```
 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 * @author Alexander V. Astapchuk
20 * @version $Revision$
21 */
22
23 package org.apache.harmony.security.tests.java.security;
24
25 import dalvik.annotation.TestTargetClass;
26 import dalvik.annotation.TestTargets;
27 import dalvik.annotation.TestLevel;
28 import dalvik.annotation.TestTargetNew;
29
30 import java.io.File;
31 import java.net.URL;
32 import java.net.InetAddress;
33 import java.net.MalformedURLException;
34 import java.net.UnknownHostException;
35 import java.security.CodeSigner;
36 import java.security.CodeSource;
37 import java.security.cert.CertPath;
38 import java.security.cert.Certificate;
39
40 import org.apache.harmony.security.tests.support.TestCertUtils;
41
42 import junit.framework.TestCase;
43 @TestTargetClass(CodeSource.class)
44 /**
45  * Unit test for CodeSource.
46  *
47  */
48
49 public class CodeSourceTest extends TestCase {
50     /**
51      *
52      * Entry point for standalone runs.
53      *
54      * @param args command line arguments
55      */
56     public static void main(String[] args) throws Exception {
57         junit.textui.TestRunner.run(CodeSourceTest.class);
```

```
    }

    private java.security.cert.Certificate[] chain = null;

    /* Below are various URLs used during the testing */
    private static URL urlSite;

    private static URL urlDir; // must NOT end with '/'

    private static URL urlDirOtherSite; // same as urlDir, but another
    site

    private static URL urlDir_port80, urlDir_port81;

    /* must be exactly the same as urlDir, but with slash added */
    private static URL urlDirWithSlash;

    //private static URL urlDirFtp;
    private static URL urlDir_FileProtocol;

    private static URL urlDirIP;

    private static URL urlFile, urlFileWithAdditionalDirs,
    urlFileDirOtherDir;

    private static URL urlFileDirMinus;

    private static URL urlFileDirStar;

    private static URL urlRef1, urlRef2;

    private boolean init = false;

    private void init() {
        if (!init) {
            try {
                String siteName = "www.intel.com";
                InetAddress addr = InetAddress.getByName(siteName);
                String siteIP = addr.getHostAddress();

                urlSite = new URL("http://"+siteName+"");
                urlDir = new URL("http://"+siteName+"/drl_test");
                urlDirOtherSite = new
                URL("http://www.any-other-site-which-is-not-siteName.com
                /drl_test");

                urlDir_port80 = new
                URL("http://"+siteName+":80/drl_test");
                urlDir_port81 = new
                URL("http://"+siteName+":81/drl_test");
                urlDirWithSlash = new URL(urlDir + "/");

                //urlDirFtp = new URL("ftp://www.intel.com/drl_test");
                urlDir_FileProtocol = new
                URL("file://"+siteName+"/drl_test");

                urlDirIP = new URL("http://"+siteIP+"/drl_test");

                urlFile = new
                URL("http://"+siteName+"/drl_test/empty.jar");
```

```
110                 urlFileWithAdditionalDirs = new URL(
111 
                            "http://"+siteName+"/drl_test/what/ever/here/emp
                            ty.jar");
112 
113                 urlFileDirMinus = new
                    URL("http://"+siteName+"/drl_test/-");
114                 urlFileDirStar = new
                    URL("http://"+siteName+"/drl_test/*");
115                 urlFileDirOtherDir = new
                    URL("http://"+siteName+"/_test_drl_/*");
116 
117                 urlRef1 = new
                    URL("http://"+siteName+"/drl_test/index.html#ref1");
118                 urlRef2 = new
                    URL("http://"+siteName+"/drl_test/index.html#ref2");
119             } catch (MalformedURLException ex) {
120                 throw new Error(ex);
121             } catch (UnknownHostException ex) {
122                 throw new Error(ex);
123             } finally {
124                 init = true;
125             }
126         }
127     }
128 
129     protected void setUp() throws Exception {
130         super.setUp();
131         init();
132         chain = TestCertUtils.getCertChain();
133     }
134 
135     /**
136      * Tests hashCode().<br>
137      * javadoc says nothing, so test DRL-specific implementation.
