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

/**
 * @author Vaishnavi Kasabwala
 *
 */
public class CryptoUtilitiesTest {

    /*
     * Tests of reduceToGCD
     */

    @Test
    public void testReduceToGCD_0_0() {
        NaturalNumber n = new NaturalNumber2(0);
        NaturalNumber nExpected = new NaturalNumber2(0);
        NaturalNumber m = new NaturalNumber2(0);
        NaturalNumber mExpected = new NaturalNumber2(0);
        CryptoUtilities.reduceToGCD(n, m);
        assertEquals(nExpected, n);
        assertEquals(mExpected, m);
    }

    @Test
    public void testReduceToGCD_30_21() {
        NaturalNumber n = new NaturalNumber2(30);
        NaturalNumber nExpected = new NaturalNumber2(3);
        NaturalNumber m = new NaturalNumber2(21);
        NaturalNumber mExpected = new NaturalNumber2(0);
        CryptoUtilities.reduceToGCD(n, m);
        assertEquals(nExpected, n);
        assertEquals(mExpected, m);
    }

    /*
     * Tests of isEven
     */

    @Test
    public void testIsEven_0() {
        NaturalNumber n = new NaturalNumber2(0);
        NaturalNumber nExpected = new NaturalNumber2(0);
        boolean result = CryptoUtilities.isEven(n);
        assertEquals(nExpected, n);
        assertEquals(true, result);
    }

    @Test
    public void testIsEven_1() {
        NaturalNumber n = new NaturalNumber2(1);
        NaturalNumber nExpected = new NaturalNumber2(1);
        boolean result = CryptoUtilities.isEven(n);
        assertEquals(nExpected, n);
        assertEquals(false, result);
    }

    /*
```

```
63      * Tests of powerMod
64      */
65
66     @Test
67     public void testPowerMod_0_0_2() {
68         NaturalNumber n = new NaturalNumber2(0);
69         NaturalNumber nExpected = new NaturalNumber2(1);
70         NaturalNumber p = new NaturalNumber2(0);
71         NaturalNumber pExpected = new NaturalNumber2(0);
72         NaturalNumber m = new NaturalNumber2(2);
73         NaturalNumber mExpected = new NaturalNumber2(2);
74         CryptoUtilities.powerMod(n, p, m);
75         assertEquals(nExpected, n);
76         assertEquals(pExpected, p);
77         assertEquals(mExpected, m);
78     }
79
80     @Test
81     public void testPowerMod_17_18_19() {
82         NaturalNumber n = new NaturalNumber2(17);
83         NaturalNumber nExpected = new NaturalNumber2(1);
84         NaturalNumber p = new NaturalNumber2(18);
85         NaturalNumber pExpected = new NaturalNumber2(18);
86         NaturalNumber m = new NaturalNumber2(19);
87         NaturalNumber mExpected = new NaturalNumber2(19);
88         CryptoUtilities.powerMod(n, p, m);
89         assertEquals(nExpected, n);
90         assertEquals(pExpected, p);
91         assertEquals(mExpected, m);
92     }
93
94     /*
95      * Test of isPrime1
96      */
97     @Test
98     // boundary, 2 is the only even prime number
99     public void testIsPrime1_2() {
100        NaturalNumber n = new NaturalNumber2(2);
101
102        boolean prime = true;
103
104        assertEquals(prime, CryptoUtilities.isPrime1(n));
105    }
106
107    // routine
108    @Test
109    public void testIsPrime1_15() {
110        NaturalNumber n = new NaturalNumber2(15);
111
112        boolean prime = false;
113
114        assertEquals(prime, CryptoUtilities.isPrime1(n));
115    }
116
117    // challenging, large number
118    @Test
119    public void testIsPrime1_3788923469() {
```

```
120         NaturalNumber n = new NaturalNumber2("3788923469");
121
122         boolean prime = true;
123
124         assertEquals(prime, CryptoUtilities.isPrime1(n));
125     }
126
127     /*
128      * Test of isPrime2
129      */
130     @Test
131     // boundary, 2 is the only prime even number
132     public void testIsPrime2_2() {
133
134         NaturalNumber n = new NaturalNumber2(2);
135
136         boolean prime = true;
137
138         assertEquals(prime, CryptoUtilities.isPrime2(n));
139     }
140
141     @Test
142     // routine
143     public void testIsPrime2_27() {
144         NaturalNumber n = new NaturalNumber2(27);
145
146         boolean prime = false;
147
148         assertEquals(prime, CryptoUtilities.isPrime2(n));
149     }
150
151     @Test
152     // challenging, large natural number
153     public void testIsPrime2_3788923469() {
154         NaturalNumber n = new NaturalNumber2("3788923469");
155
156         boolean prime = true;
157
158         assertEquals(prime, CryptoUtilities.isPrime2(n));
159     }
160
161     /*
162      * Test of isWitnessToCompositeness
163      */
164     @Test
165     // boundary,  2 is the only prime even number
166     public void testIsWitnessToCompositeness_2_30() {
167         NaturalNumber w = new NaturalNumber2(2);
168         NaturalNumber n = new NaturalNumber2(30);
169         assertEquals("2", w.toString());
170         assertEquals("30", n.toString());
171         boolean truth = true;
172         assertEquals(truth, CryptoUtilities.isWitnessToCompositeness(w, n));
173     }
174
175     @Test
176     // routine
```

```
177     public void testIsWitnessToCompositeness_15_65() {
178         NaturalNumber w = new NaturalNumber2(15);
179         NaturalNumber n = new NaturalNumber2(65);
180         assertEquals("15", w.toString());
181         assertEquals("65", n.toString());
182         boolean wrong = false;
183         assertEquals(wrong, CryptoUtilities.isWitnessToCompositeness(w, n));
184     }
185 
186     @Test
187     // challenging, uses larger numbers
188     public void testIsWitnessToCompositeness_30_990() {
189         NaturalNumber w = new NaturalNumber2(30);
190         NaturalNumber n = new NaturalNumber2(990);
191         assertEquals("30", w.toString());
192         assertEquals("990", n.toString());
193         boolean truth = true;
194         assertEquals(truth, CryptoUtilities.isWitnessToCompositeness(w, n));
195     }
196 
197     /*
198      * Test of generateNextLikelyNumber
199      */
200     @Test
201     // boundary, smallest prime numbers
202     public void testGenerateNextLikelyPrime_2() {
203         NaturalNumber n = new NaturalNumber2(2);
204         CryptoUtilities.generateNextLikelyPrime(n);
205         assertEquals("3", n.toString());
206     }
207 
208     // routine
209     public void testGenerateNextLikelyPrime_20() {
210         NaturalNumber n = new NaturalNumber2(20);
211         CryptoUtilities.generateNextLikelyPrime(n);
212         assertEquals("23", n.toString());
213     }
214 
215     // challenging, large number
216     public void testGenerateNextLikelyPrime_3788923467() {
217         NaturalNumber n = new NaturalNumber2("3788923467");
218         CryptoUtilities.generateNextLikelyPrime(n);
219         assertEquals("3788923469", n.toString());
220     }
221 
222 }
```