Department of Computer Science

CS362 – Software Engineering II

Final Project Part B

URLValidator.isValid() Testing

Albert Nim - nima
Armand Parajon - parajona
Reed Liu - liure

3/18/2018

I. Testing Methodology

Manual Testing
One method of testing we used was manual testing. This consisted of calling the isValid() function on both valid and invalid complete URLs multiple times. The function that is used for manual testing in our program is known as testManualTest(). In order to test whether or not something is a bug, we should test URLs that are both valid and invalid to see if we get the expected results. For example, we would test a base URL that should work, such as "http://google.com" and add variations to it in order to test the parts that a URL can consist of.

Some samples of the URLs used that should be true:
"http://www.google.com"
"http://www.google.com:3234"
"http://192.168.1.1"
"http://www.google.com/mail"
"http://www.google.com/mail/test/testfolder"
"www.google.com"

Example of URLs entered in manually that should be false:
"http://www.google.comsdasdsadasd"
"http://"
"randomstring"
"http://www.google.com///mail"
"http://900.3000.1.3232"
"http://www.google.com/ mail"

Running these URLs through the *isValid()* function produces either true or false, which we can use in combination with known valid and invalid URLs to detect bugs manually.

Input Partitioning Testing
A URL is comprised of three main parts: the scheme, the authority, and the path. Three partition test functions were created, each targeted at one of these URL sections. In the test program, these functions are named: *testSchemePartition(), testAuthPartition() and testPathPartition()*.

Each of these functions use an array of ResultPair objects, ResultPair[] _partition_Pairs. This array stores pairs of test partitions and whether or not these partitions are valid or not in a boolean valid statement. Each of these partitions are then appended onto a string that represents the rest of the URL which is know to be valid. For example, if we are testing the scheme "https://", then we would append the correct URL "www.amazon.com" to the scheme and test the validity of the entire URL. An example is shown in the figure below.

```
// Test partition: path
public void testPathPartition() {

   System.out.println("PARTITION TESTING: Path Partition.");
   UrlValidator urlVal = new UrlValidator( schemes: null,  authorityValidator: null,  options: UrlValidator.ALLOW_ALL_SCHEMES

   String beforeAuth = "http://www.amazon.com";

   ResultPair[] pathPairs = {new ResultPair( item: "/home",  valid: true),
           new ResultPair( item: "/home/index",  valid: true),
           new ResultPair( item: "/home/index/",  valid: true),
           new ResultPair( item: "/home/index.html",  valid: true),
           new ResultPair( item: "/\\",  valid: false),
           new ResultPair( item: " ",  valid: true),
           new ResultPair( item: "/home//",  valid: false),
           new ResultPair( item: "/home/index.html//",  valid: false),
           new ResultPair( item: "/..",  valid: false)};

   for (int i = 0; i < pathPairs.length; i++) {
      String testUrl = beforeAuth + pathPairs[i].item;
      boolean testValid = !pathPairs[i].valid;
      if (testValid == urlVal.isValid(testUrl)) {
         System.out.println("PASS: " + testUrl + " is " + testValid);
      }
      else {
         System.out.println("FAIL: " + testUrl);
      }
   }

   System.out.println("\n");
}
```

Fig. 1: Partition Test for Paths

The test function will display to console if the result of calling isValid() on the URL matches the valid parameter in ResultPair[].

Some schemes that are tested include:
“http://”, “ftp://”, “://”, and “ “

Some authorities that were tested include:
“255.255.255.255”, “300.300.300.300”, “amazon.com”, “a.a.a.a.a.a”, “www.?=.com”, “//” and “localhost“

Some paths that were tested include:
“/home”, “/home/”, “/home/index”, “/..”, “home//” and “ “

Programming Based Testing

Our programming based tests were focused on randomization. Similar to how a URL was dynamically created at run time in the UrlValidatorTest of the correct version, our tests created a dynamic URL for testing, but with slightly more randomization. Two units were created - one for testing correct URLs and one for incorrect URLs

```java
//Test if correct domain name URLs are valid
public void testIsValidCorrectUrlsDNS()
{
   UrlValidator urlVal = new UrlValidator( schemes: null,  authorityValidator: null, UrlValidator.ALLOW_ALL_SCHEMES);
   int testCount = 2000;
   int passCount = 0;
   String[] randomDomains = generateRandomWords(testCount);
   String[] randomCorrectPrefixes = generateRandomCorrectPrefixes(testCount);
   String[] randomCorrectEndings = generateRandomCorrectEndings(testCount);

   for (int i = 0; i < testCount; i++) {
      String testUrl = randomCorrectPrefixes[i] + randomDomains[i] + randomCorrectEndings[i];

      if (urlVal.isValid(testUrl)) {
         //System.out.println(testUrl + " PASS"); //uncomment to show url
         passCount++;
      }
      else {
         //System.out.println(testUrl + " FAIL"); //uncomment to show url
      }
   }
   String resultString = String.format("UNIT TEST (CORRECT URLS): %d of %d tests passed.", passCount, testCount);
   System.out.println(resultString + "\n");
}
```

