CSE 2231 – Software 2: Software Development and Design

Professor: Rob LaTour

Project #2: NaturalNumber Implementation on String

The Ohio State University

College of Engineering

Columbus, Ohio

```
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 Danny Kan (kan.74@osu.edu)
 * @author Jatin Mamtani (mamtani.6@osu.edu)
 *
 */
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);

/*
 * Complete and Systematic Test Cases:
 */

/**
 * Testing the no-argument constructor.
 */
@Test
public final void testNoArgumentConstructor() {
    NaturalNumber nActual = this.constructorTest();
    NaturalNumber nExpected = this.constructorRef();
    assertEquals(nExpected, nActual);
}

/**
 * Testing the integer constructor using the minimum {@code Integer} value
 * of 0.
 */
@Test
public final void testIntegerConstructor0() {
    NaturalNumber nActual = this.constructorTest(0);
    NaturalNumber nExpected = this.constructorRef(0);
    assertEquals(nExpected, nActual);
```

```
}

/**
 * Testing the integer constructor using the {@code Integer} value of 32.
 */
@Test
public final void testIntegerConstructor32() {
    NaturalNumber nActual = this.constructorTest(32);
    NaturalNumber nExpected = this.constructorRef(32);
    assertEquals(nExpected, nActual);
}

/**
 * Testing the integer constructor using the {@code Integer} value of 100.
 */
@Test
public final void testIntegerConstructor100() {
    NaturalNumber nActual = this.constructorTest(100);
    NaturalNumber nExpected = this.constructorRef(100);
    assertEquals(nExpected, nActual);
}

/**
 * Testing the integer constructor using the maximum {@code Integer} value
 * of Integer.MAX_VALUE = 2147483647.
 */
@Test
public final void testIntegerConstructorMaxValue() {
    NaturalNumber nActual = this.constructorTest(Integer.MAX_VALUE);
    NaturalNumber nExpected = this.constructorRef(Integer.MAX_VALUE);
    assertEquals(nExpected, nActual);
}
```

```
/**
 * Testing the string constructor using the {@code String} value of 0.
 */
@Test
public final void testStringConstructor0() {
    NaturalNumber nActual = this.constructorTest("0");
    NaturalNumber nExpected = this.constructorRef("0");
    assertEquals(nExpected, nActual);
}

/**
 * Testing the string constructor using the {@code String} value of 32.
 */
@Test
public final void testStringConstructor32() {
    NaturalNumber nActual = this.constructorTest("32");
    NaturalNumber nExpected = this.constructorRef("32");
    assertEquals(nExpected, nActual);
}

/**
 * Testing the string constructor using the {@code String} value of 100.
 */
@Test
public final void testStringConstructor100() {
    NaturalNumber nActual = this.constructorTest("100");
    NaturalNumber nExpected = this.constructorRef("100");
    assertEquals(nExpected, nActual);
}

/**
```

```
 * Testing the string constructor using the {@code String} value of
 * 9999999999.
 */
@Test
public final void testStringConstructor9999999999() {
    NaturalNumber nActual = this.constructorTest("9999999999");
    NaturalNumber nExpected = this.constructorRef("9999999999");
    assertEquals(nExpected, nActual);
}

/**
 * Testing the natural number constructor using the {@code NaturalNumber}
 * value of 0.
 */
@Test
public final void testNaturalNumberConstructor0() {
    NaturalNumber nActual = this.constructorTest(new NaturalNumber1L(0));
    NaturalNumber nExpected = this.constructorRef(new NaturalNumber1L(0));
    assertEquals(nExpected, nActual);
}

/**
 * Testing the natural number constructor using the {@code NaturalNumber}
 * value of 32.
 */
@Test
public final void testNaturalNumberConstructor32() {
    NaturalNumber nActual = this.constructorTest(new NaturalNumber1L(32));
    NaturalNumber nExpected = this.constructorRef(new NaturalNumber1L(32));
    assertEquals(nExpected, nActual);
}
```

```
/**
 * Testing the natural number constructor using the {@code NaturalNumber}
 * value of 100.
 */
@Test
public final void testNaturalNumberConstructor100() {
    NaturalNumber nActual = this.constructorTest(new NaturalNumber1L(100));
    NaturalNumber nExpected = this.constructorRef(new NaturalNumber1L(100));
    assertEquals(nExpected, nActual);
}

/**
 * Testing the natural number constructor using the {@code NaturalNumber}
 * value of 9999999999.
 */
@Test
public final void testNaturalNumberConstructor9999999999() {
    NaturalNumber nActual = this
            .constructorTest(new NaturalNumber1L("9999999999"));
