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

import org.junit.Test;

import components.naturalnumber.NaturalNumber;

/**
 * JUnit test fixture for {@code NaturalNumber}'s constructors and kernel
 * methods.
 *
 * @author Chloe Feller and Krish Patel
 *
 */
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);

```

```
60      /**
61       * 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     // TODO - add test cases for four constructors, multiplyBy10, divideBy10, isZero
105
106     /**
107      * No-argument constructor.
108      */
109     @Test
110     public void testNoArgumentConstructor() {
111
112         NaturalNumber q = this.constructorTest();
113         NaturalNumber qExpected = this.constructorRef();
114
115         assertEquals(qExpected, q);
116     }
117
118     /**
```

```
119      * Constructor from {@code int}.
120      */
121     @Test
122     public void testNoArgumentConstructorInt() {
123         NaturalNumber q = this.constructorRef(0);
124         NaturalNumber qExpected = this.constructorTest(0);
125
126         assertEquals(qExpected, q);
127     }
128
129     /**
130      * Constructor from {@code int}.
131      */
132     @Test
133     public void testNoArgumentConstructorIntWithOneDigit() {
134         NaturalNumber q = this.constructorRef(5);
135         NaturalNumber qExpected = this.constructorTest(5);
136
137         assertEquals(qExpected, q);
138     }
139
140     /**
141      * Constructor from {@code int}.
142      */
143     @Test
144     public void testNoArgumentConstructorIntWithTwoDigits() {
145         NaturalNumber q = this.constructorRef(42);
146         NaturalNumber qExpected = this.constructorTest(42);
147
148         assertEquals(qExpected, q);
149     }
150
151     /**
152      * Constructor from {@code int}.
153      */
154     @Test
155     public void testNoArgumentConstructorIntWithMaxDigits() {
156         NaturalNumber q = this.constructorRef(Integer.MAX_VALUE);
157         NaturalNumber qExpected = this.constructorTest(Integer.MAX_VALUE);
158
159         assertEquals(qExpected, q);
160     }
161
162     /**
163      * Constructor from {@code String}.
164      */
165     @Test
166     public void testNoArgumentConstructorStringZeroString() {
167         NaturalNumber q = this.constructorRef("0");
168         NaturalNumber qExpected = this.constructorTest("0");
169
170         assertEquals(qExpected, q);
171     }
172
173     /**
174      * Constructor from {@code String}.
175      */
176     @Test
177     public void testNoArgumentConstructorWithOneDigitString() {
```

```
178         NaturalNumber q = this.constructorRef("4");
179         NaturalNumber qExpected = this.constructorTest("4");
180
181         assertEquals(qExpected, q);
182     }
183
184     /**
185      * Constructor from {@code String}.
186      */
187     @Test
188     public void testNoArgumentConstructorWithTwoDigitsString() {
189         NaturalNumber q = this.constructorRef("98");
190         NaturalNumber qExpected = this.constructorTest("98");
191
192         assertEquals(qExpected, q);
193     }
194
195     /**
196      * Constructor from {@code NaturalNumber}.
197      */
198     @Test
199     public void testNoArgumentConstructorNN() {
200         NaturalNumber q = this.constructorRef(this.constructorRef("0"));
201         NaturalNumber qExpected = this
202                 .constructorTest(this.constructorTest("0"));
203
204         assertEquals(qExpected, q);
205     }
206
207     /**
208      * Constructor from {@code NaturalNumber}.
209      */
210     @Test
211     public void testNoArgumentConstructorNNWithOneDigit() {
212         NaturalNumber q = this.constructorRef(this.constructorRef("7"));
213         NaturalNumber qExpected = this
214                 .constructorTest(this.constructorTest("7"));
215
216         assertEquals(qExpected, q);
217     }
218
219     /**
220      * Constructor from {@code NaturalNumber}.
221      */
222     @Test
223     public void testNoArgumentConstructorNNWithTwoDogits() {
224         NaturalNumber q = this.constructorRef(this.constructorRef("96"));
225         NaturalNumber qExpected = this
226                 .constructorTest(this.constructorTest("96"));
227
228         assertEquals(qExpected, q);
229     }
230
231     /**
232      * Constructor from {@code NaturalNumber}.
233      */
234     @Test
235     public void testNoArgumentConstructorNNWithALotOfNumbers() {
236         NaturalNumber q = this
```

```
237                 .constructorRef(this.constructorRef("98765432123456789"));
238         NaturalNumber qExpected = this
239                 .constructorTest(this.constructorTest("98765432123456789"));
240
241         assertEquals(qExpected, q);
242     }
243
244     // TODO - multipleBy10 Test Cases
245
246     /**
247      * Basic case of k = 0 and this = 0.
248      */
249     @Test
250     public void testMultiplyBy10WithZero() {
251
252         /**
253          * Variables.
254          */
255         NaturalNumber q = this.constructorTest(0);
256         NaturalNumber qExpected = this.constructorRef(0);
257
258         /**
259          * Call Method.
260          */
261         q.multiplyBy10(0);
262
263         /**
264          * Assert Values.
265          */
266         assertEquals(qExpected, q);
267     }
268
269     /**
270      * Similar to above test case, but with k = 5.
271      */
272     @Test
273     public void testMultiplyBy10WithZeroWithAddition() {
274
275         /**
276          * Variables.
277          */
278         NaturalNumber q = this.constructorTest(0);
279         NaturalNumber qExpected = this.constructorRef(0);
280
281         /**
282          * Call Method.
283          */
284         q.multiplyBy10(5);
285
286         /**
287          * Assert Values.
288          */
289         assertEquals(qExpected, q);
290     }
291
292     /**
293      * Multiply with one digit.
294      */
295     @Test
```

