CS 362
Group Project part B
Thomas Mathew
Nicholas Wong
Enrique Fernandez

For part b, you will be provided a buggy version of URL Validator.

Use URLValidator folder under the base directory (where base dominion folder is). Create your own copy of the directory and all the files. Don't use files under the base directory.

You are provided a buggy version of URLValidator. You need to find out as many bugs as you can in this bad URLValidator. In the last assignment, I have provided the current test framework that apache commons team had to test URLValidator. We need to assume that all those tests don't exist. Your team is a testing company and client comes to you with URLValidator implementation and asks for your help to make it bug-free. You need to just concentrate on valid() method, the one that is tested in testIsValid() method of last assignment. Your task is to find out all the bugs, find out failure causes and provide bug reports. Though for this project, as long as you find out three bugs that should be sufficient. You don't need to fix any of the bugs. Developers will do it. Bug reports will contain

1. What is the failure?
2. How did you find it?
3. What is the cause of that failure? Explain what part of the code is causing it?

You can use any methodology that you learn during the class to test it. To stay consistent let us do it this way.

First just do manual testing. Call the valid method of URLValidator with different possible valid/invalid inputs and see if you find a failure. (2 points)

**We ran manual tests with a wide range of different URLs, both valid and invalid. Some of the tests we ran included URLs with resource identifiers, URLs from different countries, with or without "http://" and "https://", and with or without "www". In all cases, the isValid() method returned the correct value and so we didn't find any bugs.**

Second, come up with good input partitioning. Try to provide a varying set of inputs that partition the overall input set well. Did you find any failures? You can call valid method once or more for each partition. (3 points)

**A URL is made up of up to some significant parts, including Scheme, Authority, Path, and Query. So, we chose these four parts as our input parameters, and iterated through many different values, including characters, punctuations, and more for each of the four partitions.**

Third, do programming based testing. Write few unit test cases. You can have some sort of loop in your unit test and test different URL with each instance of the loop. Something very similar to testIsValid() but your own logic and idea. Even a single test will be sufficient if you write it like testIsValid() method. Did you find any failures? Submit your test files and test cases as part of your work under your onid/URLValidator folder. (5 points)

**We set up different arrays for each section of a url a strings. We then ran through each of the arrays, concatenated the strings and tested that string as a URL. Due to the nature of URLs, it was relatively easy to choose a range of "strange" values to push the limits of isValid(), one could call these edge cases. This also helped keep the number of tests relatively low and thus test execution time was insignificant. There was a mix of failures and successes with the test depending on parameters.**

When you find out any failure, debug using Eclipse debugger or any other tool and try to localize its cause. Provide at what line/lines in what file the failure manifested itself. Did you use any of Agan's principle in debugging URLValidator? (5 points)

**Bug #1: We found one repeatable bug when it comes to the Query. When a tested URL had no query, isValid() performed as expected, namely it would return "true" for valid URLs, and "false" otherwise. But, when a query was included in a URL, such as "?action=view", isValid() would always return false. The bug in question was in line 446 of UrlValidator.java:**

```
   protected boolean isValidQuery(String query) {

      if (query == null) {
         return true;
}

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

   }
```

**We found this bug by taking out partitions of the URL one by one until we found that taking Queries out would result in a higher number of passed URL strings. We then looked into the test Queries function, and that revealed this bug.**

**Bug #2: Another bug we found was within the authority. We ran tests on a number of different authority parameters, and found that when the authority contained 3 or more periods, we would return false, even if the URL was valid. We did some digging and found that at line 158 or UrlValidator, this line of code limits the authority to having no more than 3 "parts", separated by periods:**

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

**So, in any case where there were 3 or more periods, the authority would be divided into at least 4 parts, and the test would return false, even though that URL would be valid, assuming other parts are valid.**

**Bug #3:A third bug we got was found in the InetAddressValidator.java file on line 96:**

```
if (iIpSegment > 255) {
        return true;
```

**This checks the format of ipv4 addresses. We found this bug with test cases that were invalid against the ipv4 format, but returned true anyways. This was anything that had more than 1 byte in an address with numbers. An example we used for our test case was 256.256.256.256, in which case all are 1 bit larger than 1 byte. This test case passed, even though it shouldn't have.**

(5 points) Provide and submit a report called **ProjectPartB.pdf** in canvas. You need to provide following details in the report. Clearly, mention your methodology of testing. For manual testing, provide some of your (not all) urls.

**Our methodology involved combining a wide range of pre-defined parameters to create URLs, then calling isValid() on the resulting URL. We set up four different arrays containing values for the Scheme, Authority, Path, and Query. Then we set up nested "for" loops to iterate through the arrays, testing all possible combinations of values. Here are just a few sample URLs we used:**

| URL | Expected result | Actual Result |
|---|---|---|
| https://www.google.com.et/?gws_rd=ssl | pass | fail |
| http://256.256.256.256 | fail | pass |
| http://www.amazon.com | pass | pass |

Also, mention how did you work in the team? How did you divide your work? How did you collaborate?

**One of us was sitting in front of the computer coding while the other two looked over the shoulder to make sure there weren't any errors (peer programming). Another of us made up specific test cases that we wanted for each section (unit testing, manual testing, etc). The other group member made sure whether or not the test cases should be valid, or invalid and test it against the output from the test functions we ran.**

**We mostly used Google Hangouts to work together as meetup locations weren't available to some group members. This proved to be very useful because we could fit the meetup with anyone's schedule and it was far more convenient than trying to set a time to meet up physically.**