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)
total	100% (24/ 24)	100% (274/ 274)

```
  1 /*
  2
  * Licensed to the Apache Software Foundation (ASF) under one or more
  3
  * contributor license agreements.  See the NOTICE file distributed with
  4
  * this work for additional information regarding copyright ownership.
  5
  * The ASF licenses this file to You under the Apache License, Version 2.0
  6
  * (the "License"); you may not use this file except in compliance with
  7  * the License.  You may obtain a copy of the License at
  8  *
  9  *      http://www.apache.org/licenses/LICENSE-2.0
 10  *
 11
  * Unless required by applicable law or agreed to in writing, software
 12  * distributed under the License is distributed on an "AS IS" BASIS,
 13
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 14
  * See the License for the specific language governing permissions and
 15  * limitations under the License.
 16  */
 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
          private boolean printStatus = false;
```

```
    private boolean printIndex = false;//print index that indicates current scheme,h
41
42      public UrlValidatorTest(String testName) {
            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

    static final String[] validStd = {"http://", "www.google.com", "", "/somefile.cs

    static final String[] invalidStd = {"ffa/??://", "zzz?", ":32ze", "/wrong /Path"
56
57
58      public void testManualTest()
59      {

        UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_LOCAL_
61
62          // test null url
            System.out.println("Check null url");
            System.out.println(!urlVal.isValid(null));
            System.out.println("Check empty string");
            System.out.println(!urlVal.isValid(""));
67
68          // Check basic schemes
            System.out.println("Check basic schemes without port.");

        System.out.println(urlVal.isValid("http://www.amazon.com") == true);

        System.out.println(urlVal.isValid("https://www.amazon.com") == true);

        System.out.println(urlVal.isValid("ftp://www.amazon.com") == true);

        System.out.println(urlVal.isValid("http://www.amazon.com/homepage.html") ==

        System.out.println(urlVal.isValid("https://www.amazon.com/homepage.html") ==

        System.out.println(urlVal.isValid("ftp://www.amazon.com/homepage.html") == t
76
77          // Check ports
            System.out.println("\nCheck basic schemes with ports.");

        System.out.println(urlVal.isValid("http://www.amazon.com:") == false);

        System.out.println(urlVal.isValid("https://www.amazon.com:") == false);

        System.out.println(urlVal.isValid("ftp://www.amazon.com:") == false);

        System.out.println(urlVal.isValid("http://www.amazon.com:1") == true);

        System.out.println(urlVal.isValid("https://www.amazon.com:22") == true);

        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") =

        System.out.println(urlVal.isValid("https://www.amazon.com:65?action=view") =

        System.out.println(urlVal.isValid("ftp://www.amazon.com:45?action") == true)

        System.out.println(urlVal.isValid("http://www.google.com?action=view") == tr
 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#r
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("htttp://www.amazon.com/") == false);

        System.out.println(urlVal.isValid("dfssfsdsdfsdf://www.amazon.com/") == fals
113       }
114
115       void testHost(UrlValidator v, String host, boolean expected){

        String url = validStd[SCHEME] + host + validStd[PORT] + validStd[PATH] + val
            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
        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
131
        String[] validHostnameTests = {"www.google.com", "w2w.google.com", "google.c
        String[] invalidHostnameTests = { "www.google", ".www.google.com", "www,goog
134
        String testCase[][] = {validNumberTests, validHostnameTests, invalidNumberTe
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] + val
              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
160           String scheme;
161           String testUrl;
162           boolean expected;
163           boolean result;
164
        String validBefore[] = {"http", "ftp", "ft2p", "https", "irc", "redis", "ut2
        String invalidBefore[] = {"", "3nc", "..", "/htpp", "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; befo
                for (int semiColonIndex = 0; semiColonIndex < semiColon.length; semi
                    for (int suffixIndex = 0; suffixIndex < suffix.length; suffixInd
                        scheme = before[beforeArray][beforeIndex] + semiColon[semiCo
                        expected = (beforeArray == 0 && semiColonIndex == 0 && suffi
                            testScheme(validator, scheme, expected);
                    }
                }
            }
        }


    // 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);
        }
    }

    class Case {
        private String message;
        private String testCase;
        private boolean validity;

    public Case (String _message, String _testCase, boolean _validity) {
            message = _message;
            testCase = _testCase;
            validity = _validity;
        }
    }


private static int _assertEquals(String message, boolean expected, boolean actua
        if (expected != actual) {
            System.out.println("Failed: " + message + " Expected: " + expected);
            return 1;
        }
        return 0;
    }


private void _runThroughCases(Case[] cases, UrlValidator validator, int componen
        String[] validParts = validStd.clone();
        String[] invalidParts = invalidStd.clone();
        String url;
        boolean result;

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

