

[ [all classes](#) ] [ [<empty package name>](#) ]

## Coverage Summary for Class: UrlValidatorTest (<empty package name>)

Class	Method, %	Line, %
UrlValidatorTest	100% (22/ 22)	100% (269/ 269)
UrlValidatorTest\$Case	100% (2/ 2)	100% (5/ 5)
<b>total</b>	100% (24/ 24)	100% (274/ 274)

```
1  /*
2
3  * Licensed to the Apache Software Foundation (ASF) under one or more
4  * contributor license agreements. See the NOTICE file distributed with
5  * this work for additional information regarding copyright ownership.
6  * The ASF licenses this file to You under the Apache License, Version 2.0
7  * (the "License"); you may not use this file except in compliance with
8  * the License. You may obtain a copy of the License at
9  *
10  *      http://www.apache.org/licenses/LICENSE-2.0
11  *
12  * Unless required by applicable law or agreed to in writing, software
13  * distributed under the License is distributed on an "AS IS" BASIS,
14  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15  * See the License for the specific language governing permissions and
16  * limitations under the License.
17  */
18
19 import com.sun.tools.doclets.formats.html.SourceToHTMLConverter;
20 import junit.framework.TestCase;
21 import org.junit.Ignore;
22 import org.junit.Test;
23
24 import java.net.URL;
25 import java.util.ArrayList;
26 import java.util.Arrays;
27 import java.util.List;
28 import java.util.StringJoiner;
29 import java.util.stream.Collectors;
30
31
32 /**
33  * Performs Validation Test for url validations.
34  *
35  * @version $Revision: 1128446 $ $Date: 2011-05-27 13:29:27 -0700 (Fri, 27 May 2011)
36  */
37 public class UrlValidatorTest extends TestCase {
38
39     private boolean printStatus = false;
```

```

private boolean printIndex = false;//print index that indicates current scheme,h
41
42     public UrlValidatorTest(String testName) {
43         super(testName);
44     }
45
46
47     static final int SCHEME = 0;
48     static final int HOST = 1;
49     static final int PORT = 2;
50     static final int PATH = 3;
51     static final int QUERY_STRING = 4;
52     static final int FRAGMENT = 5;
53
54     static final String[] validStd = {"http://", "www.google.com", "", "/somefile.cs
55     static final String[] invalidStd = {"ffa/?:?://", "zzz?", ":32ze", "/wrong /Path"
56
57
58     public void testManualTest()
59     {
60
61         UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_LOCAL_
62         // test null url
63         System.out.println("Check null url");
64         System.out.println(!urlVal.isValid(null));
65         System.out.println("Check empty string");
66         System.out.println(!urlVal.isValid(""));
67
68         // Check basic schemes
69         System.out.println("Check basic schemes without port.");
70
71         System.out.println(urlVal.isValid("http://www.amazon.com") == true);
72         System.out.println(urlVal.isValid("https://www.amazon.com") == true);
73         System.out.println(urlVal.isValid("ftp://www.amazon.com") == true);
74         System.out.println(urlVal.isValid("http://www.amazon.com/homepage.html") ==
75         System.out.println(urlVal.isValid("https://www.amazon.com/homepage.html") ==
76         System.out.println(urlVal.isValid("ftp://www.amazon.com/homepage.html") == t
77         // Check ports
78         System.out.println("\nCheck basic schemes with ports.");
79
80         System.out.println(urlVal.isValid("http://www.amazon.com:") == false);
81         System.out.println(urlVal.isValid("https://www.amazon.com:") == false);
82         System.out.println(urlVal.isValid("ftp://www.amazon.com:") == false);
83         System.out.println(urlVal.isValid("http://www.amazon.com:1") == true);
84         System.out.println(urlVal.isValid("https://www.amazon.com:22") == true);
85         System.out.println(urlVal.isValid("ftp://www.amazon.com:333") == true);

