Team Members:
Brandon Gipson
Stephen Heng
CS 362, Winter 2016

URLValidator Group Project B

Overall Methodology for Testing:

Beyond the required manual and input partition testing we chose to use random unit testing as our main testing method. This method revealed what bugs we would later focus our debugging efforts on. Using Eclipse debugger we were able to isolate the bugs a create three solid bug reports for our developers to fix. We did this using the Agan's Principle of debugging methodology.

We as a group tried to understand the UrlValidator files. As in project A, we watched the videos and the lecture and also did a review of UrlValidatorTest.java to see how the program was running and all the parts of url that it was checking for. Even before debugging, we looked over the files to see the constraints and what was considered a valid or invalid part of the URL. Our tests were made to stimulate failure and go back and tracing the root of the cause. When we did find bugs, we tested an array of the similar conditions to validate the bug with our divide and conquer technique as seen with our partition tests.

Manual Testing:

For manual testing we tried to focus on URLs that would provide results we were expecting for easier validation testing. This included a few fringe cases (such as //$$.com and ht1q:1337.c0m) and many more common cases (www.google.com, https://amazon.co.uk). Most results were as expected except for a few outliers. These outliers were authorities ending in co.uk or in other country codes and when the scheme was left blank.

Partition Tests:

Team Member, Stephen Heng, tested for the input partition. The tests were done by splitting the URL into multiple components, ie. Schemes, authorities, ports, paths, path options and queries. We had 6 partition tests for each of the components. We could use partition 1 test as an example. For this scheme test, we would create a String Array of valid schemes, String goodschemes[] = new String[5], and invalid schemes, String badschemes[] = new String[4]. Then we would create Strings for all other components and set them as valid/true.

The first part of the partition test would run a for loop combining the goodschemes array with the valid components and test that with the isvalid function. We created condition statements assuming the expected results would return true/valid. If it didn't, a "Fail" statement would print out highlighting that the expected valid URL come out as invalid. For the second part of the test, we did the same test in the other direction. We ran a loop with the badschemes with the valid URL components, expecting an invalid return. If it came out as valid, we knew we had a bug. We ran these partition tests 6 times, testing for expected valid and invalid components.

By running the partition tests, we can across many bugs. We had bugs related to valid queries that returned invalid. We also came across valid port numbers that we detected as valid. The partition test was a great way to pinpoint which parts of URL components were leading to failures.

Random Unit Testing: Through random unit testing we were able to confirm the bugs found via manual testing but to a greater degree. The larger quantity and randomness of the tests lead to a much better sample size when compared to our manual testing. This allowed us to further pinpoint what areas of the URLValidator code was flawed which can be seen in the bug reports below.

Team Dynamic:

Overall we had a great team dynamic. Our primary form of communication was email which we used to divide project responsibilities and periodically provide progress reports for each other. Brandon had an idea of how to program the random tests so he spearheaded that and Stephen took on the input partitioning. We constantly communicated with one another to see if we detected the same bugs and how to go about documenting it in the the bug report. We did have trouble with one of our group members dropping out part way through the quarter but our group proved quite resilient. The two of us worked hard, worked smart, and worked well.

Test Names:

Manual Tests: manualTest.java
Partition Tests: UrlValidatotTest.java
Random Unit Tests: testIsValid.java
Altogether the tests can also be found in UrlValidatotTest.java

Bug Reportrs:

Bug #1

1. What is the failure?

The Port number contains an error. A port is valid if it is a semicolon followed by a zero or positive numbers, no negative integers. Also the port can be an empty set and it will still be valid. However, the failure comes when the port number is longer than 3 digits. For example, port ":8", ":80", and ":800" are valid but port ":8000" is invalid when it should be valid.

2. How did you find it?
We ran our testing during the partition part and tested multiple valid and invalid port numbers. We tested different port digit lengths and found that any port longer than 3 digits are coming up invalid.
Code:
goodport[5] = ":800"; // valid
goodport[6] = ":65535"; // valid but shows up as invalid
goodport[7] = ":800000"; // valid but shows up as invalid

Output:
```
FAIL!!! Incorrect Expecation, URL is invalid:
http://www.amazon.com:65535/test123/test1/file
```

3. What is the cause of that failure? Explain what part of the code is causing it?
The cause of the failure is that the program only considers port numbers of size length 1 to 3 digits long.
File/Location of Bug: URLValidator.java, line 158
Error: private static final String PORT_REGEX = "^:(\\d{1,3})$";
Fix: change the range {1,3} to include the allowable valid port number length

Bug #2

1. What is the failure?
The authority URL is only allowed to go to the max number of 255, however, 256.256.256.256 is listed as a valid authority when it should return invalid.