    NaturalNumber nExpected = this
            .constructorRef(new NaturalNumber1L("9999999999"));
    assertEquals(nExpected, nActual);
}

/*
 * Testing .multiplyBy10() in this section:
 */

/**
 * Testing .multiplyBy10(0) on an initial {@code NaturalNumber} value of 0,
 * resulting in the {@code NaturalNumber} value of 0.
 */
```

```
@Test
public final void testMultiplyBy10Using0On0() {
    NaturalNumber nActual = this.constructorTest(0);
    nActual.multiplyBy10(0);
    NaturalNumber nExpected = this.constructorRef(0);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .multiplyBy10(3) on an initial {@code NaturalNumber} value of 0,
 * resulting in the {@code NaturalNumber} value of 3.
 */
@Test
public final void testMultiplyBy10Using3On0() {
    NaturalNumber nActual = this.constructorTest(0);
    nActual.multiplyBy10(3);
    NaturalNumber nExpected = this.constructorRef(3);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .multiplyBy10(9) on an initial {@code NaturalNumber} value of 0,
 * resulting in the {@code NaturalNumber} value of 9.
 */
@Test
public final void testMultiplyBy10Using9On0() {
    NaturalNumber nActual = this.constructorTest(0);
    nActual.multiplyBy10(9);
    NaturalNumber nExpected = this.constructorRef(9);
    assertEquals(nExpected, nActual);
}
```

```
/**
 * Testing .multiplyBy10(0) on an initial {@code NaturalNumber} value of 1,
 * resulting in the {@code NaturalNumber} value of 10.
 */
@Test
public final void testMultiplyBy10Using0On1() {
    NaturalNumber nActual = this.constructorTest(1);
    nActual.multiplyBy10(0);
    NaturalNumber nExpected = this.constructorRef(10);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .multiplyBy10(0) on an initial {@code NaturalNumber} value of 32,
 * resulting in the {@code NaturalNumber} value of 320.
 */
@Test
public final void testMultiplyBy10Using0On32() {
    NaturalNumber nActual = this.constructorTest(32);
    nActual.multiplyBy10(0);
    NaturalNumber nExpected = this.constructorRef(320);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .multiplyBy10(9) on an initial {@code NaturalNumber} value of 99,
 * resulting in the {@code NaturalNumber} value of 999.
 */
@Test
public final void testMultiplyBy10Using9On99() {
    NaturalNumber nActual = this.constructorTest(99);
    nActual.multiplyBy10(9);
```

```
    NaturalNumber nExpected = this.constructorRef(999);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .multiplyBy10(3) on an initial {@code NaturalNumber} value of 12,
 * resulting in the {@code NaturalNumber} value of 123.
 */
@Test
public final void testMultiplyBy10Using3On12() {
    NaturalNumber nActual = this.constructorTest(12);
    nActual.multiplyBy10(3);
    NaturalNumber nExpected = this.constructorRef(123);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .multiplyBy10(0) once on an initial {@code NaturalNumber} value
 * of 10, resulting in the {@code NaturalNumber} value of 100.
 */
@Test
public final void testMultiplyBy10Using0On10Once() {
    NaturalNumber nActual = this.constructorTest("10");
    nActual.multiplyBy10(0);
    NaturalNumber nExpected = this.constructorRef("100");
    assertEquals(nExpected, nActual);
}

/**
 * Testing .multiplyBy10(0) twice on an initial {@code NaturalNumber} value
 * of 10, resulting in the {@code NaturalNumber} value of 1000.
 */
```

```
@Test
public final void testMultiplyBy10Using0On10Twice() {
    NaturalNumber nActual = this.constructorTest("10");
    nActual.multiplyBy10(0);
    nActual.multiplyBy10(0);
    NaturalNumber nExpected = this.constructorRef("1000");
    assertEquals(nExpected, nActual);
}

/*
 * Testing .divideBy10() in this section:
 */

/**
 * Testing .divideBy10() on an initial {@code NaturalNumber} value of 0,
 * resulting in the {@code NaturalNumber} value of 0.
 */
@Test
public final void testDivideBy10On0() {
    NaturalNumber nActual = this.constructorTest(0);
    NaturalNumber nExpected = this.constructorRef(0);
    int lastDigit = nActual.divideBy10();
    assertEquals(0, lastDigit);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .divideBy10() on an initial {@code NaturalNumber} value of 9,
 * resulting in the {@code NaturalNumber} value of 0.
 */
@Test
public final void testDivideBy10On9() {
```

```
    NaturalNumber nActual = this.constructorTest(9);
    NaturalNumber nExpected = this.constructorRef(0);
    int lastDigit = nActual.divideBy10();
