hr, ht, hu, id, ie, il, im, in, io, iq, ir, is, it]

- So the bug is truncated country list in file DomainValidator.java for COUNTRY_CODE_TLDS which starts on line 248:
    - private static final String[] COUNTRY_CODE_TLDS = new String[]

Actual Results:
The URL validator returns false on what should have be a true result.

Expected Results:
The validator should have returned true.

Build Date & Platform:
3/10/2016 on Windows 10 using Eclipse Mars .2 and JUnit - URL validator version created for CS362-w16

Additional Builds and Platforms:
- Occurs On
  Windows 10
Should occur on all platforms, due to this being a missing part of a list.

Additional Information:
Adding the missing elements of the array, should fix this piece of the bug.

Bug 2

Overview Description: More detailed expansion of summary.
Regex matcher doesn't properly match query. This means in more general terms, that the URL validator is unable to validate queries.

Steps to Reproduce:
- Set breakpoint at UrlValidatorTest.isValidQuery() line: 442
- Debug As JUnit Test
- Here on line 442 the variable query has the value "node=5" which seems correct meaning the java regex matching pattern is probably working ok.
- The variable query is not null, so passes the null check, and the only other thing to do is line 446 return !QUERY_PATTERN.matcher(query).matches();
- On line 446, hovering over QUERY_PATTERN shows value ^(.*)$
- Step Into goes to Pattern.class and matcher() which says creates a matcher that will match the given input against this pattern. The regex pattern should match anything, in our case, node=5.
- Step Return
- Back on line 446 of UrlValidator.java, use Inspect, or Ctrl-Shift-I.
- Putting the cursor on variable query and Ctrl-Shift-I shows node=5 so it is working (same value as just hovering over variable query).
- Putting cursor on matches() and Ctrl-Shift-I says cannot resolve variable.
- Putting cursor on QUERY_PATTERN and Ctrl-Shift-I shows regex ^(.*)$
- Highlighting QUERY_PATTERN.matcher(query).matches() and Ctrl-Shift-I shows 'true' but looks like return will negate that to false, see line 446: return !QUERY_PATTERN.matcher(query).matches();
- Back at line 314 in UrlValidator.java the if statement has a negation:
  if (!isValidQuery(urlMatcher.group(PARSE_URL_QUERY))) {
  so if the if evaluates true it will enter the braces where it has "return false;".

- And in the end, false is printed, failing our test.
- So either the negation on isValidQuery in the if statement is a bug or the return value negation is a bug for the valid query node=5. Based on the structure of the other statements in isValid(), and the "speech" of saying if not isValidQuery then return false, the if statement is ok.
- So the bug is the negation in the return statement on line 446 in isValidQuery() of UrlValidate.java as shown below:
  return !QUERY_PATTERN.matcher(query).matches();

Actual Results:
The URL validator returns false on what should have be a true result whenever a query is included.

Expected Results:
The validator should have returned true.

Build Date & Platform:
3/10/2016 on Windows 10 using Eclipse Mars .2 and JUnit - URL validator version created for CS362-w16

Additional Builds and Platforms:
- Occurs On
  Windows 10
Should occur on all platforms, due to this being a missing part of a list.

Additional Information:
NA

Bug 3

Overview Description: More detailed expansion of summary.
Unable to work with ports over 999, which is a problem with the regex matcher.

Steps to Reproduce:
- Set breakpoint at UrlValidatorTest.testManualTest() line: 95
- Debug As JUnit Test
- At breakpoint, Step Into isValid()
- Use Step Into instead of Resume after toggling off breakpoint at UrlValidator.isValid(String) line: 305 because the step into resulted in hitting line 306, return false.
- Hovering over authority at this point see its value is www.amazon.com:8080
- So toggled breakpoint back on for at UrlValidator.isValid(String) line: 305
- Restarted debug by Terminate and Debug As JUnit Test
- Unfortunately, the breakpoint at 305 pauses for all authority checks, so hit Resume many times until stop at UrlValidatorTest.testManualTest() line: 95
- Resume and check Variables window for authority = "www.amazon.com:8080"
- ok so Step Into isValidAuthority() and continue stepping

- AUTHORITY_PATTERN.matcher which has value of ^([\p{Alnum}\-\.]*)(:\d*)?(.*)? passes ok
- Used Step Over for the section of code about hostname or IP Address
- When land on line 391 set breakpoint there in case I mess up and so can get back here faster. It is the line in isValid() saying: String port = authorityMatcher.group(PARSE_AUTHORITY_PORT);
- Step Into goes into Matcher.class so Step Return.
- Port now assigned the value :8080
- Now on line 393 that reads if (!PORT_PATTERN.matcher(port).matches()) {
- Hover over PORT_PATTERN and see ^:(\d{1,3})$
- Step Into goes to Pattern.class, code external to project, so Step Return
- Back on line 393. Step Into again goes to Matcher.class, again code external to project, so Step Return
- Back on line 393. Step Into again advances to line 394 - a return false.
- So the pattern matching for port 8080 is not working.
- The pattern is ^:(\d{1,3})$ so what does that mean?
    - ^ means beginning of line or pattern and $ means end of line or pattern
    - between begin and end must be a colon (explicitly the character :) and a grouping by parens ( )
    - The parentheses group digits by \d
    - the braces beside \d provide a range
    - in this case the range has been defined as 1 to 3 digits in the group
    - so any port integer greater than 999 will not match
- So the bug is in the pattern to verify the port number - it only allows ports up to 999 - and 8080 is a valid port especially observed with websites on servers using Tomcat.
- PORT_PATTERN is setup on line 158 of file UrlValidator.java as shown below
    - private static final String PORT_REGEX = "^:(\\d{1,3})$";

Actual Results:
Any port over 999 with return false.

Expected Results:
All ports up to max port should be allowed.

Build Date & Platform:
3/10/2016 on Windows 10 using Eclipse Mars .2 and JUnit - URL validator version created for CS362-w16

Additional Builds and Platforms:
- Occurs On
    Windows 10
Should occur on all platforms, due to this being a missing part of a list.

Additional Information:

NA

References

[1] Apache Commons Validator 1.5.0 API - UrlValidator documentation
- https://commons.apache.org/proper/commons-validator/apidocs/org/apache/commons/validator/routines/UrlValidator.html

[2] Bug Writing Guidelines
- https://issues.apache.org/bugwritinghelp.html

[3] Commons Validator Issue Tracking
- http://commons.apache.org/proper/commons-validator/issue-tracking.html

[4] Commons Validator JIRA project page
- http://issues.apache.org/jira/browse/VALIDATOR