```

```

        System.out.println(urlVal.isValid("http://www.amazon.com:4444") == true);
        System.out.println(urlVal.isValid("https://www.amazon.com:55555") == true);
        System.out.println(urlVal.isValid("https://www.amazon.com:65535") == true);
        System.out.println(urlVal.isValid("https://www.amazon.com:65536") == false);
        System.out.println(urlVal.isValid("ftp://www.amazon.com:666666") == false);
90
91        // Check queries
        System.out.println("\nCheck queries.");
        System.out.println(urlVal.isValid("https://www.amazon.com:12?action=view") == true);
        System.out.println(urlVal.isValid("https://www.amazon.com:65?action=view") == true);
        System.out.println(urlVal.isValid("ftp://www.amazon.com:45?action") == true);
        System.out.println(urlVal.isValid("http://www.google.com?action=view") == true);
97
98        // Check local host
        System.out.println("\nCheck local hosts.");
        System.out.println(urlVal.isValid("http://localhost/") == true);
        System.out.println(urlVal.isValid("http://machine/") == true);
        System.out.println(urlVal.isValid("http://127.0.0.1/") == true);
103
104        // Check fragments
        System.out.println("\nCheck fragments.");
        System.out.println(urlVal.isValid("http://www.somesite.org:33/somepage.csv#random")) == true);
107
108        // Check random stuff
        System.out.println("\nCheck random stuff.");
        System.out.println(urlVal.isValid("http://amazon.com/") == true);
        System.out.println(urlVal.isValid("http://www.amazon.com/") == false);
        System.out.println(urlVal.isValid("dfssfsdsdfsdf://www.amazon.com/") == false);
113    }
114
115    void testHost(UrlValidator v, String host, boolean expected){
        String url = validStd[SCHEME] + host + validStd[PORT] + validStd[PATH] + validStd[QUERY] + validStd[FRAGMENT];
        boolean result = v.isValid(url);
        _assertEquals("Host Test: " + host + " URL: " + url, expected, result);
        url = invalidStd[SCHEME] + host;
        result = v.isValid(url);
        _assertEquals("Host Test: " + host + " URL: " + url, false, result);
122    }
123
124    public void testUrlValidatorValidatesHost() {
        System.out.println("Host Partition Tests");
        UrlValidator validator = new UrlValidator(null, null, UrlValidator.ALLOW_ALL);

```

```

String[] validNumberTests = {"255.255.255.255", "0.0.0.0", "4.3.2.1", "01.00.00.00"};
String[] invalidNumberTests = {"-1.-10.-100.-256", "256.256.256.256", "1111.1111.1111.1111"};
String[] invalidAlphaNumTests = {"1.a.1.1", "a.a.a.a", "1.!.1.1", "1a.1.1.1"};
String[] invalidIPStructureTests = {"1..1.1.1", ".1.1.1.1", "1.1.1.1.", "1.1.1.1."};
131
String[] validHostnameTests = {"www.google.com", "w2w.google.com", "google.com"};
String[] invalidHostnameTests = {"www.google", ".www.google.com", "www,google.com"};
134
String testCase[][] = {validNumberTests, validHostnameTests, invalidNumberTests};
136     String host;
137     boolean expected;
138     for (int i = 0; i < testCase.length; i++){
        for (int j = 0; j < testCase[i].length; j++){
            expected = i < 2;
            host = testCase[i][j];
            testHost(validator, host, expected);
144         }
145     }
146 }
147
148 void testScheme(UrlValidator v, String scheme, boolean expected){
    String url = scheme + validStd[HOST] + validStd[PORT] + validStd[PATH] + validStd[QUERY];
    boolean result = v.isValid(url);
    _assertEquals("Scheme Test: " + scheme + " URL: " + url, expected, result);
    url = scheme + invalidStd[HOST];
    result = v.isValid(url);
    _assertEquals("Scheme Test: " + scheme + " URL: " + url, false, result);
155 }
156
157 public void testUrlValidatorValidatesScheme() {
    System.out.println("Scheme Partition Tests");
    UrlValidator validator = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
160     String scheme;
161     String testUrl;
162     boolean expected;
163     boolean result;
164
    String validBefore[] = {"http", "ftp", "ft2p", "https", "irc", "redis", "ut2p"};
    String invalidBefore[] = {"", "3nc", "..", "/http", "12", "!.http"};
    String before[][] = {validBefore, invalidBefore};
    String suffix[] = {"//", "/", ""};
    String semiColon [] = {":", ""};
170
171     // test combinations of valid and invalid sections of the scheme
    for (int beforeArray = 0; beforeArray < before.length; beforeArray++ ){

