SOFTWARE ENGINEERING II (CS_362_400_W2019)

Project Part B

Team Members

Name	ONID
Arthur Liou	lioua
Chu Wang	wangch7
Sean Lo	lohs

Part-B, Testing URL Validator: Instructions

1. (CHU) Do manual testing. Call the isValid() method of URLValidator with different possible valid/invalid inputs. Document any failures that you find. (15 points)
2. (ARTHUR) Do programming based testing. Write several unit test cases that implement various methods to test different URLs. Document any failures that you find. (20 points)
3. (SEAN) Do random testing. Come up with good input partitioning. Try to provide a varying set of inputs that partition the overall input set well. Document any failures that you find. (15 points)
4. Submit a report called ProjectPartB.pdf in Canvas that contains the following Sections: (40 points)
    - Methodology Testing (15 points) Write in detail about each methodology you used for testing (manual, partition and programming based).
        - Write in detail the test functions that implement each methodology.
        - For manual testing, provide your URLs.
        - For random testing, describe your partitions and list some of the URLs that represent each partition.
    - Bug Report (10 points)
        - ▪ Write a bug report for each of the bugs you found.
          • Describe the failure.
          • Describe how you found it terms of the test case that detected it.
          • Describe the cause of the failure. Explain what part of the code is causing it and provide a screenshot of the faulty code.
    - Debugging (10 points)
        - ▪ When you find a failure, debug it using Eclipse/IntelliJ debugger or any other tool and find its cause.
          ▪ Did you use any of Agan's principle in debugging URLValidator?
          ▪ Describe the failure and your debugging details for each bug. Provide the line numbers and file names where the failure manifested itself.
    - Team Work (5 points)
        - Write about how worked as a team? How did you divide your work? How did you collaborate? List the contributions of each of your team members.
5. The class GitHub repository (10 points)
    - Submit your unit tests/random tests on GitHub under projects/your-onid/URLValidatorInCorrect/ folder.

- Create a new branch of your repository called "youronid-finalproject" that contains your final submission. This branch must be created before the due date to receive credit.

Methodology Testing (15 points):

Write in detail about each methodology you used for testing (manual, partition and programming based).

Manual Testing

We looked at the structure of valid url. Then, manually created some urls that are valid, as well as some that are invalid, run isValid() function and System.out.print() to see if the result matches with expected outcome.

URLs tested:

- (NULL)
- http::/manualTest_1.com/lol
- http://manualTest_1.com/lol
- 3ht://manualTest_1.com/lol
- https://manualTest_1.com:1080/lol
- ftp://manualTest_1.com:1080/lol
- 3ht://aaa/lol
- http://manualTest_1.com//lol
- http://manualTest_1.com/lol/#page=123

Programming Testing

For our programming-based testing, we looked at the various java files and functions within each file to determine a way to suitably create a few unit tests.

1. In testEdgeCase(), we focused on building a miniaturized test suite consistent of a few unit tests to test some edge cases, such as null (from manual testing above), an empty string/URL, and pure numbers. We thought this was an important step to start with (from a programming testing perspective) because looking for and considering edge cases for the program is an aspect of development every developer and software engineering team needs to consider. While these edge cases may not be used often in application testing and actual usage, the times they are entered can sometime be disastrous in real-world software applications. Hence, our decision to focus on edge cases from a programming tester perspective came first.
2. In testBasicValidator(), we replicated replicate the intended usage of the UrlValidator.java file via the Example of Usage in the usage comments at the top. This usage specifically touches upon schemes. In this case, the usage played around with scheme manipulation, where the default valid schemes are "http", "https", and "ftp". But in this test case, we specified only "http" and "https", leaving out ftp and allowing us to verify that schemes is working correctly.

Random Testing

Random testing is all about partitioning the total inputs into distinct categories and then testing combinations of different partitions together. When I was analyzing the structure of URLs, I decided to take a look at the correct validator test as well as the validator structure. From these information, I have partition a website URL into the following structure,

URL = a scheme + a authority + a port and a path

Within this categorization, each partition can be grouped into a true or a false section. A true or correctly outputted URL would have the format of trueScheme + trueAuthroity + truePort + truePath and a false URL would have the format of falseScheme + falseAuthority + falsePort + falsePath.

Having decided on the methodology of my partitioning of website URLS, I then declared many arrays of string variables of each partition. Once this done, I utilized Java's Random utility to randomly select a string from each partition and then combining them together. By doing some, I can guarantee that my selection each time will be either TRUE or FALSE. I then test the supposedly TRUE statement to see if it is in fact a TRUE statement. Vice Versa, I do the same for the false statement as well, but this time, checks if they're false.

