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

import org.junit.Test;

import components.naturalnumber.NaturalNumber;
import components.naturalnumber.NaturalNumber1L;

/**
 * JUnit test fixture for {@code NaturalNumber}'s constructors and kernel
 * methods.
 *
 * @author David P. and Zach B.
 *
 */
public abstract class NaturalNumberTest {

    /**
     * Invokes the appropriate {@code NaturalNumber} constructor for the
     * implementation under test and returns the result.
     *
     * @return the new number
     * @ensures constructorTest = 0
     */
    protected abstract NaturalNumber constructorTest();

    /**
     * Invokes the appropriate {@code NaturalNumber} constructor for the
     * implementation under test and returns the result.
     *
     * @param i
     *            {@code int} to initialize from
     * @return the new number
     * @requires i >= 0
     * @ensures constructorTest = i
     */
    protected abstract NaturalNumber constructorTest(int i);

    /**
     * Invokes the appropriate {@code NaturalNumber} constructor for the
     * implementation under test and returns the result.
     *
     * @param s
     *            {@code String} to initialize from
     * @return the new number
     * @requires there exists n: NATURAL (s = TO_STRING(n))
     * @ensures s = TO_STRING(constructorTest)
     */
    protected abstract NaturalNumber constructorTest(String s);

    /**
     * Invokes the appropriate {@code NaturalNumber} constructor for the
     * implementation under test and returns the result.
     *
     * @param n
     *            {@code NaturalNumber} to initialize from
     * @return the new number
     * @ensures constructorTest = n
     */
    protected abstract NaturalNumber constructorTest(NaturalNumber n);

    /**
     * Invokes the appropriate {@code NaturalNumber} constructor for the
     * reference implementation and returns the result.
     *
     * @return the new number
     * @ensures constructorRef = 0
     */
    protected abstract NaturalNumber constructorRef();

    /**
     * Invokes the appropriate {@code NaturalNumber} constructor for the
     * reference implementation and returns the result.
     *
     * @param i
     *            {@code int} to initialize from
     * @return the new number
     * @requires i >= 0
     * @ensures constructorRef = i
     */
    protected abstract NaturalNumber constructorRef(int i);

    /**
     * Invokes the appropriate {@code NaturalNumber} constructor for the
     * reference implementation and returns the result.
     *
     * @param s
     *            {@code String} to initialize from
     * @return the new number
     * @requires there exists n: NATURAL (s = TO_STRING(n))
     * @ensures s = TO_STRING(constructorRef)
     */
    protected abstract NaturalNumber constructorRef(String s);

    /**
     * Invokes the appropriate {@code NaturalNumber} constructor for the
     * reference implementation and returns the result.
     *
     * @param n
     *            {@code NaturalNumber} to initialize from
     * @return the new number
     * @ensures constructorRef = n
     */
    protected abstract NaturalNumber constructorRef(NaturalNumber n);

    /**
     * Verify no-argument constructor initializes to zero.
     */
    @Test
    public void testNoArgumentConstructor() {
        NaturalNumber n1 = this.constructorTest();

        NaturalNumber n2 = this.constructorRef();

        assertEquals(n1, n2);
    }

    /**
     * Check if constructor with int 0 initializes correctly.
     */
    @Test
    public void testConstructorWithIntZero() {
        int value = 0;
        NaturalNumber n1 = this.constructorTest(value);
        NaturalNumber n2 = this.constructorRef(value);
        assertEquals(n1, n2);
    }

    /**
     * Check if constructor with small int initializes correctly.
     */
    @Test
    public void testConstructorWithSmallInt() {
        int value = 17;
        NaturalNumber n1 = this.constructorTest(value);
        NaturalNumber n2 = this.constructorRef(value);
        assertEquals(n1, n2);
    }