2. How did you find it?
During the partition tests, we listed 256.256.256.256 as an invalid authority but it returned as valid.
Code:
badauthority[0] = "256.256.256.256";

Output:
```
FAIL!!! Incorrect Expecation, URL is valid:
http://256.256.256.256:80/test123/test1/file
```

3. What is the cause of that failure? Explain what part of the code is causing it?
The failure is a result of a bug in the InetAddressValidator.java file. There is an typo the returns digits over 255 to be true instead of false.

File/Location of Bug: InetAddressValidator.java, line 94-98

Error:

```
if (iIpSegment > 255) {

                    return true;

          }
```

Fix: change return true to return false.

Bug #3

1. What is the failure?

The Option Path “/../” and “#” returns as a valid URL when it is invalid.

2. How did you find it?

During the partition test, we assigned invalid path options to string arrays and tested it with all other valid URL parts and ran it against the isValid test.

Code:

```
badoption[1] = "/../";
badoption[4] = "/#/file";
badoption[5] = "/#";
```

Output:

```
FAIL!!! Incorrect Expecation, URL is valid:
http://www.amazon.com:80/test123/../
FAIL!!! Incorrect Expecation, URL is valid:
http://www.amazon.com:80/test123/#/file
```

3. What is the cause of that failure? Explain what part of the code is causing it?

The cause of the failure is that the program does not take into account the “#” and “/../” as invalid URL paths. There is a section in where “..” and “/..” are listed as invalid but not the entire path of “/../” is included in the return of false.

File/Location of Bug: UrlValidator.java , line 411-434

Error:

```
int slashCount = countToken("/", path);
          int dot2Count = countToken("..", path);
          if (dot2Count > 0) {
          if ((slashCount - slash2Count - 1) <= dot2Count) {
                    return false;
          }
          }
```

```
            return true;
```
Fix: needs to take into account other invalid symbols “#” and “/../”.

Bug #4

1. What is the failure?

Schemes- an empty “” set for the scheme is listed as invalid when it is valid URL.

2. How did you find it?

During testing, we listed the empty set “” as a valid scheme and ran it with valid URL parts and the result was an invalid URL.

Code:

```
goodschemes[3] = "";
```

Output:

```
FAIL!!! Incorrect Expecation, URL is invalid:
www.amazon.com:80/test123/test1/file
```

3. What is the cause of that failure? Explain what part of the code is causing it?

The cause stems from the parameters of String SCHEME_REGEX. It seems the format for a scheme must first contain an alpha, followed by either numbers, then two back slashes \\. It needs to allow for an empty “” as a scheme.

File/Location of Bug: UrlValidator.java, line 131

Error:

```
private static final String SCHEME_REGEX = "^\\p{Alpha}[\\p{Alnum}\\+\\-\\.]*";
```

Fix: It seems the code allows for any alpha as the starting scheme with numbers followed by slashes and/or periods. It needs to allow for an empty set as well.

Bug #5

1. What is the failure?

Authority – the program takes into account the country code only after the first period. However, if there are multiple “.”, it will return invalid when it is a valid authority string. An example, is “amazon.ca” is valid but “google.co.uk” is outputted as invalid when it is valid.

2. How did you find it?

We tested for multiple kinds of country code authority as valid.

Code:

```
goodauthority[7] = "www.google.co.uk";
```

goodauthority[8] = "amazon.ca";

Output:
```
FAIL!!! Incorrect Expecation, URL is invalid:
http://www.google.co.uk:80/test123/test1/file

SUCCESS!!! Correct Expecation, URL is valid:
http://amazon.ca:80/test123/test1/file
```

3. What is the cause of that failure? Explain what part of the code is causing it?
The failure is a result of the program only accept the country code in the first ".ca" and not accounting for a longer authority after the first period, ie ".co.ca" or "co.uk".

File/Location of Bug: UrlValidator.java
Error: The parser does not take into account multiple periods after the authority name. It accounts for google.com but not google.co.uk
Fix: Need to change the parser to incorporate multiple periods after the authority as valid part.

Bug #6

1. What is the failure?
Query – "`file?action=view" returns invalid when it is a valid query. The query accepts an empty "" set as a valid query but not a query containing question marks.`

2. How did you find it?
During the partition phase, we ran tests on goodquery vs badquery and noticed that any queries detailing the form below with be returned as invalid even though it was a valid URL.

Code:
goodquery[0] = "?action=view";

Output:
```
FAIL!!! Incorrect Expecation, URL is invalid:
http://www.amazon.com:80/test123/test1/file?action=view
```

3. What is the cause of that failure? Explain what part of the code is causing it?
The program does not allow a starting delimiter such as a question mark to streamline the routes for queries. It will always return invalid.

File/Location of Bug: URLValidator.java, line 151
Error: private static final String QUERY_REGEX = "^(.*)$";

Fix: The program needs to allow for a delimiter and other functions need to be change to accommodate such changes.