Project Report Part 2

Corinna Huffaker
Jeromie Clark
Bolan Peng
Damian Mecham

Github

https://github.com/OSU-CS362-F16/f16-project-cs362-assignment-5-west-coast

Code Baseline

The project test plan is based on evaluating the BuggyUrlValidator code provided at http://web.engr.oregonstate.edu/~baldwdav/BuggyURLValidator.zip

Test Techniques

Describe the techniques you used and include snippets of relevant testing code

Requirements-Based Input Domain Testing

Requirements for URL Validator

A specification of the BuggyURLValidator codebase was not provided. Comments in the code imply that the validator should be compliant with RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax which is available at: http://www.ietf.org/rfc/rfc2396.txt

The following discussion of these requirements are the basis for our unit test input domain testing. The RFC 2396 standard allows for many variants of the syntax. The input domain testing focused on character set and syntax for segments of the URI. A valid returned value from the URL Validator may not match the requirements for a specific scheme. As stated in our plan, ideas for valid and not valid URL inputs were drawn from the following two websites as well as generated from review of the standard. In each subsection below, the requirement and corresponding input files that cover the test case are outlined.

Example sources of known test cases (not necessarily rfc2396.txt compliant):
http://greenbytes.de/tech/webdav/urldecomp.xml
https://mathiasbynens.be/demo/url-regex

Note: Excerpts from the standards outlined below are italicized. The standard uses "|" to indicate alternatives.

Null Input

Unit tests were constructed to verify that null input is handled by the URLValidator for all constructor types. Examples include:

```
UrlValidator uv1 = new UrlValidator(schemes);
assertFalse(uv1.isValid(null));
```

Character Set

Characters allowed by the rfc2396.txt standard that are 'unreserved' include upper and lower case letters, decimal digits, and a limited set of punctuation marks and symbols.

unreserved = alphanum | mark
mark = "-" | "_" | "." | "!" | "~" | "" | "'" | "(" | ")"*

From rfc2396.txt, some characters are reserved and require an escape sequence to use in some but not all contexts. The following is the reserved character set:

reserved = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","

Additionally, data must be properly encoded if it does not correspond to printable characters in the US-ASCII coded character set. Escape encoding is comprised of a percent character "%" followed by two hexadecimal digits representing the octet code.

The example given in rfc2396.txt is "%20" representing the space character. The "%" character must always be escaped as "%25" to represent the character. Excluded characters as defined by rfc2396.txt must be escaped for proper representation in the URI and include the following sets of characters:

control = <US-ASCII coded characters 00-1F and 7F hexadecimal>
space = <US-ASCII coded character 20 hexadecimal>
delims = "<" | ">" | "#" | "%" | <">
unwise = "{" | "}" | "|" | "\" | "^" | "[" | "]" | "`"

The following input domains were defined to test this character set:

1. Correct syntax
   1.1 Using unreserved characters

A bug was identified where the "mark" list of characters are not allowed in the authority section.

   1.2 Using unreserved and reserved escaped characters
   1.3 Using reserved escaped characters

1.4 Using unreserved characters and escaped excluded characters

2. Incorrect syntax
   1.1 Using unreserved characters and excluded characters (see "URLValidatorTest.urlWithCommaTest" for examples)
   1.2 Using unreserved and excluded (not escaped) characters
   1.3 Using reserved (not escaped) characters

These conditions were tested with the random input generator described in the next section as well as input files listed in ./TestData/input directory. UrlValidatorTest class reads the strings in from file in input directory and tests "isValid" condition. An example test case:

```
public void urlValidatorAllowTwoSlashesTest() {
        List<String> inputListStrings = readStrings("./TestData/input/rfc2396URI_02.txt");
        UrlValidator uvAll = new UrlValidator(UrlValidator.ALLOW_2_SLASHES);
        UrlValidator uv = new UrlValidator(); // default does not allow two slashes
        for (int j=0; j<inputListStrings.size(); j++) {
                assertTrue("ALLOW_2_SLASHES setting ON, should return True for \"" +
                        inputListStrings.get(j) + "\" " , uvAll.isValid(inputListStrings.get(j)));
                assertFalse("ALLOW_2_SLASHES setting OFF, should return False for \"" +
                        inputListStrings.get(j) + "\" " , uv.isValid(inputListStrings.get(j)));
        }
}
```

Syntax

Syntax for the URI is dependent on syntax of the scheme. The rfc2396.txt defines the absolute URI as <scheme>:<scheme-specific-part>. The scheme-specific part that represents hierarchical relationships within the namespace or "generic URI" consists of four main components:

<scheme>://<authority><path>?<query>

The authority, path and query components are dependent on the scheme type and may not be allowed. The "/" character may be used for separating hierarchical components. The hierarchical component can be further divided into subcomponents as follows:

hier_part = (net_path | abs_path) ["?" query]
net_path = "//" authority [abs_path]
abs_path = "/" path_segments

