CS 362 Final Project (Testing the URLValidator)
Part b: Report

Group Members:

Dustin Dinh
Jason Flaig
Kelvin Watson

Test Methodology

Prior to testing, we discussed and designed our tests to cover as much as the input domain as possible. We combined manual testing and programmatic testing to ensure that significant portions of the input domain would be covered. Manual testing involved calling the isValid() function with a variety of syntactically-correct URL's, as well as syntactically-incorrect URL's. We then took a more systematic approach by partitioning the input domain into well-defined sections, and defining test functions that represented the input domain. The division of the input space resulted in the following "categories" of URL's being tested (see below for a more thorough explanation of the rationale behind these partitions):

1. testMissingComponents(): URL's with missing elements (schemes, hosts, top-level domains, ports, paths, queries, fragments)
    a. An absence of certain components in URL's is the most basic starting point
    b. If omissions result in errors we know what to rule out or find easy failures
2. testExtraSlashes(): URL's with extra forward slashes
    a. Obvious class of syntax errors to check for bogus errors in the system under test
3. testTopLevelDomains(): URL's containing a variety of top-level domains (TLD)
    a. The Web is global and better work for a variety of domains in all locations
4. testPorts(): URL's containing a variety of port numbers
    a. Different ports allow different ways to connect to the web, and are common
5. testQueries(): URL's containing query strings in the "?query=value" format were tested
    a. Although queries are optional parts of the URI, they often appear in URLs generated by an web form, which can generate single or multiple queries.
6. testEncodings(): URL's containing percent-encodings, i.e. hexadecimal notation to represent reserved characters
    a. Due to certain characters disallowed in URL's this is an essential test
7. testLetterCase(): URL's containing a lower and upper-case character combinations
    a. Testing case sensitivity ensures that case doesn't matter in URL's

These 7 categories of errors are representative of either common URL requirements or rule out basic errors. Among the categories, top level domains, ports, encodings, queries and letter case are representative of the legal input domains for URLs. As a result, we intended to get good coverage of these domains. Finally, depending on their location within the URL, missing components and extra slashes were used to test the input partition of illegal characters.

To ensure thoroughness, we then added programmatic tests using brute-force enumeration to cover all permutations of missing URL components, and other syntactic errors.

Manual Testing

We began by performing manual tests on the Apache Commons UrlValidator, to verify that the isValid() method can be called without the URLValidator crashing, and that the isValid() method returns the expected boolean result for at least some trivial tests. All manual tests were performed using assertion checks against the expected return value of isValid() in the following format for valid URL's:

```
try{
  testURL = "http://www.ebay.arpa";
  assertEquals(testURL, true, uv.isValid(testURL));
  System.out.println("   PASS:  "+testURL+"   expected=true,isValid()="+uv.isValid(testURL));
} catch(AssertionError e){
  System.out.println("   FAIL:  "+testURL+"   expected=true,isValid()="+uv.isValid(testURL));
}
```

Conversely, the following code was used to verify that that the expected return value of isValid() when the expected return value is false:

```
try{
  testURL = "https:////www.google.co";
  assertEquals(testURL, false, uv.isValid(testURL)); //should be an invalid url due to missing
required components
  System.out.println("   PASS:  "+testURL+"   expected=false, isValid()="+uv.isValid(testURL));
} catch(AssertionError e){
  System.out.println("   FAIL:  "+testURL+"   expected=false, isValid()="+uv.isValid(testURL));
}
```

The following are some of the URL's that were tested in the testManualTest() method:

```
ftp://www.amazon.com
http://www.google.com
http://www.go.com/test1
http://www.ebay.arpa
http://www.go.com/test1?qr=ans#fra
```

Partitioning

We decided to partition URLs (uniform resource locator)'s based on their various components. Every URL conforms to the syntax of a generic URI (uniform resource identifier). The following is the general form of URI:

scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

As such, we partitioned our code by protocol (URL scheme), hostname, top level domain name, port, and path.

