FINAL PROJECT PART A

Jens Bodal, Emily Snyder, William George

For Part a, You will be provided a correctly working version of URL Validator

Go through instructor video of explaining URL Validation testing.

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

   aburasali/UrlValidatorfolder.

Correct filepath: cs362sp16/URLValidatorCorrect/test/UrlValidatorTest.java

There are two functions, ***testIsValid()*** and an overloaded method call for ***testIsValid(Object[] testObjects, long options)***. The method call without function parameters calls the overloaded method with the parameters ***testUrlParts*** and ***UrlValidator.ALLOW_ALL_SCHEMES*** then calls **setUp()**. **setUp()** resets the object's propery ***testPartsIndex***; this property is an array and all of its values will be set to 0 after running ***setUp()***.

1. Create Validator object with parameters ALLOW_ALL_SCHEMES and ALLOW_2_SLASHES as options
2. Assert validator object works on valid urls
3. Set statusPerLine to 60 and initialize printed to zero
4. If printIndex set status per line to 6
5. Enter do while loop
    a. Do
        i. Create Stringbuffer object
        ii. Set expected to true
        iii. For each item in the testPartsIndex
            1. Set index to value in testPartsIndex
            2. Use ResultPair object with members item and valid to cast testObjects[testPartsIndexIndex] to ResultPair array named part
            3. Append item from testObjects[testPartsIndexIndex].item to testbuffer
            4. Make expected = ResultPair object valid parameter && expected
                a. So if ResultPair object valid is false expected is false and vice versa
            5. Continue until url is constructed
        iv. Set stringbuffer to url variable as string
        v. Use isValid member of URLvalidator class is valid and set result to result
        vi. If result is true, output valid url
        vii. If printStatus is true and printIndex is false check if both result and expected are true, if so print a period, if not an X
        viii. Increment printed
        ix. If printed is either 6 or 60 print new line and set printed to zero
    b. While
        i. You can still increment the test parts index
6. If printStatus is true print a new line

2. Give how many total number of urls it is testing.

   Scheme: 9
   Authority: 19
   Port: 7
   Path: 10
   // Path Options: 15 - not being used, commented out testIsValid(**testUrlPartsOptions**, options)
   Query : 3
   9 * 19 * 7 * 10 * 3 = 35,910
   1,890 valid URLs displayed in console

3. Explain how it is building all the urls.

   In order to create composite URLs for testing it breaks them down into four parts:
   <scheme>://<authority><path>?<query>
   Each of these four parts must be correct in order for the composite URL to be correct. The URL validator creates a series of pairs for each of these parts which each holds whether that piece is valid individually. When they are put together to form a composite URL it can then test whether it is valid as a whole. There are further path options that can be tested to verify in instances when two slashes are allowed, but this option is commented out in the testIsValid function.

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

   Valid: http://www.google.com:80
   Not valid: http://www.google.com:-1

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

A real world test, such as ***URL Validator's testIsValid()*** test, is not very different than the unit tests and card tests that we have written. Our unit tests were designed to test specific aspects of the dominion code to ensure that the constraints of the game were being adhered to and that other changes in the code would not affect other areas of the code as it is worked on. That is not to say that the URL validator testIsValid() method is identical in nature however. It performs many functions before testing an aspect of the program, which is sometimes necessary to ensure that the code being tested conforms to the requirements needed. The primary goal of a unit test is to test a specific functionality in its entirety and it would appear that the testIsValid() method accomplishes this through its implementation.

6. Submit a file called ProjectPartA.txt with your writeup. You can submit the file under the folder URLValidator within your ONID directory. (How to setup this folder will be explained soon.)

Commented out testisValid method

```
public void testIsValid(Object[] testObjects, long options) {
    //create URL Validator object url val with parameters schemes set as null and
RegexValidator authority validator set as null and
    //the Allow2Slashes, allow all schems and no fragments options selected
    UrlValidator urlVal = new UrlValidator(null, null, options);
    //calls isValid using validator isValid function for two known valid urls
    assertTrue(urlVal.isValid("http://www.google.com"));
    assertTrue(urlVal.isValid("http://www.google.com/"));
    //preset printed and status per line
    int statusPerLine = 60;
    int printed = 0;
    //print index that indicates current scheme,host,port,path, query test were using. Set as
    //false in URLValidator test class
    if (printIndex)  {
      statusPerLine = 6;
    }
    //do while loop
    do {
      //create stringbuffer object
      StringBuffer testBuffer = new StringBuffer();
      //set expected to ture
      boolean expected = true;
      //uses testPartsIndexIndex as looping variable while less than testPartsIndex.length
      //initialized in setUp() functions
      for (int testPartsIndexIndex = 0; testPartsIndexIndex < testPartsIndex.length;
++testPartsIndexIndex) {
        //set index variable
        int index = testPartsIndex[testPartsIndexIndex];
        //creates ResultPair array part from ResultPair object which groups tests and expected
results
```

```
//http://grepcode.com/file/repo1.maven.org/maven2/commons-validator/commons-validator/1.4.1
/org/apache/commons/validator/ResultPair.java
          //public class ResultPair {
          //public String item;
          //public boolean valid;

          //public ResultPair(String item, boolean valid) {
              //this.item = item;
              //this.valid = valid;  //Weather the individual part of url is valid.
            //}
          //}
          ResultPair[] part = (ResultPair[]) testObjects[testPartsIndexIndex];
          //appends item from result pair to test buffer
          testBuffer.append(part[index].item);
          //makes expected = expected and part[index].valid...expected set to true earlier
          expected &= part[index].valid;
        }
        //System.out.println(testPartsIndex[0]);
        //changes items from previous parts of constructed url to a string
        String url = testBuffer.toString();
        //tests string to see if it is a valid url
        boolean result = urlVal.isValid(url);
        //if valid print url
        if(result == true)
           System.out.println(url);
        //asserts that url is valid, expected and part[index].valid is true and that result exists
        assertEquals(url, expected, result);
        //if printstatus is true (set as false in URL validator test move on)
        if (printStatus) {
          //if print index is true, move on
          if (printIndex) {
            //System.out.print(testPartsIndextoString());
          } else {
            //check if result == expected and print a period
            if (result == expected) {
              System.out.print('.');
            } else {
              //if result is not expected print an X
              System.out.print('X');
            }
          }
          //increment printed
```

```
          printed++;
          //check if the printed int == statusPerLine which is either 60 or 6 then println and change
printed back to zero
          if (printed == statusPerLine) {
            //output new line
            System.out.println();
            printed = 0;
          }
        }
        //move to next url
      } while (incrementTestPartsIndex(testPartsIndex, testObjects));
      //print a new line
      if (printStatus) {
        System.out.println();
      }
    }
```