Alternatively, instead of the abs_path, an 'opaque_part' where the slash character is not used, may define the path. The opaque_part can be represented as:

opaque_part = unreserved | escaped | ";" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","

A list of potentially valid URLs was developed from two source websites, the standard, and known good links on the internet. This list was further categorized by the modifiers:

ALLOW_ALL_SCHEMES (see TestData/input/rfc2396URI_01.txt and TestData/input/rfc2396URI_01_default.txt)
ALLOW_2_SLASHES (see TestData/input/rfc2396URI_02.txt)
NO_FRAGMENTS (see TestData/input/rfc2396URI_04.txt)
ALLOW_LOCAL_URLS (see TestData/input/rfc2396URI_08.txt)

The URLValidatorUnitTest tests each of the following sections by extending the URLValidator class to allow access to protected methods.

Scheme

Scheme names begin with a lowercase letter and are followed by any combination of lowercase letters, digits, plus ("+"), period ("."), or hyphen ("-"). The specification requires upper case to be interpreted as lower case.

*scheme = alpha *(alpha | digit | "+" | "-" | ".")*

Authority

The rfc2396.txt standard doesn't actually define specific authority elements, as these are defined by an Internet-based server or a scheme-specific registry of naming authorities.

The general form of an authority is:

authority = server | reg_name

The authority component is preceded by a "//" and is terminated by the next slash, question-mark or by the end of the URI. Reserved characters within the authority are ";", ":", "@", "?", and "/".

The authority element is not required for relative references. There are also Registry and Server based Naming Authorities. A server based Naming Authority using IP-based protocol has the following format:

<userinfo>@<host>:<port>

Only the host is required. If the userinfo is present it is followed by an "@" sign. Some URL schemes use the format "user:password" in the userinfo field.

hostport = host [":" port]
host = hostname | IPv4address
*hostname = *(domainlabel ".") toplabel ["."]*
*domainlabel = alphanum | alphanum *(alphanum | "-") alphanum*
*toplabel = alpha | alpha *(alphanum | "-") alphanum*

*IPv4address = 1*digit "." 1*digit "." 1*digit "." 1*digit*
*port = *digit*

Hostnames are a sequence of domain labels separated by a ".". Domain labels start and end with an alphanumeric character and possibly contain "-" characters. The rightmost domain label of a fully qualified domain name will never start with a digit. If a port number other than the default is required, the port may be specified in decimal separated from the host by a colon.

Every HTTP URL conforms to the syntax of a generic URI. A generic URI is of the form:
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
The user:password and port are optional components.

In addition to the random tests, tests for the Authority include:

```
assertFalse("null input to isValidAuthority returns true",uv.isValidAuthority(null));
assertFalse("authorityMatcher should not match but returns true",uv.isValidAuthority(""));
assertTrue("www.cnn.com:123",uv.isValidAuthority("www.cnn.com:123"));
assertTrue("http://www.cnn.com:123/foo/", uv.isValid("http://www.cnn.com:123/foo/"));
assertFalse("http://www.cnn.com:123:abcde/foo/", uv.isValid("http://www.cnn.com:123:abcde/foo/"));
// BUG: Authentication information is part of the authority, but always fails
// assertTrue("bob:foo@www.cnn.com:123",uv.isValidAuthority("bob:foo@www.cnn.com:123"));
assertFalse("authority \"www\" is not valid",uv.isValid("http://www/Addressing/"));
assertTrue("authority \"www.w3.org\" is valid",uv.isValid("http://www.w3.org/Addressing/"));
```

Tests with authority validator set:

```
RegexValidator av = new RegexValidator("foo");
UrlValidator uv = new UrlValidator(null, av, 0);
assertTrue("custom validator expects foo", uv.isValidAuthority("foo"));
assertFalse("custom validator - no match", uv.isValidAuthority("blerp"));
assertFalse("custom validator - null", uv.isValidAuthority(null));
assertFalse("custom validator - no match", uv.isValidAuthority("foo:foo.blerp asdf"));
```

Path

The path must begin with a single slash (/) if an authority part was present, and may also if one was not, but must not begin with a double slash.

path = [abs_path | opaque_part]
*path_segments = segment *("/" segment)*
*segment = *pchar *(";" param)*
*param = *pchar*
pchar = unreserved | escaped | ":" | "@" | "&" | "=" | "+" | "$" | ","
the characters "/", ";", "=", and "?" are reserved

The path segment can contain parameters separated by a ";".