```

```

        for (int beforeIndex = 0; beforeIndex < before[beforeArray].length; beforeIndex++)
            for (int semiColonIndex = 0; semiColonIndex < semiColon.length; semiColonIndex++)
                for (int suffixIndex = 0; suffixIndex < suffix.length; suffixIndex++)
                    scheme = before[beforeArray][beforeIndex] + semiColon[semiColonIndex] + suffix[suffixIndex];
                    expected = (beforeArray == 0 && semiColonIndex == 0 && suffixIndex == 0) ? before[beforeArray][beforeIndex] : scheme;
                    testScheme(validator, scheme, expected);
179         }
180     }
181 }
182 }
183 }
184

// test out of order scheme sections that would be valid in order
String orderTests[] = {"//:http", "http//:", "://http", " :http//"};
for (int i = 0; i < orderTests.length; i++){
    testScheme(validator, orderTests[i], false);
188 }
189 }
190

class Case {
192     private String message;
193     private String testCase;
194     private boolean validity;
195

    public Case (String _message, String _testCase, boolean _validity) {
        message = _message;
        testCase = _testCase;
        validity = _validity;
200     }
201 }
202
203 private static int _assertEquals(String message, boolean expected, boolean actual) {
    if (expected != actual) {
        System.out.println("Failed: " + message + " Expected: " + expected);
        return 1;
207     }
    return 0;
209 }
210
211 private void _runThroughCases(Case[] cases, UrlValidator validator, int componentUnderTest) {
    String[] validParts = validStd.clone();
    String[] invalidParts = invalidStd.clone();
214     String url;
215     boolean result;
216

    int totalTests = 0;
    int failingTests = 0;
    for (Case _case: cases) {
        validParts[componentUnderTest] = _case.testCase;
        invalidParts[componentUnderTest] = _case.testCase;
222

```

```

        url = Arrays.stream(validParts).collect(Collectors.joining(""));
        result = validator.isValid(url);

        failingTests += _assertEquals(_case.message + " URL: " + url, _case.valid, result);
        totalTests++;
227
    }

    url = Arrays.stream(invalidParts).collect(Collectors.joining(""));
    result = validator.isValid(url);

    failingTests += _assertEquals(_case.message + " URL: " + url, false, result);
    totalTests++;
232     }
233

    System.out.println("Total tests: " + totalTests + " Failing Tests: " + failingTests);
235 }
236 /**
237  * partition the ports and test them with control urls
238  */
239 public void testUrlValidatorValidatesPorts()
240 {
    System.out.println("Port Partition Tests");
    Case[] cases = {
243         new Case("min port", ":1", true),
244         new Case("zero port", ":0", true),
245         new Case("negative port not allowed", ":-1", false),
246         new Case("http port is good", ":80", true),
247
        new Case("well known python app port should be good", ":8888", true),
248
        new Case("runescape port should be good", ":43594", true),
249         new Case("maximum port", ":65535", true),
250         new Case("larger than maximum", ":65536", false),
251
        new Case("port has foreign characters", ":45ZXZ", false),
252
        new Case("all letters not allowed as well", ":ABCDEFG", false),
253         new Case("port has spaces", ": 3423", false),
254
        new Case("port is empty, signifies default port", "", true)
255     };
256 }
257
258
    UrlValidator validator = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
260
    _runThroughCases(cases, validator, PORT);
262 }
263
264 public void testValidatorValidatesPathsAndOptions()
265 {
    System.out.println("Paths and Options Partition Test");
    Case[] cases = {
268         new Case("root path is okay", "/", true),
269
        new Case("top level path is okay", "/purchases", true),
270
        new Case("even longer path is fine", "/purchases/by-date/tomorrow", true),
271
    };
}

