URLValidator Debug

Rosa Tung (tungr), Sam Nelson (nelsons3) and Kara Franco (lallyk)

The following is the bug report and project documentation for our Final Project Part B. Team members Rosa, Sam and Kara worked together in several Google hangout meetings to design, implement and analyze our testing and debugging of the URLValidator project. We split up work during the meeting so that members designed, tested and analyzed the tests.

Manual Testing

The table below lists out our manual tests for the URLValidator. We decided to test 11 URLs that either had one invalid component or valid components. Our method of testing included one group member creating the URLs, one testing their validity and deciding on an Expected Result, and the last member testing the URL string in the isValid() method. The results are below:

URL to Test	Expected Result	Actual Result
http://www.amazon.com	True	True
ttp://www.amazon.com	False	True
://www.amazon.com	False	False
htt://www.amazon.com	False	True
http://www.amazon	False	False
ftp://google.com	False	True
http://www.cnn.com	True	True
http://www.amazon.com:5555/	True	False
ht://www.amazon.com/	False	True

http://amazon.com/	True	True
http://www.google.com:80000000	False	False

Input-Partitioning:

We have constructed the table below to explain the partitions that we tested for the Input-Partition Testing part of our debugging. As we stated in Part A of our project, the isValid() method is taking apart each URL string and testing one of five URL components (Scheme, Authority, Port, Path and Queries). We decided to test each of these partitions by putting together a URL that contains all the correct pieces of a URL with the exception of the partition to test. We all determined what a correct piece of a URL contains by using our past experience with URLs. The Expected Result column of our table represents whether we thought a test would return true or false.

Partition: URL Schemes	**Partition to Test**	**Expected Result**	**Actual Result**
http://www.google.com	http://	True	True
htt://www.google.com	htt://	False	True
https://www.google.com	https://	True	True
ftp://www.google.com	ftp://	True	True
htts://www.google.com	htts://	False	True
http:www.google.com	http:	False	!isValidScheme(scheme) False
https/www.google.com	https/	False	!isValidScheme(scheme) False
///www.google.com	///	False	!isValidScheme(scheme) False
https//www.google.com	https//	False	!isValidScheme(scheme)

			False
Partition: URL Authorities	**Partition to Test**	**Expected Result**	**Actual Result**
http://www.google.com	www.google.com	True	True
http://amazon.com	amazon.com	True	True
http://uk.yahoo.com	uk.yahoo.com	True	True
http://www.yahoo.co.uk	www.yahoo.co.uk	True	!isValidAuthority(authority) False
http://yahoo.co.uk	yahoo.co.uk	True	!isValidAuthority(authority) False
http://0.0.0.0	0.0.0.0	True	True
http://192.175.456.1	192.175.456.1	False	True
http://192.174.456.1.	192.174.456.1.	False	!isValidAuthority(authority) False
http://.4.3.4.5	.4.3.4.5	False	!isValidAuthority(authority) False
http://www.google.444	www.google.444	False	!isValidAuthority(authority) False
http://spam.google.com	spam.google.com	True	True
Partition: URL Ports	**Partition to Test**	**Expected Result**	**Actual Result**

http://www.google.com:6000	:6000	True	!isValidAuthority(authority) False
http://www.google.com:39r4	:39r4	False	!isValidAuthority(authority) False
http://www.google.com	“”	True	True
http://www.google.com:0	:0	True	True
http://www.google.com:8000000	:8000000	False	!isValidAuthority(authority) False
http://www.google.com:-244	:-244	False	!isValidAuthority(authority) False
Partition: URL Paths	**Partition to Test**	**Expected Result**	**Actual Result**
http://spam.google.com/mail/u/2	/mail/u/2	True	True
http://spam.google.com/$!37	/$!37	True	True
http://spam.google.com/spam/	/spam/	True	True
http://spam.google.com/..	/..	False	!isValidPath(urlMatcher.group(PARSE_URL_PATH)) False
http://spam.google.com/../	/../	False	!isValidPath(urlMatcher.group(PARSE_URL_PATH)) False
http://spam.google.com/..//mail	/..//mail	False	!isValidPath(urlMatcher.group(PARSE_URL_PATH)) False
http://spam.google.com/inbox//mail	/inbox//mail	False	!isValidPath(urlMatcher.group(PARSE_URL_PATH))

			False
Partition: URL Queries	**Partition to Test**	**Expected Result**	**Actual Result**
http://www.google.com/?twenty=20&UnitID=439	?twenty=20&UnitID=439	True	True
http://www.google.com?twenty=20&UnitID=439	?twenty=20&UnitID=439	False	isValidQuery(urlMatcher.group(PARSE_URL_QUERY)) False
http://www.google.com/?twenty=20&UnitID=439&Kara=see&Cant=kara&This=Kara&Rosa=can't&See=can't&This=Rosa&Sam=Can't&sEe=this	?twenty=20&UnitID=439&Kara=see&Cant=kara&This=Kara&Rosa=can't&See=can't&This=Rosa&Sam=Can't&sEe=this	True	isValidQuery(urlMatcher.group(PARSE_URL_QUERY)) False

