Oregon State - CS362-004

Final Project

APACHE COMMONS URLVALIDATOR

Jon-Eric Cook, Amelia Le, Jung Eun Hwang

Part A

Review of Existing Test Case(s)

1. Explain testIsValid Function of UrlValidator test code.

Line 85: testIsValid takes 2 parameters, an array of objects and a long integer

- objects form URL
- integer signifies valid url formatting

Line 86: create a new UrlValidator object urlVal with the formatting scheme passed in

Line 87: assert that the formatting scheme allows http://www.google.com

Line 88: assert that the formatting scheme allows http://www.google.com/

Line 89-93: format console log

Line 94-127: do while URL parts exist

Line 95: create new stringBuffer object testBuffer

Line 97-102: combine passed in array objects

Line 101: check the validity of each array object, store the result in expected

Line 104: stringBuffer object to string

Line 105: boolean check if urlVal returns the URL as valid, according to the passed in validation option

Line 107-108: if 105 is true, print URL

Line 109: compare expected and result

- The rest of the function handles printing
- Function also calls setUp() to make each index of the object array

2. Give how many total number of the URLs it is testing.

- testUrlScheme(line 341): 9
- testUrlAuthority: 19
- testUrlPort: 7
- testPath: 10
- testUrlQuery: 3
- Total number of Urls it is testing: 9*19*7*10*3 = 35,910 urls.

The above calculated value of 35,910 is the correct number of urls that should be tested. In reality though, when the URLValidatortest.java file is ran, it only tests 31,920 urls. This is due to a bug in the code that was already addressed in the following bug report:

https://issues.apache.org/jira/browse/VALIDATOR-418?jql=text%20~%20%22incrementTestPartsIndex%22

The code we were provided in the URLValidatorCorrect folder does not have the patch that fixed the above mentioned bug.

3. Explain how it is building all the URLs.

Line 413 creates an array of possible URL parts. Each part consists of an array of ResultPair objects. Each ResultPair object contains a string and a boolean value. The string is a part of the URL. The boolean value indicates whether the string is valid or not.

```
Object[] testUrlParts = {testUrlScheme, testUrlAuthority, testUrlPort,
testPath, testUrlQuery};
```

testUrlParts is an array of ResultPair arrays.

testUrlScheme is an array of ResultPair objects.

One possible pair in testUrlScheme is “ResultPair("http://", true)” which indicates that “http://” is a valid URL part.

The for loop in testIsValid iterates through each URL part: scheme, authority, port, path, and query. The while loop in testIsValid calls the incrementTestPartsIndex function to loop through each possible string and boolean pair for each ResultPair array, for the number of elements in the current array of the for loop.

The incrementTestPartsIndex function adds 1 to the lowest order part. Starting at the last element in testPartsIndex, incrementTestPartsIndex increments through each object in the ResultPair array, resetting to 0 once all ResultPair elements have been iterated through. incrementTestPartsIndex then decrements to the next testPartsIndex element and repeats the iteration through each ResultPair array.

The combination of the for loop and while loop in the testIsValid function tests every possible permutation of URL parts.

4. Give an example of valid URL being tested and an invalid URL being tested by testIsValid() method.

Valid url: http://www.google.com:80/test1

Invalid url: ://test1?action=edit&mode=up

5. Do you think that a real world test (URL Validator's testIsValid() test in this case) is very different than the unit tests that we wrote (in terms of concepts & complexity)?

The real world test and unit tests we wrote for Dominion project where we checked if the program would return the correct value from what we gave it are similar. Both tests used assertions and compared the premade expected results with the output. testIsValid() knows what it considers correct and knows what it considers incorrect, so it creates a set of permutations from lists of correct URL parts and checks the validity of the URL.