In addition to the random tests, tests for the Path include:

```
assertFalse("null input to isValidPath returns true",uv.isValidPath(null));
assertTrue("empty path \"\" returns false",uv.isValidPath(""));
assertFalse("IsValidPath - Complex, Double and Single Dots, Double-Slash Disallowed",
        uv.isValidPath("/F/LUUGNNPWO/MUMSS/../DFYH./MARWDO/RHN//./JIBPWDJHFDOGW/G/QCJ.html"));
assertTrue("IsValidPath - Complex, Double and Single Dots, No Double-Slash",
        uv.isValidPath("/F/LUUGNNPWO/MUMSS/../DFYH./MARWDO/RHN/./JIBPWDJHFDOGW/G/QCJ.html"));
assertTrue("IsValidPath - Complex, Single Dots",
       uv.isValidPath("/F/PSEJK/LUUGNNPWO.MUMSS/DFYHMARWDO/RHN/./JIBPWDJHFDOGW/G/QCJ"));
assertTrue("IsValidPath - Complex, multiple parameters", uv.isValidPath("/F/PSEJK/a=b;a=c"));
assertFalse("IsValidPath - doesn't start with /",
       uv.isValidPath("F/PSEJK/LUUGNNPWO.MUMSS/DFYHMARWDO/RHN/./JIBPWDJHFDOGW/G/QCJ"));
assertTrue("http://news.example.com/money/$5.2-billion-merger is valid",
        uv.isValid("http://news.example.com/money/$5.2-billion-merger"));
assertTrue("http://blog.example.com/wutchoo-talkin'-bout-willis!? is valid",
        uv.isValid("http://blog.example.com/wutchoo-talkin'-bout-willis!?"));
assertTrue("https://spam.example.com/viagra-only-$2-per-pill* is valid",
        uv.isValid("https://spam.example.com/viagra-only-$2-per-pill*"));
```

Query

The query component is an optional string.

> *Within a query component, the characters ";", "/", "?", ":", "@", "&", "=", "+", ",", and "$" are reserved.*

In addition to the random tests, tests for the Query include:

```
assertTrue("null input to isValidQuery should return true",uv.isValidQuery(null));
assertTrue("query of blank space \"\" returns true",uv.isValidQuery(""));
assertTrue("valid query \"?([^#]*)\" returns false for \"?abc=1\"", uv.isValidQuery("abc=1") );
assertTrue("valid query, mulitple params", uv.isValidQuery("abc=1&def=2"));
// BUG (3): isValidQuery() doesn't really check to see if the punctuation between parameters makes sense
// assertFalse("invalid query section, crazy punctuation", uv.isValidQuery("abc==123&&&=def=1234"));
// BUG (9): isValidQuery doesn't flag queries with disallowed characters as invalid
// assertFalse("invalid query - disallowed characters", uv.isValidQuery("?kp.ndvrlq1Q,M'+UZYK?`zqhzb%a>~A\""));
```

Fragment

An optional string preceded by a “#” and character limitations as defined by the character set.
In addition to the random tests, tests for the Fragment include:

Tests with the NO_FRAGMENTS modifier set on:

```
assertFalse("NO_FRAGMENTS - otherwise valid fragment", uvNoFragments.isValidFragment("WARNING"));
assertFalse("NO_FRAGMENTS - random punctuation",
        uvNoFragments.isValidFragment("./?$23&%*(@#``!!($)@[];,,..//\\*&%^$%#@!*())"));
assertTrue("NO_FRAGMENTS - isValidFragment is NULL",uvNoFragments.isValidFragment(null));
```

Tests with the NO_FRAGMENTS modifier set off:

```
 assertTrue("NO_FRAGMENTS is not set",uv.isValidFragment("WARNING"));
assertTrue("Valid Fragment",uv.isValidFragment("WARNING"));
```

```
// BUG (4): isValidFragment() fails to detect improper usage of % (there's probably more bugs to mine in this vein...)
 // assertFalse("FRAGMENTS - random punctuation",
//            uv.isValidFragment("./?%$2%3&%*%(@#`%`!!(%$)@[];,,%..//\\*%&%^$%%#@!*%())%"));
assertTrue("FRAGMENTS - percent encoding",
            uv.isValidFragment("If%20you%20haven%27t%20got%20anything%20nice%20to%20say%20about%20anybody%2C%20come%20sit%20next%20to%20me.%20"));
 assertTrue("FRAGMENTS - NULL Fragment is valid",uv.isValidFragment(null));
 assertTrue("FRAGMENTS - Empty Fragment is valid",uv.isValidFragment("#"));
// BUG (8): isValidFragment() fails to detect special characters in the fragment
// assertFalse("FRAGMENTS - Special Characters",
//            uv.isValidFragment("http://www.ics.uci.edu/pub/ietf/uri/historical.html#EnZ^b1XXL"));
 assertTrue("FRAGMENTS - No Special Characters",
            uv.isValidFragment("http://www.ics.uci.edu/pub/ietf/uri/historical.html#EnZb1XXL"));
```

Domain Names

A hostname assigned to a host computer.

