Oregon State University: CS 362, Winter 2016

Final Group Project, Part B

Yunsik Choi
Jennifer Frase
Charles Hardes
Adam Pippert

Part 1: Manual Testing

Method:

Built a driver called testMan.java that takes one url as a command line argument. The driver feeds the url to the testManualTest method of UrlValidatorTest.java, which in turn runs isValid() on that url, returning "true" for valid and "false" for invalid. Using this driver, tested several urls, one at a time. The approach was to implement some basic input partitioning by first testing known valid and invalid url schemes. Using the same method, different valid and invalid authorities were tested, then port, then path, then query. Agan's principles of "Divide and Conquer" and "Change One Thing at a Time" were used in trying to separate and isolate bugs in different parts of the url.

Also, for use in cases that were unknown to whether they were actually valid or invalid, another driver was built called testManCorrect, which was built and written exactly the same as testMan, except with the correct version of UrlValidator.java from the UrlValidatorCorrect directory of the repo. Running a url through both testMan and testManCorrect would show exactly where the differences were in output, and therefore, which urls produced erroneous output in the buggy version of UrlValidator.java. Once a few bugs were found, testMan and all of its dependencies were run through Eclipse, and debugged line-by-line in order to locate where the bug occurred in the code.

Example of several urls used in manual testing:

Scheme testing:

http://www.google.com	(this was used as the default and tested valid as expected)
http56789://www.google.com	(this initially returned valid until it was discovered the driver had a bug)
https://www.google.com	(valid, correct)
ftp://www.google.com	(valid, correct)

Authority testing:

http://	(invalid, correct)

http://a.b.c.d.e.f.g.h.i.j	(invalid, correct)
http://a.b.c.d.e.f.g.h.com	(valid, correct)
http://wwww.google.com	(valid, correct)
http://www..google.com	(invalid, correct)
http://www.google.com.	(invalid, correct)

Port testing:

http://www.google.com:80	(valid, correct)
http://www.google.com:65535	(invalid, incorrect)
http://www.google.com:10000	(invalid, incorrect)
http://www.google.com:1000	(invalid, incorrect)
http://www.google.com:100	(valid, correct)
http://www.google.com:999	(valid, correct)

Path testing:

http://www.google.com/a/b/c	(valid, correct)
http://www.google.com/a//b/c	(invalid, correct)

Query testing:

http://www.google.com?query=true	(invalid, incorrect)

Bugs Found:

Port numbers > 999 were invalid
Queries are never valid

Part 2: Partition Testing

Method:

Since the url's are made up of several different parts, specifically the scheme, authority, port, path, and query, there is a partition for each one of these parts. In each partition the part being tested is given a good, bad, and empty value. These are then added onto the minimum requered url parts, so for the scheme, the authority is added and vice versa, but for the other three parts both the scheme and the authority are added, since both parts are required for a valid url. In this way as long as the valid scheme and authority partitions pass, then the tester can check that each part of the url is acting correctly.

Once all the parts have been verified to act correctly, then the last partition, which is checking that all the valid combinations of parts also work together, becomes effective in finding bugs that are a result of the interaction of these parts. Since the previous partitions only checked each part with the bare minimum required to check it, the way the validator works when more than those parts are present is unknown. Thus the last partition becomes necessary to make sure all the parts interact correctly.

In the process of debugging the agan's principles that I used were:
1 - Understand the system
In order to understand the system we were asked to be able to describe what the system does when working correctly from part A of the project. This helped in allowing for the easier comprehension of where a logical partition would occur. In addition, during the process of debugging the test came across some unexpected behavior, but after reading the comments about the functionality of the validator it was found to be working correctly even though the url was not necessarily invalid.

2 - Make it fail
In the process of debugging the test could repeatably make the validator fail. After examining the failure points, they were found to be the same issue repeatably.

3 - Quit thinking and look
Noticing that a bug occured in a test, a break point was set and from there the code was stepped through until a bug was found.

4 - Divide and Conquer
Break points were used to break up the code to see in which parts the test was failing.

Example Partition urls:

Basic layout for partitions 1-5 is scheme+authority+tested_part_if_not_already_there

Good: http://www.google.com:80
Bad: http://www.google.com:-1
No: http://www.google.com

The first 5 partitions all have a scheme and authority, except for 1 and 2 while testing for no scheme and no authority respectively.

Partition 6 goes through all the various valid combinations of 3 or more parts, for example:
http://www.google.com/test1?action=view
http://www.google.com:80/test1

Sample Test Results:

Testing Port Partition

Good port: http://www.google.com:80
Url worked

Bad port: http://www.google.com:-1
Url worked

No port: http://www.google.com
Url worked

Testing Valid Combination Partition

Scheme + Authority + Port: http://www.google.com:80
Url worked

Scheme + Authority + Path: http://www.google.com/test1
Url worked

Scheme + Authority + Query: http://www.google.com?action=view
URL is valid but failed

Note: Url worked is short for Url worked as expected, this was shortened to make it easier to see error messages

Found Bugs:

Queries are never valid

Part 3: Programming Based Testing

Method:
A method, testIsValid() generates a set of input urls by combining the sample url elements provided by commons apache. Each sample url element is associated with a boolean value of valid or invalid (valid is represented as true, and invalid is represented as false) to indicate whether the element is valid or not.

