Final Project Part B

Tom Gariepy, Caleb Bernard, Jonathon Hicke, Chelsea Colvin

Manual Tests

Due to the size of our team, we divided the manual tests into four sections: IP addresses, domain names, port numbers, and queries. We decided that we will individually work on one of the four sections and present our results to everyone. Each section needed to include tests we know should succeed and tests we know should fail.

IP Address Testing	Expected Results	Actual Results
http://52.33.74.97/	true	true
http://255.255.255.256	false	true
http://255.256.255.255	false	true

Domain Name Testing	Expected Results	Actual Results
http://www.google.com	true	true
http://www.▯.com	false	false
http://spaces in name.com	false	false

Port Number Testing	Expected Results	Actual Results
http://52.33.74.97:3000/	true	false
http://www.google.com:443	true	true
https://google.com:80	true	true

Query Testing	Expected Results	Actual Results
http://www.staggeringbeauty.com/?a=yes	true	false
www.google.com?a=yes	false	false
http://burymewithmymoney.com/?a=yes&b=75&stuff=some+text+here	true	false

Input Partitioning:

Based on how the url validator code works, we needed to divide the urls into these subsections:

- protocol
- subdomain
- domain
- port
- path
- query and parameters
- fragments

For each of these sections, we needed to test scenarios where:

- the section was null
- the section had an acceptable value
- the section was using non-ASCII value
- the value was too large

In addition, we needed to allow for unique tests for specific sections when:

- an IP domain contained numbers that are greater than 225
- a port was larger than 3 digits

Based on how the code validates urls and our manual test results, covering all of these conditions should allow for us to find areas where the code fails.

Link to the randomized testing code on Github:
https://github.com/amchristi/cs362w16/blob/master/projects/gariepyt/URLValidator/src/unitTesting.java

Agan's Rules Used:

Rule 2: Make it fail (find inputs that reliably trigger bugs)

Based on our manual tests, we knew of three possible ways to break the code:

1. use IP address with numbers larger than 255
2. use queries
3. use port numbers larger than 3 digits

Since we knew that these errors exist, we intentionally included them in our input partitioning so that it would break and we could determine why.

Rule 4: Divide and Conquer (Use breakpoints to pinpoint where in the code the error occurs)

Using breakpoints we were able to narrow down the methods in the original code that invoked the bug. By repeatedly narrowing the scope of our search, we were able to pinpoint the bug to a single line.

Rule 9: If you didn't fix it, it ain't fixed

In order to ensure that we found the right area where the bug is, we fixed the code and ran our tests again. If the error no longer appeared, then we knew that we not only found the area where the code was broken, but how to fix it.

Methods / Division of Labor:

For Manual Tests

- Jonathon: IP Addresses
- Caleb: Domain names
- Chelsea: Port numbers
- Tom: Queries

For Input Partitioning

- Jonathon: provided requirements based on analysis of the code

For Unit Testing:

- Tom: wrote code and executed

For Eclipse Testing:

- Caleb: determined location of errors in the code

For Error Reports:

- Jonathon: wrote out error reports

For Final Project Document:

- All: completed document

Bug Reports

Title:

IP address error

Summary:

Gives a false positive for invalid IP address.

Description:

False positives are given when validating IP address that should not be valid. If the IP address includes numbers larger than 255 it should be return a False value, but it returns true.

The function does return the correct value for if the number is between 0-255 (true). This happens if the invalid number is in any position of the IP address. It also happens if there is addition paths on the address (pages, ports, etc)

Localization:

Error appears to be in the IsValidInet4Address() in the InetAddressValidator.java file.

The exact problem is at line 96, it should return false if an IP segment is > 255.

Reproducible:

Every time

Code:

```
      UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);


         System.out.println(urlVal.isValid("http://255.255.255.255"));
         System.out.println(urlVal.isValid("http://256.255.255.255"));
         System.out.println(urlVal.isValid("http://255.256.255.255"));
         System.out.println(urlVal.isValid("http://255.255.256.255"));
         System.out.println(urlVal.isValid("http://255.255.255.256"));
Results:
      True
      True
      True
      True
      True
Should be:
      True
      False
      False
      False
      False
```

Title:

Port validation error

Summary:

Gives a false negative for ports >= 1000

Description:

Validation gives a false negative for ports greater than or equal to 1000. This appears to happen no matter what the hostname is. Ports from 0 to 999 are correctly validated.

Localization:

The error appears to be in the isValidAuthority() function of the UrlValidatior. As that is returning false when it should be returning true for the ports.

Within the isValidAuthority() function, the following code is triggering the return value of false.

The exact faulty line of code is line 158 in UrlValidator.java: PORT_REGEX = "^:(\\d{1,3})$";

Reproducible:

Every time for ports greater than 999

Code:

The following code should return true, but it does not

```
UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);



   System.out.println(urlVal.isValid("http://255.255.255.255:3000"));
   System.out.println(urlVal.isValid("http://255.255.255.255:1000"));
   System.out.println(urlVal.isValid("http://255.255.255.255:999"));
   System.out.println(urlVal.isValid("http://255.255.255.255:80"));
   System.out.println(urlVal.isValid("http://google.com:3000"));
   System.out.println(urlVal.isValid("http://google.com:1000"));
   System.out.println(urlVal.isValid("http://google.com:999"));
   System.out.println(urlVal.isValid("http://google.com:80"));
```

Result:

```
False
False
True
True
False
False
True
True
```

Title:

URL Query Error

Summary:

False negative on Query requests

Description:

When a query is included with the URL it results in a false. It appears to happen on every query call, as the random testing shows.

Localization:

Within the Validator.java fail the method isValidQuery() is giving resulting in a false when it should be returning a true.

Within the isValidQuery() the following code is returning the wrong value. The Matcher object is an external library so the error is most likely in the pattern sent to it (QUERY_PATTERN).

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

Reproducible:

Every time with every query call

Code:

```
UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);

System.out.println(urlVal.isValid("https://google.com?foo=bar"));
```

Result:

False

Should be:

True