138      */
139     @TestTargetNew(
140         level = TestLevel.COMPLETE,
141         notes = "",
142         method = "hashCode",
143         args = {}
144     )
145     public void testHashCode() {
146         // when nothing is specified, then hashCode obviously must be 0.
147                 assertTrue(new CodeSource(null, (Certificate[])
            null).hashCode() == 0);
148         // only URL.hashCode is taken into account...
149         assertTrue(new CodeSource(urlSite, (Certificate[])
            null).hashCode() == urlSite
150                 .hashCode());
151         // ... and certs[] does not affect it
152         assertTrue(new CodeSource(urlSite, chain).hashCode() == urlSite
153                 .hashCode());
154     }
155 
156     /**
157      * Tests CodeSource(URL, Certificate[]).
158      */
159     @TestTargetNew(
160         level = TestLevel.COMPLETE,
```

```
161         notes = "",
162         method = "CodeSource",
163         args = {java.net.URL.class,
            java.security.cert.Certificate[].class}
164     )
165     public void testCodeSourceURLCertificateArray() {
166         new CodeSource(null, (Certificate[]) null);
167         new CodeSource(urlSite, (Certificate[]) null);
168         new CodeSource(null, chain);
169         new CodeSource(urlSite, chain);
170     }
171
172     /**
173      * Tests CodeSource(URL, CodeSigner[]).
174      */
175     @TestTargetNew(
176         level = TestLevel.PARTIAL,
177         notes = "Verifies method with null parameters only",
178         method = "CodeSource",
179         args = {java.net.URL.class, java.security.CodeSigner[].class}
180     )
181     public void testCodeSourceURLCodeSignerArray() {
182         if (!has_15_features()) {
183             return;
184         }
185         new CodeSource(null, (CodeSigner[]) null);
186
187     }
188
189     /**
190      * equals(Object) must return <code>false</code> for null
191      */
192     @TestTargetNew(
193         level = TestLevel.PARTIAL_COMPLETE,
194         notes = "Null parameter checked",
195         method = "equals",
196         args = {java.lang.Object.class}
197     )
198     public void testEqualsObject_00() {
199         CodeSource thiz = new CodeSource(urlSite, (Certificate[]) null);
200         assertFalse(thiz.equals(null));
201
202     }
203
204     /**
205      * equals(Object) must return <code>true</code> for the same object
206      */
207     @TestTargetNew(
208         level = TestLevel.PARTIAL_COMPLETE,
209         notes = "Same objects checked",
210         method = "equals",
211         args = {java.lang.Object.class}
212     )
213     public void testEqualsObject_01() {
214         CodeSource thiz = new CodeSource(urlSite, (Certificate[]) null);
215         assertTrue(thiz.equals(thiz));
216     }
217
218     /**
219      * Test for equals(Object)<br>
```

```
220      * The signer certificate chain must contain the same set of
         certificates, but
221      * the order of the certificates is not taken into account.
222      */
223     @TestTargetNew(
224         level = TestLevel.PARTIAL_COMPLETE,
225         notes = "",
226         method = "equals",
227         args = {java.lang.Object.class}
228     )
229     public void testEqualsObject_02() {
230         Certificate cert0 = new TestCertUtils.TestCertificate();
231         Certificate cert1 = new TestCertUtils.TestCertificate();
232         Certificate[] certs0 = new Certificate[] { cert0, cert1 };
233         Certificate[] certs1 = new Certificate[] { cert1, cert0 };
234         CodeSource thiz = new CodeSource(urlSite, certs0);
235         CodeSource that = new CodeSource(urlSite, certs1);
236         assertTrue(thiz.equals(that));
237     }
238
239     /**
240      * Test for equals(Object)<br>
241      * Checks that both 'null' and not-null URLs are taken into account
         - properly.