After combining the url elements, the code constructs a sample url along with its boolean value of valid or invalid, which indicates the combined url is valid or not. For each combined url it is also used as an input for the buggy method, isValid(). Since the input url's value is predetermined, we can compare whether the buggy method, isValid() returns the expected value. If there is discrepancy between expected/predetermined and isValid() return values, the testIsValid() records those values (expected vs isValid return) as well as any string that was added from the previous input url.

Out of Agan's nine principles the following principles were used.
#1: Understand the system
#2: Make it fail
#3: Quit thinking and look
#4: Divide and conquer
#5: Change one thing at a time
#8: Get a fresh view

Example Tested urls:
The testIsValid() method prints out results (expected vs buggy isValid() return) of all the possible combined sample urls provided by commons apache. Out of the results, three test urls were selected to localize error sources using Eclipse debugger.

http://www.google.com:65535/test1?action=view
http://www.google.com:80/test1?action=edit&mode=up
ftp://256.256.256.256:80/t123

Found Bugs:
Port numbers > 999 are invalid
Queries are never valid
Ip addresses with fragments above 255 are valid

Bug Report:

Bug 1: *Port numbers > 999 are invalid*

Details: Port numbers entered in the url correctly are only validated up to 999, whereas they should be valid up to the maximum port number, which is 65535.

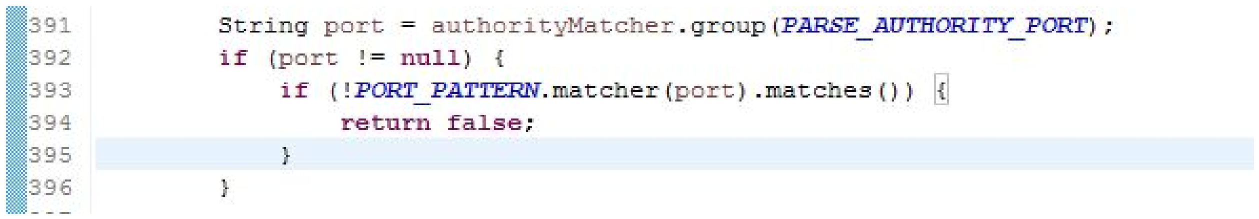
File: UrlValidator
Line: 158

Description:
http://www.google.com:65535/test1?action=view
[Expected] true [Buggy] false [Diff Str] 65535/test1?action=view

This is a valid url, but buggy isValid() returns false meaning invalid.
The Eclipse debugger indicates the error comes from the following return statement.

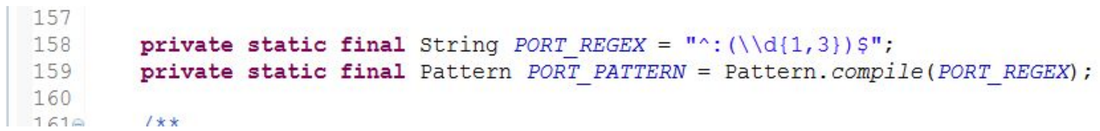
```
if (!isValidAuthority(authority)) {
        return false;
}
```

Through the use of breakpoints and stepping through the code, it was found that line 393 of UrlValidator.java: ‘PORT_PATTERN.matcher(port).matches()’ returns a false condition when it should return true



```
391         String port = authorityMatcher.group(PARSE_AUTHORITY_PORT);
392         if (port != null) {
393             if (!PORT_PATTERN.matcher(port).matches()) {
394                 return false;
395             }
396         }
```

Since the conditional seems to be correct, it is surmised that the problem lies in the matcher. Walking through this code it is found that the match checks the code using Pattern.compile(PORT_REGEX);
Therefore, the error is likely to be originated from PORT_REGEX, which is defined as "^:(\\d{1,3})$".



```
157
158     private static final String PORT_REGEX = "^:(\\d{1,3})$";
159     private static final Pattern PORT_PATTERN = Pattern.compile(PORT_REGEX);
160
161     /**
```

<u>Activity:</u> Line 158 PORT_REGEX = “^:(\\d{1,3})$” means that the port can only be a ‘:’ followed by 1 to 3 digits. This means that the highest allowed port is 999. However, since the port is an unsigned 16-bit integer, the highest possible port number is 65535, which is 5 digits.

Bug 2: *<u>Queries are never valid</u>*

<u>Details:</u> No discovered correctly entered queries have successfully returned a correct valid value. No mattering which method tests this section of a url, the query is always invalid.

