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

import org.junit.Test;

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

/**
 * @author David Park
 *
 */
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);
    }

    /*
     * Tests of powerMod
     */

    @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 isPrime2_50() {
        NaturalNumber n = new NaturalNumber2(50);
        NaturalNumber nExpected = new NaturalNumber2(50);
        boolean result = CryptoUtilities.isPrime2(n);
        assertEquals(nExpected, n);
        assertEquals(false, result);
    }

    @Test
    public void isWitness2_30() {
        NaturalNumber two = new NaturalNumber2(2);
        NaturalNumber twoExpected = new NaturalNumber2(2);
        NaturalNumber thirty = new NaturalNumber2(30);
        NaturalNumber thirtyExpected = new NaturalNumber2(30);
        boolean result = CryptoUtilities.isWitnessToCompositeness(two, thirty);
        assertEquals(twoExpected, two);
        assertEquals(thirtyExpected, thirty);
        assertEquals(true, result);
    }

```

```java
    @Test
    public void testReduceToGCD_1_1() {
        NaturalNumber n = new NaturalNumber2(1);
        NaturalNumber nExpected = new NaturalNumber2(1);
        NaturalNumber m = new NaturalNumber2(1);
        NaturalNumber mExpected = new NaturalNumber2(0);
        CryptoUtilities.reduceToGCD(n, m);
        assertEquals(nExpected, n);
        assertEquals(mExpected, m);
    }

    @Test
    public void testReduceToGCD_100_10() {
        NaturalNumber n = new NaturalNumber2(100);
        NaturalNumber nExpected = new NaturalNumber2(10);
        NaturalNumber m = new NaturalNumber2(10);
        NaturalNumber mExpected = new NaturalNumber2(0);
        CryptoUtilities.reduceToGCD(n, m);
        assertEquals(nExpected, n);
        assertEquals(mExpected, m);
    }

    @Test
    public void testReduceToGCD_25_15() {
        NaturalNumber n = new NaturalNumber2(25);
        NaturalNumber nExpected = new NaturalNumber2(5);
        NaturalNumber m = new NaturalNumber2(15);
        NaturalNumber mExpected = new NaturalNumber2(0);
        CryptoUtilities.reduceToGCD(n, m);
        assertEquals(nExpected, n);
        assertEquals(mExpected, m);
    }

    @Test
    public void testIsEven_2() {
        NaturalNumber n = new NaturalNumber2(2);
        boolean result = CryptoUtilities.isEven(n);
        assertEquals(true, result);
    }

    @Test
    public void testIsEven_999() {
        NaturalNumber n = new NaturalNumber2(999);
        boolean result = CryptoUtilities.isEven(n);
        assertEquals(false, result);
    }

    @Test
    public void testIsEven_22() {
        NaturalNumber n = new NaturalNumber2(22);
        boolean result = CryptoUtilities.isEven(n);
        assertEquals(true, result);
    }

    @Test
    public void testIsEven_45() {
        NaturalNumber n = new NaturalNumber2(45);
        boolean result = CryptoUtilities.isEven(n);
        assertEquals(false, result);
    }

    @Test
    public void testPowerMod_2_5_13() {
        NaturalNumber n = new NaturalNumber2(2);
        NaturalNumber p = new NaturalNumber2(5);
        NaturalNumber m = new NaturalNumber2(13);
        CryptoUtilities.powerMod(n, p, m);
        NaturalNumber nExpected = new NaturalNumber2(6); // 2^5 mod 13 = 6
        assertEquals(nExpected, n);
    }

    @Test
    public void testPowerMod_7_3_10() {
        NaturalNumber n = new NaturalNumber2(7);
        NaturalNumber p = new NaturalNumber2(3);
        NaturalNumber m = new NaturalNumber2(10);
        CryptoUtilities.powerMod(n, p, m);
        NaturalNumber nExpected = new NaturalNumber2(3); // 7^3 mod 10 = 3
        assertEquals(nExpected, n);
    }

    @Test
    public void testPowerMod_3_4_5() {
        NaturalNumber n = new NaturalNumber2(3);
        NaturalNumber p = new NaturalNumber2(4);
        NaturalNumber m = new NaturalNumber2(5);
        CryptoUtilities.powerMod(n, p, m);
        NaturalNumber nExpected = new NaturalNumber2(1); // 3^4 mod 5 = 1
        assertEquals(nExpected, n);
    }

    @Test
    public void testPowerMod_5_3_23() {
        NaturalNumber n = new NaturalNumber2(5);
        NaturalNumber p = new NaturalNumber2(3);
        NaturalNumber m = new NaturalNumber2(23);
        CryptoUtilities.powerMod(n, p, m);
        NaturalNumber nExpected = new NaturalNumber2(10); // 5^3 mod 23 = 10
        assertEquals(nExpected, n);
    }