Bug Report

ID	Description and Location of Failure (Details)	Severity	Priority	Replication	Status
1	isValid() returns false for all URLs containing queries. For instance, URLs such as http://www.go.com/test1?q=a will return false, even though the URL is syntactically correct. Bug detected in UrlValidator.java and reported on 13 Nov 2015	High	High	Pass in any otherwise valid URL terminated by either a singular '?' character, or a string of the following correct query format "?query1=answer1". The error can also be replicated using longer query strings "?ques1=inp1&ques2=inp2".	Fault localized to the isValidQuery() method of UrlValidator.java, line 446 (`return !QUERY_PATTERN.matcher(query).matches();`) which incorrectly returns the opposite of the actual matching.
2	URLs containing valid port numbers equal to or above 1000 consistently fail unit tests. Bug detected in UrlValidator.java and reported on 20 Nov 2015	Low to Medium The port number is an optional part of the URI	Low	In any otherwise valid URL, add a colon character and a port number containing four or five digits up to 65535 inclusive (e.g. valid port numbers are 0 through 65535). For example, the valid URL http://www.ebay.ca:8080 results in an observable failure.	Fault localized to line 158 of UrlValidator.java `private static final String PORT_REGEX = "^:(\\d{1,3})$";` which is an incorrect regular expression for evaluating port numbers.
3	Large number of valid top level domains test invalid. Specifically, isValid() returns false for TLD's that come after ".it" alphabetically, such as ".je", ".sk", ".uk" ".zw". Bug detected in DomainValidator.java and reported on 23 Nov 2015	High	High	This bug can be reproduced by calling isValid() on otherwise valid URLs with country-level domains domains that follow after ".it" alphabetically.	Fault localized to line 358 of DomainValidator.java, where the list of country code top-level domains is incomplete (stops at Italy).

Fault Localization and Debugging

Bug ID #1 - isValid() returns false for all URLs containing syntactically-correct queries

This bug was detected by our manual tests in testManualTest(), testQueries() and in our programmatic tests in testIsValid(). The severity of this bug is high because it has the potential to impact a large number of URLs, especially since queries are common in URLs. The priority of this bug is also high as it needs to be fixed as soon as possible so as to allow URLs with valid query strings to be considered valid.

Using the debugger, we were able to step into the isValid() method just prior to the test's assertion failure. To do this, a breakpoint was set in the `catch` block of the try-catch construct where an AssertionError is caught. Running the debugger, each time an AssertionError was caught, the "step into" functionality was used to examine the isValid() method's code. Stepping into the isValid() method, it was determined that the if condition `if(!isValidQuery(urlMatcher.group(PARSE_URL_QUERY)))` on line 314 evaluates to true, resulting in a false value being returned from isValid(). Stepping into isValidQuery() quickly revealed the bug on **line 446**, `return !QUERY_PATTERN.matcher(query).matches();` which returns the opposite of the result of evaluation of the regular expression.

Bug ID #2 - Port numbers 1000 and above result in failed tests

This bug was detected by our manual tests in testPorts(), programmatic tests in testIsValid() and further investigated in testAnyOtherUnitTest(), as explained below. The severity of this bug is considered low since the port number component of the URL is optional. Accordingly, the priority is also low since it is less time-sensitive than the other two bugs identified.

To narrow the range of port numbers , we used a divide and conquer approach (David Agan's Debugging Rule #4). Since there are at most 65535 valid port numbers, exhaustively testing every single port is inefficient. Instead of checking all ports, we used a binary search, which allowed us to localize the range of ports that resulted in an observable failure by eliminating half of this particular input space with each input. We started by testing from 0 until a URL with an invalid port was identified that resulted in a failure (port number 1000). Then, we tested the middle of the ports (around 32000) to check if a failure still occurred. Because a failure was observed with port 32000, we then tested at half of the input space again (port 16000). Again, a failure was observed, so we tested 8000, then 4000, 2000, and so on, until it was determined that all ports above 1000 resulted in observable failures.

