CS362 – Final Project Part B
Date: 03/14/2016
Emmalee Jones, Tanna Richardson, Xiaohan Zeng

Manual Testing

The methodology for manually testing UrlValidator from Apache was to use good tests, bad tests, and boundary tests. The URL was divided into major sections: schemas, hosts, ports, paths, and queries. An object was created for UrlValidator.java and the isValid Method was called with various good, bad, and boundary tests within an assertTrue or assertFalse command. Bugs were found when asserts did not respond with valid or invalid Urls as expected. Correct URL tests were developed by reviewing material from Wikipedia on Uniform Resource Locator. Bugs were further investigated by setting breakpoints within Eclipse before and after asserts that did not run as expected. Additional code review was done to try to isolate the area where the code was not working correctly within the URL sections.

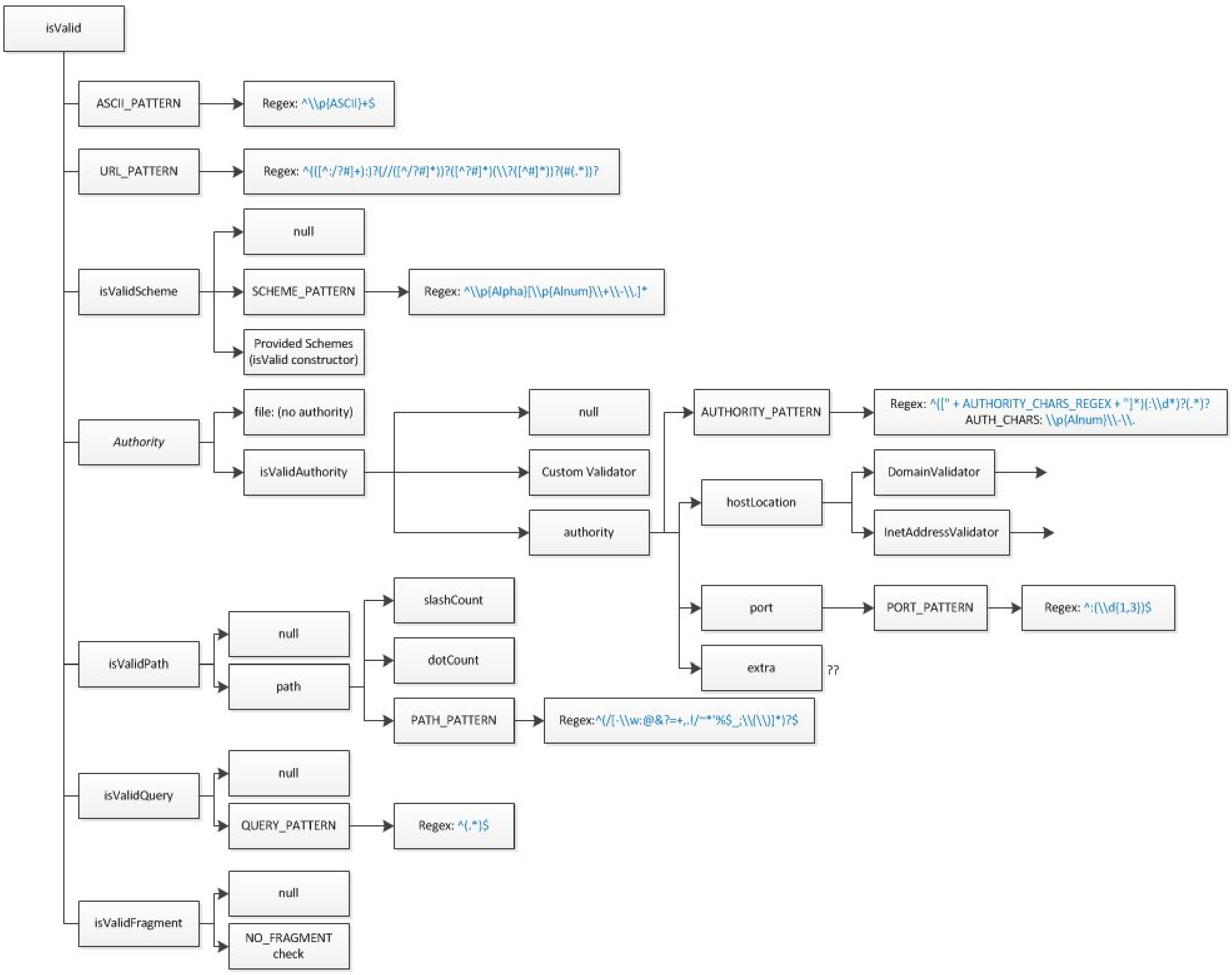
```
//Test Allow All Schemes
UrlValidator urlVal = new UrlValidator(UrlValidator.ALLOW_ALL_SCHEMES);
System.out.println("Schema Test - Allow All Schemes");
assertTrue(urlVal.isValid("http://www.amazon.com"));
assertFalse(urlVal.isValid("google"));
//Test local URLs being allowed.
urlVal = new UrlValidator(UrlValidator.ALLOW_LOCAL_URLS);
System.out.println("Local URLs being allowed");
assertTrue(urlVal.isValid("http://127.0.0.1"));
assertFalse(urlVal.isValid("http://localhost")); //Should be true, possible bug
// Test for fragments not allowed.
System.out.println("No Fragments Test");
urlVal = new UrlValidator(UrlValidator.NO_FRAGMENTS);
assertFalse(urlVal.isValid("http://www.bigdog/time.php")); //Should be true, possible bug
// Test URL with query
System.out.println("URL With Query Test");
urlVal = new UrlValidator(UrlValidator.ALLOW_ALL_SCHEMES);
assertFalse(urlVal.isValid("http://www.foo.gov/somefolder?name=bar"));   //Should be true,
possible bug
assertFalse(urlVal.isValid("http://www.foo.gov/somefolder?name=bar&name2=bar2"));
//Should be true, possible bug
assertFalse(urlVal.isValid("http://www.foo.gov/somefolder?name=bar;name2=bar2"));
//Should be true, possible bug
```

```
// Test URL with ports
System.out.println("URL With Port Test");
urlVal = new UrlValidator(UrlValidator.ALLOW_ALL_SCHEMES);
assertTrue(urlVal.isValid("http://www.apple.com:80"));
assertFalse(urlVal.isValid("http://www.apple.com:35912"));//Should be true, possible bug
```