Fig. 2: Unit Tests for Correct URLs

```java
//Test if correct domain name URLs are valid
public void testIsValidIncorrectUrlsDNS()
{
   UrlValidator urlVal = new UrlValidator( schemes: null,  authorityValidator: null, UrlValidator.ALLOW_ALL_SCHEMES);
   int testCount = 2000;
   int passCount = 0;
   String[] randomDomains = generateRandomWords(testCount);
   String[] randomCorrectPrefixes = generateRandomCorrectPrefixes(testCount); //incorrect prefixes crash the program
   String[] randomIncorrectEndings = generateRandomIncorrectEndings(testCount);

   for (int i = 0; i < testCount; i++) {
      String testUrl = randomCorrectPrefixes[i] + randomDomains[i] + randomIncorrectEndings[i];

      //URL should be invalid
      if (!urlVal.isValid(testUrl)) {
         passCount++;
         //System.out.println(testUrl + " PASS"); //uncomment to show url
      }
      else {
         //System.out.println(testUrl + " FAIL"); //uncomment to show url
      }
   }

   String resultString = String.format("UNIT TEST (INCORRECT URLS): %d of %d tests passed.", passCount, testCount);
   System.out.println(resultString + "\n");
}
```

Fig. 3: Unit Tests for Incorrect URLs

Both unit tests have helper functions to generate random domains, prefixes, and endings (e.g. *generateRandomCorrectPrefixes*()). The variable testCount allows the user to specify the amount of URLs that they want to test. The main test loop then takes place, creating a dynamic and random URL, and passes that URL into UrlValidator's *isValid()* function.

A sample execution with 10 random tests for correct URLs yields the following:

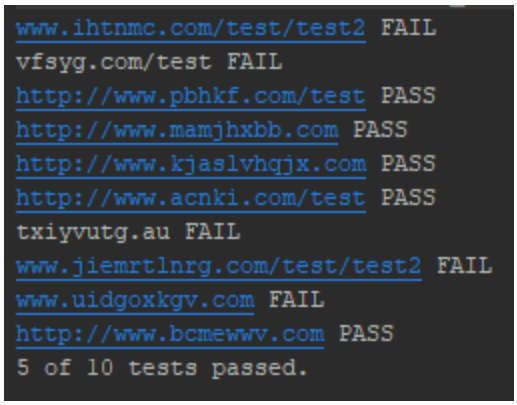

Fig. 4: Example Unit Tests after Run

In the full simulation with 2000 tests for each unit test, the following results were received:

Correct URLs:

```
746 of 2000 tests passed.
```

Incorrect URLs:

```
1516 of 2000 tests passed.
```

Not passing tests were an indication of a failure. Another failure was triggered when trying to use incorrect prefixes, which triggered an IllegalArgumentException. This is specifically covered the bug reports and debugging sections.

II. Bug Reports

Bug ID 001: Port Causes Errors
Summary: Valid URLs with valid port numbers are not being reported as correct URLs
Description: For normal URLs, you can add a colon + port number after the domain name in order to connect to it. However, any sort of port number added to a URL causes *isValid()* to return false.
Test Cases: The test cases that detected this were manual tests, which took the base URL of "http://google.com" and added a port after this. An example of this is:
`System.out.println(urlVal.isValid("http://www.google.com:3234"));`
Point of Failure: Line 318 - 321

```
if ("http".equals(scheme)) {// Special case - file: allows an empty authority
    if (authority != null) {
        if (authority.contains(":")) { // but cannot allow trailing :
            return false;
        }
    }
```

Fig. 5: Point of Failure for Bug 001

Bug ID 002: Schemes Other than http:// Crashes the Program
Summary: Any scheme other than http:// will crash the IsValid() function
Description: Using a scheme other than http:// will crash isValid(). For example, ftp:// or https:// will actuate the bug.
Test Cases: The scheme partition and manual tests caught this example. In the scheme partition test, using ftp:// or https:// in the whole URL will crash the program.
Point of Failure: RegexValidator() line 120.
Code Causing Failure: Line 317 in UrlValidator, if the scheme is not equal to http://, then the program calls !isValidAuthority() on line 326. isValidAuthority calls the function DomainValidator.unicodeToASCII() as a static class if the source UrlValidator object has a null AuthorityValidator argument in line 392.

A static object still constructs a copy of its member class functions and variables. The DomainValidator class instantiates the RegexValidator twice using non-null String[]s has its single parameter.

```
private final RegexValidator domainRegex =
        new RegexValidator(DOMAIN_NAME_REGEX);
/**
 * RegexValidator for matching a local hostname
 */
// RFC1123 sec 2.1 allows hostnames to start with a digit
private final RegexValidator hostnameRegex =
        new RegexValidator(DOMAIN_LABEL_REGEX);

/**
```

Fig. 6: First Point of Failure for Bug 002

A chain of constructors then calls a version of the RegexValidator class constructor where it takes a String[] and a boolean as its parameters. There is a line in this constructor where if the String[] parameter is != null, then it throws an exception error.