From the Domain Validator notes:

> http://www.ietf.org/rfc/rfc1034.txt, section 3
> http://www.ietf.org/rfc/rfc1123.txt, section 2.1
> Validation is also provided for top-level domains (TLDs) as defined and maintained by the Internet Assigned Numbers Authority (IANA). List at
> http://data.iana.org/TLD/tlds-alpha-by-domain.txt

Unit Testing

The high-level approach to testing was to maximize line and branch coverage of all classes in BuggyURLValidator. Initially, we divided the classes up between team members and wrote unit tests with the intent to maximize coverage as shown in the Cobertura coverage report. We then used Cobertura (and later, PIT) to guide the creation of new tests to fill the gaps. The Cobertura report shows 96% line coverage and 90% branch coverage for the 5 classes in BuggyURLValidator project.

There are a large number of test cases required for the URLValidator class so a set of input files was developed to allow checking many possible scenarios and also to allow for easily adding additional test cases. The initial approach was to develop the test cases and use the InetOracle to test for validity. Unfortunately, there were differences between the oracles and the rfc2396 requirements as defined above. Therefore, a series of tests qualifying the inputs was used and in some cases, the interpretation of correctness was a judgement call. As cases were identified that didn't pass, these were either commented out or written as separate tests in the URLValidatorUnitTest class.

Random URL Testing

Simply passing long strings of random characters would rarely result in a properly formatted URL, which would result in an ineffective, and computationally impractical approach to testing.

To inject randomness in a more efficient way, we divided the classes of input into four categories:

- Integer
- AlphaNumeric
- Printable
- Extended ASCII

The last two categories are unlikely to produce valid URLs, but they provide interesting coverage for invalid input.

We also divided the URL into its constituent parts:

- Scheme
- Authority
    - Domain
    - Port
- Query
- Fragment

We then focused on an individual section of the URL, injecting randomness from each of our classes of input.

```
@Test
public void testRandomQueryPrintableWithFragment() {
        Random r = new Random();
        UrlValidator validator = new UrlValidator();
        UrlValidatorOracle oracle = new UrlValidatorOracle();
        String testString = new String();

        for (int i = 0; i < 1000; i++) {
                testString = "http://www.osu.edu/bloop/derp/blerg/form.php?";
                testString += getRandomQuery(1);
                testString += "#asdf23857ogihsnv";

                // Provide some useful output
                String message = new String();
                message = "Test String: ";
                message += testString;
                message += " Validator: ";
                message += validator.isValid(testString);
                message += " Oracle: ";
                message += oracle.isValid(testString);
                // System.out.println(testString);
                // BUG (9): Validator doesn't flag invalid characters in the query
```

```
                // assertTrue(message,
                        (validator.isValid(testString) == oracle.isValid(testString)));
            }
        }



        // Generates a random query
        // Ensure that there are always some = and & characters
        // int group:
        // 0 - AlphaNumeric Strings (probably valid)
        // 1 - Printable ASCII (potentially valid)
        // 2 - Extended ASCII (potentially valid, but unlikely)

        private String getRandomQuery(Integer group) {
                Random r = new Random();
                String result = new String();
                String appendChar = new String();

                int len = r.nextInt(64);
                int nextChar = 0;
                if (group < 0 || group > 2) {
                        group = r.nextInt(2);
                }

                for (int i = 0; i < len; i++) {
                        switch(group) {
                                case 0:
                                        // 1/10th of the alpha string will be = or &
                                        nextChar = r.nextInt(10);
                                        if (nextChar == 0) {
                                                appendChar = "=";
                                        } else if (nextChar == 1) {
                                                appendChar = "&";
                                        } else {
                                                appendChar =
Character.toString((char)(r.nextInt(25) + 65));
                                        }
                                        break;
                                case 1:
                                        nextChar = r.nextInt(97) + 32;
                                        if (nextChar == 128) {
                                                appendChar = "=";
                                        } else if (nextChar == 129) {
                                                appendChar = "&";
                                        } else {
                                                appendChar = Character.toString((char)nextChar);
                                        }
                                        break;
                                default:
                                        nextChar = r.nextInt(257);
                                        if (nextChar == 256) {
                                                appendChar = "=";
                                        } else if (nextChar == 257) {
                                                appendChar = "&";
                                        } else {
                                                appendChar = Character.toString((char)nextChar);
```

```
                                }
                        break;
                }
                result += appendChar;
        }
return result;
}
```

While quality unit tests guided by code coverage yielded a large number of interesting bugs, random URL testing exposed some additional problems filtering invalid input.

Evaluation of New Tools

PIT Mutation Coverage

The team used the PIT mutation testing system to validate the completeness and efficacy of our unit tests. PIT mutates the actual application code to determine whether or not errors injected into the application trigger corresponding failures in the existing unit test suite.

The combination of JUNIT, PIT and Maven effectively require that all of the unit tests pass during test execution in order to get valid PIT results (otherwise execution of individual sub-tests may halt, skewing the results), so to accommodate the requirement, we commented out failing unit tests, documenting them with //BUG: <description> in the test source.

The results generated by PIT indicate that our code is covered well, and in general, the tests themselves are very effective at catching new bugs inserted into the codebase. We have identified a few blocks of code that seem to be unreachable either due to redundant validation checks (a condition sufficient to trigger the conditional would never result in a regular expression match), and the majority of the items called out by PIT correlate directly with the code we were unable to reach through unit testing.

