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

/**
 * JUnit test fixture for {@code NaturalNumber}'s constructors and kernel
 * methods.
 *
 * @author Put your name here
 *
 */
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
```

```
62      * reference implementation and returns the result.
63      *
64      * @return the new number
65      * @ensures constructorRef = 0
66      */
67     protected abstract NaturalNumber constructorRef();
68
69     /**
70      * Invokes the appropriate {@code NaturalNumber} constructor for the
71      * reference implementation and returns the result.
72      *
73      * @param i
74      *            {@code int} to initialize from
75      * @return the new number
76      * @requires i >= 0
77      * @ensures constructorRef = i
78      */
79     protected abstract NaturalNumber constructorRef(int i);
80
81     /**
82      * Invokes the appropriate {@code NaturalNumber} constructor for the
83      * reference implementation and returns the result.
84      *
85      * @param s
86      *            {@code String} to initialize from
87      * @return the new number
88      * @requires there exists n: NATURAL (s = TO_STRING(n))
89      * @ensures s = TO_STRING(constructorRef)
90      */
91     protected abstract NaturalNumber constructorRef(String s);
92
93     /**
94      * Invokes the appropriate {@code NaturalNumber} constructor for the
95      * reference implementation and returns the result.
96      *
97      * @param n
98      *            {@code NaturalNumber} to initialize from
99      * @return the new number
100     * @ensures constructorRef = n
101     */
102    protected abstract NaturalNumber constructorRef(NaturalNumber n);
103
104    /*
105     * START OF CONSTRUCTOR TEST CASES
106     */
107
108    /**
109     * Test for no argument constructor case.
110     */
111    @Test
112    public final void noConstructorTest() {
113        // Setup
114        NaturalNumber n = this.constructorTest();
115        NaturalNumber nExpected = this.constructorRef();
116        // Call
117        // Eval
118        assertEquals(nExpected, n);
```

```
119     }
120
121     /**
122      * Test for int 0 case.
123      */
124     @Test
125     public final void intZeroTest() {
126         // Setup
127         NaturalNumber n = this.constructorTest(0);
128         NaturalNumber nExpected = this.constructorRef(0);
129         // Call
130         // Eval
131         assertEquals(nExpected, n);
132     }
133
134     /**
135      * Test for single digit int case.
136      */
137     @Test
138     public final void intSingleTest() {
139         // Setup
140         NaturalNumber n = this.constructorTest(9);
141         NaturalNumber nExpected = this.constructorRef(9);
142         // Call
143         // Eval
144         assertEquals(nExpected, n);
145     }
146
147     /**
148      * Test for double digit int case.
149      */
150     @Test
151     public final void intDoubleTest() {
152         // Setup
153         NaturalNumber n = this.constructorTest(92);
154         NaturalNumber nExpected = this.constructorRef(92);
155         // Call
156         // Eval
157         assertEquals(nExpected, n);
158     }
159
160     /**
161      * Test for string "0" case.
162      */
163     @Test
164     public final void stringZeroTest() {
165         // Setup
166         NaturalNumber n = this.constructorTest("0");
167         NaturalNumber nExpected = this.constructorRef("0");
168         // Call
169         // Eval
170         assertEquals(nExpected, n);
171     }
172
173     /**
174      * Test for typical string case; no leading zeros.
175      */
```

```java
    @Test
    public final void stringTypicalTest() {
        // Setup
        NaturalNumber n = this.constructorTest("340");
        NaturalNumber nExpected = this.constructorRef("340");
        // Call
        // Eval
        assertEquals(nExpected, n);
    }

    /**
     * Test for NN 0 case.
     */
    @Test
    public final void nnZeroTest() {
        // Setup
        NaturalNumber x = this.constructorRef(0);
        NaturalNumber n = this.constructorTest(x);
        NaturalNumber nExpected = this.constructorRef(x);
        // Call
        // Eval
        assertEquals(nExpected, n);
    }

    /**
     * Test for single digit NN case.
     */
    @Test
    public final void nnSingleTest() {
        // Setup
        NaturalNumber x = this.constructorRef(6);
        NaturalNumber n = this.constructorTest(x);
        NaturalNumber nExpected = this.constructorRef(x);
        // Call
        // Eval
        assertEquals(nExpected, n);
    }

    /**
     * Test for double digit NN case.
     */
    @Test
    public final void nnDoubleTest() {
        // Setup
        NaturalNumber x = this.constructorRef(42);
        NaturalNumber n = this.constructorTest(x);
        NaturalNumber nExpected = this.constructorRef(x);
        // Call
        // Eval
        assertEquals(nExpected, n);
    }