```

```

272         new Case("path with a space prepended is wrong", " /", false),
273         new Case("path with space at end is incorrect", "/path/path/ ", false),
274         new Case("path with incorrect percent encoding incorrect", "/path/pa
275         new Case("path with correct percent encoding is correct", "/path/pat
276     };

    UrlValidator validator = new UrlValidator(null, null, UrlValidator.ALLOW_ALL
    _runThroughCases(cases, validator, PATH);
279 }
280
281 /*     public void testValidatorValidatesFragments()
282     {
283         Case[] cases = {
284             new Case("fragment is empty", "#", true),
285             new Case("fragment has an anchor", "#profile", true),
286             new Case("fragment contains a space before pound sign", " #", false),
287             new Case("fragment is long and has special characters", "#a-really-l
288             new Case("fragment is not encoded correctly", "# space-should-be-enc
289             new Case("fragment contains encoded space", "%20-space-should-be-en
290             new Case("fragment contains invalid encoding", "%zz-not-valid-encod
291         };
292
293         UrlValidator validator = new UrlValidator(null, null, UrlValidator.ALLOW_ALL
294         _runThroughCases(cases, validator, FRAGMENT);
295     }
296
297 */
298
299     // Helper to check the result and print
300
301     private boolean checkAndPrint(UrlValidator val, String url, boolean expected)
302     {
303         if(expected == true)
304         {
305             if(val.isValid(url) == expected)
306             {
307                 System.out.println(url + " CORRECTLY passes."); return true;
308             }
309             else
310             {
311                 System.out.println(url + " INCORRECTLY fails."); return false;
312             }
313         }
314         else
315         {
316             if(val.isValid(url) == expected)
317             {

```

```

        System.out.println(url + " CORRECTLY fails."); return true;
318     }
319     else
320     {

        System.out.println(url + " INCORRECTLY passes."); return false;
322     }
323 }
324 }
325
326 // Partition testing
327 public void testFragmentPartition()
328 {
329     // Tell the user whats happening
330     System.out.println("\nTesting Fragments (Section 2)");
332
333     // Spin up the validator object
334     UrlValidator urlVal = new UrlValidator();
335
336     // Input partition cases for query

    String[] inputFragmentPartsPass = {"#row=123", "#row=asda", "#row=123,128",
    String[] inputFragmentPartsFail = {"row=123", "#rowasda", "#row=123#128", "r

339     // Construct pre and post for clean code in loops
340

    String preFragmentPass = validStd[SCHEME] + validStd[HOST] + validStd[PORT]
    String postFragmentPass = validStd[FRAGMENT];

    String preFragmentFail = invalidStd[SCHEME] + invalidStd[HOST] + invalidStd[
    String postFragmentFail = invalidStd[FRAGMENT];
345
346     // Test result counters
347     int passCount = 0; int failCount = 0;
348
349     // Construct URL's for true section under test and true control
    for(String pass : inputFragmentPartsPass)
351     {
        String url = preFragmentPass + pass + postFragmentPass;

        if(checkAndPrint(urlVal, url, true)) passCount ++; else failCount ++;
354     }
355
356     // Construct URL's for true section under test and false control
    for(String pass : inputFragmentPartsPass)
358     {
        String url = preFragmentFail + pass + postFragmentFail;

        if(checkAndPrint(urlVal, url, false)) passCount ++; else failCount ++;
361     }
362
363     // Construct URL's for false section under test and true control
    for(String fail : inputFragmentPartsFail)
365     {
        String url = preFragmentPass + fail + postFragmentPass;

```



```

        if(checkAndPrint(urlVal, url, false)) passCount ++; else failCount ++;
368     }
369
370     // Construct URL's for false section under test and false control
        for(String fail : inputFragmentPartsFail)
372     {
            String url = preFragmentFail + fail + postFragmentFail;
        if(checkAndPrint(urlVal, url, false)) passCount ++; else failCount ++;
375     }
376
        System.out.println("\nTest fragments final results (Section 3) pass count = "
        Integer.toString(passCount) + " and fail count = " + Integer.toString(failCount));
379     }
380
381     public void testQueryPartition()
382     {
383         // Tell the user whats happening
        System.out.println("\nTesting queries (Section 2)");
385
386         // Spin up the validator object
        UrlValidator urlVal = new UrlValidator();
388
389         // Input partition cases for query
        String[] inputQueryPartsPass = {"?pid=21", "?pid=dsdffs", "?pid=21&risc=98",
        String[] inputQueryPartsFail = {"pid=34", "?pid?dsdffs", "?pid=21risc=98", "
392
393         // Construct pre and post for clean code in loops
        String preQueryPass = validStd[SCHEME] + validStd[HOST] + validStd[PORT] + v
        String postQueryPass = validStd[FRAGMENT];
        String preQueryFail = invalidStd[SCHEME] + invalidStd[HOST] + invalidStd[PORT] + v
        String postQueryFail = invalidStd[FRAGMENT];
398
399         // Test result counters
        int passCount = 0; int failCount = 0;
401
402         // Construct URL's for true section under test and true control
        for(String pass : inputQueryPartsPass)
404     {
            String url = preQueryPass + pass + postQueryPass;
        if(checkAndPrint(urlVal, url, true)) passCount ++; else failCount ++;
407     }
408
409         // Construct URL's for true section under test and false control
        for(String pass : inputQueryPartsPass)
411     {
            String url = preQueryFail + pass + postQueryFail;
        if(checkAndPrint(urlVal, url, false)) passCount ++; else failCount ++;
414     }
415

