Final Project: Part B

Team Members:

Sam Schultz, Sean Mulholland, Tatyana Vlaskin

First just do manual testing. Call the valid method of URLValidator with different possible valid/invalid inputs and see if you find a failure. For manual testing, provide some of your (not all) urls and why.

For the manual tests, we wanted to test each part of a URL: scheme, authority, port, path and query. We did the testing using both valid and invalid entries for each part of the URL. The intent was to make sure that the URLValidator can detect all correct and incorrect URL addresses. For this part of testing we used the following validator:
UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
For the list of all the URLs that were tested please refer to the URLValidatorTest.java file.
Few examples are provided below.

//testing VALID scheme, VALID authority, valid port

1. http://www.amazon.com
2. http://ww.amazon.com
3. www.amazon.com ---FAILED THE TEST

//testing INVALID scheme, valid authority, valid port

1. hhhh:/www.amazon.com
2. http:/www.amazon.com
3. etc

//testing valid scheme, valid authority, valid port, VALID path

1. http://www.amazon.com/
2. http://www.amazon.com/gp/goldbox/ref=nav_cs_gb
3. etc

//testing valid scheme, valid authority, valid port, INVALID path

1. http://www.amazon.com//
2. http://www.amazon.com/....
3. etc

//testing valid scheme, valid authority, VALID port

1. http://www.amazon.com:80
2. http://www.amazon.com:400
3. http://www.amazon.com:6553 ---FAILED THE TEST
4. http://www.amazon.com:65535 ---FAILED THE TEST
5. etc

//testing valid scheme, valid authority, INVALID port

1. http://www.amazon.com:-80
2. http://www.amazon.com:A80
3. etc

//testing valid scheme, valid authority, VALID query

1. http://www.amazon.com/?action=view
2. http://www.amazon.com/?
3. etc

//testing valid scheme, valid authority, INVALID query

1. http://www.amazon.com/?action=
2. http://www.amazon.com/?action22=view
3. etc

//Miscellaneous Manual Tests

1. fake://org.org
2. one://falsehost.TLD/
3. fake://org.org.org:65535/pathname/subdir?key=val
4. alFake+still://org.org.org:90
5. https://localhost:80/---FAILED THE TEST
6. http://www.google.com:99999---FAILED THE TEST
7. http://www.google.com:999990 ---FAILED THE TEST

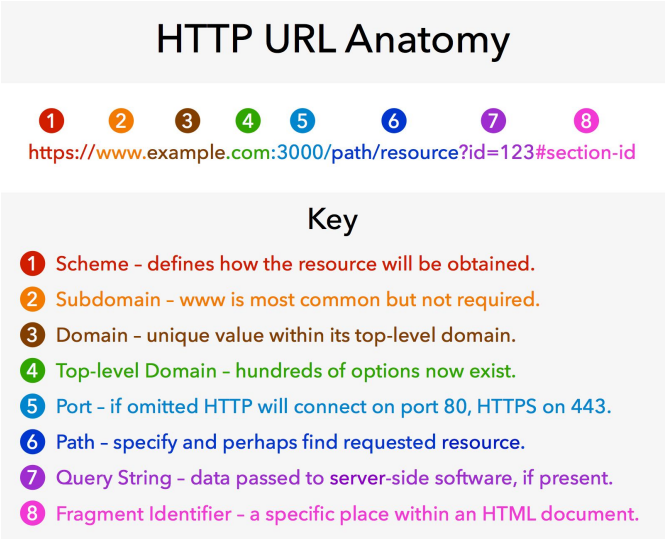
After that we created another validator:
urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_2_SLASHES);
And tested URL with 2 slashed in the path component and more than 2 slashes in the path and other components of the URL

1. http://www.amazon.com//path
2. http://www.amazon.com///path

Second, come up with good input partitioning. Try to provide a varying set of inputs that partition the overall input set well. Did you find any failures? You can call valid method once or more for each partition.For partitioning, mention your partitions with reasons.

