

```
1 import static org.junit.Assert.assertEquals;
2
3 import org.junit.Test;
4
5 import components.naturalnumber.NaturalNumber;
6 import components.naturalnumber.NaturalNumber1L;
7
8 /**
9  * JUnit test fixture for {@code NaturalNumber}'s constructors and kernel
10 * methods.
11 *
12 * @author David P. and Zach B.
13 *
14 */
15 public abstract class NaturalNumberTest {
16
17     /**
18      * Invokes the appropriate {@code NaturalNumber} constructor for the
19      * implementation under test and returns the result.
20      *
21      * @return the new number
22      * @ensures constructorTest = 0
23      */
24     protected abstract NaturalNumber constructorTest();
25
26     /**
27      * Invokes the appropriate {@code NaturalNumber} constructor for the
28      * implementation under test and returns the result.
29      *
30      * @param i
31      *         {@code int} to initialize from
32      * @return the new number
33      * @requires i >= 0
34      * @ensures constructorTest = i
35      */
36     protected abstract NaturalNumber constructorTest(int i);
37
38     /**
39      * Invokes the appropriate {@code NaturalNumber} constructor for the
40      * implementation under test and returns the result.
41      *
42      * @param s
43      *         {@code String} to initialize from
44      * @return the new number
45      * @requires there exists n: NATURAL (s = TO_STRING(n))
46      * @ensures s = TO_STRING(constructorTest)
47      */
48     protected abstract NaturalNumber constructorTest(String s);
49
50     /**
51      * Invokes the appropriate {@code NaturalNumber} constructor for the
52      * implementation under test and returns the result.
53      *
54      * @param n
55      *         {@code NaturalNumber} to initialize from
56      * @return the new number
57      * @ensures constructorTest = n
```

```

58     */
59     protected abstract NaturalNumber constructorTest(NaturalNumber n);
60
61     /**
62      * Invokes the appropriate {@code NaturalNumber} constructor for the
63      * reference implementation and returns the result.
64      *
65      * @return the new number
66      * @ensures constructorRef = 0
67      */
68     protected abstract NaturalNumber constructorRef();
69
70     /**
71      * Invokes the appropriate {@code NaturalNumber} constructor for the
72      * reference implementation and returns the result.
73      *
74      * @param i
75      *        {@code int} to initialize from
76      * @return the new number
77      * @requires i >= 0
78      * @ensures constructorRef = i
79      */
80     protected abstract NaturalNumber constructorRef(int i);
81
82     /**
83      * Invokes the appropriate {@code NaturalNumber} constructor for the
84      * reference implementation and returns the result.
85      *
86      * @param s
87      *        {@code String} to initialize from
88      * @return the new number
89      * @requires there exists n: NATURAL (s = TO_STRING(n))
90      * @ensures s = TO_STRING(constructorRef)
91      */
92     protected abstract NaturalNumber constructorRef(String s);
93
94     /**
95      * Invokes the appropriate {@code NaturalNumber} constructor for the
96      * reference implementation and returns the result.
97      *
98      * @param n
99      *        {@code NaturalNumber} to initialize from
100     * @return the new number
101     * @ensures constructorRef = n
102     */
103     protected abstract NaturalNumber constructorRef(NaturalNumber n);
104
105     /**
106      * Verify no-argument constructor initializes to zero.
107      */
108     @Test
109     public void testNoArgumentConstructor() {
110         NaturalNumber n1 = this.constructorTest();
111
112         NaturalNumber n2 = this.constructorRef();
113
114         assertEquals(n1, n2);

```

```
115     }
116
117     /**
118      * Check if constructor with int 0 initializes correctly.
119      */
120     @Test
121     public void testConstructorWithIntZero() {
122         int value = 0;
123         NaturalNumber n1 = this.constructorTest(value);
124         NaturalNumber n2 = this.constructorRef(value);
125         assertEquals(n1, n2);
126     }
127
128     /**
129      * Check if constructor with small int initializes correctly.
130      */
131     @Test
132     public void testConstructorWithSmallInt() {
133         int value = 17;
134         NaturalNumber n1 = this.constructorTest(value);
135         NaturalNumber n2 = this.constructorRef(value);
136         assertEquals(n1, n2);
137     }
138
139     /**
140      * Check if constructor with large int initializes correctly.