Analysis of Input Partitioning: Our method for testing the individual parts of the URL included inspecting the calls made in the series of if statements within isValid(). Starting from the top of the table, we see that the isValidScheme() method had two failures: htt:// was expected to be False and it returned True and htts:// was expected to be False and it returned True. The isValidAuthority() method had three failures. All of the failures were in either in authorities www.yahoo.co.uk, yahoo.co.uk or in the authority that included an incorret IP (192.175.456.1) which came back True, even though 456 is above 256. The isValidPort() method had one failure, :6000 was expected to be true, however the method returned false. The isValidPath() method had no failures, we tested a few tricky paths, such as /../, where the . is not a valid file name or end path. Lastly, the query component test, isValidQuery() method had one failure, we were expecting that the long query with correct syntax to be True, however it returned False.

Program Based Testing:

Our unit test functions are located in the UrlValidatorTest.java file in the project folder. We initially wrote a function named testIsValid(), which is commented out in the file. We intended to test the URLs by looping over URL fragments to construct a URL, then we tested that given URL with isValid(). As you may see, this function had 5 for loops and was too complicated for the tests cases we actually needed. We decided to not use testIsValid() and to write four methods that asserted our assumptions about the bugs in the UrlValidator. We narrowed down the bugs, from our manual testing and input partition testing, to be within one of the four categories: tests that had an expected outcome(fail or success) or tests that had unexpected outcomes (should be working or should not be working). The major failures that we found in our tests and debugging are listed below in our bug report. The test functions are named:

testSuccessfullyWorkingAsExpected()

testSuccessfullyFailingAsExpected()

testShouldBeWorkingButAreNotWorking()

testShouldNotBeWorkingButAreWorking()

Bug Reports:

Below are the three most significant bugs that we identified from our testing and debugging of the UrlValidator project.

Bug #1: http://www.google.com/mail/?twenty=20&UnitID=439

We assumed that the bug was in the isValidAuthority() method in this bug, however after debugging, we found that the isValidQuery() method returns

Filename: UrlValidator.java

Line Number Bug Occurs:

Lines 321 - 323

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

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

What is the failure, what is the cause of that failure, explain what part of the code is causing it? There is an exclamation mark (!) added to the return statement (the highlighted line above). This is causing the return statement to be the opposite and therefore incorrect from the QUERY.PATTERN.matcher().

How did you find it? We began our debugging of this potential bug with all of us discussing where we should narrow down our search for the bug. We then put breakpoints throughout our test file and the UrlValidator.java file. As we stepped through the program, we watched the variables change and paths that the program took. When we were in the isValidQuery() method, we noticed the bug.

Bug #2: http://www.google.com:3000

We tested a normal URL with a valid port number.

Filename: UrlValidator.java

Line Number Bug Occurs:

Line 158
```
private static final String PORT_REGEX = "^:(\\d{1,3})$";
```

What is the failure, what is the cause of that failure, explain what part of the code is causing it? In the above line we found that the PORT_REGEX variable is set to check ports that have digit length between 1 and 3 (1 to 999). Since we know that valid port numbers are from 1 - 65535, we suspect that the 3000 did not pass since it has four digits.

How did you find it? As we did with the bug in isValidQuery(), we stepped through both our UrlValidatorTest and UrlValidator programs to watch the variables related to ports. We saw that in isValidAuthority() the port number is checked against the variable PORT.PATTERN, which is the value of the PORT_REGEX (a port with the number of digits being 1 to 3 (1-999). When we discovered this, we decided this is the bug.

Bug #3: http://www.google.com:8000000

We expected that a port over 65535 would fail, which it did, however it is because it is over :999 and not because it is over 65535. This test is failing for the wrong reasons.

Filename: UrlValidator.java

Line Number Bug Occurs: After line 496, there should be a check for port numbers out of range.

What is the failure, what is the cause of that failure, explain what part of the code is causing it? The failure is that the port :800000 is failing for the wrong reason. This is a tricky bug, since we see that the invalid port number is failing, however, we found that it is failing because is it over 3 digits (999) and not because it is over 65535. We see that the program should have a statement after line 496 that checks if the port number is within bounds (at or below 65535).

How did you find it? We decided to narrow our search to the isValidAuthority() method, where the port is checked. We stepped through the function to look at the variables and realized that line 392 checks if the port segment is null, but then the program fails to check for a port that is over 65535.

Did we use Agan's Principle in debugging?

Our method of debugging slightly followed Agan's Principle. We studied the isValid() method in our Part A of the project thoroughly. This allowed us to become familiar and understand the UrlValidator program. In our manual, input-partitioning and unit tests we fed the program URL values that we knew would make it fail. We also followed the Divide and Conquer rule, for example when we narrowed our search for the port but to the isValidAuthority() method. We also kept track of the tests and debugging methods that we tried (in both the tables above in and a notebook). The Agan's rules that we did not follow were getting a fresh view and actually fixing the bugs (since it was not necessary for this assignment).