    /**
     * Check if constructor with large int initializes correctly.
     */
    @Test
    public void testConstructorWithLargeInt() {
        int value = 987654321;
        NaturalNumber n1 = this.constructorTest(value);
        NaturalNumber n2 = this.constructorRef(value);
        assertEquals(n1, n2);
    }

    /**
     * Check if constructor with string "0" initializes correctly.
     */
    @Test
    public void testConstructorWithStringZero() {
        String value = "0";
        NaturalNumber n1 = this.constructorTest(value);
        NaturalNumber n2 = this.constructorRef(value);
        assertEquals(n1, n2);
    }

    /**
     * Check if constructor with small string initializes correctly.
     */
    @Test
    public void testConstructorWithSmallString() {
        String value = "456";
        NaturalNumber n1 = this.constructorTest(value);
        NaturalNumber n2 = this.constructorRef(value);
        assertEquals(n1, n2);
    }

    /**
     * Check if constructor with large string initializes correctly.
     */
    @Test
    public void testConstructorWithLargeString() {
        String value = "12345678901234567890";
        NaturalNumber n1 = this.constructorTest(value);
        NaturalNumber n2 = this.constructorRef(value);
        assertEquals(n1, n2);
    }

    /**
     * Verify constructor from NaturalNumber.
     */
    @Test
    public void testConstructorFromNaturalNumber() {
        NaturalNumber source = this.constructorRef(100);
        NaturalNumber n1 = this.constructorTest(source);
        NaturalNumber n2 = this.constructorRef(100);
        assertEquals(n1, n2);
    }

    /**
     * Verify constructor from NaturalNumber with a small value.
     */
    @Test
    public void testConstructorFromSmallNaturalNumber() {
        NaturalNumber source = this.constructorRef(35);
        NaturalNumber n1 = this.constructorTest(source);
        NaturalNumber n2 = this.constructorRef(35);
        assertEquals(n1, n2);
    }

    /**
     * Verify constructor from NaturalNumber with a large value.
     */
    @Test
    public void testConstructorFromLargeNaturalNumber() {
        NaturalNumber source = this.constructorRef(54321);
        NaturalNumber n1 = this.constructorTest(source);
        NaturalNumber n2 = this.constructorRef(54321);
        assertEquals(n1, n2);
    }

    /**
     * Verify constructor from NaturalNumber from zero on different
     * implementation.
     */
    @Test
    public void testConstructorFromZeroFromNaturalNumber1L() {
        NaturalNumber1L temp = new NaturalNumber1L(0);

        NaturalNumber n1 = this.constructorTest(temp);
        NaturalNumber n2 = this.constructorRef(temp);
        assertEquals(n1, n2);
    }

    /**
     * Verify constructor from NaturalNumber with different implementation.
     */
    @Test
    public void testConstructorFromNaturalNumber1L() {
        NaturalNumber1L temp = new NaturalNumber1L(25);

        NaturalNumber n1 = this.constructorTest(temp);
        NaturalNumber n2 = this.constructorRef(temp);
        assertEquals(n1, n2);
    }

    /**
     * Test multiplyBy10 with zero.
     */
    @Test
    public void testMultiplyBy10WithZero() {
        int digit = 0;
        NaturalNumber n1 = this.constructorTest();
        NaturalNumber n2 = this.constructorRef(digit);
        n1.multiplyBy10(digit);
        assertEquals(n1, n2);
    }

    /**
     * Test multiplyBy10 with a small number.
     */
    @Test
    public void testMultiplyBy10WithSmallNumber() {
        int expectedValue = 52;
        NaturalNumber n1 = this.constructorTest(5);
        NaturalNumber n2 = this.constructorRef(expectedValue);
        n1.multiplyBy10(2);
        assertEquals(n1, n2);
    }

    /**
     * Test multiplyBy10 with a large number.
     */
    @Test
    public void testMultiplyBy10WithLargeNumber() {
        int number = 45;
        int expectedValue = 453;
        NaturalNumber n1 = this.constructorTest(number);
        NaturalNumber n2 = this.constructorRef(expectedValue);
        n1.multiplyBy10(3);
        assertEquals(n1, n2);
    }