```

```

416 // Construct URL's for false section under test and true control
    for(String fail : inputQueryPartsFail)
418 {
        String url = preQueryPass + fail + postQueryPass;

        if(checkAndPrint(urlVal, url, false)) passCount ++; else failCount ++;
421 }
422
423 // Construct URL's for false section under test and false control
    for(String fail : inputQueryPartsFail)
425 {
        String url = preQueryFail + fail + postQueryFail;

        if(checkAndPrint(urlVal, url, false)) passCount ++; else failCount ++;
428 }
429

    System.out.println("\nTest queries final results (Section 2) pass count = "
        + Integer.toString(passCount) + " and fail count = " + Integer.toString(failCount));
432 }
433
434 private Case[] generateSchemes() {
    String validBefore[] = {"http", "ftp", "ft2p", "https", "irc", "redis", "ut2", "ut3", "ut4", "ut5", "ut6", "ut7", "ut8", "ut9", "ut10", "ut11", "ut12", "ut13", "ut14", "ut15", "ut16", "ut17", "ut18", "ut19", "ut20", "ut21", "ut22", "ut23", "ut24", "ut25", "ut26", "ut27", "ut28", "ut29", "ut30", "ut31", "ut32", "ut33", "ut34", "ut35", "ut36", "ut37", "ut38", "ut39", "ut40", "ut41", "ut42", "ut43", "ut44", "ut45", "ut46", "ut47", "ut48", "ut49", "ut50", "ut51", "ut52", "ut53", "ut54", "ut55", "ut56", "ut57", "ut58", "ut59", "ut60", "ut61", "ut62", "ut63", "ut64", "ut65", "ut66", "ut67", "ut68", "ut69", "ut70", "ut71", "ut72", "ut73", "ut74", "ut75", "ut76", "ut77", "ut78", "ut79", "ut80", "ut81", "ut82", "ut83", "ut84", "ut85", "ut86", "ut87", "ut88", "ut89", "ut90", "ut91", "ut92", "ut93", "ut94", "ut95", "ut96", "ut97", "ut98", "ut99", "ut100", "ut101", "ut102", "ut103", "ut104", "ut105", "ut106", "ut107", "ut108", "ut109", "ut110", "ut111", "ut112", "ut113", "ut114", "ut115", "ut116", "ut117", "ut118", "ut119", "ut120", "ut121", "ut122", "ut123", "ut124", "ut125", "ut126", "ut127", "ut128", "ut129", "ut130", "ut131", "ut132", "ut133", "ut134", "ut135", "ut136", "ut137", "ut138", "ut139", "ut140", "ut141", "ut142", "ut143", "ut144", "ut145", "ut146", "ut147", "ut148", "ut149", "ut150", "ut151", "ut152", "ut153", "ut154", "ut155", "ut156", "ut157", "ut158", "ut159", "ut160", "ut161", "ut162", "ut163", "ut164", "ut165", "ut166", "ut167", "ut168", "ut169", "ut170", "ut171", "ut172", "ut173", "ut174", "ut175", "ut176", "ut177", "ut178", "ut179", "ut180", "ut181", "ut182", "ut183", "ut184", "ut185", "ut186", "ut187", "ut188", "ut189", "ut190", "ut191", "ut192", "ut193", "ut194", "ut195", "ut196", "ut197", "ut198", "ut199", "ut200", "ut201", "ut202", "ut203", "ut204", "ut205", "ut206", "ut207", "ut208", "ut209", "ut210", "ut211", "ut212", "ut213", "ut214", "ut215", "ut216", "ut217", "ut218", "ut219", "ut220", "ut221", "ut222", "ut223", "ut224", "ut225", "ut226", "ut227", "ut228", "ut229", "ut230", "ut231", "ut232", "ut233", "ut234", "ut235", "ut236", "ut237", "ut238", "ut239", "ut240", "ut241", "ut242", "ut243", "ut244", "ut245", "ut246", "ut247", "ut248", "ut249", "ut250", "ut251", "ut252", "ut253", "ut254", "ut255", "ut256", "ut257", "ut258", "ut259", "ut260", "ut261", "ut262", "ut263", "ut264", "ut265", "ut266", "ut267", "ut268", "ut269", "ut270", "ut271", "ut272", "ut273", "ut274", "ut275", "ut276", "ut277", "ut278", "ut279", "ut280", "ut281", "ut282", "ut283", "ut284", "ut285", "ut286", "ut287", "ut288", "ut289", "ut290", "ut291", "ut292", "ut293", "ut294", "ut295", "ut296", "ut297", "ut298", "ut299", "ut300", "ut301", "ut302", "ut303", "ut304", "ut305", "ut306", "ut307", "ut308", "ut309", "ut310", "ut311", "ut312", "ut313", "ut314", "ut315", "ut316", "ut317", "ut318", "ut319", "ut320", "ut321", "ut322", "ut323", "ut324", "ut325", "ut326", "ut327", "ut328", "ut329", "ut330", "ut331", "ut332", "ut333", "ut334", "ut335", "ut336", "ut337", "ut338", "ut339", "ut340", "ut341", "ut342", "ut343", "ut344", "ut345", "ut346", "ut347", "ut348", "ut349", "ut350", "ut351", "ut352", "ut353", "ut354", "ut355", "ut356", "ut357", "ut358", "ut359", "ut360", "ut361", "ut362", "ut363", "ut364", "ut365", "ut366", "ut367", "ut368", "ut369", "ut370", "ut371", "ut372", "ut373", "ut374", "ut375", "ut376", "ut377", "ut378", "ut379", "ut380", "ut381", "ut382", "ut383", "ut384", "ut385", "ut386", "ut387", "ut388", "ut389", "ut390", "ut391", "ut392", "ut393", "ut394", "ut395", "ut396", "ut397", "ut398", "ut399", "ut400", "ut401", "ut402", "ut403", "ut404", "ut405", "ut406