For partitioning we decided to do something similar that we did for the manual testing: we looked at different components of the URL. Reffer to

https://cascadingmedia.com/insites/2015/02/url-anatomy-seo-tips.html



A more simplified version of the URL components is:

1. Scheme
2. Authority = [subdomain + domain + top-level domain]
3. Port
4. Path
5. Query

Due to the time constraints we will do partitioning only for SCHEME, AUTHORITY, PORT, PATH and QUERY. We tested both the isValid() method, as well as the sub-methods that are called in the execution of isValid() (e.g. isValidScheme(), isValidAuthority(), etc.) for each partition. We also used two types of validators, one constructed by the default constructor (no arguments), and one that allowed all schemes that fit the RFC defined scheme syntax.

The validators were created using the following code:

```
UrlValidator defUrlVal = new UrlValidator();
UrlValidator urlVal = new UrlValidator(UrlValidator.ALLOW_ALL_SCHEMES);
```

We tested the following valid and invalid schema:

SCHEME	EXPECTED RESULT	ACTUAL RESULT
http://	true	false
https://	true	false
h3tp://	true	false
ftp://	true	False --FAILED THE TEST- NOT SURE WHY?????

""	true	False --FAILED THE TEST
3ht://	false	false
http:/	false	false
http:	false	false
http/	false	false
://	false	false
+http://	false	false
h-+.ps://	true	true
a%%%://	false	false

FAILED THE TEST: The following 2 schema were flagged as INVALID enthough they are VALID: **NO SCHEME** - amazon.com is an acceptable address, so is **h3tp://**. This was an interesting finding because when the validator was created using the following code: urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_2_SLASHES), **h3tp://** was labeled as VALID scheme - see manual testing.

We tested the following valid and invalid AUTHORITY:

AUTHORITY	EXPECTED RESULT	ACTUAL RESULT
www.amazon.com	true	true
amazon.com	true	true
amazon.org	true	true
amazon.edu	false	False
""	false	False
" “	false	false
ppp.amazon.com	false	true--FAILED THE TEST

www.amazon%.com	false	false
www.amazon	false	false
localhost	true	false
0.0.0.0	true	true
123.012.123.230	true	true
255.255.255.255	true	true
256.255.255.255	false	false
1.2.3.4.5	false	false
google.falseTLD	false	false
.test	false	false
test.	false	false

We tested the following valid and invalid PORTS:

PORT	EXPECTED RESULT	ACTUAL RESULT
0	true	true
11	true	true
52	true	true
89	true	true
125	true	true
265	true	true
935	true	true
1115	true	false--FAILED THE TEST

5858	true	false--FAILED THE TEST
15455	true	false--FAILED THE TEST
65530	true	false--FAILED THE TEST
65536	false	false
-0	false	false
:a11	false	false
:52a	false	false
NO PORT NUMBER [:]	false	false
154578787	false	false
65539	false	false

We tested the following valid and invalid QUERIES:
String[] listOfValidQuery = {"?action=edit&mode=up", "?newwindow=1&q=url+query", "?action=edit&mode=up", " "};

String[] listOfInValidQuery = {"/ ","??action=view"};

QUERY	EXPECTED RESULT	ACTUAL RESULT
?action=edit&mode=up	true	false--FAILED THE TEST
?newwindow=1&q=url+query	true	false--FAILED THE TEST
?action=edit&mode=up	true	false--FAILED THE TEST
" "	true	false--FAILED THE TEST
/	false	false
??action=view	false	false
?un1qu37r4cking	true	true

key#val	false	true--FAILED THE TEST

Third, do programming based testing. Did you find any failures? Submit your test files and test cases as part of your work under your onid/URLValidator folder. For unit tests/random tests, submit your unit tests using GitHub under URLValidator folder.

For the programming based testing we took valid URL components and created an array for each of the component: Scheme, Authority, Port, Path and Query.Then we made loops, where we iterate through all elements in the URL components arrays to make valid address. Basically, we are running through all permutations of the arrays combinations.
The address is than tested using the isValid() function. If "false" is returned , this is an indication that bug was detected [because all the created addressed are valid]. All the failures that were detected using programming based testing have already been detected using Manual testing and partitioning testing [this is due to the fact that we have done very thorough testing at that stage of testing].

