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

import org.junit.Test;

import components.naturalnumber.NaturalNumber;
import components.naturalnumber.NaturalNumber2;

/**
 * @author Gabe Azzarita
 *
 */
public class CryptoUtilitiesTest {

    /*
     * Tests of reduceToGCD
     *
     * 0_0 as a boundary case, 30_21 as a routine case, 56183649_34567893 as a
     * challenging case
     *
     */

    @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);
    }

    @Test
    public void testReduceToGCD_56183649_34567893() {
        NaturalNumber n = new NaturalNumber2(56183649);
        NaturalNumber nExpected = new NaturalNumber2(3);
        NaturalNumber m = new NaturalNumber2(34567893);
        NaturalNumber mExpected = new NaturalNumber2(0);
        CryptoUtilities.reduceToGCD(n, m);
        assertEquals(nExpected, n);
```

```
    assertEquals(mExpected, m);
}

/*
 * Tests of isEven (at least one even and one odd)
 *
 * 0 as a boundary case, 1 as a boundary case, 15234 as a routine case,
 * 9365401456712629461 as a challenging case
 *
 */

@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);
}

@Test
public void testIsEven_15234() {
    NaturalNumber n = new NaturalNumber2(15234);
    NaturalNumber nExpected = new NaturalNumber2(15234);
    boolean result = CryptoUtilities.isEven(n);
    assertEquals(nExpected, n);
    assertEquals(true, result);
}

@Test
public void testIsEven_9365401456712629461() {
    NaturalNumber n = new NaturalNumber2("9365401456712629461");
    NaturalNumber nExpected = new NaturalNumber2("9365401456712629461");
    boolean result = CryptoUtilities.isEven(n);
    assertEquals(nExpected, n);
    assertEquals(false, result);
}

/*
 * Tests of powerMod (at least one odd and one even)
 *
 * 0_0_2 as a boundary case, 17_18_19 as a boundary case (bigger than maxInt
```

```
 * value), 7_11_33 as a routine case
 *
 *
 */

@Test
public void testPowerMod_0_0_2() {
    NaturalNumber n = new NaturalNumber2(0);
    NaturalNumber nExpected = new NaturalNumber2(1);
    NaturalNumber p = new NaturalNumber2(0);
    NaturalNumber pExpected = new NaturalNumber2(0);
    NaturalNumber m = new NaturalNumber2(2);
    NaturalNumber mExpected = new NaturalNumber2(2);
    CryptoUtilities.powerMod(n, p, m);
    assertEquals(nExpected, n);
    assertEquals(pExpected, p);
    assertEquals(mExpected, m);
}

@Test
public void testPowerMod_17_18_19() {
    NaturalNumber n = new NaturalNumber2(17);
    NaturalNumber nExpected = new NaturalNumber2(1);
    NaturalNumber p = new NaturalNumber2(18);
    NaturalNumber pExpected = new NaturalNumber2(18);
    NaturalNumber m = new NaturalNumber2(19);
    NaturalNumber mExpected = new NaturalNumber2(19);
    CryptoUtilities.powerMod(n, p, m);
    assertEquals(nExpected, n);
    assertEquals(pExpected, p);
    assertEquals(mExpected, m);
}

@Test
public void testPowerMod_7_11_33() {
    NaturalNumber n = new NaturalNumber2(7);
    NaturalNumber nExpected = new NaturalNumber2(7);
    NaturalNumber p = new NaturalNumber2(11);
    NaturalNumber pExpected = new NaturalNumber2(11);
    NaturalNumber m = new NaturalNumber2(33);
    NaturalNumber mExpected = new NaturalNumber2(33);
    CryptoUtilities.powerMod(n, p, m);
    assertEquals(nExpected, n);
    assertEquals(pExpected, p);
    assertEquals(mExpected, m);
}

/*
 * Tests of isWitnessToCompositeness
 *
 * 2_4 as a boundary case, 4_10 as a routine case, 345_56912 as a
```

```
 * challenging case
 *
 */

@Test
public void testIsWitnessToCompositeness_2_4() {
    NaturalNumber n = new NaturalNumber2(2);
    NaturalNumber nExpected = new NaturalNumber2(2);
    NaturalNumber p = new NaturalNumber2(4);
    NaturalNumber pExpected = new NaturalNumber2(4);
    boolean result = CryptoUtilities.isWitnessToCompositeness(n, p);
    boolean expectedResult = true;
    assertEquals(expectedResult, result);
    assertEquals(nExpected, n);
    assertEquals(pExpected, p);
}

