Oregon State University

CS 362-400 SOFTWARE ENGINEERING II
Winter 2018
FINAL PROJECT
Jonathan Grocott (grocottj), Zheng Zhu (zhuzhe),
Drew Wolfe (wolfedr)

Final Project: Part A

Review of Existing Test Case(s)

---

1. Explain *testIsValid* Function of *UrlValidator* test code. It is available under *URLValidatorCorrect* folder.

Summary:

The `testIsValid()` method is the unit test driver for `public boolean isValid(String value)`. It is being used to verify that a given URL is a valid URL input. It accomplishes this by calling an overloaded version taking two parameters. One is an array of objects, and the other is a long int data type value which represents the associated options (in this case `allowAllSchemes`). The method iterates through all the possible URLs that are composed of five sections: scheme, authority, port, path and query. The `ResultPair[]` arrays are used to provide the strings along with the expected validity values of individual sections of the URL input. If any part of the URL is invalid, the boolean `expected` is set to false. The `testIsValid()` method then calls the `isValid` method of the `UrlValidator` class on the composite URL to test its validity and compares the result with the `expected` value. The `testIsValid()` method asserts that both results should match; otherwise, an exception is thrown. It also prints out all the URLs that are considered valid by the `isValid` method. After the `testIsValid()` method executes the overloaded version with parameters it calls the setUp() method, which resets the `testPartsIndex[]` values.

Line Number	Code	Description
83	`public void testIsValid(Object[] testObjects, long allowAllSchemes){`	Calls the *testIsValid* method, passing an array *testObjects* and a `long int` indicating which types of URL schemes are allowed
84	`UrlValidator urlVal = new UrlValidator(null, null, allowAllSchemes);`	Creates an `UrlValidator` object, named *urlVal*
86 & 87	`assertTrue(urlVal.isValid("http://www.google.com"));`	Sanity check for the *isValid* method of the *urlVal* instance
88	`int statusPerLine = 60;`	Sets the maximum number of status that can be printed on one line
89	`int printed = 0;`	Initializes the counter for printed status
90	`if (printIndex)`	Checks if the indices of the URL parts need to be printed out

91	`statusPerLine = 6;`	If index needs to be printed out, sets the maximum number of printed status per line to 6
93	`do {`	The starting point of the do-while loop that iterates through all the possible URL combinations
94	`StringBuilder testBuffer = new StringBuilder();`	Creates a `StringBuilder` object named *testBuffer* that will hold the URL being tested
95	`boolean expected = true;`	Initializes the value of a `boolean` named *expected* to true, which is used to store the expected result of the URL validity check
96	`for (int testPartsIndexIndex = 0; testPartsIndexIndex < testPartsIndex.length; ++testPartsIndexIndex)`	Starts the loop that iterates each value of the *testPartsIndex*
97	`int index = testPartsIndex[testPartsIndexIndex];`	Sets the current *index*
98	`ResultPair[] part = (ResultPair[]) testObjects[testPartsIndexIndex];`	Sets the `ResultPair` array named *part* to the item at index *testPartsIndexIndex* of *testObjects* (which is the same as *testUrlParts* in this program)
99	`testBuffer.append(part[index].item);`	Appends the URL part to the `StringBuilder` object *testBuffer*
100	`expected &= part[index].valid;`	Conducts the AND operation on the `boolean` *expected*. *expected* is true if and only if each part of the URL is valid
102	`String url = testBuffer.toString();`	Converts *testBuffer* to a `String` *url*
103	`boolean result = urlVal.isValid(url);`	Calls the *isValid* method of the *urlVal* instance to check the validity of *url* and stores the returned `boolean` value in *result*
104	`if(result == true)`	If the *url* is considered valid by the judgement of the *isValid* method
105	`System.out.println(url);`	Prints out the *url*
106	`assertEquals(url, expected, result);`	Calls the JUnit method *assertEquals*

		to confirm the consistency of the values of *expected* and *result*. Otherwise, an `AssertionError` is thrown with the *url* being tested
107	`if (printStatus) {`	If test status needs to be printed out
108	`  if (printIndex) {`	If the indices of the URL parts need to be printed out
109	`  System.out.print(testPartsIndextoString());`	Prints out the indices of the URL parts
110	`  } else {`	If the indices of the URL parts do not need to be printed out
111	`  if (result == expected) {`	If the test *result* matches the *expected* value
112	`  System.out.print('.');`	Prints out "."
113	`  } else {`	If the test *result* does not match the *expected* value
114	`  System.out.print('X');`	Prints out "X"
117	`printed++;`	Increments the *printed* counter
118	`if (printed == statusPerLine) {`	If the number of printed status equals the maximum number per line
119	`  System.out.println();`	Starts a new line
120	`  printed = 0;`	Resets the *printed* counter to 0
123	`} while (incrementTestPartsIndex(testPartsIndex, testObjects));`	The end of the do-while loop that iterates through all the possible URL combinations. The loop terminates when the *incrementTestPartsIndex* method reaches the last index and returns false
124	`if (printStatus) {`	If test status needs to be printed out
125	`  System.out.println();`	Starts a new line

