CS362-004 Final Project Part B
June 10, 2017

Jessica Huang - huangjes
Steven Quagliata - quaglias
Jacqueline Francik - francikj

Methodology Testing

The manual testing is comprised of a series of different urls that were created by the group. The idea was to create a range of urls that would test each section of the url with valid and invalid values. This lead to a grouping of urls that were valid, completely invalid, or a combination of valid and invalid sections. These were then passed through the code to compare to our own expected values.

Sample URLs Manual Testing		
URL	**Expected Outcome**	**Actual Outcome**
http://www.amazon.com	True	True
http://www.google.com	True	True
http://www.amazon.org	True	True
http:///www.amazon.com	False	False
http://www. .com	False	False
http://amazon.com.uy	True	False
http://amazon.us	True	False
https://google.com	True	True
http://12.36.231.167	True	True
http://12.36.231.167:80	True	True
http://361.569.291.357	False	True
http://www.amazon.arpa	True	True
www.amazon.com	True	False
http://google.com/index	True	True
http://www.google.com/?q=url	True	False

tp://google.com	False	False

For the partitioning section we broke each component of the url into its own function. Scheme, authority, port, path, and query were all tested independently using a general array of strings that was duplicated in each function. This lead to having consistent test variables that had only the partition being tested changed. For instance, our first partition was testing scheme and we provided a scheme with valid and invalid values, valid authority, valid port, valid path, and valid query sections that were combined to create url string values that the function would test. When testing for authority in the second partition, a new series of string array values were added with invalid authority and ran through the function just as the scheme function (this time with all valid scheme values instead). This process was repeated for all partitions.

Sample partitions:

validScheme:	“http://”	invalidScheme:	“a2pp://”
validAuthority:	“www.amazon.com”	invalidAuthority:	“361.569.291.357”
validPort:	“:80”	invalidPort:	“-1”
validPath:	“/index”	invalidPath:	“/..”
validQuery:	“?q=URL”	invalidQuery:	“?action=view&”

Sample URLs PartitionTesting		
URL	**Expected Outcome**	**Actual Outcome**
http://www.amazon.com:80/index?q=URL	True	False
a2pp://google.com	False	False
125.5.8.107:80	True	False
http://361.569.291.357	False	False
www.amazon.com?q=URL	True	False
http://125.5.8.107	True	True
http://125.5.8.107:-1	False	False
http://google.com/..	False	False

http://google.com/?action=view&	False	False
http://amazon.com/index	True	True

For the programming section arrays were created for each url component using elements that were both valid and invalid. For example:

```
String[] urlSchemes = {"http://", "http:", "https:", "batman", "://", ""};
String[] urlAuthority = {"www.google.com", "go.au", "255.255.255.255", "1.2.3", ""};
String[] urlPort = {":80", ":65535", ";0", ":-1", "65a"};
String[] urlPath = {"/test1", "/..", ""};
String[] urlQueries = {"?action=view", "?action=edit&mode=up", ""};
```

For our programming test, testIsValid(), elements from the Scheme, Authority and Path arrays were randomly chosen and then concatenated to form a url. This process was completed within a for loop which gave us a different URL to test with each instance of the loop.

If the result was valid, or true, “Passed” was printed out in front of the url that was tested and if the result was invalid, or false, a “Failed” was printed out in front of the url. This allowed is to see which arrays were coming back as invalid and evaluate if this was a valid result or not.

For future programming tests, a different combination of the arrays could be used to create a different set of urls to test.

Our testing was implemented in the UrlValidatorTest.java and named testManualTest(), testYourFirstPartition(), testYourSecondPartition(), testYourThirdPartition(), testYourFourthPartition(), testYourFifthPartition(), and testIsValid().

Bug Report / Debugging

Title: Not All Country Domain Codes Recognized
Platform: Eclipse
Severity of the problem: Medium

Description: When URLs are entered with certain country codes, some of them are not recognized as valid URLs. This was discovered during manual testing when the group entered country codes that were located at the lower end of the alphabet such as .us.

Set-up of test:

1. Go into the UrlValidatorTest.java file.

2. In the public void testManualTest() function, create a new Url Validator with: UrlValidator urlVal = new UrlValidator();
3. Test a Url and print out the results with System.out.println(urlVal.isValid("http://www.amazon.us"));

We tested the URLs: “http://www.amazon.us” and “http://www.amazon.au”, expecting both of them to be valid. The first URL was shown as invalid while the second one returned “true” for a valid URL.