242      */
243     @TestTargetNew(
244         level = TestLevel.PARTIAL_COMPLETE,
245         notes = "",
246         method = "equals",
247         args = {java.lang.Object.class}
248     )
249     public void testEqualsObject_04() {
250         CodeSource thiz = new CodeSource(urlSite, (Certificate[]) null);
251         CodeSource that = new CodeSource(null, (Certificate[]) null);
252         assertFalse(thiz.equals(that));
253         assertFalse(that.equals(thiz));
254
255         that = new CodeSource(urlFile, (Certificate[]) null);
256         assertFalse(thiz.equals(that));
257         assertFalse(that.equals(thiz));
258     }
259
260     /**
261      * Tests CodeSource.getCertificates().
262      */
263     @TestTargetNew(
264         level = TestLevel.PARTIAL_COMPLETE,
265         notes = "",
266         method = "getCertificates",
267         args = {}
268     )
269     public void testGetCertificates_00() {
270         assertNull(new CodeSource(null, (Certificate[])
            null).getCertificates());
271         java.security.cert.Certificate[] got = new CodeSource(null,
            chain)
272                 .getCertificates();
273         // The returned array must be clone()-d ...
274         assertNotSame(got, chain);
275         // ... but must represent the same set of certificates
```

```
276         assertTrue(checkEqual(got, chain));
277     }
278
279     /**
280      * Tests whether the getCertificates() returns certificates obtained
         from
281      * the signers.
282      */
283     @TestTargetNew(
284         level = TestLevel.PARTIAL_COMPLETE,
285         notes = "",
286         method = "getCertificates",
287         args = {}
288     )
289     public void testGetCertificates_01() {
290         if (!has_15_features()) {
291             return;
292         }
293         CertPath cpath = TestCertUtils.getCertPath();
294         Certificate[] certs = (Certificate[])
            cpath.getCertificates().toArray();
295         CodeSigner[] signers = { new CodeSigner(cpath, null) };
296         CodeSource cs = new CodeSource(null, signers);
297         Certificate[] got = cs.getCertificates();
298         // The set of certificates must be exactly the same,
299         // but the order is not specified
300         assertTrue(presented(certs, got));
301         assertTrue(presented(got, certs));
302     }
303
304     /**
305      * Checks whether two arrays of certificates represent the same same
         set of
306      * certificates - in the same order.
307      * @param one first array
308      * @param two second array
309      * @return <code>true</code> if both arrays represent the same set
         of
310      * certificates,
311      * <code>false</code> otherwise.
312      */
313     private static boolean checkEqual(java.security.cert.Certificate[]
        one,
314             java.security.cert.Certificate[] two) {
315
316         if (one == null) {
317             return two == null;
318         }
319
320         if (two == null) {
321             return false;
322         }
323
324         if (one.length != two.length) {
325             return false;
326         }
327
328         for (int i = 0; i < one.length; i++) {
329             if (one[i] == null) {
330                 if (two[i] != null) {
```

```
331                     return false;
332                 }
333             } else {
334                 if (!one[i].equals(two[i])) {
335                     return false;
336                 }
337             }
338         }
339         return true;
340     }
341 
342     /**
343      * Performs a test whether the <code>what</code> certificates are
         all
344      * presented in <code>where</code> certificates.
345      *
346      * @param what - first array of Certificates
347      * @param where  - second array of Certificates
348      * @return <code>true</code> if each and every certificate from
         'what'
349      * (including null) is presented in 'where' <code>false</code>
         otherwise
350      */
351     private static boolean presented(Certificate[] what, Certificate[]
        where) {
352         boolean whereHasNull = false;
353         for (int i = 0; i < what.length; i++) {
354             if (what[i] == null) {
355                 if (whereHasNull) {
356                     continue;
357                 }
358                 for (int j = 0; j < where.length; j++) {
359                     if (where[j] == null) {
360                         whereHasNull = true;
361                         break;
362                     }
363                 }
364                 if (!whereHasNull) {
365                     return false;
366                 }
367             } else {
368                 boolean found = false;
369                 for (int j = 0; j < where.length; j++) {
370                     if (what[i].equals(where[j])) {
371                         found = true;
372                         break;
373                     }
374                 }
375                 if (!found) {
376                     return false;
377                 }
378             }
379 
380         }
381         return true;
382     }
383 
384     /**
385      * Tests CodeSource.getCodeSigners().