```

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

            failingTests += _assertEquals(_case.message + " URL: " + url, _case.vali
                  totalTests++;
227
            url = Arrays.stream(invalidParts).collect(Collectors.joining(""));
                  result = validator.isValid(url);

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

        System.out.println("Total tests: " + totalTests + " Failing Tests: " + faili
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
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",
271
```

```
                new Case("path with a space prepended is wrong", " /", false),
272
                new Case("path with space at end is incorrect", "/path/path/ ", fals
273
                new Case("path with incorrect percent encoding incorrect", "/path/pa
274
                new Case("path with correct percent encoding is correct", "/path/pat
275             };
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
    private boolean checkAndPrint(UrlValidator val, String url, boolean expected)
301         {
                if(expected == true)
303             {
                    if(val.isValid(url) == expected)
305                 {

                    System.out.println(url + " CORRECTLY passes."); return true;
307                 }
308                 else
309                 {

                    System.out.println(url + " INCORRECTLY fails.");       return false;
311                 }
312             }
313             else
314             {
                    if(val.isValid(url) == expected)
316                 {
```

```
                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
327       // Partition testing
328       public void testFragmentPartition()
329       {
330           // Tell the user whats happening
              System.out.println("\nTesting Fragments (Section 2)");
332
333           // Spin up the validator object
              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
340           // Contruct pre and post for clean code in loops

      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
              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.toStrin
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           // Contruct 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[POR
              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.toStrin
432       }
433
434       private Case[] generateSchemes() {

        String validBefore[] = {"http", "ftp", "ft2p", "https", "irc", "redis", "ut2

        String invalidBefore[] = {"", "3nc", "..", "/htpp", "12", "!http"};
              String beforeSchemes[][] = {validBefore, invalidBefore};
              String suffix[] = {"//", "/", ""};
              String semiColon [] = {":", ""};
              String scheme = "";
              String msg = "";
442           boolean expected;
              ArrayList<Case> cases = new ArrayList<>();
444

        for (int beforeSchemesArray = 0; beforeSchemesArray < beforeSchemes.length;

             for (int beforeSchemesIndex = 0; beforeSchemesIndex < beforeSchemes[befo

                 for (int semiColonIndex = 0; semiColonIndex < semiColon.length; semi

                     for (int suffixIndex = 0; suffixIndex < suffix.length; suffixInd

                         scheme = beforeSchemes[beforeSchemesArray][beforeSchemesInde

                         expected = (beforeSchemesArray == 0 && semiColonIndex == 0 &

                         msg = scheme + " should " + (expected ? " pass": " fail");
                               cases.add(new Case(msg, scheme, expected));
453                      }
454                  }
455              }
456           }
457
              return cases.toArray(new Case[cases.size()]);
459       }
460
461       private Case[] generateQueryStringCases()
```

```
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
479     private Case[] generateHostCases()
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": "fai
504             }
505         }
506
            return cases.toArray(new Case[cases.size()]);
508     }
509
510     private Case[] generateFragmentCases()
```

```
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
527     private Case[] generatePortCases()
528     {
529
            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", ":ABCDEFG", 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
549     private Case[] generatePathCases() {
            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/ ", fals
556
              new Case("path with incorrect percent encoding incorrect", "/path/pa
557
```

```
                new Case("path with correct percent encoding is correct", "/path/pat
            };

            return pathCases;
        }


    public static int testCombo(UrlValidator validator, List<String> parts, boolean[
            boolean expected = true;
            boolean result;
            for (boolean b: validityOfComponents) {
                expected &= b;
            }
            String url = parts.stream().collect(Collectors.joining(""));
            result = validator.isValid(url);
            if (result != expected) {

           System.out.println("FAILURE: " + url + " EXPECTED: " + expected);
                return 1;
            }
            return 0;
        }

        public void testCombos()
        {
            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 = "";

            Case[] schemeCases = generateSchemes();
            Case[] queryCases =  generateQueryStringCases();
            Case[] fragmentCases =  generateFragmentCases();
            Case[] hostCases =  generateHostCases();
            Case[] portCases = generatePortCases();
            Case[] pathCases = generatePathCases();

            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.validi

                                    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