File: UrlValidator
Line:446

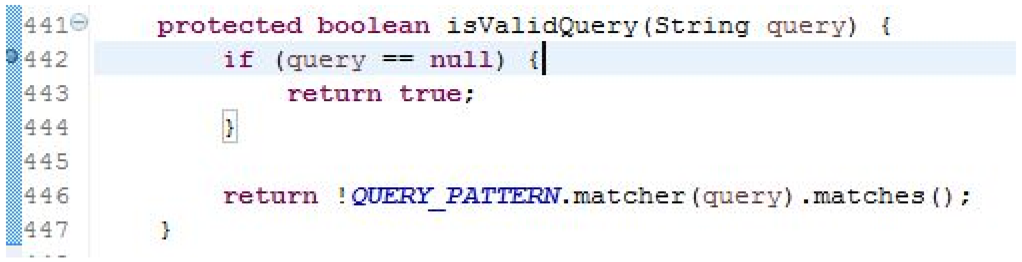
<u>Description:</u>
http://www.google.com:80/test1?action=edit&mode=up
[Expected] true [Buggy] false [Diff Str] ?action=edit&mode=up

The Eclipse debugger indicates the error comes from the following return statement.

```
if (!isValidQuery(urlMatcher.group(PARSE_URL_QUERY))) {
        return false;
}
```

It is then from !QUERY_PATTERN.matcher(query).matches(); statement inside isValidQuery(urlMatcher.group(PARSE_URL_QUERY)).

Line 446 of UrlValidator.java : ‘return !QUERY_PATTERN.matcher(query).matches();’ returns a false value when it should return true



```
441     protected boolean isValidQuery(String query) {
442         if (query == null) {
443             return true;
444         }
445 
446         return !QUERY_PATTERN.matcher(query).matches();
447     }
```

<u>Activity:</u> Since the matcher checks if the query is correct, returning true if it is, in that case, the isValidQuery will return false and vice versa. This means that the ! (not) in front of the expression is the cause of the error.

Bug 3: *<u>Ip addresses with fragments above 255 are valid</u>*

<u>Details:</u> isValid() returns true even though the input url is not valid.

File: InetAddressValidator
Line:96

<u>Description:</u>

ftp://256.256.256.256:80/t123
[Expected] false [Buggy] true [Diff Str] 256.256.256.256:80/t123

The test evaluates the above url as valid. It means this error was not caught by isValid() method where there is only one return true statement at the end.

The Eclipse debugger initially stops at line 305

```
// Validate the authority
if (!isValidAuthority(authority)) {
	return false;
}
```

inside isValid() method.

Inside isValidAuthority() method the debugger stops at line 385

```
if (!inetAddressValidator.isValid(hostLocation)) {
	// isn't either one, so the URL is invalid
	return false;
}
```

However, hostLocation value of 256.256.256.256 is still considered as valid. This invalid hostLocation value is not evaluated as false.

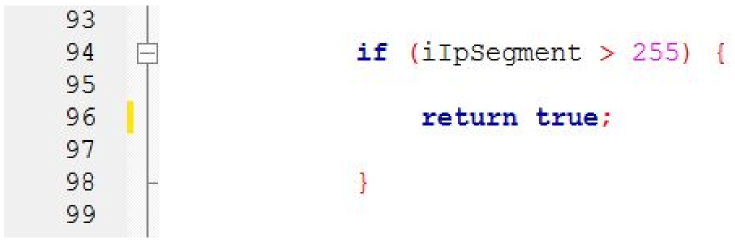
It is likely that the error is originated from inetAddressValidator.isValid(hostLocation).

According to commons apache,

source:
https://commons.apache.org/proper/commons-validator/apidocs/src-html/org/apache/commons/validator/routines/InetAddressValidator.html#line.65

if an ip segment value is greater than 255, it is invalid with a return value of false.

Activity: When looking at the inetAddressValidator.isValid function, it is found that the function redirects to isValidInet4Address and isValidInet6Address functions that are connected via OR operation. Inside the functions there is a check for if the ip segment is greater that 255. This conditional was found to erroneously return true in that cause.



```
            if (iIpSegment > 255) {

                return true;

            }
```

Teamwork:

While doing part A we decided that the best course for our group was to work separately, and then bring our contributions together. As a result, for part B we did a similar thing. We broke the parts up, with part 1 being manual testing, part 2 being partitioning, and part 3 being the programming based testing. In the end Charles did part 1, Jennifer did part 2, and Yunsik did part 3. Since Adam came into our group late in the course. He did not have a part at first. Later when he joined he was assigned to help Charles's test. Accordingly Charles sent Adam what he had completed and his results and Adam replied that he got the same results. However, this is the extent of his participation in the group as Charles (who is the only one he had contact with) did all the coding/debugging and write up for his part. Since each person had different parts, we each did our own debugging as well, and then brought all our reports together, similar to part A.