The failure was caused by the DomainValidator.java’s COUNTRY_CODE_TLDS array. This array does not include all the country codes, only the first half until “it” for Italy. Therefore, any URL using country code after “it” will be invalid.

Title: URL Works With Invalid IP Address
Platform: Eclipse
Severity of the problem: Medium

Description: During our manual test, we tried testing some valid and invalid IP addresses in our URLs. The valid IPv4 addresses have numbers between 0 and 255 while invalid IPv4 addresses have numbers outside that range. We were seeing the invalid addresses returning as valid URLs.

Set-up of test:
1. Go into the UrlValidatorTest.java file.
2. In the public void testManualTest() function, create a new Url Validator with: UrlValidator urlVal = new UrlValidator();
3. Test a Url and print out the results with System.out.println(urlVal.isValid("http://12.36.231.167"));

During manual testing, we tested the URLs "http://12.36.231.167" and "http://361.569.291.357". The first returned true as expected while the second URL returned true as well when we expected false.

The cause of the failure is in the InetAddressValidator.java’s isValidInet4Address(String inet4Address) function. In particular, this section of code is resulting in incorrect behavior.

```
if (iIpSegment > 255) {

                return true;

        }
```

Here iIPSegment is one of the segments of the IPv4 address. This should be returning false because these segments should not be greater than 255.

Title: Queries Are Invalid
Platform: Eclipse
Severity of the problem: Medium

Description: Manual testing revealed that adding a query to our valid URL such as “/?q=url” makes the URL invalid. However, this is a valid query so it should return true.

Set-up of test:

4. Go into the UrlValidatorTest.java file.
5. In the public void testManualTest() function, create a new Url Validator with: UrlValidator urlVal = new UrlValidator();
6. Test a Url and print out the results with System.out.println(urlVal.isValid("http://www.google.com/?q=url”));

We tested the URL above with the query, expecting it to return true. The actual outcome was false, flagging it as an invalid URL.

The cause of the failure is in the URLValidator.java’s isValidQuery(String query) function. In particular, this line of code causes incorrect behavior.

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

Here, instead of returning true that the pattern matches, it returns true if the pattern does not match and false if the pattern matches.

Debugging - Agan’s Rules

In order to better get an understanding of the code we were tasked with testing we first needed to understand what the actual code was meant to do. Using Agan’s rules we were able to capitalize on a handful of them to help us get through the assignment. For starters, rule one applies to spending time on the Apache Commons website and reading the implied usage of the code. Rule two applies to finding failure states, and we have states some of the bugs found and their cause (rule three) as we had to find what was causing the failure in the code. Rule four was easily applied since from the beginning our task was the break down and create partitions. This allowed us to focus on very specific parts of the url validator until we could find the parts that were causing the bugs. Although not all of Agan’s rules were specifically used for this project, many of them provided a solid set of guiding principles that we were able to use to accomplish out debugging.

For each bug, we used the debugger for Eclipse by putting a breakpoint at the line that was giving us a bug. Then, we stepped into in order to execute one statement at a time. We continued stepping over until we discovered which section of code was giving us an error. For example, to find the bug “Not All Country Domain Codes Recognized”, we placed a breakpoint at System.out.println("Result:" + urlVal.isValid("http://www.amazon.us")). Then, we used the debugger on Eclipse and stepped into each statement until we found a suspect code. In this case case, we were noticing that something in the domainValidator was not valid, which led us to look more closely at the DomainValidator.java code. From there, we were able to discover that there were missing country codes. The remaining bugs were discovered in much the same way, by setting breakpoints, stepping through the code, and finding statements that did not make sense in respect to how the URL Validator is supposed to work.

Team Work

Teamwork was easily managed with the use of Google docs and the use of Google Hangouts to provide ease of communication between the three of us. The code itself lended to be copied and pasted between partitions with some modifications between each section, but once one team member created an outline of the code another team member was able to create the functional code for the partitions. The third team member focused on the last testing function and created their own version to test the code. All code was shared and reviewed by all members with minimal changes. Once we all had a standardized data set from the combined code we were able to run the programs locally and see if any found bugs could be duplicated at each machine. Given that our code provided a printout of each url and the rest of the test, it was easy to verify where the bugs were and that we were all getting the same bug.