As an example, the line “for (int i = 0; i <= 3; i++)...” loops over a group of regular expression matches. PIT injects faults into the conditional of this for loop, which doesn't necessarily make a lot of sense, as the conditional logic is self-contained.

In another instance the mutation coverage report identifies an area of unreachable code, which it flags as “replaced return of integer sized value with (x == 0 ? 1 : 0) → NO_COVERAGE”. Further inspection shows that the statement “String[] groups = ipv4Validator.match(inet4Address);” returns null if there are less than four groups, therefore, the

method exits before reaching the test condition. While it is a bug, unused code would most likely be treated as a low priority issue by the engineering team. (See Bug Report 10: Unused code in isValidInet4Address method)

That said, for an automated analysis, PIT provided valuable assurance that our tests were reasonably comprehensive, with a low amount of overhead in terms of triaging false-positives.

FindBugs

The team used findbugs-maven-plugin version 3.0.4. FindBugs does a static analysis of the code and provides a report identifying code that does not meet correctness, style and bad practice criteria. The report generated for BuggyURLValidator only identified two issues with the ResultPair class where the item and valid parameters don't meet the Style requirements due to "Unread public/protected field". This was interesting but not particularly useful for resolving issues with the URL validator.

Bug Report List:

Bug ID: `1`

Title: `Java classes are missing package designators`

Product: `URLValidator main classes`

Platform: `CentOS Linux Relea se 7.2.1511 (core)`

Reproducibility: `Always`

Description
BuggyURLValidator classes are not included in a package. This could result in potential naming conflicts with java system calls. This bug fix eliminates a potential problem with syntax that would prevent compilation.

Steps to Reproduce: Found using the Eclipse syntax checker.

Commit Text:
`BUGFIX:` Add missing "package osu.cs362.URLValidator;" to java classes in project.

Bug ID: `2`

Title: InetAddressValidator.isValid() erroneously returns true when address segment values are larger than 255.

Product: `BuggyURLValidator`

Platform: `CentOS Linux Relea se 7.2.1511 (core)`

Reproducibility: Always

Description: A valid IP address is a tuple of 4 integers from 0 - 255 separated by dots, but InetAddressValidator.isValid() erroneously returns true for invalid IP addresses, like 192.168.999.256

Steps to Reproduce:
InetAddressValidator iv = InetAddressValidator.getInstance();
assertFalse("192.168.999.256", iv.isValid("192.168.999.256"));

Expected Behavior:
iv.isValid() returns false

Actual Behavior:
iv.isValid() returns true

Test Case:
```
 @Test
 // Invoking the getInstance() without a parameter should treat local URLs as invalid
 public void testIsValidInet4Address() {
        InetAddressValidator iv = InetAddressValidator.getInstance();
        assertFalse("192.168.999.256", iv.isValid("192.168.999.256"));
 |
```

Bug ID: 3

Title: URLValidator.isValidQuery() doesn't check for valid punctua tion

Product: BuggyURLValidator

Platform: CentOS Linux Relea se 7.2.1511 (core)

Reproducibility: Always

Description:
As long as the string supplied to isValidQuery() contains allowed characters, the string is validated regardless of whether or not those characters are in a meaningful contextual placement.

Steps to Reproduce:
```
assertFalse("invalid query section, crazy punctuation",
uv.isValidQuery("abc==123&&&=def=1234"));
```

Expected Behavior:
isValidQuery() returns true

Actual Behavior:
isValidQuery() returns false

Test Case:
```
        @Test
        public void IsValidQueryTest() {
                UrlValidatorExtension uv = new UrlValidatorExtension();
                assertTrue("valid query, mulitple params", uv.isValidQuery("abc=1&def=2"));
```

```
                // BUG: isValidQuery() doesn't really check to see if the punctuation between
                // parameters makes sense
                assertFalse("invalid query section, crazy punctuation",
                        uv.isValidQuery("abc==123&&&=def=1234"));
        }
```

Commit Text:
```
BUGFIX: UrlValidator.isValidQuery() returned an inverse of the correct result.
It was really blocking any interesting testing, so I just fixed it.
```

Code Snippet:

```
 -
 -        return !QUERY_PATTERN.matcher(query).matches();
 +
 +        // BUG: Output of isValidQuery was inverted
 +        return QUERY_PATTERN.matcher(query).matches();
```

Bug ID: `4`

Title: `URLValidator.isValidFragment() fails to detect improper usa  ge of %`

Product: `BuggyURLValidator`

Platform: `CentOS Linux Relea se 7.2.1511 (core)`

Reproducibility: `Always`

Description:
```
isValidFragment() validates that only allowed characters are included, but it does not
validate that they make contextual sense.  In this instance, percent-encoded characters are
allowed, but you shouldn't be able to just toss percents followed by random special characters
around...
```