When you find out any failure, debug using Eclipse debugger and try to localize its cause. Provide at what line/lines in what file the failure manifested itself. Did you use any of Agan's principle in debugging URLValidator?

To answer this question, we referred to the following website
http://looksgoodworkswell.blogspot.com/2012/12/indispensable-principles-for-debugging.html
to get a better understanding of the Agan's principles.

AGAN'S PRINCIPLES:
Understand the System:
Part A of the project made us familiar with the system. At the same time all the members of the group took time to the notes of the programmer of the URLValidator. We also walked through the URLValidator using the Eclipse debugger and we also studied the code in detail to make sure that we understand all purpose of all the functions in the code.

Make it Fail:
During the manual and partition stages of the testing, we initially were testing invalid URLs. This was done because we wanted to see expected failure to make sure that the validator detects

invalid URLs correctly. At the same time, we also tested large number of valid URL results hopping that is there is a bug one of the valid URL will be marked as invalid.

Quit Thinking and Look:
Initially, we tried to follow this principle and stepped through the debugger several times, but came to the conclusion that it was taking too much times. As a result of that we decided to think and come up with numerous tests, print out result of the test on the console long and after the bug was detected, go back to the code and use the debugger to figure out the source of the bug.

Divide and Conquer:
For all the testing of the URLValidator, we were partitioning URL down to smallest component. This has helped us a lot when it came to narrowing down the source of the problem [a scheme, authority, port, ect].

Change One Thing at a Time:
We were following this principle especially during the programming based stage of the testing. We were keeping all the components of the ULR address the same, except the one component that we were testing. This has helped us to detect a problem.

Keep an Audit Trail:
After each test, we kept really good documentation to make sure that we will remember what has already been done.

how did you work in the team? How did you divide your work? How did you collaborate?

The amount of work required to locate the minimum three bugs did not warrant significant collaboration. We agreed over email that we would each formulate our own methods for each style of testing and collectively bring our bug reports together to compare and contrast findings. This allowed different methodologies within the same categories to expand our test coverages within the different styles of testing. For example, instead of agreeing on one set of manual tests we can Tatyana's manual tests included manual test combinations that Sean's did not and vice versa. This may not be how a professional team would work as they may not allocate extra man hours to repeating the same partitions, but for our testing it allowed us to quickly broaden the actual testing space.

Further, by each solving the same problem individually and checking the agreement of our solutions we have confirmation for our findings making them more reliable. It's too easy to call the code we're testing buggy when in actuality it may be the tests themselves. For the

scope of the project and amount of testing requested writing 3 separate sets of tests that intend to find the same thing gives us confirmation when all 3 parties agree but also allows further investigation if one of the parties does not agree.

BUG REPORTS:

****

BUG 1

****

Title: Ports with 4 or 5 digits as invalid

Details:

Type: Bug
Status: OPEN
Priority: Major
Resolution: Unresolved
Affects Version/s: Revision:1227719 Validator: 1.4
Fix Version/s: None
Component/s: DomainValidator::isValidTld(String tld);
Labels: easyfix
Environment: Windows 8.1, JRE 1.8.0_92

People:

Reporter: Tatyana Vlaskin
Date: 6/4/16

Description: The isValid() method incorrectly flags ports that are longer than 3 digits as incorrect ports.

Code Line(s): URLValidator.java line 158

```
private static final String PORT_REGEX = "^:(\\d{1,3})$";
```

Debug Details:

Some of the test that have detected this problem:
System.out.println(urlVal.isValid("http://www.amazon.com:6553")); Returns false, while the correct result should be true.

System.out.println(urlVal.isValid("http://www.amazon.com:65535"));Returns false, while the correct result should be true.