Examples of URLs for Random Testing

True URLs that tested true	**False URLs that tested false**
http:go.1aa:65a/..//file	http:256.256.256.256:999999999999999999/..//file
http:/256.256.256.256:-1/../	http:/.1.2.3.4:65a/..
http:256.256.256.256:999999999999999999/..//file	http:.aaa:-1/..
http:go.a1a:65a/..//file	http:/go.a:65a/..//file
http:go.a1a:65a/../	http:go.a1a:65a/../

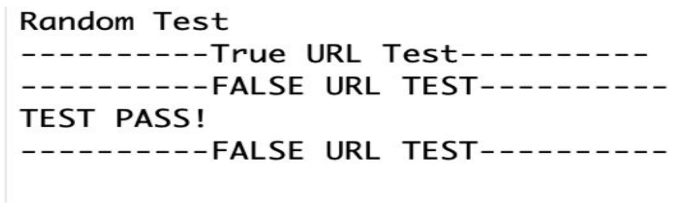
Bug Report (10 points):

Note: Our group decided do the revised Part B version. Introducing the two bugs reported below.

Bug #1

Title: Logical error in isValid() of urlvalidator.java
Product: UrlValidator.java of Dominion
Classification: This is a high priority as it is logical error that prevents accurate result to be outputted.
Platform: Eclipse for Java
Can it be reproduced?: Yes, it can be reproduced by removing ! from !isValidScheme function within isValid function on line 313 of urlvalidator.java

1. **Failure Description:** Whenever I compile my random testing generator, it does not consistently output expected results. It would occasionally output a few lines before it crashes and stop working. Below is a screenshot of my random testing failing to compute true url test.



```
Random Test
----------True URL Test----------
----------FALSE URL TEST----------
TEST PASS!
----------FALSE URL TEST----------
```

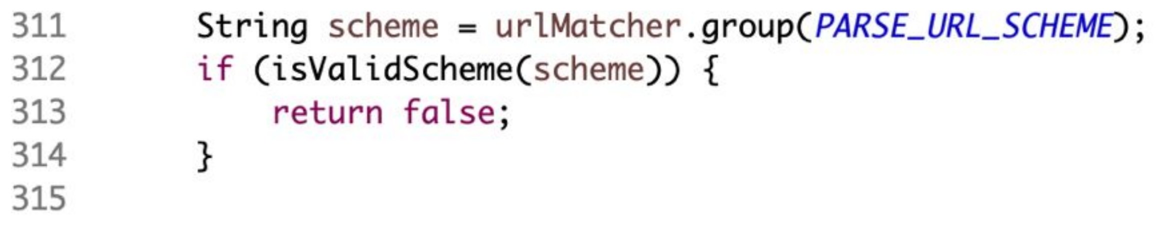
Expected Results: All random tests should generate result whether a test passed or failed.
Actual Results: True URL tests sometimes return test fail information but mostly omit and skips result reporting.

2. Describe how you found it terms of the test case that detected it.

My first clue that this was potentially caused by isValidScheme due to the fact that my test case True URL test never returns true whenever random test is ran yet it also does not return TEST FAIL either. This hinted that there is potentially an error regarding validation.

3. .Describe the cause of the failure. Explain what part of the code is causing it and provide screenshot of the faulty code.

I looked through my code and verified that all of my TRUE partitions are valid when compared to the correct validator test file provided. Judging by this, it was evident that within isValid() function, there must be a logical error in regards the return whether a particular partition was valid. With Scheme being the first one tested, it can be observed that isValidScheme returns false when it is still valid. The correct condition statement should've been !isValidScheme.



```
311        String scheme = urlMatcher.group(PARSE_URL_SCHEME);
312        if (isValidScheme(scheme)) {
313            return false;
314        }
315
```

Can it be reproduced?: Yes, it can be reproduced by removing ! from !isValidScheme function within isValid function on line 313 of urlvalidator.java

__Bug #2__
Title: Edge Case Tester Runs into a Null Pointer Exception (NPE)
Product: UrlValidator.java of Dominion
Classification: This is a low (maybe medium) priority as it is a development error where edge cases where not fully considered and caught. This is an issue that is lower impact (ie: how many users are actually going to enter null as their input), as other bugs (such as Bug #1 above) are high priority.
Platform: Eclipse for Java
Can it be reproduced?: Yes, this error can be reproduced. Either comment out or remove, the edge case guard in the isValid() method in and/or around L301-303.
Description:
1) Failure
Expected Results:
"Unit Test - Edge Case: Null
Passed! Found a null URL to be inValid"
Actual Results:
"Unit Test - Edge Case: Null
…" (where nothing was outputted)