Steps to Reproduce:
```
assertTrue("FRAGMENTS - random punctuation",
uv.isValidFragment("./?%$2%3&%*%(@#`%`!!(%$)@[];,,%..//\\*%&%^$%%#@!*%())%"));
```

Expected Behavior:
```
isValidFragment() returns true
```

Actual Behavior:
```
isValidFragment() returns false
```

Test Case:
```
        @Test
        public void IsValidQueryTest() {
                UrlValidatorExtension uv = new UrlValidatorExtension();
                assertTrue("valid query, mulitple params", uv.isValidQuery("abc=1&def=2"));

                // BUG: isValidQuery() doesn't really check to see if the punctuation between
                // parameters makes sense
```

```
                assertFalse("invalid query section, crazy punctuation",
                        uv.isValidQuery("abc==123&&&=def=1234"));
        }
```

Commit Text:
```
BUGFIX: UrlValidator.isValidQuery() returned an inverse of the correct result.
It was really blocking any interesting testing, so I just fixed it.
```

Code Snippet:

```
 -
 -         return !QUERY_PATTERN.matcher(query).matches();
 +
 +         // BUG: Output of isValidQuery was inverted
 +         return QUERY_PATTERN.matcher(query).matches();
```

Bug ID: `5`

Title: `DomainValidator.isValid() fails to detect unicode URLs`

Product: `BuggyURLValidator`

Platform: `CentOS Linux Relea se 7.2.1511 (core)`

Reproducibility: `Always`

Description:
```
Only ASCII characters are allowed in domains, but DomainValidator.isValid returns true for
Unicode strings.
```

Steps to Reproduce:
```
assertTrue("名がドメイン.com", dv.isValid("名がドメイン.com")
```

Expected Behavior:
```
isValid() returns true
```

Actual Behavior:
```
isValid() returns false
```

Test Case:

```
public void testIsValid() {
    DomainValidator dv = DomainValidator.getInstance(true);

    // BUG (5): Unicode -- Only ASCII is allowed in URLs, and I don't think we're validating
    // IRIs here
    assertTrue("名がドメイン.com", dv.isValid("名がドメイン.com"));
  }
```

Bug ID: `6`

Title: `DomainValidator.isValid() returns incorrect result for "."`

Product: `BuggyURLValidator`

Platform: `CentOS Linux Release 7.2.1511 (core)`

Reproducibility: `Always`

Description:

```
According to RFC 2181: The zero length full name is defined as representing the root of the
DNS tree, and is typically written and displayed as "."
```

Steps to Reproduce:

```
dv.isValid(".");
dv.isValid("");
```

Expected Behavior:

```
It's not completely clear to me what the correct answer here is, but per RFC 2181, either a
zero-length domain "", "." or both should validate.  In this instance, neither validate.

assertTrue(".", dv.isValid("."));
assertFalse("Empty String", dv.isValid(""));
```

Actual Behavior:

```
DomainValidator.isValid(".") returns false
DomainValidator.isValid("") returns false
```

Test Case:

```
public void testIsValid() {

  DomainValidator dv = DomainValidator.getInstance(true);
  // BUG (6): RFC 2181: The zero length full name is defined as representing the root of
  // the DNS tree, and is typically written and displayed as ".

  assertFalse("Empty String", dv.isValid(""));
  assertTrue(".", dv.isValid("."));
}
```

Bug ID: `7`

Title: `DomainValidator.isValid() returns true for URLs > 255 characters`

Product: `BuggyURLValidator`

Platform: `CentOS Linux Release 7.2.1511 (core)`

Reproducibility: `Always`

Description:

```
BUG (7): Per RFC 1123, 255 character hostnames should be supported.  We validate hostnames
that are longer, but this may not be compatible with all clients.
```

Steps to Reproduce:

```
dv.isValid("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.com"));
```

Expected Behavior:
DomainValidator.isValid() returns false

Actual Behavior:
DomainValidator.isValid() returns true

Test Case:

```
public void testIsValid() {

  DomainValidator dv = DomainValidator.getInstance(true);
  // BUG (7): Per RFC 1123 says 255 character hostnames should be supported.
  // We validate hostnames that are longer, but this may not be compatible with all clients.

assertFalse("256 Character Hostname",
dv.isValid("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.com"));

}
```

Bug ID: 8

Title: DomainValidator.isValidFragment() erroneously returns true for fragments with special characters

Product: BuggyURLValidator

Platform: CentOS Linux Relea se 7.2.1511 (core)

Reproducibility: Always

Description:
isValidFragment() fails to detect special characters in the fragment. This causes URLValidator.isValid() to miss this class of malformed URLs.