Our manual testing is implemented in the testManualTest() method in UrlValidatorTest.java.

Input Partitioning

Our methodology for domain input partitioning was to understand the system under test and identify as many partitions as possible to ensure we were covering all of the valid and invalid URLs. We started with the basic partitions that are apparent in the UrlValidator code: scheme, authority, path, query and fragment. We then reviewed RFC 3986: Uniform Resource Identifier (URI): Generic Syntax to understand the detailed requirements for each of these partitions and identify valid and invalid values that we should be testing. Finally, we took a detailed look through the code and identified each of the checks that are performed in the individual isValid* methods. The diagram on the following page shows each of these checks:



We then identified values that should fail and pass each of these checks.

Once we had established partitions for testing, we added all of our valid and invalid values to the appropriate ResultPair[] array (testSchemas, testHosts, testPorts, testPaths, testQueries and testFragments). For each partition test, we looped through all the corresponding test* array values and used the default value (index 0; always valid) for the other components. For example, our code for testing the scheme partition is:

```
for (int i = 0; i < testSchemas.length; i++){
    ResultPair url = URLmaker(i,0,0,0,0,0);
    assertEquals(url.valid, urlVal.isValid(url.item));
}
```

URLmaker is a helper method we wrote that gets a value from each test* array at the specified index and combines them into a URL. It also checks the validity of each value and if any of them are false, it sets the entire URL to be false. The combined URL and true/false value are returned as a ResultPair.

This approach allowed us to keep one master list of valid/invalid values and use it for both partition and random testing. In addition to the ResultPair arrays, we added manual testing to handle special input partitions such as custom validators and the full URL checks.

Our partition testing is implemented in the following methods in UrlValidatorTest.java:

- testYourFirstPartition() - full URL
- testYourSecondPartition() - schemes
- testYourThirdPartition() - authorities
- testYourFourthPartition() - paths
- testYourFifthPartition() - queries
- testYourSixthPartition() - fragments

Unit Tests / Random Tests

In this part we call isValid multiple times with different URLs and test whether return is what we expect. The difference from partitioning testing is that we leverage random test methodology to generate URL. To do that we re-use different partitioning arrays we have above and randomly pick up element from each array. Then we use URLmaker to assemble the URL and get the ResultPair. See pseudo code below,

```
int schemaIndex = ran.nextInt(testSchemas.length);
int hostIndex = ran.nextInt(testHosts.length);
int portIndex = ran.nextInt(testPorts.length);
int pathIndex = ran.nextInt(testPaths.length);
int queryIndex = ran.nextInt(testQueries.length);
int fragmentIndex = ran.nextInt(testFragments.length);
ResultPair url = URLmaker(schemaIndex, hostIndex, portIndex, pathIndex, queryIndex,
fragmentIndex);
```

This allows us to get good coverage and often find unexpected bug but with less code needed to cover all possible permutation.