Jonathan Grocott (grocottj), Zheng Zhu (zhuzhe),
Drew Wolfe (wolfedr)

2. Give how many total number of the URLs it is testing.
   - LINE 86 & 87: 2 (The same URL is also being tested later in the do-while loop.)
   - LINE 202 | testUrlScheme: 9
   - LINE 212 | testUrlAuthority: 18
   - LINE 231 | testUrlPort: 7
   - LINE 239 | testPath: 10
   - LINE 268 | testUrlQuery: 3
   - TOTAL NUMBER OF URLs that **SHOULD** be tested: $2 + 9 \times 18 \times 7 \times 10 \times 3 = 34022$. If we exclude the duplicate URLs from the total count, the number becomes 34020.
   - HOWEVER, a bug exists in the *incrementTestPartsIndex()* method [1], which returns false when the index in *testPartsIndex[0]* first reaches 8 and terminates the do-while loop in the *testIsValid()* method earlier than it should. As a result, none of the URLs whose first part is the 8th item in the *testUrlScheme* array get tested when running the *testIsValid()* method. Therefore, the **ACTUAL** number of not necessarily pairwise distinct URLs that are being tested in the program is: $2 + 8 \times 18 \times 7 \times 10 \times 3 = 30242$. Again, if we exclude the duplicate URLs from the total count, the number becomes 30240.

   a. Also, explain how it is building all the URLs.
   - As per the parameters passed into the *testIsValid()* method in line 42 and the code comment beginning at line 195 in UrlValidatorTest.java, a URL is composed of the following five (5) parts arranged in this order: scheme, authority, port, path, and query, all of which must be valid in order for the composite URL to be considered valid.

3. Give an example of valid URL being tested and an invalid URL being tested by *testIsValid()* method.
   - A valid URL being tested: ftp://go.au:65535/test1/file?action=view
     The corresponding indices for the individual sections of the above URL are {1,2,1,7,0}. Each part is valid, so the URL comprised of these five parts is also valid.
   - An invalid URL being tested: http:/www.google.com:80/..//file
     The corresponding indices for the individual sections of the above URL are {4,0,0,8,2}. Among the five sections, the scheme and path are both invalid, making the entire URL invalid.

4. *UrlValidator* code is a direct copy paste of apache commons URL validator code. The test file/code is also direct copy paste of apache commons test code. 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)? Explain in few lines.
   - The fundamental concepts between the real-world *testIsValid()* test and the unit tests we wrote for the dominion code are not very different. In both kinds of tests, we generate test cases as inputs, pass them to the function being tested, and compare the expected results with those we get from running the function. Tests are marked as passed if the results match the expected output; otherwise, tests are marked as failed.
   - On the other hand, the complexity of the real-world *testIsValid()* test is much greater than that of our unit tests written for the dominion code. This has to do with the huge number of test cases

created in the *testIsValid()* test, which are designed to cover as many scenarios as possible and to guarantee the effectiveness of the test. We feel that this level of complexity may actually be required in many real-world unit tests, and we can learn from it when writing our own tests in the future.

Team Work

---

1. Write about how did you work in the team?
    a. How did you divide your work?

    Our team took a proactive approach to the assignment in the sense that each member contributed to a question/task as time and understanding of the subject permitted, leading up to the due date. No one member delegated or assigned the work. Question #1 is obviously foundational to subsequent questions and required a disproportionate amount of effort in order to fully describe the function. As a result, each member of the team was required to become familiar with it as a means to adequately respond to Question #1, check the work, and respond effectively to subsequent questions.

    b. How did you collaborate?

    The team uses a private channel (*6remlin-hunt3rs*) within a *Slack* workspace shared by a small group of peers. The team communicates approaches towards problems, scheduling, and the broad strokes of division of labor within the channel.

2. The contribution of each of your team members.

- **Jonathan Grocott**: Wrote the initial summary (in question #1) as well as initial line-by-line descriptions for the function's code. Reviewed all question responses for accuracy prior to submission.
- **Zheng Zhu**: Wrote the answers to questions #3 and #4. Made corrections and revisions to the answers of questions #1 and #2.
- **Drew Wolfe**: Initiated the Google Doc, provided the *expected* number of URLs to be tested within question #2, reviewed all question responses for accuracy, and provided the *Team Work* section responses.

References

[1]: Issues.apache.org. (2017). *[VALIDATOR-418] UrlValidatorTest: testIsValid() does not run all tests - ASF JIRA.* [online] Available at: https://issues.apache.org/jira/browse/VALIDATOR-418 [Accessed 23 Feb. 2018].