Steps to Reproduce:
uv.isValidFragment("http://www.ics.uci.edu/pub/ietf/uri/historical.html#EnZ^b1XXL");

Expected Behavior:
DomainValidator.isValidFragment() returns false

Actual Behavior:
DomainValidator.isValid() returns true

Test Case:

```
@Test
public void IsValidFragmentTest() {
       UrlValidatorExtension uv = new UrlValidatorExtension();  // by default allow fragments

// BUG (8): isValidFragment() fails to detect special characters in the fragment
assertFalse("FRAGMENTS - Special Characters",
uv.isValidFragment("http://www.ics.uci.edu/pub/ietf/uri/historical.html#EnZ^b1XXL"));
```

```
}
```

Bug ID: 9

Title: DomainValidator.isValidQuery() erroneously returns true for queries with special characters

Product: BuggyURLValidator

Platform: CentOS Linux Relea se 7.2.1511 (core)

Reproducibility: Always

Description:
isValidQuery() fails to detect special characters in the query. This causes URLValidator.isValid() to erroneously call this class of malformed URLs valid.

Steps to Reproduce:
```
uv.isValidQuery("?kp.ndvrlq1Q,M'+UZYK?`zqhzb%a>~A\"");
```

Expected Behavior:
DomainValidator.isValidQuery() returns false

Actual Behavior:
DomainValidator.isValidQuery() returns true

Test Case:

```
@Test
public void IsValidQueryTest() {
        UrlValidator uv = new UrlValidator();  // by default allow fragments

        // BUG (9): isValidQuery doesn't flag queries with disallowed characters as invalid
        // assertFalse("invalid query - disallowed characters",
            uv.isValidQuery("?kp.ndvrlq1Q,M'+UZYK?`zqhzb%a>~A\""));
}
```

Bug ID: 10

Title: Unused code in isValidInet4Address method

Product: BuggyURLValidator

Platform: CentOS Linux Relea se 7.2.1511 (core)

Reproducibility: Always

Description:
The isValidInet4Address should return false if there are fewer than 4 groups provided in the IP address. The test case does pass but a failure occurs in the PIT mutation coverage because a test for this condition results in a surviving mutation. In fact, the method tests the same condition twice resulting in unused code that can’t be reached with the unit tests.

Steps to Reproduce:

```
   assertFalse("10.12.100", iv.isValidInet4Address("10.12.100"));
```

This test passes, but does not result in code coverage of the "if (ipSegment == null || ipSegment.length() <= 0)" condition. The 'ipSegment' variable is set from the array groups. The method isValidInet4Address exits on a prior line "if (groups == null) return false;"

Expected Behavior:
The method has the proper response, returns false if the ip address has less than four segments.

Actual Behavior:
Unused code.

Bug ID: 11

Title: Some valid file schemes do not validate correctly

Product: BuggyURLValidator

Platform: CentOS Linux Relea se 7.2.1511 (core)

Reproducibility: Always

Description:

PER: RFC 2396
The path may consist of a sequence of path segments separated by a single slash "/" character. Within a path segment, the characters "/", ";", "=", and "?" are reserved. Each path segment may include a sequence of parameters, indicated by the semicolon ";" character. The parameters are not significant to the parsing of relative references.

```
assertTrue("file:localhost/etc/fstab", uv.isValid("file:localhost/etc/fstab"));
assertTrue("file:/path/index.html", uv.isValid("file:/path/index.html"));
assertTrue("file://path/index.html", uv.isValid("file://path/index.html"));
```

Bug ID: 12

Title: URLValidator.IsValidAuthority() erroneously returns false w hen authentication information is included in the authority section of a URL

Product: BuggyURLValidator

Platform: CentOS Linux Relea se 7.2.1511 (core)

Reproducibility: Always

Description:
When passing authentication information in a URL, the isValidAuthority() check fails. even though authentication information is permissible in URLs.

Steps to Reproduce:
Run the following unit test:

```
assertTrue("bob:foo@www.cnn.com:123",uv.isValidAuthority("bob:foo@www.cnn.com:123"));
```

Expected Behavior:
`DomainValidator.isValidQuery() returns false`

Actual Behavior:
`DomainValidator.isValidQuery() returns true`

Test Case:

```
        @Test
        public void IsValidAuthorityTest() {
                UrlValidator uv = new UrlValidator();
assertTrue("www.cnn.com:123",uv.isValidAuthority("www.cnn.com:123"));
assertTrue("http://www.cnn.com:123/foo/", uv.isValid("http://www.cnn.com:123/foo/"));
        }
```

Bug ID: `13`

Title: `DomainValidator.isValid() was looking at the wrong index in  the regex match for a TLD`

Product: `BuggyURLValidator`

Platform: `CentOS Linux Relea se 7.2.1511 (core)`

Reproducibility: `Always`

Description:
`DomainValidator.isValid() passes the supplied string to a regular expression evaluation, which returns an array of matches. The check for isValidTld() was being performed on the first index of the array instead of the last, so the isValidTld() check would fail, causing all domains to be flagged as invalid. Since this effectively made the class unusable, I simply made the fix to facilitate more interesting testing.`