Once the range of port numbers was sufficiently narrowed using an approach similar to delta debugging as described above, we were able to localize the fault by using the debugger. Line 393 `if (!PORT_PATTERN.matcher(port).matches())` evaluates to true, and as a result, isValid() returns false. Further investigation into the PORT_REGEX String object on line 158 reveals that the following regular expression used to evaluate port numbers is incorrect. `"^:(\\d{1,3})$"` This regular expression incorrectly matches a number containing at least a single digit and at most three digits. This is the reason why a number such as 1000 would fail the regex evaluation, since it exceeds the maximum 3 digits matched by this regular expression. Instead, the regex should be `"^:(\\d{1,5})$"` in order to accommodate valid port numbers up to and including 65535.

Bug ID #3 - Country code top-level domains after .it results in test failure

This bug was detected by our manual tests in testTopLevelDomains(). The top-level domain is a required component of the URL, and thus, the impact of this bug is very high, especially for countries whose top-level domains come after .it alphabetically. The priority is also set to high since it should be fixed urgently to allow URLs containing country code TLDs after .it alphabetically to be considered valid.

For Bug #3, we used Agan's rules to find faults. Following rule 2 , we attempted to make the program fail consistently. We discovered through manual testing that only the secondary top level domain of .uk failed

while others did not. For example, ac.uk failed while asn.au did not fail. As a result, we followed rule number 4, divide and conquer to localize the problem by stepping over and into relevant functions as follows: Breakpoints were set each time a new function was called in the chain of function calls. On discovering an incorrect return value, we returned to the breakpoint and stepped into function. This process was repeated until the code defect was found.

The fault was localized to the isValidAuthority() method. Using "step into" functionality, the code defect was further localized to line 381, then line 385 when both domainValidator.isValid(hostLocation) and inetAddressValidator.isValid(hostLocation) both return false. Using www.myURL.co.uk as an example, inetAddressValidator.isValid(hostLocation) correctly returns false as the URL is not provided as an IP address. The function isValidCountryCodeTld() searches for top level domains is located on line 191 of DomainValidator.java using the contains() method on the COUNTRY_CODE_TOLD_LIST list variable. The list is itself located on line 248. Watching isValidCountryCodeTld(tld), isValidGenericTld(tld) and isValidInfrastructureTld(tld) revealed that these expressions all evaluate to false, resulting in a cascade of erroneous return values for the DomainValidator class' isValidTld() method, and subsequently the DomainValidator class' isValid() method, then the UrlValidator class' isValidAuthority() method, and finally the UrlValidator class' isValid() method.

Teamwork

The group collaborated via email, Google Hangouts, Github and Google Docs. We used email to coordinate meeting times and Google Hangouts for our video meetings and text chat. In the video meetings we met under the assumption that everyone understood the requirements of the project, so work began rather quickly. Uncertainty about the input partitioning lead us to a high-level discussion of what inputs our tests should cover. These input partitions, which are listed 1-6 at the beginning of this document, became our plan or template that we covered in our code. We found our 3 bugs quickly using our template in Eclipse and we uploaded our work to a separate Github repository hosted by Kelvin. Google Docs was our editor of choice for writing this document.

We worked well together, getting done with our work earlier for a few reasons. Kelvin was very motivated to get the work done, in turn Dustin and Jason were motivated to do more sooner. Kelvin and Jason had also worked with Dustin in other course projects, so we all wanted to pitch in. We each found 3 bugs: Kelvin found a bug in queries, Jason found a bug for common ports and Dustin found an error in top level domains. Kelvin also dug a little deeper and found that ports greater than 1000 failed. We did not actually divide up work but we were all very interested in finding bugs so we proceed more organically and worked and combined our efforts via Github. The same effort went into this report. Dustin started it, then Kelvin and Jason volunteered and took sections followed by Dustin helping more.