    @Test
    public void testPowerMod_6_2_7() {
        NaturalNumber n = new NaturalNumber2(6);
        NaturalNumber p = new NaturalNumber2(2);
        NaturalNumber m = new NaturalNumber2(7);
        CryptoUtilities.powerMod(n, p, m);
        NaturalNumber nExpected = new NaturalNumber2(1); // 6^2 mod 7 = 1
        assertEquals(nExpected, n);
    }

    @Test
    public void isWitnessToCompositeness_2_4() {
        NaturalNumber two = new NaturalNumber2(2);
        NaturalNumber four = new NaturalNumber2(4);
        boolean result = CryptoUtilities.isWitnessToCompositeness(two, four);
        assertEquals(true, result); // 2 is a witness to 4 being composite
    }

    @Test
    public void isWitnessToCompositeness_3_7() {
        NaturalNumber three = new NaturalNumber2(3);
        NaturalNumber seven = new NaturalNumber2(7);
        boolean result = CryptoUtilities.isWitnessToCompositeness(three, seven);
        assertEquals(false, result); // 3 is not a witness to 7 being composite
    }

    @Test
    public void isWitnessToCompositeness_2_9() {
        NaturalNumber two = new NaturalNumber2(2);
        NaturalNumber nine = new NaturalNumber2(9);
        boolean result = CryptoUtilities.isWitnessToCompositeness(two, nine);
        assertEquals(true, result);
    }

    @Test
    public void isWitnessToCompositeness_10_12() {
        NaturalNumber ten = new NaturalNumber2(10);
        NaturalNumber twelve = new NaturalNumber2(12);
        boolean result = CryptoUtilities.isWitnessToCompositeness(ten, twelve);
        assertEquals(true, result);
    }

    @Test
    public void isPrime1_2() {
        NaturalNumber two = new NaturalNumber2(2);
        boolean result = CryptoUtilities.isPrime1(two);
        assertEquals(true, result);
    }

    @Test
    public void isPrime1_4() {
        NaturalNumber four = new NaturalNumber2(4);
        boolean result = CryptoUtilities.isPrime1(four);
        assertEquals(false, result);
    }

    @Test
    public void isPrime1_5() {
        NaturalNumber five = new NaturalNumber2(5);
        boolean result = CryptoUtilities.isPrime1(five);
        assertEquals(true, result);
    }

    @Test
    public void isPrime1_10() {
        NaturalNumber ten = new NaturalNumber2(10);
        boolean result = CryptoUtilities.isPrime1(ten);
        assertEquals(false, result);
    }

```

```java
    @Test
    public void isPrime1_97() {
        NaturalNumber ninetySeven = new NaturalNumber2(97);
        boolean result = CryptoUtilities.isPrime1(ninetySeven);
        assertEquals(true, result);
    }

    @Test
    public void isPrime2_3() {
        NaturalNumber three = new NaturalNumber2(3);
        boolean result = CryptoUtilities.isPrime2(three);
        assertEquals(true, result); // 3 is a prime number
    }

    @Test
    public void isPrime2_9() {
        NaturalNumber nine = new NaturalNumber2(9);
        boolean result = CryptoUtilities.isPrime2(nine);
        assertEquals(false, result); // 9 is not a prime number
    }

    @Test
    public void isPrime2_11() {
        NaturalNumber eleven = new NaturalNumber2(11);
        boolean result = CryptoUtilities.isPrime2(eleven);
        assertEquals(true, result);
    }

    @Test
    public void isPrime2_15() {
        NaturalNumber fifteen = new NaturalNumber2(15);
        boolean result = CryptoUtilities.isPrime2(fifteen);
        assertEquals(false, result);
    }

    @Test
    public void isPrime2_101() {
        NaturalNumber oneHundredOne = new NaturalNumber2(101);
        boolean result = CryptoUtilities.isPrime2(oneHundredOne);
        assertEquals(true, result);
    }

    @Test
    public void generateNextLikelyPrime_startingFrom2() {
        NaturalNumber n = new NaturalNumber2(2);
        CryptoUtilities.generateNextLikelyPrime(n);
        NaturalNumber expected = new NaturalNumber2(3); // The next prime after 2 is 3
        assertEquals(expected, n);
    }

    @Test
    public void generateNextLikelyPrime_startingFrom14() {
        NaturalNumber n = new NaturalNumber2(14);
        CryptoUtilities.generateNextLikelyPrime(n);
        NaturalNumber expected = new NaturalNumber2(17); // The next prime after 14 is 17
        assertEquals(expected, n);
    }

    @Test
    public void generateNextLikelyPrime_startingFrom25() {
        NaturalNumber n = new NaturalNumber2(25);
        CryptoUtilities.generateNextLikelyPrime(n);
        NaturalNumber expected = new NaturalNumber2(29); // The next prime after 25 is 29
        assertEquals(expected, n);
    }

    @Test
    public void generateNextLikelyPrime_startingFrom1() {
        NaturalNumber n = new NaturalNumber2(1);
        CryptoUtilities.generateNextLikelyPrime(n);
        NaturalNumber expected = new NaturalNumber2(2); // The next prime after 1 is 2
        assertEquals(expected, n);
    }

}

```