Steps to Reproduce:
`Call DomainValidator.isValid("www.google.com");`

Expected Behavior:
`DomainValidator.isValid() returns true`

Actual Behavior:
`DomainValidator.isValid() returns false`

Patch
```
@ DomainValidator.java @151
-            return isValidTld(groups[groups.length - 1]);
+            return isValidTld(groups[groups.length - 1]);
```

Bug ID 14

Title: DomainValidator list of international TLDs was truncaed

Product: `BuggyURLValidator`

Platform: `CentOS Linux Relea se 7.2.1511 (core)`

Reproducibility: `Always`

Description:
`The list of valid international TLDs was truncated after Italy, causing any domain with a suffix higher than "it" to be marked as invalid... (e.g. amazon.jp or amazon.co.uk)`

Steps to Reproduce:
`Call DomainValidator.isValid("www.google.com");`

Expected Behavior:
`DomainValidator.isValid() returns true`

Actual Behavior:
`DomainValidator.isValid() returns false`

Patch
```
@ DomainValidator.java @359+
// BUG: The list of valid country codes was truncated.
 +        // It was really annoying so I added it back
 +        "je",                // Jersey
 +        "jm",                // Jamaica
 +        "jo",                // Jordan
 +        "jp",                // Japan
 +        "ke",                // Kenya
 +        "kg",                // Kyrgyzstan
 +        "kh",                // Cambodia (Khmer)
 +        "ki",                // Kiribati
 +        "km",                // Comoros
 +        "kn",                // Saint Kitts and Nevis
 +        "kp",                // North Korea
 +        "kr",                // South Korea
 +        "kw",                // Kuwait
 +        "ky",                // Cayman Islands
```

Bug ID 14

Title: UrlValidator return value of isValidQuery() was inverted

Product: `BuggyURLValidator`

Platform: `CentOS Linux Relea se 7.2.1511 (core)`

Reproducibility: `Always`

Description:
`Someone threw us an easy bug to discover.  The return value of this function was prefaced with !, so all values returned were incorrect.  This caused valid domains with supplied queries to be marked as invalid.`

Steps to Reproduce:

```
Call URLValidator.isValidQuery("abc=1&def=2"));
```

Expected Behavior:
```
isValidQuery() returns true
```

Actual Behavior:
```
isValidQuery() returns false
```

Patch
```
URLValidator.java @453
-        return !QUERY_PATTERN.matcher(query).matches();
 +
 +        // BUG: Output of isValidQuery was inverted
 +        return QUERY_PATTERN.matcher(query).matches();
```

Bug ID 15

Title: DomainValidator return value of isValidLocalTld() was inverted

Product: `BuggyURLValidator`

Platform: `CentOS Linux Relea se 7.2.1511 (core)`

Reproducibility: `Always`

Description:
```
Someone threw us an easy bug to discover.  The return value of this function was prefaced with
!, so all values returned were incorrect.  This caused valid domains with supplied queries to
be marked as invalid.
```

Steps to Reproduce:
```
Call DomainValidator.isValidLocalTld("localhost"));
```

Expected Behavior:
```
isValidLocalTldy() returns true
```

Actual Behavior:
```
isValidLocalTld() returns false
```

Patch
```
DomainValidator.java @ 205
-       return !LOCAL_TLD_LIST.contains(chompLeadingDot(iTld.toLowerCase()));
 +
 +      return LOCAL_TLD_LIST.contains(chompLeadingDot(iTld.toLowerCase()));
```

Other Observations:

It's not completely clear that this is a bug, but cursory research indicates that an IP address of 0 is actually valid in some circumstances [1], yet iv.isValid would return false in this instance.

Depending on the context, this might be a valid outcome, but it seems like something the engineering team would want to make a conscious decision about.

[1] https://en.wikipedia.org/wiki/0.0.0.0

InetAddressValidator iv = InetAddressValidator.getInstance();
assertFalse("000.000.000.000", iv.isValid("000.000.000.000"));

References

http://pitest.org/
http://docs.oracle.com/javase/7/docs
http://findbugs.sourceforge.net/demo.html
https://maven.apache.org/plugins/maven-site-plugin/index.html
http://web.engr.oregonstate.edu/~baldwdav/BuggyURLValidator.zip
http://junit.sourceforge.net/javadoc/org/junit/Assert.html
http://greenbytes.de/tech/webdav/urldecomp.xml
https://mathiasbynens.be/demo/url-regex
https://en.wikipedia.org/wiki/Uniform_Resource_Locator

Project Report Instructions

Implement the improvements described above. Submit your code and write a test report as a text or PDF file called REPORT.(pdf/md) describing your experiences. Guidelines:

- Your code must include a README.md file describing how to run your code and tests
- Describe the techniques you used and include snippets of relevant testing code
- Bug reports for any issues you find in the code
- An evaluation of the new tool you used. Include a description of what the tool added to your test suites and a comparison to the techniques covered in the exercises.

Your test report should be at least 3 pages (750 words) but no more than 6 (1500 words). Check your report in to your repository and also submit the report via Canvas.