Our random testing is implemented in the testIsValid() method in UrlValidatorTest.java.

Team Collaboration

An initial meeting was held with Google Hangouts to review the project and to decide how to split up the assignment. We decided to let everyone take a look at how the UrlValidator was supposed to work and to understand how the Uniform Resource Locator (URL) domain was constructed. We came up with a chart for understanding the URL composition which helped in coming up with the testing method of good tests, bad tests, and boundary tests. Each team member volunteered for various portions of the assignment with one person taking the manual tests, another input portioning, and the program based tests by the final team member. We all added different ResultPair Tests. The work was stored on Google Drive so that each team member could review and modify the assignments, the code was managed in Git. We collaborated through Google Mail as needed, Google Hangouts on a weekly basis, and Google Drive as necessary. The team worked well as a group.

Bug Reports

VALIDATOR-1	Invalid port size for validation of port number.
Details	
Type: Bug	Affects Version/s: 2.0 Revision: 1128446
Status: Open	Fix Version/s: None
Priority: Major	Component/s: Routines
Resolution: Unresolved	Labels: None
Description	
UrlValidator does not accept port numbers over 3 digits. The following code returns false. urlVal = new UrlValidator(); assertFalse(urlVal.isValid("http://www.example.com:8080")) A URL’s port is either null or a 16-bit unsigned integer that identifies a networking port. It is initially null. The unsigned range is 0 through 65,535. https://url.spec.whatwg.org/ Five digit numbers up to 65,535 should be accepted as valid ports.	

How to Reproduce
Steps to Reproduce: 1. urlVal = new UrlValidator(); 2. assertFalse(urlVal.isValid("http://www.example.com:8080")); Expected Result: True Result Actual Result: False Result We found the error by good, bad, and boundary testing for ports.
Debugging

We used the Eclipse debugger to set breakpoints before and after the code that did not work as expected. We reviewed the UrlValidator to look for code related to port and set additional breakpoints. We looked up information about what valid ports should be to set the tests. The cause of the error is an incorrect setting for the size of a port, it is set to “3” instead of “5” for port range. The code that is bad is lines 158 & 159
`private static final String PORT_REGEX = "^:(\\d{1,3})$";`
`private static final Pattern PORT_PATTERN = Pattern.compile(PORT_REGEX);`
The PORT_PATTERN is used in lines 393 & 394 to test the port.
`if (!PORT_PATTERN.matcher(port).matches()) {`
`return false;`
We used many of Agan’s principles in our test of this bug. We tried to “understand the system” since any tests require some domain knowledge using the internet to access UrlValidator information and information about URL. We “tried to make it” by providing good, bad, and boundary tests. We modified the tests one test at a time to try to isolate the problem which allowed us to be able to search the code for the possible problem. We “kept an audit trail” as a team to be able to share with other team members so they could improve on the tests. In the end we “did not really fix” the problem but we could with the information we found.

VALIDATOR-2	No support for latest gTLD’s
Details	
Type: Improvement	Affects Version/s: 1.4 Revision 1227719
Status: Open	Fix Version/s: None
Priority: Minor	Component/s: Routines
Resolution: Unresolved	Labels: None
Description	

New generic top-level domains are being approved everyday by ICANN to provide businesses with more branding opportunities and break domains into smaller categories. The current list of approved names includes almost one thousand gTLDs and can be found here: https://newgtlds.icann.org/en/program-status/delegated-strings.

The UrlValidator currently only recognizes the following domains:
- Infrastructure: arpa, root
- Generic: aero, asia, biz, cat, com, coop, info, jobs, mobi, museum, name, net, org, pro, tel, travel, gov, edu, mil, int
- Country codes: ac-it, 109 total

How to Reproduce

Steps to Reproduce:
1. urlVal = new UrlValidator();
2. assertTrue(urlVal.isValid("http://www.amazon.rocks"));

Expected Result: assert passes and code continues execution
Actual Result: assert fails and execution halts

We found the bug through manual testing of good, bad, and boundary values.