Suggested fix:

URLValidator.java line 158 private static final String PORT_REGEX = "^:(\\d{1,5})$";

****

BUG 2

****

Title: NULL scheme not accepted

Details:
Type: Bug
Status: OPEN
Priority: Minor
Resolution: Unresolved
Affects Version/s: Revision:1227719 Validator: 1.4
Fix Version/s: None
Component/s: DomainValidator::isValidTld(String tld);
Labels: easyfix
Environment: Windows 8.1, JRE 1.8.0_92

People:
Reporter: Tatyana Vlaskin
Date: 6/4/16

Description: Missing scheme returns false when it should return true because null scheme is valid.

Code Line(s): URLValidator.java line 336

```
if (scheme == null) {
return false;
}
```

Debug Details:
Some of the test that have detected this problem:
System.out.println(urlVal.isValid("www.amazon.com")); Returns false, while the correct result should be true.

Suggested fix:

URLValidator.java line 336

```
if (scheme == null) {
return true;
}
```

****

BUG 3

****

Title: Acceptable queries fail validation

Details:
Type: Bug
Status: OPEN
Priority: Major
Resolution: Unresolved
Affects Version/s: Revision:1227719 Validator: 1.4
Fix Version/s: None
Component/s: DomainValidator::isValidTld(String tld);
Labels: easyfix
Environment: Windows 8.1, JRE 1.8.0_92

People:
Reporter: Tatyana Vlaskin
Date: 6/4/16

Description: UrlValidator considers any URL with a query string to be invalid

Code Line(s):

URLValidator.java line 446

return !QUERY_PATTERN.matcher(query).matches();

URLValidator.java line 314

```
        if (!isValidQuery(urlMatcher.group(PARSE_URL_QUERY))) {
        return false;
        }
```

And 446

return !QUERY_PATTERN.matcher(query).matches();

Debug Details:
Some of the tests that have detected this problem:

```
  String validAuthority = "www.amazon.com";
  String validScheme = "http://";
  String validPort = ":80";
  String validPath = "/path";
  String validQuery = "?action=view";
 public void testYourFourthPartition(){
        System.out.println("\nTESTING Valid QUERY:\n");
        System.out.println("\n*****************************************\n");
        System.out.println("\nEverything should be true:\n");
        String[] listOfValidQuery = {"?action=edit&mode=up", "?newwindow=1&q=url+query",
"?action=edit&mode=up", " "};
        UrlValidator urlVal2 = new UrlValidator();
```

```
        for (int i = 0; i < listOfValidQuery.length; i++) {
        String curQuery = validScheme  + validAuthority + validPort + validPath +
listOfValidQuery[i];
        System.out.println("Testing " + curQuery);
        System.out.println(urlVal2.isValid(curQuery));
        }
```

This function test URLs with different valid queries. Returns false all the time, while the correct result should be true.

Suggested fix:

Remove negation.

****

BUG 4

****

Title: Invalid "/" truncation testing

Details:

Type: BUG

Status: OPEN

Priority: Major

Resolution: Unresolved

Affects Version/s: Revision:1227719 Validator: 1.4

Fix Version/s: None

Component/s: UrlValidator.isValid("String")

Labels: None

Environment: Windows 8.1, JRE 1.8.0_92

People:

Reporter: Sam Schultz

Date: 6/5/2016

Description: The parsing for the full URL does not take into account that extra slashes will be truncated after the scheme

Code Line(s): Line 340

```
if (!SCHEME_PATTERN.matcher(scheme).matches()) {
return false;
}
```

Debug Details:. Bug triggered with https://///www.yahoo.com/.

The error coming from the isValidScheme(String) method. More specifically I think the error is stemming from line 340 when the scheme pattern is parsed against the matcher.

Suggested Fix: Allow multiple characters of '/' after the scheme in scheme string validation.

****

BUG 5

****

Title: Invalid scheme validation

Details: **People:**

Type: BUG Reporter: Sam Schultz

Status: OPEN Date: 6/5/2016

Priority: Blocker

Resolution: Unresolved

Affects Version/s: Revision:1227719 Validator: 1.4

Fix Version/s: None

Component/s: UrlValidator.isValidScheme(String)

Labels: None

Environment: Windows 8.1, JRE 1.8.0_92

Description: The scheme validation is failing against known scheme valid scheme strings such as Http:// .

