Oregon State - CS362-004

Final Project

APACHE COMMONS URLVALIDATOR

Jon-Eric Cook, Amelia Le, Jung Eun Hwang

Part B

Methodology Testing

Manual Testing

Manual testing was conducted on the isValid() function in the URLValidator.java file. Check each url parts if it is valid or invalid. Found one bug from ip segment that is valid only when is larger than 255. Also, found one bug from all valid queries printed as invalid. This testing method was carried out by simply calling the isValid() function multiple times with both valid and invalid urls as inputs. This was done in an attempt to find any failures in the code. Before testing took place, research was first done to fully understand the different parts of a url. The different url parts are as follows: scheme, host, path and query. Armed with this information, the limits of what each url part could be was explored. From this exploration it was discovered that the maximum port number was 65535, 5 digits. Using a valid url with a port number of 65535 as the input to isValid() caused it to return false, when it should have returned true. This was again tested with a port number of 6553 and 655. The 6553 port number returned false but the 655 port number returned true. Due to these finding, the portion of the URLValidator.c that dealt with the acceptable port numbers was found. Line 158 specifies that only port numbers with 3 digits are accepted. This is where the bug lies.

```
System.out.println(urlVal.isValid("http://www.amazon.com:65535"));
System.out.println(urlVal.isValid("http://www.amazon.com:6553"));
System.out.println(urlVal.isValid("http://www.amazon.com:655"));
System.out.println(urlVal.isValid("http://2.2.2.895"));
System.out.println(urlVal.isValid("http://2.2.2.255"));
System.out.println(urlVal.isValid("http://2.2.2.256"));
System.out.println(urlVal.isValid("http://www.amazon.com?x=1"));
```

System.out.println(urlVal.isValid("http://www.amazon.com?x"));
System.out.println(urlVal.isValid("http://www.amazon.com?x\n"));
System.out.println(urlVal.isValid("http://www.amazon.com?x\0"));

Partitioning
Five functions were created to test URL parts: scheme, authority, port, path, and query. If any one of these parts should fail, the whole URL would fail validation. Testing these parts individually allows us to see if bugs occur at specific points in the validator. Additionally, a sixth function was created to test alternative paths: the ALLLOW_2_SLASHES option allows URLS which contain two slashes.

Each partition was tested with a selection of valid or invalid URL parts, prepended or appended with known valid URL parts to make a complete URL. For each partition, valid URL parts should return true when passed to the validator, and invalid URL parts should return false. Some valid URL examples are:

Scheme:
    Part: http://
    URL: http://www.google.com
Authority:
    Part: amazon.com
    URL: http://amazon.com
Port:
    Part: :80
    URL: http://www.google.com:80
Path:
    Part: /test1
    URL: http://www.google.com/test1
Alternative path:
    Part: /test1//file
    URL: http://www.google.com/test1//file
Query:
    Part: ?action=view
    URL: http://www.google.com?action=view

Base Testing
Generates random strings to attach to url. Some of them passed only when they have “\n”. Also, as for other unit testing, investigated further on the bugs we found above manual and partitioning testing. The integers generated randomly as port partition to be tested.

Agan’s Principal for Debugging

Most of the principals has been applied on our testing except “If you didn’t fix it, it ain’t fixed”. We took a look the issues and changed the inputs gradually to find out the lines of bugs in the program.

Bug Report

Bug ID: Bug 01
Summary: Valid ports fail
Description: Valid ports range from 0 to 65535. “A port number is a 16-bit unsigned integer, thus ranging from 0 to 65535.” https://en.wikipedia.org/wiki/Port_(computer_networking)
During manual testing, ports 6553 and 65535 caused isValid() to return false when they should have returned true. The maximum number of digits in a port is 5, not 3.
Bug code: Line 158 in URLValidator.java only allows ports between 1 and 3 digits.

```
private static final String PORT_REGEX = "^:(\\d{1,3})$";
private static final Pattern PORT_PATTERN = Pattern.compile(PORT_REGEX);

if (!PORT_PATTERN.matcher(port).matches()) {
   return false;
}
```

Test Code:

```
// Expected Results: pass     Actual Results: fail
System.out.println(urlVal.isValid("http://www.amazon.com:65535"));
// Expected Results: pass     Actual Results: fail
System.out.println(urlVal.isValid("http://www.amazon.com:6553"));
```

Debugging Detail: A handful of Agan’s principles were in fact employed during the hunt for the above port bug. Before any formal debugging occurred, a substantial amount of effort was put forth to understand the system. This included researching Java functions such as “Pattern.compile() and the Pattern type “private static final Pattern”. Also, research was done on the use of regular expressions. Once the above was accomplished, multiple variations of urls were testing. Eventually a bug was found, meaning that a known valid url was said to be invalid but the isValid function. At this point a divide and conquer approach was used to narrow the search for the culprit line of code. Due to this tactic, the bug was found on line 158 in URLValidator.java.

Bug ID: Bug 02
Summary: Valid IP Address Fail
Description: IP address which is greater than 255 became incorrectly valid.
Bug code: Line 94 in InetAddressValidator.java
In *public boolean isValidInet4Address(String inet4Address)* function,

```
 if (ilpSegment > 255) {
return true;
}
```

Test Code:

```
//Expected: Invalid  Result: Valid
System.out.println(urlVal.isValid("http://2.2.2.255"));
//Expected: Invalid  Result: Valid
System.out.println(urlVal.isValid("http://2.2.2.256"));
//Expected: Invalid  Result: Valid
System.out.println(urlVal.isValid("http://2.2.2.895"));
```

Debugging Detail:
Traced where IP address been testing in InetAddressValidator.java. Found line 94 had incorrect if statement. It should have been if(iIpSegment < 255).

Bug ID: Bug 03
Summary: Valid Query Fail
Description: Valid Query became incorrectly invalid.
Bug code: Line 446 in UrlValidator.java
In ***protected boolean*** *isValidQuery(String query)* function,
return *!**QUERY_PATTERN**.matcher(query).matches();*

Test Code:

```
//Expected: Valid    Result: Invalid
System.out.println(urlVal.isValid("http://www.amazon.com?x=1"));
//Expected: Valid    Result: Invalid
System.out.println(urlVal.isValid("http://www.amazon.com?x"));
//Expected: Invalid  Result: Valid
System.out.println(urlVal.isValid("http://www.amazon.com?x\n"));
//Expected: Valid    Result: Invalid
System.out.println(urlVal.isValid("http://www.amazon.com?x\0"));
```

Debugging Detail:
In Line 314 of IsValid() called IsValidQuery() and IsValidQuery() was returning true when it is false and vice versa. isValidQuery() should return
"***QUERY_PATTERN***.matcher(query).matches();" instead of
"***!QUERY_PATTERN***.matcher(query).matches();"

Teamwork

As a whole, our teamwork and collaboration was a success. During our first meeting on Google Hangout we divided the work into three parts, manual testing, partitioning testing and base testing. Each team member was responsible for writing the test code for their assigned part as well as write the corresponding sections for their test. During our second meeting, we were able to wrap up what we missed and compile all our code and written sections into a single report. All this was made easy by using Google Chat and Google Hangouts for all communications and codeshare.io for all code related sharing and compiling.