    /*
     * START OF KERNEL METHOD TEST CASES
     */

    // MultiplyBy10 ------------------------------------------------------------
```

```
233
234     /**
235      * Testing n.multiplyBy10(0); n == 0.
236      */
237     @Test
238     public final void testMultipling0Adding0() {
239         // Setup
240         NaturalNumber n = this.constructorTest("0");
241         NaturalNumber nExpected = this.constructorRef("0");
242         // Call
243         n.multiplyBy10(0);
244         // Eval
245         assertEquals(nExpected.toString(), n.toString()); // <- works
246         //assertEquals(true, nExpected.equals(n)); < -- does not work
247         //assertEquals(nExpected, n); <-- does not work
248         // if assertEquals/ .equals(); has the same behavior as ==,
249         // then it will not work as objA == objB compares reference values,
250         // I do not know how to get around this check without knowing how it
251         // checks. If it uses .equals(); then without knowing how it compares
252         // I dont know how to get around this check. I have checked the
253         // implementation of NaturalNumber2 and there doesnt seem to be any
254         // mismatch with our. Both implementation's kernel methods have
255         // the same return values for the same n == 0.
256     }
257
258     /**
259      * Testing n.multiplyBy10(1); n == 0.
260      */
261     @Test
262     public final void testMultipling0Adding1() {
263         // Setup
264         NaturalNumber n = this.constructorTest("0");
265         NaturalNumber nExpected = this.constructorRef("1");
266         // Call
267         n.multiplyBy10(1);
268         // Eval
269         assertEquals(nExpected, n);
270     }
271
272     /**
273      * Testing n.multiplyBy10(0); n == 1.
274      */
275     @Test
276     public final void testMultipling1Adding0() {
277         // Setup
278         NaturalNumber n = this.constructorTest("1");
279         NaturalNumber nExpected = this.constructorRef("10");
280         // Call
281         n.multiplyBy10(0);
282         // Eval
283         assertEquals(nExpected, n);
284     }
285
286     /**
287      * Testing n.multiplyBy10(1); n == 1.
288      */
289     @Test
```

```
290     public final void testMultipling1Adding1() {
291         // Setup
292         NaturalNumber n = this.constructorTest("1");
293         NaturalNumber nExpected = this.constructorRef("11");
294         // Call
295         n.multiplyBy10(1);
296         // Eval
297         assertEquals(nExpected, n);
298     }
299
300     /**
301      * Testing n.multiplyBy10(0); n == Integer.MAX.
302      */
303     @Test
304     public final void testMultiplingINTMAXAdding0() {
305         // Setup
306         NaturalNumber n = this.constructorTest(Integer.MAX_VALUE);
307         NaturalNumber nExpected = this
308                 .constructorRef(Integer.toString(Integer.MAX_VALUE) + "0");
309         // Call
310         n.multiplyBy10(0);
311         // Eval
312         assertEquals(nExpected, n);
313     }
314
315     // DivideBy10 -----------------------------------------------------------
316
317     /**
318      * Testing n.divideBy10(); n == 0.
319      */
320     @Test
321     public final void testDividing0Return0() {
322         // Setup
323         NaturalNumber n = this.constructorTest("0");
324         NaturalNumber nExpected = this.constructorRef("0");
325         int remainder;
326         // Call
327         remainder = n.divideBy10();
328         // Eval
329         assertEquals(0, remainder);
330         assertEquals(nExpected, n);
331     }
332
333     /**
334      * Testing n.divideBy10(); n == 1.
335      */
336     @Test
337     public final void testDividing1Return1() {
338         // Setup
339         NaturalNumber n = this.constructorTest("1");
340         NaturalNumber nExpected = this.constructorRef("0");
341         int remainder;
342         // Call
343         remainder = n.divideBy10();
344         // Eval
345         assertEquals(1, remainder);
346         assertEquals(nExpected, n);
```

```
347     }
348
349     /**
350      * Testing n.divideBy10(); n == 10.
351      */
352     @Test
353     public final void testDividing10Return0() {
354         // Setup
355         NaturalNumber n = this.constructorTest("10");
356         NaturalNumber nExpected = this.constructorRef("1");
357         int remainder;
358         // Call
359         remainder = n.divideBy10();
360         // Eval
361         assertEquals(0, remainder);
362         assertEquals(nExpected, n);
363     }
364
365     /**
366      * Testing n.divideBy10(); n == 11.
367      */
368     @Test
369     public final void testDividing11Return1() {
370         // Setup
371         NaturalNumber n = this.constructorTest("11");
372         NaturalNumber nExpected = this.constructorRef("1");
373         int remainder;
374         // Call
375         remainder = n.divideBy10();
376         // Eval
377         assertEquals(1, remainder);
378         assertEquals(nExpected, n);
379     }
380
381     // IsZero ----------------------------------------------------------------
382     /**
383      * Testing n.isZero();.
384      */
385     @Test
386     public final void testIsZeroTrue() {
387         // Setup
388         NaturalNumber n = this.constructorTest();
389         NaturalNumber nExpected = this.constructorRef();
390         // Call
391         // Eval
392         assertEquals(true, n.isZero());
393         assertEquals(nExpected, n);
394     }
395
396     /**
397      * Testing n.isZero();.
398      */
399     @Test
400     public final void testIsZeroFalse() {
401         // Setup
402         NaturalNumber n = this.constructorTest("345");
403         NaturalNumber nExpected = this.constructorRef("345");
```

```
404         // Call
405         // Eval
406         assertEquals(false, n.isZero());
407         assertEquals(nExpected, n);
408     }
409 }
410
```