Code Line(s): Line 131 of UrlValidator

```
private static final String SCHEME_REGEX = "^\\p{Alpha}[\\p{Alnum}\\+\\-\\.]*";
```

Debug Details: The regular expression query against the scheme is in the incorrect format, therefore rejecting valid schemes.

Suggested Fix: Adjust the regular expression parsing to follow the correct pattern using ^([a-z][a-z0-9+\-.]*) regex pattern.

****

BUG 6

****

Title: Invalid Query Validation

Details: **People:**

Type: BUG Reporter: Sam Schultz

Status: OPEN Date: 6/5/2016

Priority: Blocker
Resolution: Unresolved
Affects Version/s: Revision:1227719 Validator: 1.4
Fix Version/s: None
Component/s: UrlValidator.isValidQuery(string)
Labels: None
Environment: Windows 8.1, JRE 1.8.0_92
Description: Known valid query syntax is not being validated correctly
Code Line(s): Line 151 of UrlValidator
```
private static final String QUERY_REGEX = "^(.*)$";
```
Debug Details: Test was failing when a query is in valid syntax is within the url. The query is denoted by a '?'. The test is failing on this input :
Http://google.com:443/path?trueQuery=1#truefrag.

Suggested Fix: The solution to this bug is to adjust the Regular expression parse pattern to except the valid syntax denoted with the '?' .

****

BUG 7 --- Bug located in "Correct" code, but exists in incorrect code as well

****

Title: Ports 99999 > 65535 are returning Valid

Details:
Type: Bug
Status: OPEN
Priority: Minor
Resolution: Unresolved
Affects Version/s: Revision:1227719 Validator: 1.4
Fix Version/s: None
Component/s: isValidAuthority(String authority);
Labels: None
Environment: Windows 8.1, JRE 1.8.0_92

People:
Reporter: Sean Mulholland
Date: 6/4/16

Description: isValidAuthority returns true for ports greater than 65535. While the URL RFC doesn't define port range, the implicit port range defined by the most common transport layer protocols (16 bit unsigned int) makes URLs with ports higher than 65535 invalid.

This was found by partition testing the isValidAuthority function with the test string:

www.google.com:65536

The offending code is listed below, and only functions to check the pattern of a port number. If there is a ':' character in authority, the following 5 characters are validated as digits. A range is not specified so ports from 00000-99999 will all return valid.

Code Line(s): Lines 393-394 and Line 158-159

Lines 393-394: Calling Location

```
if (!PORT_PATTERN.matcher(port).matches()) {
    return false;
```

158-159: Errant Code

```
private static final String PORT_REGEX = "^:(\\d{1,5})$";
private static final Pattern PORT_PATTERN = Pattern.compile(PORT_REGEX);
```

Debug Details:

string authority = "www.google.com:65536"
string port = ":65536"
UrlValidator was Default constructed
isValidAuthority(authority) returns TRUE

Suggested Fix:

Correct function to check if collective int is < 65536. Since it is unspecified in the allowed port specifications of the RFC for URLs, if larger values than 65535 are to be allowed, expand the REGEX to allow for either any number of digits or some other sensible (range of max unsigned 32 bit int value) size.

****

BUG 8 --- Bug located in "Correct" code, but exists in incorrect code as well

****

Title: Authority Regex groupings inaccurate

Details:

Type: Bug
Status: OPEN
Priority: Major
Resolution: Unresolved
Affects Version/s: Revision:1227719 Validator: 1.4
Fix Version/s: None
Component/s: isValidAuthority(String authority);
Labels: None
Environment: Windows 8.1, JRE 1.8.0_92

People:

Reporter: Sean Mulholland
Date: 6/4/16

Description: isValidAuthority doesn't properly validate domain names. When testing the authority string:

"www.google.com::::"

Parsing the string returns:

Domain = "www.google.com"

Port = ":"

The extra colons get dropped during parsing by the Java Pattern matcher. The hostLocation gets parsed by the pattern matcher which groups the first set of alphanumeric characters, '-', and '.' into the first group (PARSE_AUTHORITY_HOST_IP is an alias for 1). This first group is returned by the matcher. The port is parsed by returning the second group defined by a colon followed by any number of repeating digits. This ignores the intermediate extra colon characters that don't end up in a matched group (just the whole authority string).