386      */
```

```
387     @TestTargetNew(
388         level = TestLevel.PARTIAL_COMPLETE,
389         notes = "",
390         method = "getCodeSigners",
391         args = {}
392     )
393     public void testGetCodeSigners_00() {
394         if (!has_15_features()) {
395             return;
396         }
397         CodeSigner[] signers = { new
            CodeSigner(TestCertUtils.getCertPath(),
398                 null) };
399         CodeSource cs = new CodeSource(null, signers);
400         CodeSigner[] got = cs.getCodeSigners();
401         assertNotNull(got);
402         assertTrue(signers.length == got.length);
403         // not sure whether they must be in the same order
404         for (int i = 0; i < signers.length; i++) {
405             CodeSigner s = signers[i];
406             boolean found = false;
407             for (int j = 0; j < got.length; j++) {
408                 if (got[j] == s) {
409                     found = true;
410                     break;
411                 }
412             }
413             assertTrue(found);
414         }
415     }
416
417     /**
418      * Tests CodeSource.getCodeSigners() for null.
419      */
420     @TestTargetNew(
421         level = TestLevel.PARTIAL_COMPLETE,
422         notes = "",
423         method = "getCodeSigners",
424         args = {}
425     )
426     public void testGetCoderSignersNull() throws Exception{
427         assertNull(new CodeSource(new URL("http://url"),
            (Certificate[])null).getCodeSigners()); //$NON-NLS-1$
428     }
429
430     /**
431      * Tests CodeSource.getLocation()
432      */
433     @TestTargetNew(
434         level = TestLevel.COMPLETE,
435         notes = "",
436         method = "getLocation",
437         args = {}
438     )
439     public void testGetLocation() {
440         assertTrue(new CodeSource(urlSite, (Certificate[])
            null).getLocation() == urlSite);
441         assertTrue(new CodeSource(urlSite, chain).getLocation() ==
            urlSite);
442         assertNull(new CodeSource(null, (Certificate[])
```

```
            null).getLocation());
443         assertNull(new CodeSource(null, chain).getLocation());
444     }
445
446     /**
447      * Tests CodeSource.toString()
448      */
449     @TestTargetNew(
450         level = TestLevel.COMPLETE,
451         notes = "",
452         method = "toString",
453         args = {}
454     )
455     public void testToString() {
456         // Javadoc keeps silence about String's format,
457         // just make sure it can be invoked.
458         new CodeSource(urlSite, chain).toString();
459         new CodeSource(null, chain).toString();
460         new CodeSource(null, (Certificate[]) null).toString();
461     }
462
463     /**
464      * Tests whether we are running with the 1.5 features.<br>
465      * The test is preformed by looking for (via reflection) the
        CodeSource's
466      * constructor  {@link CodeSource#CodeSource(URL, CodeSigner[])}.
467      * @return <code>true</code> if 1.5 feature is presented,
        <code>false</code>
468      * otherwise.
469      */
470     private static boolean has_15_features() {
471         Class klass = CodeSource.class;
472         Class[] ctorArgs = { URL.class, new CodeSigner[] {}.getClass()
            };
473         try {
474             klass.getConstructor(ctorArgs);
475         } catch (NoSuchMethodException ex) {
476             // NoSuchMethod == Not RI.v1.5 and not DRL
477             return false;
478         }
479         return true;
480     }
481
482     /**
483      * must not imply null CodeSource
484      */
485     @TestTargetNew(
486         level = TestLevel.PARTIAL_COMPLETE,
487         notes = "",
488         method = "implies",
489         args = {java.security.CodeSource.class}
490     )
491     public void testImplies_00() {
492         CodeSource cs0 = new CodeSource(null, (Certificate[]) null);
493         assertFalse(cs0.implies(null));
494     }
495
496     /**
497      * CodeSource with location=null && Certificate[] == null implies
        any other
```

```
498      * CodeSource
499      */
500     @TestTargetNew(
501         level = TestLevel.PARTIAL_COMPLETE,
502         notes = "",
503         method = "implies",
504         args = {java.security.CodeSource.class}
505     )
506     public void testImplies_01() throws Exception {
507         CodeSource thizCS = new CodeSource(urlSite, (Certificate[])
            null);
508         CodeSource thatCS = new CodeSource(null, (Certificate[]) null);
509         assertTrue(thatCS.implies(thizCS));
510         assertTrue(thatCS.implies(thatCS));
511
512         assertFalse(thizCS.implies(thatCS));
513     }
514
515     /**
516      * If this object's location equals codesource's location, then
            return true.