@Test
public void testIsWitnessToCompositeness_4_11() {
    NaturalNumber n = new NaturalNumber2(4);
    NaturalNumber nExpected = new NaturalNumber2(4);
    NaturalNumber p = new NaturalNumber2(11);
    NaturalNumber pExpected = new NaturalNumber2(11);
    boolean result = CryptoUtilities.isWitnessToCompositeness(n, p);
    boolean expectedResult = false;
    assertEquals(expectedResult, result);
    assertEquals(nExpected, n);
    assertEquals(pExpected, p);
}

@Test
public void testIsWitnessToCompositeness_345_56912() {
    NaturalNumber n = new NaturalNumber2(2);
    NaturalNumber nExpected = new NaturalNumber2(2);
    NaturalNumber p = new NaturalNumber2(10);
    NaturalNumber pExpected = new NaturalNumber2(10);
    boolean result = CryptoUtilities.isWitnessToCompositeness(n, p);
    boolean expectedResult = true;
    assertEquals(expectedResult, result);
    assertEquals(nExpected, n);
    assertEquals(pExpected, p);
}

/*
 * Tests of isPrime2 (at least one even and odd, and at least one prime)
 *
 * 2 as a boundary, 5124 as a routine case, 4641 as a routine case,
 * 29996224275833 as a challenging case
 *
 */
```

```java
@Test
public void testIsPrime2_2() {
    NaturalNumber n = new NaturalNumber2(2);
    NaturalNumber nExpected = new NaturalNumber2(2);
    boolean result = CryptoUtilities.isPrime2(n);
    boolean expectedResult = true;
    assertEquals(expectedResult, result);
    assertEquals(nExpected, n);
}

@Test
public void testIsPrime2_5124() {
    NaturalNumber n = new NaturalNumber2(5124);
    NaturalNumber nExpected = new NaturalNumber2(5124);
    boolean result = CryptoUtilities.isPrime2(n);
    boolean expectedResult = false;
    assertEquals(expectedResult, result);
    assertEquals(nExpected, n);
}

@Test
public void testIsPrime2_4641() {
    NaturalNumber n = new NaturalNumber2(4641);
    NaturalNumber nExpected = new NaturalNumber2(4641);
    boolean result = CryptoUtilities.isPrime2(n);
    boolean expectedResult = false;
    assertEquals(expectedResult, result);
    assertEquals(nExpected, n);
}

@Test
public void testIsPrime2_29996224275833() {
    NaturalNumber n = new NaturalNumber2("29996224275833");
    NaturalNumber nExpected = new NaturalNumber2("29996224275833");
    boolean result = CryptoUtilities.isPrime2(n);
    boolean expectedResult = true;
    assertEquals(expectedResult, result);
    assertEquals(nExpected, n);
}

/*
 * Tests of generateNextLikelyPrime (at least one even and odd)
 *
 * 2 as a boundary case, 6917 as a routine case, 6860 as a routine case,
 * 29996224275832 as a challenging case
 *
 */

@Test
public void testGenerateNextLikelyPrime_2() {
    NaturalNumber n = new NaturalNumber2(2);
```

```
        NaturalNumber nExpected = new NaturalNumber2(2);
        CryptoUtilities.generateNextLikelyPrime(n);
        assertEquals(nExpected, n);
    }

    @Test
    public void testGenerateNextLikelyPrime_6917() {
        NaturalNumber n = new NaturalNumber2(6947);
        NaturalNumber nExpected = new NaturalNumber2(6947);
        CryptoUtilities.generateNextLikelyPrime(n);
        assertEquals(nExpected, n);
    }

    @Test
    public void testGenerateNextLikelyPrime_6860() {
        NaturalNumber n = new NaturalNumber2(6860);
        NaturalNumber nExpected = new NaturalNumber2(6863);
        CryptoUtilities.generateNextLikelyPrime(n);
        assertEquals(nExpected, n);
    }

    @Test
    public void testGenerateNextLikelyPrime_29996224275832() {
        NaturalNumber n = new NaturalNumber2("29996224275832");
        NaturalNumber nExpected = new NaturalNumber2("29996224275833");
        CryptoUtilities.generateNextLikelyPrime(n);
        assertEquals(nExpected, n);
    }

}
```