", "ut407", "ut408", "ut409", "ut410", "ut411", "ut412", "ut413", "ut414", "ut415", "ut416", "ut417", "ut418", "ut419", "ut420", "ut421", "ut422", "ut423", "ut424", "ut425", "ut426", "ut427", "ut428", "ut429", "ut430", "ut431", "ut432", "ut433", "ut434", "ut435", "ut436", "ut437", "ut438", "ut439", "ut440", "ut441", "ut442", "ut443", "ut444", "ut445", "ut446", "ut447", "ut448", "ut449", "ut450", "ut451", "ut452", "ut453", "ut454", "ut455", "ut456", "ut457", "ut458", "ut459", "ut460", "ut461", "ut462", "ut463", "ut464", "ut465", "ut466", "ut467", "ut468", "ut469", "ut470", "ut471", "ut472", "ut473", "ut474", "ut475", "ut476", "ut477", "ut478", "ut479", "ut480", "ut481", "ut482", "ut483", "ut484", "ut485", "ut486", "ut487", "ut488", "ut489", "ut490", "ut491", "ut492", "ut493", "ut494", "ut495", "ut496", "ut497", "ut498", "ut499", "ut500", "ut501", "ut502", "ut503", "ut504", "ut505", "ut506", "ut507", "ut508", "ut509", "ut510", "ut511", "ut512", "ut513", "ut514", "ut515", "ut516", "ut517", "ut518", "ut519", "ut520", "ut521", "ut522", "ut523", "ut524", "ut525", "ut526", "ut527", "ut528", "ut529", "ut530", "ut531", "ut532", "ut533", "ut534", "ut535", "ut536", "ut537", "ut538", "ut539", "ut540", "ut541", "ut542", "ut543", "ut544", "ut545", "ut546", "ut547", "ut548", "ut549", "ut550", "ut551", "ut552", "ut553", "ut554", "ut555", "ut556", "ut557", "ut558", "ut559", "ut560", "ut561", "ut562", "ut563", "ut564", "ut565", "ut566", "ut567", "ut568", "ut569", "ut570", "ut571", "ut572", "ut573", "ut574", "ut575", "ut576", "ut577", "ut578", "ut579", "ut580", "ut581", "ut582", "ut583", "ut584", "ut585", "ut586", "ut587", "ut588", "ut589", "ut590", "ut591", "ut592", "ut593", "ut594", "ut595", "ut596", "ut597", "ut598", "ut599", "ut600", "ut601", "ut602", "ut603", "ut604", "ut605", "ut606", "ut607", "ut608", "ut609", "ut610", "ut611", "ut612", "ut613", "ut614", "ut615", "ut616", "ut617", "ut618", "ut619", "ut620", "ut621", "ut622", "ut623", "ut624", "ut625", "ut626", "ut627", "ut628", "ut629", "ut630", "ut631", "ut632", "ut633", "ut634", "ut635", "ut636", "ut637", "ut638", "ut639", "ut640", "ut641", "ut642", "ut643", "ut644", "ut645", "ut646", "ut647", "ut648", "ut649", "ut650", "ut651", "ut652", "ut653", "ut654", "ut655", "ut656", "ut657", "ut658", "ut659", "ut660",
```

```

462     {
        String[] inputQueryPartsPass = {"?pid=21", "?pid=dsdffs", "?pid=21&risc=98",
        String[] inputQueryPartsFail = {"pid=34", "?pid?dsdffs", "?pid=21risc=98", "
465         ArrayList<Case> cases = new ArrayList<>();
467         for (int i = 0; i < inputQueryPartsPass.length; i++) {
            cases.add(new Case("Query " + inputQueryPartsPass[i] + " should pass", i
470         }
471         for (int i = 0; i < inputQueryPartsPass.length; i++) {
            cases.add(new Case("Query " + inputQueryPartsFail[i] + "should fail", in
474         }
475         return cases.toArray(new Case[cases.size()]);
477     }
478     private Case[] generateHostCases()
479     {
480         String[] validNumberTests = {"255.255.255.255", "0.0.0.0", "4.3.2.1", "01.00
        String[] invalidNumberTests = {"-1.-10.-100.-256", "256.256.256.256", "1111
        String[] invalidAlphaNumTests = {"1.a.1.1", "a.a.a.a", "1!.1.1", "1a.1.1.1"
        String[] invalidIPStructureTests = {"1..1.1.1", ".1.1.1.1", "1.1.1.1.", "1.1
485         String[] validHostnameTests = {"www.google.com", "w2w.google.com", "google.c
        String[] invalidHostnameTests = { "www.google", ".www.google.com", "www,goog
        String hostTestCase[][] = {validNumberTests, validHostnameTests, invalidNumb
489         ArrayList<Case> cases = new ArrayList<>();
            String msg = "";
492         String host;
493         boolean expected;
            for (int i = 0; i < hostTestCase.length; i++) {
                expected = i < 2;
                if (i == 1 || i == 5) {
                    msg = "Hostname test ";
498                } else {
                    msg = "Dotted decimal test ";
500                }
                for (int j = 0; j < hostTestCase[i].length; j++) {
                    host = hostTestCase[i][j];
                    cases.add(new Case(msg + host + " should " + (expected? "pass": "fail
504                }
505            }
506            return cases.toArray(new Case[cases.size()]);
508        }
509        private Case[] generateFragmentCases()
510