517      */
518     @TestTargetNew(
519         level = TestLevel.PARTIAL_COMPLETE,
520         notes = "",
521         method = "implies",
522         args = {java.security.CodeSource.class}
523     )
524     public void testImplies_02() throws Exception {
525         CodeSource thizCS = new CodeSource(urlSite, (Certificate[])
            null);
526         CodeSource thatCS = new CodeSource(thizCS.getLocation(),
527                 (Certificate[]) null);
528         assertTrue(thizCS.implies(thatCS));
529         assertTrue(thatCS.implies(thizCS));
530
531     }
532
533     /**
534      * This object's protocol (getLocation().getProtocol()) must be
            equal to
535      * codesource's protocol.
536      */
537     /*
538      * FIXME
539      * commented out for temporary, as there is no FTP:// protocol
            supported yet.
540      * to be uncommented back.
541      public void testImplies_03() throws Exception {
542      CodeSource thizCS = new CodeSource(urlDir, (Certificate[]) null);
543      CodeSource thatCS = new CodeSource(urlDirFtp, (Certificate[])
            null);
544      assertFalse(thizCS.implies(thatCS));
545      assertFalse(thatCS.implies(thizCS));
546      }
547      */
548     @TestTargetNew(
549         level = TestLevel.PARTIAL_COMPLETE,
550         notes = "",
551         method = "implies",
```

```
552         args = {java.security.CodeSource.class}
553     )
554     public void testImplies_03_tmp() throws Exception {
555         CodeSource thizCS = new CodeSource(urlDir, (Certificate[])
            null);
556         CodeSource thatCS = new CodeSource(urlDir_FileProtocol,
557                 (Certificate[]) null);
558         assertFalse(thizCS.implies(thatCS));
559         assertFalse(thatCS.implies(thizCS));
560     }
561
562     /**
563      * If this object's host (getLocation().getHost()) is not null, then
            the
564      * SocketPermission constructed with this object's host must imply
            the
565      * SocketPermission constructed with codesource's host.
566      */
567     @TestTargetNew(
568         level = TestLevel.PARTIAL_COMPLETE,
569         notes = "",
570         method = "implies",
571         args = {java.security.CodeSource.class}
572     )
573     public void testImplies_04() throws Exception {
574         CodeSource thizCS = new CodeSource(urlDir, (Certificate[])
            null);
575         CodeSource thatCS = new CodeSource(urlDirIP, (Certificate[])
            null);
576
577         assertTrue(thizCS.implies(thatCS));
578         assertTrue(thatCS.implies(thizCS));
579
580         //
581         // Check for another site - force to create SocketPermission
582         //
583         thatCS = new CodeSource(urlDirOtherSite, (Certificate[]) null);
584         assertFalse(thizCS.implies(thatCS));
585
586         //
587         // also check for getHost() == null
588         //
589         thizCS = new CodeSource(new URL("http", null, "file1"),
590                 (Certificate[]) null);
591         thatCS = new CodeSource(new URL("http", "another.host.com",
            "file1"),
592                 (Certificate[]) null);
593         // well, yes, this is accordint to the spec...
594         assertTrue(thizCS.implies(thatCS));
595         assertFalse(thatCS.implies(thizCS));
596     }
597
598     /**
599      * If this object's port (getLocation().getPort()) is not equal to
            -1 (that
600      * is, if a port is specified), it must equal codesource's port.