    /**
     * Test multiplyBy10 with multiple of 10/100.
     */
    @Test
    public void testMultiplyBy10With100() {
        int number = 10;
        int expectedValue = 100;
        NaturalNumber n1 = this.constructorTest(number);
```

```java
        NaturalNumber n2 = this.constructorRef(expectedValue);
        n1.multiplyBy10(0);
        assertEquals(n1, n2);
    }

    /**
     * TEST multiplyBy10 with zero and two.
     */
    @Test
    public void testMultiplyBy10WithZeroAndTwo() {
        int number = 0;
        int expectedValue = 2;
        NaturalNumber n1 = this.constructorTest(number);
        NaturalNumber n2 = this.constructorRef(expectedValue);
        n1.multiplyBy10(2);
        assertEquals(n1, n2);
    }

    /**
     * Test divideBy10 with a small number.
     */
    @Test
    public void testDivideBy10WithSmallNumber() {
        int number = 36;
        int expectedQuotient = 3;
        int expectedRemainder = 6;
        NaturalNumber n1 = this.constructorTest(number);
        NaturalNumber n2 = this.constructorRef(expectedQuotient);
        int remainder = n1.divideBy10();
        assertEquals(remainder, expectedRemainder);
        assertEquals(n1, n2);
    }

    /**
     * Test divideBy10 with a large number.
     */
    @Test
    public void testDivideBy10WithLargeNumber() {
        int number = 78912;
        int expectedQuotient = 7891;
        int expectedRemainder = 2;
        NaturalNumber n1 = this.constructorTest(number);
        NaturalNumber n2 = this.constructorRef(expectedQuotient);
        int remainder = n1.divideBy10();
        assertEquals(remainder, expectedRemainder);
        assertEquals(n1, n2);
    }

    /**
     * Test divideBy10 with a single digit.
     */
    @Test
    public void testDivideBy10WithSingleDigit() {
        int number = 8;
        int expectedQuotient = 0;
        int expectedRemainder = 8;
        NaturalNumber n1 = this.constructorTest(number);
        NaturalNumber n2 = this.constructorRef(expectedQuotient);
        int remainder = n1.divideBy10();
        assertEquals(remainder, expectedRemainder);
        assertEquals(n1, n2);
    }

    /**
     * Test divideBy10 with zero.
     */
    @Test
    public void testDivideBy10WithZero() {
        int number = 0;
        int expectedQuotient = 0;
        int expectedRemainder = 0;
        NaturalNumber n1 = this.constructorTest(number);
        NaturalNumber n2 = this.constructorRef(expectedQuotient);
        int remainder = n1.divideBy10();
        assertEquals(remainder, expectedRemainder);
        assertEquals(n1, n2);
    }

    /**
     * Check if isZero returns true for the default value.
     */
    @Test
    public void testIsZeroTrueForDefault() {
        NaturalNumber n1 = this.constructorTest();
        boolean isZero = n1.isZero();
        assertEquals(isZero, true);
    }

    /**
     * Check if isZero returns true for zero value.
     */
    @Test
    public void testIsZeroTrueForZeroValue() {
        NaturalNumber n1 = this.constructorTest(0);
        boolean isZero = n1.isZero();
        assertEquals(isZero, true);
    }

    /**
     * Check if isZero returns false for a small non-zero value.
     */
    @Test
    public void testIsZeroFalseForSmallValue() {
        NaturalNumber n1 = this.constructorTest(8);
        boolean isZero = n1.isZero();
        assertEquals(isZero, false);
    }

    /**
     * Check if isZero returns false for a large non-zero value.
     */
    @Test
    public void testIsZeroFalseForLargeValue() {
        NaturalNumber n1 = this.constructorTest(98765);
        boolean isZero = n1.isZero();
        assertEquals(isZero, false);
    }
}

```