Failure Trace

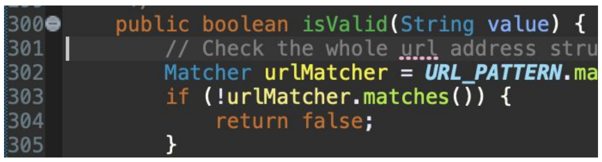
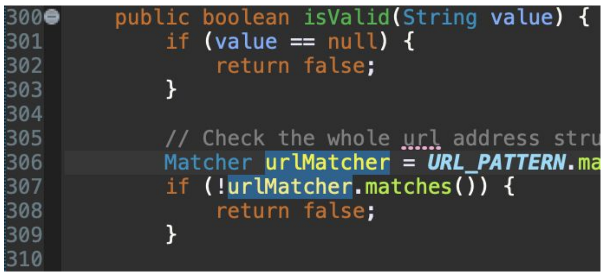
```
"java.lang.NullPointerException
        at java.util.regex.Matcher.getTextLength(Matcher.java:1283)
        at java.util.regex.Matcher.reset(Matcher.java:309)
        at java.util.regex.Matcher.<init>(Matcher.java:229)
        at java.util.regex.Pattern.matcher(Pattern.java:1093)
        at UrlValidator.isValid(UrlValidator.java:306)
        at UrlValidatorTest.testEdgeCase(UrlValidatorTest.java:37)
…."
```

2. Describe how you found it terms of the test case that detected it.

In the specific test case that detected it, testEdgeCase(), one of the unit test was testing null inputs. When JUnit threw out the NPE, it indicates that somewhere there was a null causing the unit test to crash.

3..Describe the cause of the failure. Explain what part of the code is causing it and provide screenshot of the faulty code.

The cause of the NPE, there isn't a edge case that captured null values. See below, in Line 301-303, the introduced bull was to remove the null edge case capture.

Code - Bug is introduced	Code - Original / Correct
	

```
300     public boolean isValid(String value) {
301         // Check the whole url address stru
302         Matcher urlMatcher = URL_PATTERN.ma
303         if (!urlMatcher.matches()) {
304             return false;
305         }
```

```
300     public boolean isValid(String value) {
301         if (value == null) {
302             return false;
303         }
304
305         // Check the whole url address stru
306         Matcher urlMatcher = URL_PATTERN.ma
307         if (!urlMatcher.matches()) {
308             return false;
309         }
310
```

Summary of Bugs Introduced:

- Bug introduced on Lines 301-303, the null edge case guard was removed
- Bug introduced on Line 312 (the ! was removed), where the correct code is
    - ```if (!isValidScheme(scheme)) {```

Debugging (10 points)

For Debugging Bug #1, the failure was a logical error that led to incorrect output of results of the random testing function. AGAN's principles were applied by fully understanding and analyzing the code first before dividing and testing each section of the code. Notes of each round of changes were recorded in order to ensure that changes made won't affect other functions. Using AGAN's principles and intelliJ's ultimately allowed me to isolate the source of the problem. Line Number 311 was the source of the problem within UrlValidator.IsValid function of the UrlValidator.java file.

For Debugging Bug #2, the failure was a null pointer exception (NPE), and when debugging this, we use some of Agan's principles to debug and also used the Eclipse debugger to trace the origin of the error. In the failure trace above, we could see these following two lines, which indicates which testing function and method failed and at which line. Line number 306 in UrlValidator and L37 in UrlValidatorTest.

at UrlValidator.isValid(UrlValidator.java:306)
at UrlValidatorTest.testEdgeCase(UrlValidatorTest.java:37)

<u>**Team Work: (5 points)**</u> Describe the contribution of each of the team member.

- Arthur Liou:

  - Q2, and it's respective Q4
  - Bug Report #2
  - Proofread before submission
  - Github/Github URLs/Submission
- Sean Lo:
  - Q3, and it's respective Q4
  - Bug Report #1
  - Debugging #1
  - Proofread before submission
- Chu Wang:
  - Google Doc Setup
  - Q1, and it's respective Q4
  - Proofread before submission

Github URL:

- https://github.com/artliou/CS362-W2019/tree/lioua-finalproject/projects/lioua/FinalProject
- https://github.com/artliou/CS362-W2019/blob/lioua-finalproject/projects/lioua/FinalProject/URLValidatorInCorrect/src/UrlValidator.java
- https://github.com/artliou/CS362-W2019/blob/lioua-finalproject/projects/lioua/FinalProject/URLValidatorInCorrect/test/UrlValidatorTest.java