141      */
142     @Test
143     public void testConstructorWithLargeInt() {
144         int value = 987654321;
145         NaturalNumber n1 = this.constructorTest(value);
146         NaturalNumber n2 = this.constructorRef(value);
147         assertEquals(n1, n2);
148     }
149
150     /**
151      * Check if constructor with string "0" initializes correctly.
152      */
153     @Test
154     public void testConstructorWithStringZero() {
155         String value = "0";
156         NaturalNumber n1 = this.constructorTest(value);
157         NaturalNumber n2 = this.constructorRef(value);
158         assertEquals(n1, n2);
159     }
160
161     /**
162      * Check if constructor with small string initializes correctly.
163      */
164     @Test
165     public void testConstructorWithSmallString() {
166         String value = "456";
167         NaturalNumber n1 = this.constructorTest(value);
168         NaturalNumber n2 = this.constructorRef(value);
169         assertEquals(n1, n2);
170     }
171
```

```
172  /**
173   * Check if constructor with large string initializes correctly.
174   */
175  @Test
176  public void testConstructorWithLargeString() {
177      String value = "1234567890123456789";
178      NaturalNumber n1 = this.constructorTest(value);
179      NaturalNumber n2 = this.constructorRef(value);
180      assertEquals(n1, n2);
181  }
182
183  /**
184   * Verify constructor from NaturalNumber.
185   */
186  @Test
187  public void testConstructorFromNaturalNumber() {
188      NaturalNumber source = this.constructorRef(100);
189      NaturalNumber n1 = this.constructorTest(source);
190      NaturalNumber n2 = this.constructorRef(100);
191      assertEquals(n1, n2);
192  }
193
194  /**
195   * Verify constructor from NaturalNumber with a small value.
196   */
197  @Test
198  public void testConstructorFromSmallNaturalNumber() {
199      NaturalNumber source = this.constructorRef(35);
200      NaturalNumber n1 = this.constructorTest(source);
201      NaturalNumber n2 = this.constructorRef(35);
202      assertEquals(n1, n2);
203  }
204
205  /**
206   * Verify constructor from NaturalNumber with a large value.
207   */
208  @Test
209  public void testConstructorFromLargeNaturalNumber() {
210      NaturalNumber source = this.constructorRef(54321);
211      NaturalNumber n1 = this.constructorTest(source);
212      NaturalNumber n2 = this.constructorRef(54321);
213      assertEquals(n1, n2);
214  }
215
216  /**
217   * Verify constructor from NaturalNumber from zero on different
218   * implementation.
219   */
220  @Test
221  public void testConstructorFromZeroFromNaturalNumber1L() {
222      NaturalNumber1L temp = new NaturalNumber1L(0);
223
224      NaturalNumber n1 = this.constructorTest(temp);
225      NaturalNumber n2 = this.constructorRef(temp);
226      assertEquals(n1, n2);
227  }
228
```

```
229  /**
230   * Verify constructor from NaturalNumber with different implementation.
231   */
232  @Test
233  public void testConstructorFromNaturalNumber1L() {
234      NaturalNumber1L temp = new NaturalNumber1L(25);
235
236      NaturalNumber n1 = this.constructorTest(temp);
237      NaturalNumber n2 = this.constructorRef(temp);
238      assertEquals(n1, n2);
239  }
240
241  /**
242   * Test multiplyBy10 with zero.
243   */
244  @Test
245  public void testMultiplyBy10WithZero() {
246      int digit = 0;
247      NaturalNumber n1 = this.constructorTest();
248      NaturalNumber n2 = this.constructorRef(digit);
249      n1.multiplyBy10(digit);
250      assertEquals(n1, n2);
251  }
252
253  /**
254   * Test multiplyBy10 with a small number.
255   */
256  @Test
257  public void testMultiplyBy10WithSmallNumber() {
258      int expectedValue = 52;
259      NaturalNumber n1 = this.constructorTest(5);
260      NaturalNumber n2 = this.constructorRef(expectedValue);
261      n1.multiplyBy10(2);
262      assertEquals(n1, n2);
263  }
264
265  /**
266   * Test multiplyBy10 with a large number.
