Chido Nguyen (nguychid)
Eric Newton (newtoner)
Alex Johnson (johnsal8)
CS 362
Final Project: Part B
November 28, 2018

Testing Methodology

Manual testing

For our manual testing, we created an array of strings wherein each string represented a candidate URL. Then from within a for-loop we called isValid() on each URL. The output of this function call was then printed to standard output.

```
for (String u : urls) {
    if (urlValidator.isValid(u)) {
        System.out.println(u + " is valid");
    } else {
        System.out.println(u + " is invalid");
    }
}
```

As this was manual testing, we did not exhaust the possibilities of different types of strings to test. We did, however, cover some basic inputs like varying cases, special characters, etc.

We found that the URL validator correctly identified many valid URLs, like http://www.google.com/. And it correctly identified invalid URLs, like not-a-url. However, we discovered many false positives – URLs that were identified as valid when they weren't. These included http://notaurl and http://watch-everything-go-off-the-rails, and any other permutation that begins with the string "http://".

Interestingly, the URL validator in Google Docs also incorrectly identifies these as valid URLs.

Input partitioning

For input partitioning tests we were going to examine the scheme and authority parts of a URL. We copied the scheme of having ResultPair of a string URL part and a boolean value. We then change the tested variable via for loop that iterates through the specific partition we're testing while keeping the other URL parts static on a valid string value. For instance, if we were testing scheme x it would be changed via each for loop iteration and we use "www.google.com" as the static valid variable: x + "www.google.com".

```
for(ResultPair duo: testUrlScheme ) {

                    if(duo.valid) {
                            assertTrue(urlVal.isValid(duo.item + "www.google.com"));
                    }
                    else
                            assertFalse(urlVal.isValid(duo.item + "www.google.com"));

            }
```

We were going to use the built-in assertTrue and assertFalse to test each URL permutation to see if it came out to be what we expected. To regulate which is used we check the validity of the result pair, for valid ones we assert true and invalid ones we assert false.

The idea behind the input partitioning test is to generate ResultPairs where you have pairs of items and values. The items are string parts of a URL and values are either true false booleans designating if the item is a legal URL part. We can use these arrays of ResultPairs to either randomly generate inputs or access them directly as needed.

While generating testScheme codes we were going to go with a for-loop. It would iterate through the array of schemes and append a legal "www.google.com" to the end of each scheme. In theory, it should work, but we found that this strategy resulted in many errors.

We moved away from the for-loop and created direct assert lines with direct assert to each scheme part. This still failed the test. We proceeded to comment out all but 1 test per run to see if we could isolate the cause. It turned out that there is a possible error in processing the URL. The first scheme "http://" works but every subsequent scheme part failed.

We tried swapping out all scheme parts with http:// and the test passed. This is akin to delta debugging: "trapping the wolf" by halving its area for each iteration approach.

We took the same approach for testing Authority. We avoided the for-loop and went directly into repetitive assert statements. We assumed this test would've been more straightforward if we could dictate the scheme and keep it to "http://". But errors came up again. Basic debugging via eliminating variables suggested that isValid() might be flawed.

The program was erroring out/failing due to assertFalse() usages. Commenting out the assertFalse statements ran the program without a hitch. Adding them back generated errors, the thing about these errors is that these URLs should be registering as false when checked via isValid(). We swapped the assertFalse to assertTrue and the program run successfully.

Programmatic testing

For the unit testing of isValid (testIsValid() function) we took a similar approach to the original Apache tests. The main difference was that we replaced the incrementing test indices with a

series of for loops. This can effectively address the same range of tests as the original method, though here we use a more limited range of inputs resulting in 939 tests.

```
// Iterate through every test URL combination
       for(int a = 0; a < testUrlScheme.length; a++) {
          for(int b = 0; b < testUrlAuthority.length; b++) {
               for(int c = 0; c < testUrlPort.length; c++) {
                  for(int d = 0; d < testPath.length; d++) {
                       for(int e = 0; e < testUrlQuery.length; e++) {
                        ...
```

The valid field of each of these components is used to create an expected result (whether the URL should be valid or not). The combined URL created from each of these components is then tested with the urlVal.isValid() method and compared against the expected result.