```

```

511     {
String[] inputFragmentPartsPass = {"#row=123", "#row=asda", "#row=123,128",
String[] inputFragmentPartsFail = {"row=123", "#rowasda", "#row=123#128", "r
514     ArrayList<Case> cases = new ArrayList<>();
516     for (String s: inputFragmentPartsPass) {
cases.add(new Case("Fragment: " + s + " should pass", s, true));
519     }
520     for (String s: inputFragmentPartsFail) {
cases.add(new Case("Fragment: " + s + " should not pass", s, false));
523     }
return cases.toArray(new Case[cases.size()]);
525     }
526     private Case[] generatePortCases()
527     {
528     Case[] portCases = {
531         new Case("min port", ":1", true),
532         new Case("zero port", ":0", false),
533         new Case("negative port not allowed", ":-1", false),
534         new Case("http port is good", ":80", true),
535         new Case("well known python app port should be good", ":8888", true),
536         new Case("runescape port should be good", ":43594", true),
537         new Case("maximum port", ":65535", true),
538         new Case("larger than maximum", ":65536", false),
539         new Case("port has foreign characters", ":45ZXZ", false),
540         new Case("all letters not allowed as well", ":ABCDEFGF", false),
541         new Case("port has spaces", ": 3423", false),
542         new Case("port is empty, signifies default port", "", true)
543     };
544     };
545     return portCases;
547     }
548     private Case[] generatePathCases() {
549     Case[] pathCases = {
551         new Case("root path is okay", "/", true),
552         new Case("top level path is okay", "/purchases", true),
553         new Case("even longer path is fine", "/purchases/by-date/tomorrow",
554         new Case("path with a space prepended is wrong", " /", false),
555         new Case("path with space at end is incorrect", "/path/path/ ", false),
556         new Case("path with incorrect percent encoding incorrect", "/path/pa
557