Code Line(s): Lines 377, 391, 134-135, and 102

Line 377: Calling Location

```
String hostLocation = authorityMatcher.group(PARSE_AUTHORITY_HOST_IP);
```

Lines 391: Calling Location

```
String port = authorityMatcher.group(PARSE_AUTHORITY_PORT);
```

Lines 134-135: Errant Code

```
private static final String AUTHORITY_REGEX =
    "^([" + AUTHORITY_CHARS_REGEX + "]*)(:\\d*)?(.*)?";
```

Line 102: Relevant Regex

```
private static final String AUTHORITY_CHARS_REGEX = "\\p{Alnum}\\-\\.";
```

Debug Details:

String authority = "www.google.com::::"
String hostLocation = "www.google.com"
String port = ":"
Url validator is from default constructor

Suggested Fix:

Correct Regex to include all colon characters except the last one (if they exist) in group 1. Group 2 should start at the last colon character.

****

BUG 9

****

Title: Invalid Query Regex

Details:
Type: Bug
Status: OPEN
Priority: Major
Resolution: Unresolved
Affects Version/s: Revision:1227719 Validator: 1.4
Fix Version/s: None
Component/s: isValidQuery(String query);
Labels: None
Environment: Windows 8.1, JRE 1.8.0_92

People:
Reporter: Sean Mulholland
Date: 6/4/16

Description: Per RFC 3986, the '#' is considered a general delimeter and is not allowed in a query string. The isValidQuery method checks if a passed in query string matches the prescribed regex. The current regex is:

QUERY_REGEX = "^(.*)$"

This can be interpreted as any character from the start of the string to the end of the string. This is not the true allowable query syntax as described by RFC 3986.

Code Line(s): UrlValidator Line 446, 151, and 153,

Line 446: Calling Location
```
return QUERY_PATTERN.matcher(query).matches();
```
Line 151: Errant Regex
```
private static final String QUERY_REGEX = "^(.*)$";
```
Line 153: Builds Pattern from Regex
```
private static final Pattern QUERY_PATTERN = Pattern.compile(QUERY_REGEX);
```
Debug Details:
String query = "key#val"
return value = true

Suggested Fix:
Correct Regex to not allow for general delimiters in query

****

BUG 10

****

Title: Invalid return value for "localhost" domain

Details:
Type: Bug

People:
Reporter: Sean Mulholland

Status: OPEN Date: 6/4/16
Priority: Major
Resolution: Unresolved
Affects Version/s: Revision:1227719 Validator: 1.4
Fix Version/s: None
Component/s: DomainValidator::isValidTld(String tld);
Labels: easyfix
Environment: Windows 8.1, JRE 1.8.0_92

Description: When checking if a top level domain (TLD) is valid, the wrong boolean is returned. There is an incorrect "!" prior to the return value, so when a valid TLD is located from the known list the isValidTld() return value is returned false which is incorrect behaviour.

Code Line(s): UrlValidator Line 381, DomainValidator lines 137, 154, and 205

```
UrlValidator::Line 381: Calling Location
      if (!domainValidator.isValid(hostLocation)) {
DomainValidator::Line 137: Calling Location
        return isValidTld(groups[0]);
DomainValidator::Line 154: Calling Location
         if(allowLocal && isValidLocalTld(tld)) {
DomainValidator::Line 205: Errant Code
         return !LOCAL_TLD_LIST.contains(chompLeadingDot(iTld.toLowerCase()));
```

Debug Details:

String query = "https://localhost:80/"
Return value = false

Suggested Fix:

Remove the '!' operator from the return value