Due to this, the program will throw a "Regular expressions are missing" exception every time "http://" is not the scheme of the URL.

```
public RegexValidator(String[] regexs, boolean caseSensitive) {
    if (regexs != null || regexs.length == 0) {
        throw new IllegalArgumentException("Regular expressions are missing");
    }
    patterns = new Pattern[regexs.length];
    int flags =  (caseSensitive ? 0: Pattern.CASE_INSENSITIVE);
    for (int i = 0; i < regexs.length-1; i++) {
        if (regexs[i] == null || regexs[i].length() == 0) {
            throw new IllegalArgumentException("Regular expression[" + i + "] is missing");
        }
        patterns[i] =  Pattern.compile(regexs[i], flags);
    }
}

/**
```

Fig. 7: Second Point of Failure for Bug 002

III. Debugging

Bug ID 001: Port Causes Errors

From the multiple unit tests that involved ports, it was clear that there was an error in processing the ports due to all URLS, even the valid ports failing. Causing the *isValid()* function to fail multiple times was an example of Agan's Principle #2, Make It Fail as we got many valid functions to get a false result everytime we used a colon.. First, we looked at the *isValid()* function in order to see where the ports would be processed inside the function. i*sValid()* utilizes multiple false statements in order to detect different parts of the problem. By putting print statements to check for which one was causing a colon to return false, we determined that it was the special case where http was the scheme. It contains an if-statement that returns false everytime that a colon is in the authority. In the correct version of the file, the scheme is "file" rather than http.

Bug ID: Schemes Other than http:// Crashes the Program

To diagnose this bug, we heavily relied on breakpoints and step-into's + step-throughs using IntelliJ's debugger tool. We felt using manual testing for this bug was a good choice because it allowed us more control over our testing. One example of this was that we substituted various schemes such as FTP and HTTPS one at a time, exemplifying Agan's Principle #5 Change One Thing at a Time.

With those schemes, we noticed that an IllegalArgumentException was thrown by the regex validator class for schemes other than "http", thus also exemplifying Agan's Principle #2 Make It Fail.

```
httpd://iqlseaeg.com/..

java.lang.ExceptionInInitializerError
    at UrlValidator.isValidAuthority(UrlValidator.java:393)
    at UrlValidator.isValid(UrlValidator.java:327)
    at UrlValidatorTest.testIsValidInorrectUrlsDNS(UrlValidatorTest.java:75) <10 internal calls>
    at junit.textui.TestRunner.doRun(TestRunner.java:116) <1 internal calls>
    at junit.textui.TestRunner.doRun(TestRunner.java:109) <4 internal calls>
Caused by: java.lang.IllegalArgumentException: Regular expressions are missing
    at RegexValidator.<init>(RegexValidator.java:121)
    at RegexValidator.<init>(RegexValidator.java:96)
```

To get to the bottom of this exception, we started analyzing the root level function call of *IsValid()* from an instance of the UrlValidator class (with a null AuthorityValidator argument). Following the order of execution, all non-http schemes call the function isValidAuthority. Once in that function, we put a watch on the authorityValidator instance member to make sure it was null.

```
    if (authorityValidator != null && authorityValidator.isValid(authority)) {
        return true;
    }
```

AuthorityValidator stayed null, and execution proceeded to line 393 with a call to a static class method:

```
393             final String authorityASCII = DomainValidator.unicodeToASCII(authority);
```

As detailed in the bug report, calling a static class method requires that the static class instantiate each of its members, most importantly in this case two instances of the RegexValidator class with a non-null argument.

```
private final RegexValidator domainRegex =
        new RegexValidator(DOMAIN_NAME_REGEX);
/**
 * RegexValidator for matching a local hostname
 */
// RFC1123 sec 2.1 allows hostnames to start with a digit
private final RegexValidator hostnameRegex =
        new RegexValidator(DOMAIN_LABEL_REGEX);

/**
```

Through a chain of constructors, the following RegexValidator class constructor gets called and throws an exception on line 121.

```
119 @   public RegexValidator(String[] regexs, boolean caseSensitive) {
120         if (regexs != null || regexs.length == 0) {
121             throw new IllegalArgumentException("Regular expressions are missing");
122         }
```

The bug itself is in the expression "regexs != null", it should instead be "regexs == null" as seen in the correct version. Both conditions on line 120 would evaluate to false and execution would proceed without exception had this bug not been here, and hence all schemes other than http lead to this exception.

IV. Teamwork

Armand - Programming Based Testing
Reed - Partition Programming
Albert - Manual Programming

This group collaborated by communicating through Google Hangouts video/voice chat throughout multiple weekly meetings. All members were present in the multiple meetings and contributed to the discussions and the final product. Each member of the group worked on a specific portion of the methodology testing and everyone collaborated on the bug reports, debugging and teamwork sections throughout the project.

The group shared a Google Doc file for the final product. In addition, the group used GitHub to share code with each other and work on debugging with. The overall collaboration went smoothly and the final product was completed on time.

V. Other Information

The test class for the method is located at the following URL:
https://github.com/liureOSU/CS362-004-W2018/tree/liure-finalproject/projects/liure/URLValidatorInCorrect/test