```

```

        new Case("path with correct percent encoding is correct", "/path/pat
558     });
559
        return pathCases;
561     }
562
563
    public static int testCombo(UrlValidator validator, List<String> parts, boolean
        boolean expected = true;
565     boolean result;
        for (boolean b: validityOfComponents) {
            expected &= b;
568     }
        String url = parts.stream().collect(Collectors.joining(""));
        result = validator.isValid(url);
        if (result != expected) {
            System.out.println("FAILURE: " + url + " EXPECTED: " + expected);
            return 1;
574     }
        return 0;
576     }
577
578     public void testCombos()
579     {
        boolean expected = true;
        List<String> urlParts = Arrays.asList(validStd); // initialize with valid co
        boolean[] validityOfComponents = new boolean[validStd.length];
        UrlValidator validator = new UrlValidator(null, null, UrlValidator.ALLOW_ALL
        String scheme = "";
585
        Case[] schemeCases = generateSchemes();
        Case[] queryCases = generateQueryStringCases();
        Case[] fragmentCases = generateFragmentCases();
        Case[] hostCases = generateHostCases();
        Case[] portCases = generatePortCases();
        Case[] pathCases = generatePathCases();
592
        int totalTestCases = 0;
        int failedTestCases = 0;
        for (Case schemeCase: schemeCases) {
            scheme = schemeCase.testCase;
            validityOfComponents[SCHEME] = schemeCase.validity;
            urlParts.set(SCHEME, scheme);
            for (Case hostCase: hostCases ) {
                validityOfComponents[HOST] = hostCase.validity;
                urlParts.set(HOST, hostCase.testCase);
                for (Case portCase : portCases) {
                    validityOfComponents[PORT] = portCase.validity;
                    urlParts.set(PORT, portCase.testCase);
                    for (Case pathCase : pathCases) {
                        validityOfComponents[PATH] = pathCase.validity;
                        urlParts.set(PATH, pathCase.testCase);
                        for (Case queryCase : queryCases) {
                            validityOfComponents[QUERY_STRING] = queryCase.validity;

```

```

        urlParts.set(QUERY_STRING, queryCase.testCase);
        for (Case fragmentCase : fragmentCases) {
            validityOfComponents[FRAGMENT] = fragmentCase.validity;
            urlParts.set(FRAGMENT, fragmentCase.testCase);
            failedTestCases += testCombo(validator, urlParts, va
            totalTestCases++;
616         }
617     }
618 }
619 }
620 }
621 }
        System.out.println("SUMMARY");
        System.out.println("TOTAL TEST CASES: " + totalTestCases);
        System.out.println("FAILED TEST CASES: " + failedTestCases);
625     }
626 }
627 }

```

generated on 2017-08-11 14:35