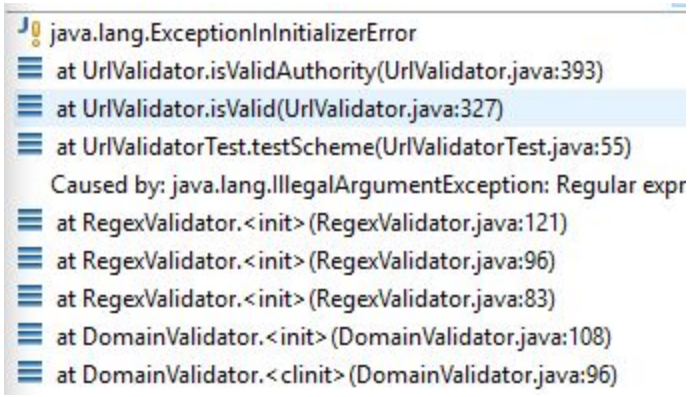
```
boolean expectedResult = testUrlScheme[a].valid && testUrlAuthority[b].valid &&
testUrlPort[c].valid && testPath[d].valid && testUrlQuery[e].valid;

String url = testUrlScheme[a].item + testUrlAuthority[b].item + testUrlPort[c].item +
testPath[d].item + testUrlQuery[e].item;

if (expectedResult == true) {
       assertTrue(urlVal.isValid(url));
       System.out.println("Expected Result: true  Actual Result: " +
Boolean.toString(urlVal.isValid(url)));
       System.out.println("URL: " + url);
} else {
// isValid appears to return inconsistent results with bad URLs
//assertFalse(urlVal.isValid(url));
System.out.println("Expected Result: false  Actual Result: " +
Boolean.toString(urlVal.isValid(url)));
System.out.println("URL: " + url);
 }
}
```

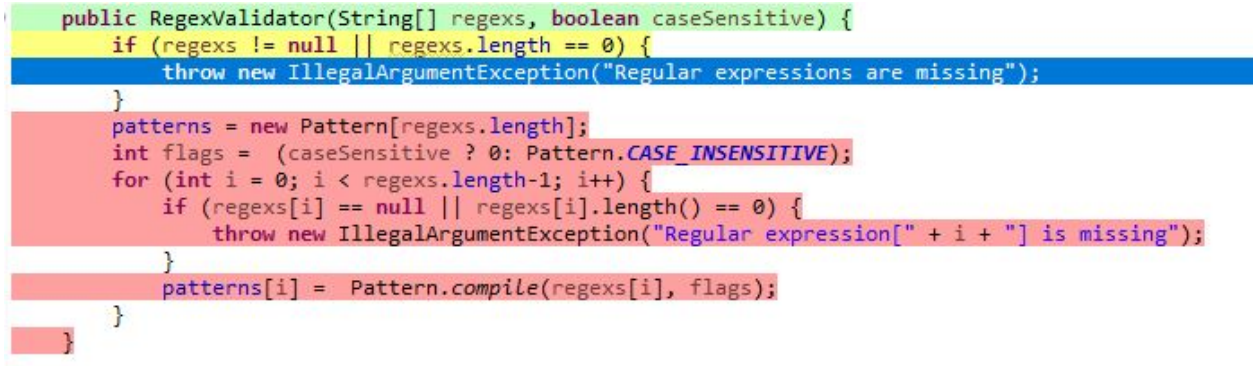
As with manual testing above, inconsistent results were returned with bad URLs. A number of false positives were found, results that should have been false but instead returned as true. For instance:

```
Expected Result: false  Actual Result: true
URL: http://www.------------.com/test1?action=view
Expected Result: false  Actual Result: true
URL: http://www.------------.com/test1?action=edit&mode=up
```

