```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

/**
 * Sample JUnit test fixture for FactoringUtility.
 *
 * @author Paolo Bucci
 *
 */
public final class FactoringUtilityTest {

    /**
     * Test aFactor with input 0.
     */
    @Test
    public void aFactorTest1() {
        /*
         * Set up variables and call method under test
         */
        int n = 1;
        int factor = FactoringUtility.aFactor(n);
        /*
         * Assert that values of variables match expectations
         */
        assertTrue(factor > 0);
        assertEquals(0, n % factor);
        assertEquals(1, n); // not necessary to check "restores n"; why?
    }

    /**
     * Test aFactor with input 1.
     */
    @Test
    public void aFactorTest2() {
        /*
         * Set up variables and call method under test
         */
        int n = 1;
        int factor = FactoringUtility.aFactor(n);
        /*
         * Assert that values of variables match expectations
         */
        assertTrue(factor > 0);
        assertEquals(0, n % factor);
    }

    /**
     * Test aFactor with input 2.
     */
    @Test
    public void aFactorTest3() {
        /*
         * Set up variables and call method under test
         */
        int n = 2;
```

```
58          int factor = FactoringUtility.aFactor(n);
59          /*
60           * Assert that values of variables match expectations
61           */
62          assertTrue(factor > 0);
63          assertEquals(0, n % factor);
64      }
65
66      /**
67       * Test aFactor with input 4.
68       */
69      @Test
70      public void aFactorTest4() {
71          /*
72           * Set up variables and call method under test
73           */
74          int n = 4;
75          int factor = FactoringUtility.aFactor(n);
76          /*
77           * Assert that values of variables match expectations
78           */
79          assertTrue(factor > 0);
80          assertEquals(0, n % factor);
81      }
82
83      /**
84       * Test aFactor with input 12.
85       */
86      @Test
87      public void aFactorTest5() {
88          /*
89           * Set up variables and call method under test
90           */
91          int n = 12;
92          int factor = FactoringUtility.aFactor(n);
93          /*
94           * Assert that values of variables match expectations
95           */
96          assertTrue(factor > 0);
97          assertEquals(0, n % factor);
98      }
99
100     /**
101      * Test aNonTrivialFactorV1 with input 15.
102      */
103     @Test
104     public void aNonTrivialFactorV1Test1() {
105         /*
106          * Set up variables and call method under test
107          */
108         int n = 15;
109         int factor = FactoringUtility.aNonTrivialFactorV1(n);
110         /*
111          * Assert that values of variables match expectations
112          */
113         assertTrue(1 < factor);
114         assertTrue(factor < n);
```

```
115         assertEquals(0, n % factor);
116     }
117
118     /**
119      * Test aNonTrivialFactorV1 with input 17.
120      */
121     @Test
122     public void aNonTrivialFactorV1Test2() {
123         /*
124          * Set up variables and call method under test
125          */
126         int n = 17;
127         int factor = FactoringUtility.aNonTrivialFactorV1(n);
128         /*
129          * Assert that values of variables match expectations
130          */
131         assertTrue(1 < factor);
132         assertTrue(2 < n);
133         assertEquals(0, n % factor);
134     }
135
136     /**
137      * Test aNonTrivialFactorV1 with input 32.
138      */
139     @Test
140     public void aNonTrivialFactorV1Test3() {
141         /*
142          * Set up variables and call method under test
143          */
144         int n = 32;
145         int factor = FactoringUtility.aNonTrivialFactorV1(n);
146         /*
147          * Assert that values of variables match expectations
148          */
149         assertTrue(1 < factor);
150         assertTrue(factor < n);
151         assertEquals(0, n % factor);
152     }
153
154     /**
155      * Test aNonTrivialFactorV2 with input 12.
156      */
157     @Test
158     public void aNonTrivialFactorV2Test1() {
159         /*
160          * Set up variables and call method under test
161          */
162         int n = 12;
163         int factor = FactoringUtility.aNonTrivialFactorV2(n);
164         /*
165          * Assert that values of variables match expectations
166          */
167         assertTrue(1 < factor);
168         assertTrue(factor < n);
169         assertEquals(0, n % factor);
170     }
171
```

```
172     /**
173      * Test aNonTrivialFactorV2 with input 15.
174      */
175     @Test
176     public void aNonTrivialFactorV2Test2() {
177         /*
178          * Set up variables and call method under test
179          */
180         int n = 15;
181         int factor = FactoringUtility.aNonTrivialFactorV2(n);
182         /*
183          * Assert that values of variables match expectations
184          */
185         assertTrue(1 < factor);
186         assertTrue(factor < n);
187         assertEquals(0, n % factor);
188     }
189 
190     /**
191      * Test aNonTrivialFactorV2 with input 25.
192      */
193     @Test
194     public void aNonTrivialFactorV2Test3() {
195         /*
196          * Set up variables and call method under test
197          */
198         int n = 25;
199         int factor = FactoringUtility.aNonTrivialFactorV2(n);
200         /*
201          * Assert that values of variables match expectations
202          */
203         assertTrue(1 < factor);
204         assertTrue(factor < n);
205         assertEquals(0, n % factor);
206     }
207 
208     /**
209      * Test aNonTrivialFactorV3 with input 16.
210      */
211     @Test
212     public void aNonTrivialFactorV3Test1() {
213         /*
214          * Set up variables and call method under test
215          */
216         int n = 16;
217         int factor = FactoringUtility.aNonTrivialFactorV3(n);
218         /*
219          * Assert that values of variables match expectations
220          */
221         assertTrue(1 < factor);
222         assertTrue(factor < n);
223         assertEquals(0, n % factor);
224     }
225 
226 }
```