601      */
602     @TestTargetNew(
603         level = TestLevel.PARTIAL_COMPLETE,
604         notes = "",
```

```
605         method = "implies",
606         args = {java.security.CodeSource.class}
607     )
608     public void testImplies_05() throws Exception {
609         CodeSource thizCS = new CodeSource(urlDir_port80,
            (Certificate[]) null);
610         CodeSource thatCS = new CodeSource(urlDir, (Certificate[])
            null);
611
612         assertTrue(thizCS.implies(thatCS));
613         assertTrue(thatCS.implies(thizCS));
614
615         thizCS = new CodeSource(urlDir, (Certificate[]) null);
616         thatCS = new CodeSource(urlDir_port81, (Certificate[]) null);
617         //assert*True* because thizCS has 'port=-1'
618         assertTrue(thizCS.implies(thatCS));
619
620         thizCS = new CodeSource(urlDir_port81, (Certificate[]) null);
621         thatCS = new CodeSource(urlDir, (Certificate[]) null);
622         assertFalse(thizCS.implies(thatCS));
623         //
624         thizCS = new CodeSource(urlDir_port80, (Certificate[]) null);
625         thatCS = new CodeSource(urlDir_port81, (Certificate[]) null);
626         assertFalse(thizCS.implies(thatCS));
627     }
628
629     /**
630      * If this object's file (getLocation().getFile()) doesn't equal
631      * codesource's file, then the following checks are made: ...
632      */
633     @TestTargetNew(
634         level = TestLevel.PARTIAL_COMPLETE,
635         notes = "",
636         method = "implies",
637         args = {java.security.CodeSource.class}
638     )
639     public void testImplies_06() throws Exception {
640         CodeSource thizCS = new CodeSource(urlFile, (Certificate[])
            null);
641         CodeSource thatCS = new CodeSource(urlFile, (Certificate[])
            null);
642         assertTrue(thizCS.implies(thatCS));
643     }
644
645     /**
646      * ... If this object's file ends with "/-", then codesource's file
       must
647      * start with this object's file (exclusive the trailing "-").
648      */
649     @TestTargetNew(
650         level = TestLevel.PARTIAL_COMPLETE,
651         notes = "",
652         method = "implies",
653         args = {java.security.CodeSource.class}
654     )
655     public void testImplies_07() throws Exception {
656         CodeSource thiz = new CodeSource(urlFileDirMinus,
            (Certificate[]) null);
657         CodeSource that = new CodeSource(urlFile, (Certificate[]) null);
658         assertTrue(thiz.implies(that));
```

```
659
660         that = new CodeSource(urlFileWithAdditionalDirs, (Certificate[])
            null);
661         assertTrue(thiz.implies(that));
662
663         that = new CodeSource(urlFileDirOtherDir, (Certificate[]) null);
664         assertFalse(thiz.implies(that));
665     }
666
667     /**
668      * ... If this object's file ends with a "/*", then codesource's
         file must
669      * start with this object's file and must not have any further "/"
670      * separators.
671      */
672     @TestTargetNew(
673         level = TestLevel.PARTIAL_COMPLETE,
674         notes = "",
675         method = "implies",
676         args = {java.security.CodeSource.class}
677     )
678     public void testImplies_08() throws Exception {
679         CodeSource thiz = new CodeSource(urlFileDirStar, (Certificate[])
            null);
680         CodeSource that = new CodeSource(urlFile, (Certificate[]) null);
681         assertTrue(thiz.implies(that));
682         that = new CodeSource(urlFileWithAdditionalDirs, (Certificate[])
            null);
683         assertFalse(thiz.implies(that));
684         //
685         that = new CodeSource(urlFileDirOtherDir, (Certificate[]) null);
686         assertFalse(thiz.implies(that));
687         // must not have any further '/'
688         that = new CodeSource(new URL(urlFile.toString() + "/"),
689                 (Certificate[]) null);
690         assertFalse(thiz.implies(that));
691     }
692
693     /**
694      * ... If this object's file doesn't end with a "/", then
         codesource's file
695      * must match this object's file with a '/' appended.
696      */
697     @TestTargetNew(
698         level = TestLevel.PARTIAL_COMPLETE,
699         notes = "",
700         method = "implies",
701         args = {java.security.CodeSource.class}
702     )
703     public void testImplies_09() throws Exception {
704         CodeSource thizCS = new CodeSource(urlDir, (Certificate[])
            null);
705         CodeSource thatCS = new CodeSource(urlDirWithSlash,
706                 (Certificate[]) null);
707         assertTrue(thizCS.implies(thatCS));
708         assertFalse(thatCS.implies(thizCS));
709     }
710
711     /**
712      * If this object's reference (getLocation().getRef()) is not null,
```

```
          it must
  713      * equal codesource's reference.