267   */
268  @Test
269  public void testMultiplyBy10WithLargeNumber() {
270      int number = 45;
271      int expectedValue = 453;
272      NaturalNumber n1 = this.constructorTest(number);
273      NaturalNumber n2 = this.constructorRef(expectedValue);
274      n1.multiplyBy10(3);
275      assertEquals(n1, n2);
276  }
277
278  /**
279   * Test multiplyBy10 with multiple of 10/100.
280   */
281  @Test
282  public void testMultiplyBy10With100() {
283      int number = 10;
284      int expectedValue = 100;
285      NaturalNumber n1 = this.constructorTest(number);
```

```
286     NaturalNumber n2 = this.constructorRef(expectedValue);
287     n1.multiplyBy10(0);
288     assertEquals(n1, n2);
289 }
290
291 /**
292  * TEST multiplyBy10 with zero and two.
293  */
294 @Test
295 public void testMultiplyBy10WithZeroAndTwo() {
296     int number = 0;
297     int expectedValue = 2;
298     NaturalNumber n1 = this.constructorTest(number);
299     NaturalNumber n2 = this.constructorRef(expectedValue);
300     n1.multiplyBy10(2);
301     assertEquals(n1, n2);
302 }
303
304 /**
305  * Test divideBy10 with a small number.
306  */
307 @Test
308 public void testDivideBy10WithSmallNumber() {
309     int number = 36;
310     int expectedQuotient = 3;
311     int expectedRemainder = 6;
312     NaturalNumber n1 = this.constructorTest(number);
313     NaturalNumber n2 = this.constructorRef(expectedQuotient);
314     int remainder = n1.divideBy10();
315     assertEquals(remainder, expectedRemainder);
316     assertEquals(n1, n2);
317 }
318
319 /**
320  * Test divideBy10 with a large number.
321  */
322 @Test
323 public void testDivideBy10WithLargeNumber() {
324     int number = 78912;
325     int expectedQuotient = 7891;
326     int expectedRemainder = 2;
327     NaturalNumber n1 = this.constructorTest(number);
328     NaturalNumber n2 = this.constructorRef(expectedQuotient);
329     int remainder = n1.divideBy10();
330     assertEquals(remainder, expectedRemainder);
331     assertEquals(n1, n2);
332 }
333
334 /**
335  * Test divideBy10 with a single digit.
336  */
337 @Test
338 public void testDivideBy10WithSingleDigit() {
339     int number = 8;
340     int expectedQuotient = 0;
341     int expectedRemainder = 8;
342     NaturalNumber n1 = this.constructorTest(number);
```

```
343     NaturalNumber n2 = this.constructorRef(expectedQuotient);
344     int remainder = n1.divideBy10();
345     assertEquals(remainder, expectedRemainder);
346     assertEquals(n1, n2);
347 }
348
349 /**
350  * Test divideBy10 with zero.
351  */
352 @Test
353 public void testDivideBy10WithZero() {
354     int number = 0;
355     int expectedQuotient = 0;
356     int expectedRemainder = 0;
357     NaturalNumber n1 = this.constructorTest(number);
358     NaturalNumber n2 = this.constructorRef(expectedQuotient);
359     int remainder = n1.divideBy10();
360     assertEquals(remainder, expectedRemainder);
361     assertEquals(n1, n2);
362 }
363
364 /**
365  * Check if isZero returns true for the default value.
366  */
367 @Test
368 public void testIsZeroTrueForDefault() {
369     NaturalNumber n1 = this.constructorTest();
370     boolean isZero = n1.isZero();
371     assertEquals(isZero, true);
372 }
373
374 /**
375  * Check if isZero returns true for zero value.
376  */
377 @Test
378 public void testIsZeroTrueForZeroValue() {
379     NaturalNumber n1 = this.constructorTest(0);
380     boolean isZero = n1.isZero();
381     assertEquals(isZero, true);
382 }
383
384 /**
385  * Check if isZero returns false for a small non-zero value.
386  */
387 @Test
388 public void testIsZeroFalseForSmallValue() {
389     NaturalNumber n1 = this.constructorTest(8);
390     boolean isZero = n1.isZero();
391     assertEquals(isZero, false);
392 }
393
394 /**
395  * Check if isZero returns false for a large non-zero value.
396  */
397 @Test
398 public void testIsZeroFalseForLargeValue() {
399     NaturalNumber n1 = this.constructorTest(98765);
```

```
400         boolean isZero = n1.isZero();
401         assertEquals(isZero, false);
402     }
403 }
404
```