    assertEquals(9, lastDigit);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .divideBy10() on an initial {@code NaturalNumber} value of 32,
 * resulting in the {@code NaturalNumber} value of 3.
 */
@Test
public final void testDivideBy10On32() {
    NaturalNumber nActual = this.constructorTest(32);
    NaturalNumber nExpected = this.constructorRef(3);
    int lastDigit = nActual.divideBy10();
    assertEquals(2, lastDigit);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .divideBy10() on an initial {@code NaturalNumber} value of 999,
 * resulting in the {@code NaturalNumber} value of 99.
 */
@Test
public final void testDivideBy10On999() {
    NaturalNumber nActual = this.constructorTest(999);
    NaturalNumber nExpected = this.constructorRef(99);
    int lastDigit = nActual.divideBy10();
    assertEquals(9, lastDigit);
    assertEquals(nExpected, nActual);
}
```

```
/**
 * Testing .divideBy10() once on an initial {@code NaturalNumber} value of
 * 3680, resulting in the {@code NaturalNumber} value of 368.
 */
@Test
public final void testDivideBy10On3680Once() {
    NaturalNumber nActual = this.constructorTest(3680);
    NaturalNumber nExpected = this.constructorRef(368);
    int lastDigit = nActual.divideBy10();
    assertEquals(0, lastDigit);
    assertEquals(nExpected, nActual);
}

/**
 * Testing .divideBy10() twice on an initial {@code NaturalNumber} value of
 * 3680, resulting in the {@code NaturalNumber} value of 36.
 */
@Test
public final void testDivideBy10On3680Twice() {
    NaturalNumber nActual = this.constructorTest(3680);
    NaturalNumber nExpected = this.constructorRef(36);
    int lastDigit1 = nActual.divideBy10();
    int lastDigit2 = nActual.divideBy10();
    assertEquals(0, lastDigit1);
    assertEquals(8, lastDigit2);
    assertEquals(nExpected, nActual);
}

/*
 * Testing .isZero() in this section:
 */
```

```
/**
 * Testing .isZero() on the no-argument constructor.
 */
@Test
public final void testIsZeroNoArgumentConstructor() {
    NaturalNumber nActual = this.constructorTest();
    NaturalNumber nExpected = this.constructorRef();
    assertEquals(true, nActual.isZero());
    assertEquals(nExpected, nActual);
}

/**
 * Testing .isZero() on the integer constructor using {@code Integer} value
 * of 0.
 */
@Test
public final void testIsZeroIntegerConstructor() {
    NaturalNumber nActual = this.constructorTest(0);
    NaturalNumber nExpected = this.constructorRef(0);
    assertEquals(true, nActual.isZero());
    assertEquals(nExpected, nActual);
}

/**
 * Testing .isZero() on the string constructor using {@code String} value of
 * 0.
 */
@Test
public final void testIsZeroStringConstructor() {
    NaturalNumber nActual = this.constructorTest("0");
    NaturalNumber nExpected = this.constructorRef("0");
```

```
    assertEquals(true, nActual.isZero());
    assertEquals(nExpected, nActual);
}

/**
 * Testing .isZero() on the natural number constructor using
 * {@code NaturalNumber} value of 0.
 */
@Test
public final void testIsZeroNaturalNumberConstructor() {
    NaturalNumber nActual = this.constructorTest(new NaturalNumber1L());
    NaturalNumber nExpected = this.constructorRef(new NaturalNumber1L());
    assertEquals(true, nActual.isZero());
    assertEquals(nExpected, nActual);
}

/**
 * Testing .isZero() on the integer constructor using {@code Integer} value
 * of 32.
 */
@Test
public final void testIsZeroOn32UsingIntegerConstructor() {
    NaturalNumber nActual = this.constructorTest(32);
    NaturalNumber nExpected = this.constructorRef(32);
    assertEquals(false, nActual.isZero());
    assertEquals(nExpected, nActual);
}

/**
 * Testing .isZero() on the string constructor using {@code String} value of
 * 32.
 */
```

```
    @Test
    public final void testIsZeroOn32UsingStringConstructor() {
        NaturalNumber nActual = this.constructorTest("32");
        NaturalNumber nExpected = this.constructorRef("32");
        assertEquals(false, nActual.isZero());
        assertEquals(nExpected, nActual);
    }

    /**
     * Testing .isZero() on the natural number constructor using
     * {@code NaturalNumber} value of 32.
     */
    @Test
    public final void testIsZeroOn32UsingNaturalNumberConstructor() {
        NaturalNumber nActual = this.constructorTest(new NaturalNumber1L(32));
        NaturalNumber nExpected = this.constructorRef(new NaturalNumber1L(32));
        assertEquals(false, nActual.isZero());
        assertEquals(nExpected, nActual);
    }

}
```