Debugging

We used the Eclipse debugger and set a breakpoint at the assert statement. We then stepped into isValid > isValidAuthority and found that the code uses DomainValidator to check the domain. We stepped into DomainValidator > isValid > isValidTld and found that this method returns true if isValidInfrastructureTld, isValidGenericTld, or isValidCountryCodeTld return true. Each of these methods compares the input TLD to a list of known, valid TLDs as follows:
- INFRASTRUCTURE_TLDS: arpa, root
- GENERIC_TLDS: aero, asia, biz, cat, com, coop, info, jobs, mobi, museum, name, net, org, pro, tel, travel, gov, edu, mil, int
- COUNTRY_CODE_TLDS: ac-it, 109 total

The definition for isValidTld occurs in lines 153-160 of DomainValidator.java. The list definitions for valid TLDs is found in lines 220-357.
We used many of Agan’s rules in our debugging process, particularly understand the system, make it fail, and quit thinking and look. To understand the system, we researched the RFCs and reviewed wikipedia and the ICANN website to understand the history and current state of TLD approval. We made the system fail by asserting valid TLDs that were approved but did not fall in the original group of gTLDs. Before forming a hypothesis about why the newer gTLDs were failing, we did the debugging steps listed above. This was a very useful approach because it led us to the specific list of TLDs being allowed. This allowed us to understand the bug beyond just the single TLD that we tested.

VALIDATOR-3	IP address with invalid octet (>255) passes as valid
Details	
Type: Bug	Affects Version/s: 1.4 Revision 1227719
Status: Open	Fix Version/s: None
Priority: Major	Component/s: Routines
Resolution: Unresolved	Labels: None
Description	
Valid IPv4 addresses are written in dot notation and composed of 4 octets, each with a value between 0 and 255. The full range of valid IP addresses is 0.0.0.0 - 255.255.255.255. When testing an IP address with an octet greater than 255 (e.g. 5.5.5.256), isValid should return false but is currently returning true.	
How to Reproduce	
Steps to Reproduce: 1. urlVal = new UrlValidator(); 2. assertFalse(urlVal.isValid("http://5.5.5.256")); Expected Result: assert passes and code continues execution Actual Result: assert fails and execution halts We found the bug through manual testing of good, bad, and boundary values.	
Debugging	
We used the Eclipse debugger and set a breakpoint at the assert statement. We then stepped into isValid > isValidAuthority which then uses DomainVaidator to check the hostname. Domain Validator correctly returns false (because the authority does not include a hostname), so the code then checks to see if the URL uses a valid IP address instead using InetAddressValidator. We stepped into InetAddressValidator isValid > isValidInet4Address. This method breaks the IP address into its 4 separate segments and checks that each one is valid. The last check in the sequence, on lines 94-98 of InetAddressValidator.java, is “if (iIpSegment > 255) { return true; }”, which is incorrect. Valid segments must be between 0 and 255. The condition should return false if a segment is greater than 255. We used many of Agan’s rules in our debugging process, particularly understand the system, make it fail, and quit thinking and look. To understand the system, we used our knowledge from the Intro to Computer Networks class about what constitutes a valid IP address. We then made the system fail by asserting invalid IP addresses that did not follow these rules. When seeing the failure, our original assumption may have been that the code was not even checking for values over the limits, but rather than forming a hypothesis we looked at the code first. What we found is that the code does actually do a check, but the logic was implemented incorrectly.	

VALIDATOR-4	Custom authority validator does not override built-in validator
Details	
Type: Bug	Affects Version/s: 1.4 Revision 1227719
Status: Open	Fix Version/s: None
Priority: Major	Component/s: Routines
Resolution: Unresolved	Labels: None

Description

The methods isValidScheme and isValidAuthority both allow the built-in validator to be replaced by custom validators supplied to the isValid constructor. When using a custom scheme validator, it completely replaces the built-in validator. When using a custom authority validator, we would expect the same but found that not to be the case. isValidAuthority uses the custom validator, but if the authority fails, it then checks the authority using the built-in validator rather than just returning false.

How to Reproduce

Steps to Reproduce:

1. Create a custom RegexValidator (e.g. 1-5 digits, then 1-3 alpha separated by colon)
   RegexValidator myAuthorityRegex = new RegexValidator("^[0-9]{1,5}:[a-zA-z]{1,3}$");
2. Include the RegexValidator parameter when creating a UrlValidator object
   UrlValidator customUrlVal = new UrlValidator(myAuthorityRegex,
   UrlValidator.ALLOW_ALL_SCHEMES);
3. AssertFalse on a URL that does not match the custom Regex, but does pass the standard isValidAuthority test
   assertFalse(customUrlVal.isValid("http://www.google.com"));

Expected Result: assert passes and code continues execution
Actual Result: assert fails and execution halts

We found the bug through domain input partitioning. We attempted to test a value for each one of the checks performed in the code, including custom validators.