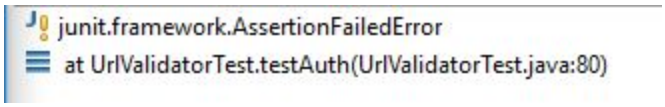
Bug Reports

Bug: PARTITION_SCHEME_001

- Failure/Error: Regular Expressions are missing (Caused by java.lang.IllegalArgumentException: Regular expressions are missing)
- The bug was found while trying to test different parts of a URL. Attempting to manually test valid URL components led to the discovery of the bug when different components were used.
- The program appears to be failing to process the regular expression properly. Anything aside from “http://” triggers this error even if the url as a whole should be valid. The trace implicates “RegexValidator” is at fault.
- 

- 

Bug: PARTITION_AUTH_001

- Error/Failure: isValid function is not detecting invalid (false) URLs (junit.framework.AssertionFailedError)
- The bug was found when expected false results were failing the assertFalse function causing the program to hang.
- The cause seems to be isValid is returning true for an invalid (false) URL structure. For example in the screenshot below “http://” is a valid component and the 3rd index component of authority is "www.----------.com" which should be false. But when the test is run with assertFalse it fails whereas if we swapped the assert to assertTrue the program does not hang. As you can see the code below line 75 and line 80 are essentially the same, except for the assert statement. This URL permutation at index 3 should be false.
- 


```
74        //This is wrong//
75        assertTrue(urlVal.isValid("http://"+testUrlAuthority[3].item));
76        assertTrue(urlVal.isValid("http://"+testUrlAuthority[4].item));
77        assertTrue(urlVal.isValid("http://"+testUrlAuthority[5].item));
78        assertTrue(urlVal.isValid("http://"+testUrlAuthority[6].item));
79        //this is right//
80      assertFalse(urlVal.isValid("http://"+testUrlAuthority[3].item));
```

Debugging

Agan's Principals

Agan's Principals are as follows:

- Understand the System
- Make it Fail
- Quit Thinking and Look
- Divide and Conquer
- Change One Thing at a Time
- Keep an Audit Trail
- Check the Plug
- Get a Fresh View
- If You Didn't Fix It, It Ain't Fixed

We found the first two items, understanding the system and making it fail, to be a critical part of this project. Indeed, understanding the system and making it fail were a critical part of the manual testing detailed above. There's was also a degree of "checking the plug" included in manual testing. Does it compile? Does a valid URL show as valid?

Getting a fresh view was made possible by having multiple people looking at the same code and checking one another's work. And since we did not fix any of the bugs we found, we consider them to be not fixed.

Tracing the http bug

As detailed above, we found that starting any URL with the string "http://" would cause the test to pass, even if the trailing contents of the URL were not valid. In order to debug this problem, we employed the built-in debugger in IntelliJ. This led us to the Matcher object in UrlValidator class.

Our suspicion was that the Regex expressions had been changed and that this was causing an invalid URL to appear as valid. However, we found the Regex expressions to be identical between the Correct and Incorrect validators, so this was unlikely to be the root issue.

A few lines beneath the Regex expression is a special case that allows any scheme starting with "http" to pass. We believe this to be the root of the bug, especially given the fact that in the correct version of the code this is meant to catch files, rather than the string "http".

Team Work!

We collaborated via Slack as our main form of communication. We each took charge of one of the three major parts to this project for testing (manual [Alex], partition [Chido], and validity [Eric]).

After completing our work we came back together to discuss our findings what errors/bugs we saw how we found them, what we thought the causes were etc. Finally, we collaborated via Google Docs and generated the final report aggregating all of our thoughts and ideas.

The GitHub repository can be found here:
https://github.com/ChidoNguyen/CS362-004-F2018/tree/partB/projects/nguychid/FinalProject/URLValidatorInCorrect