  714      */
  715     @TestTargetNew(
  716         level = TestLevel.PARTIAL_COMPLETE,
  717         notes = "",
  718         method = "implies",
  719         args = {java.security.CodeSource.class}
  720     )
  721     public void testImplies_0A() throws Exception {
  722         CodeSource thizCS = new CodeSource(urlRef1, (Certificate[])
          null);
  723         CodeSource thatCS = new CodeSource(urlRef1, (Certificate[])
          null);
  724         assertTrue(thizCS.implies(thatCS));
  725
  726         thizCS = new CodeSource(urlRef1, (Certificate[]) null);
  727         thatCS = new CodeSource(urlRef2, (Certificate[]) null);
  728         assertFalse(thizCS.implies(thatCS));
  729
  730     }
  731
  732     /**
  733      * If this certificates are not null, then all of this certificates
          should
  734      * be presented in certificates of that codesource.
  735      */
  736     @TestTargetNew(
  737         level = TestLevel.PARTIAL_COMPLETE,
  738         notes = "",
  739         method = "implies",
  740         args = {java.security.CodeSource.class}
  741     )
  742     public void testImplies_0B() {
  743
  744         Certificate c0 = new TestCertUtils.TestCertificate("00");
  745         Certificate c1 = new TestCertUtils.TestCertificate("01");
  746         Certificate c2 = new TestCertUtils.TestCertificate("02");
  747         Certificate[] thizCerts = { c0, c1 };
  748         Certificate[] thatCerts = { c1, c0, c2 };
  749
  750         CodeSource thiz = new CodeSource(urlSite, thizCerts);
  751         CodeSource that = new CodeSource(urlSite, thatCerts);
  752         // two CodeSource-s with different set of certificates
  753         assertTrue(thiz.implies(that));
  754
  755         //
  756         that = new CodeSource(urlSite, (Certificate[]) null);
  757         // 'thiz' has set of certs, while 'that' has no certs. URL-s are
          the
  758         // same.
  759         assertFalse(thiz.implies(that));
  760         assertTrue(that.implies(thiz));
  761     }
  762
  763     /**
  764      * Testing with special URLs like 'localhost', 'file://' scheme ...
  765      * These special URLs have a special processing in implies(),
  766      * so they need to be covered and performance need to be checked
  767      */
```

```
768     @TestTargetNew(
769         level = TestLevel.PARTIAL_COMPLETE,
770         notes = "",
771         method = "implies",
772         args = {java.security.CodeSource.class}
773     )
774     public void testImplies_0C() throws Exception {
775         URL url0 = new URL("http://localhost/someDir");
776         URL url1 = new URL("http://localhost/someOtherDir");
777 
778         CodeSource thizCS = new CodeSource(url0, (Certificate[]) null);
779         CodeSource thatCS = new CodeSource(url1, (Certificate[]) null);
780         assertFalse(thizCS.implies(thatCS));
781         assertFalse(thatCS.implies(thizCS));
782     }
783 
784     /**
785      * Testing with special URLs like 'localhost', 'file://' scheme ...
786      * These special URLs have a special processing in implies(),
787      * so they need to be covered and performance need to be checked
788      */
789     @TestTargetNew(
790         level = TestLevel.PARTIAL_COMPLETE,
791         notes = "",
792         method = "implies",
793         args = {java.security.CodeSource.class}
794     )
795     public void testImplies_0D() throws Exception {
796         URL url0 = new URL("file:///" +
            System.getProperty("java.io.tmpdir")
797                 + File.separator + "someDir");
798         URL url1 = new URL("file:///" +
            System.getProperty("java.io.tmpdir")
799                 + File.separator + "someOtherDir");
800         CodeSource thizCS = new CodeSource(url0, (Certificate[]) null);
801         CodeSource thatCS = new CodeSource(url1, (Certificate[]) null);
802         assertFalse(thizCS.implies(thatCS));
803         assertFalse(thatCS.implies(thizCS));
804     }
805 }
```