Debugging

We used the Eclipse debugger and set a breakpoint at the assert statement. We then stepped into isValid > isValidAuthority. At line 367, authorityValidator.isValid(authority) returns false. At that point, the isValidAuthority method should return false also, but it doesn't have an else statement. It continues on and runs the built-in authority check.
We used many of Agan's rules in our debugging process, particularly understanding the system. As

we thought about input partitioning, we went through the entire code base and discovered every check that a URL must pass before being determined valid. We also studied the system to understand how to create a custom regular expression and pass it into isValid. This allowed us to test non-standard cases and find this bug that may have been missed with just the typical valid and invalid URL testing.

VALIDATOR-5	Query Validation is showing true for false and false for true.
Details	
Type: Bug	Affects Version/s: 2.0 Revision: 1128446
Status: Open	Fix Version/s: None
Priority: Major	Component/s: Routines
Resolution: Unresolved	Labels: None
Description	

UrlValidator is returning true for false and false for true.
The following code returns false when it should return true.
 	
urlVal = new UrlValidator(UrlValidator.ALLOW_ALL_SCHEMES);
assertFalse(urlVal.isValid("http://www.foo.gov/somefolder?name=bar"));
 	
"?name=bar" is a valid query within a URL but returns a false.	

How to Reproduce	
Steps to Reproduce: 1. urlVal = new UrlValidator(UrlValidator.ALLOW_ALL_SCHEMES); 2. assertFalse(urlVal.isValid("http://www.foo.gov/somefolder?name=bar")); Expected Result: True Result Actual Result: False Result We found the error by good, bad, and boundary testing for ports.	
Debugging	
We used the Eclipse debugger to set breakpoints before and after the code that did not work as expected. We reviewed the UrlValidator to look for code related to URL Query and set additional breakpoints. We looked up information about what valid queries should be to set the tests. The cause of the error is an "!" on the QUERY_PATTERN.matcher(query).matches();. The code that is	

bad is line 446. This returns false for true and true for false.

```
 protected boolean isValidQuery(String query) {
            if (query == null) {
            return true;
            }

            return !QUERY_PATTERN.matcher(query).matches(); //Line 446
            }
```

We used many of Agan’s principles in our test of this bug. We tried to “understand the system” since any tests require some domain knowledge using the internet to access UrlValidator information and information about URL. We “tried to make it” by providing good, bad, and boundary tests. We modified the tests one test at a time to try to isolate the problem which allowed us to be able to search the code for the possible problem. We “kept an audit trail” as a team to be able to share with other team members so they could improve on the tests. In the end we “did not really fix” the problem but we could with the information we found.

VALIDATOR-6	User information is not supported (part of authority)
Details	
Type: Bug	Affects Version/s: 2.0 Revision: 1128446
Status: Open	Fix Version/s: None
Priority: Major	Component/s: Routines
Resolution: Unresolved	Labels: None
Description	
RFC 3986 defines a valid authority as: authority = [userinfo "@"] host [":" port] The user info allows users to provide an optional username and password (not recommended) for authentication. The current isValidAuthority routine does not recognize valid user info and returns false.	
How to Reproduce	
Steps to Reproduce: 1. urlVal = new UrlValidator(UrlValidator.ALLOW_ALL_SCHEMES); 2. assertTrue(urlVal.isValid("http://user@foo.gov")); Expected Result: assert passes and code continues execution Actual Result: assert fails and execution halts	

We found the error by good, bad, and boundary testing for authorities.

Debugging

We used the Eclipse debugger and set a breakpoint at the assert statement. We then stepped into isValid > isValidAuthority. At line 372, isValidAuthority calls for a pattern match against AUTHORITY_PATTERN. In line 137, AUTHORITY_PATTERN points to the regular expression AUTHORITY_REGEX on line 134. This regular expression only allows AUTHORITY_CHARS_REGEX, which is defined on line 102 as "\\p{Alnum}\\-\\.", or alphanumeric, -, and . characters. It does not include the @ symbol required for user info.

We used many of Agan's rules in our debugging process, particularly understanding the system. We carefully studied the RFC's to understand what was required for each piece of the URL. We also studied the system to understand the regular expressions used and how these compared to the expected values from the RFC. We also "quit thinking and look". There could have been any number of reasons user info was returning false, but rather than creating a hypothesis we looked into the code first and found the regular expression.