```
296     public void testMultiplyBy10WithOneDigit() {
297
298         /**
299          * Variables.
300          */
301         NaturalNumber q = this.constructorTest(3);
302         NaturalNumber qExpected = this.constructorRef(30);
303
304         /**
305          * Call Method.
306          */
307         q.multiplyBy10(0);
308
309         /**
310          * Assert Values.
311          */
312         assertEquals(qExpected, q);
313     }
314
315     /**
316      * Multiply with one digit with addition of 4.
317      */
318     @Test
319     public void testMultiplyBy10WithOneDigitWithAddition() {
320
321         /**
322          * Variables.
323          */
324         NaturalNumber q = this.constructorTest(3);
325         NaturalNumber qExpected = this.constructorRef(34);
326
327         /**
328          * Call Method.
329          */
330         q.multiplyBy10(4);
331
332         /**
333          * Assert Values.
334          */
335         assertEquals(qExpected, q);
336     }
337
338     /**
339      * Multiply with two digits.
340      */
341     @Test
342     public void testMultiplyBy10WithTwoDigits() {
343
344         /**
345          * Variables.
346          */
347         NaturalNumber q = this.constructorTest(38);
348         NaturalNumber qExpected = this.constructorRef(380);
349
350         /**
351          * Call Method.
352          */
353         q.multiplyBy10(0);
354
```

```
355         /**
356          * Assert Values.
357          */
358         assertEquals(qExpected, q);
359     }
360 
361     /**
362      * Multiply with two digits with addition of 9.
363      */
364     @Test
365     public void testMultiplyBy10WithTwoDigitsWithAddition() {
366 
367         /**
368          * Variables.
369          */
370         NaturalNumber q = this.constructorTest(38);
371         NaturalNumber qExpected = this.constructorRef(389);
372 
373         /**
374          * Call Method.
375          */
376         q.multiplyBy10(9);
377 
378         /**
379          * Assert Values.
380          */
381         assertEquals(qExpected, q);
382     }
383 
384     /**
385      * Multiply with multiple digits.
386      */
387     @Test
388     public void testMultiplyBy10WithMultipleDigits() {
389 
390         /**
391          * Variables.
392          */
393         NaturalNumber q = this.constructorTest(98365492);
394         NaturalNumber qExpected = this.constructorRef(983654920);
395 
396         /**
397          * Call Method.
398          */
399         q.multiplyBy10(0);
400 
401         /**
402          * Assert Values.
403          */
404         assertEquals(qExpected, q);
405     }
406 
407     /**
408      * Multiply with multiple digits with addition of 1.
409      */
410     @Test
411     public void testMultiplyBy10WithMultipleDigitsWithAddition() {
412 
413         /**
```

```
414          * Variables.
415          */
416         NaturalNumber q = this.constructorTest(98365492);
417         NaturalNumber qExpected = this.constructorRef(983654921);
418 
419         /**
420          * Call Method.
421          */
422         q.multiplyBy10(1);
423 
424         /**
425          * Assert Values.
426          */
427         assertEquals(qExpected, q);
428     }
429 
430     // TODO - divideBy10 Test Cases
431 
432     /**
433      * test q value as zero.
434      */
435     @Test
436     public void testDivideBy10WithZero() {
437 
438         /**
439          * Variables.
440          */
441         NaturalNumber q = this.constructorTest(0);
442         NaturalNumber qExpected = this.constructorRef(0);
443 
444         /**
445          * Call Method.
446          */
447         int remainder = q.divideBy10();
448 
449         /**
450          * Assert Values.
451          */
452         assertEquals(0, remainder);
453         assertEquals(qExpected, q);
454     }
455 
456     /**
457      * test with a Remainder Option while having q have double digits.
458      */
459     @Test
460     public void testDivideBy10WithARemainderWithDoubleDigits() {
461 
462         /**
463          * Variables.
464          */
465         NaturalNumber q = this.constructorTest(13);
466         NaturalNumber qExpected = this.constructorRef(1);
467 
468         /**
469          * Call Method.
470          */
471         int remainder = q.divideBy10();
472 
```

```
473         /**
474          * Assert Values.
475          */
476         assertEquals(3, remainder);
477         assertEquals(qExpected, q);
478     }
479 
480     /**
481      * test with no Remainder Option while having q have double digits.
482      */
483     @Test
484     public void testDivideBy10WithNoRemainderWithDoubleDigits() {
485 
486         /**
487          * Variables.
488          */
489         NaturalNumber q = this.constructorTest(30);
490         NaturalNumber qExpected = this.constructorRef(3);
491 
492         /**
493          * Call Method.
494          */
495         int remainder = q.divideBy10();
496 
497         /**
498          * Assert Values.
499          */
500         assertEquals(0, remainder);
501         assertEquals(qExpected, q);
502     }
503 
504     /**
505      * test with a Remainder Option while having q have triple digits.
506      */
507     @Test
508     public void testDivideBy10WithARemainderWithTripleDigits() {
509 
510         /**
511          * Variables.
512          */
513         NaturalNumber q = this.constructorTest(235);
514         NaturalNumber qExpected = this.constructorRef(23);
515 
516         /**
517          * Call Method.
518          */
519         int remainder = q.divideBy10();
520 
521         /**
522          * Assert Values.
523          */
524         assertEquals(5, remainder);
525         assertEquals(qExpected, q);
526     }
527 
528     /**
529      * test with no Remainder Option while having q have triple digits.
530      */
531     @Test
```

```
532     public void testDivideBy10WithNoRemainderWithTripleDigits() {
533
534         /**
535          * Variables.
536          */
537         NaturalNumber q = this.constructorTest(720);
538         NaturalNumber qExpected = this.constructorRef(72);
539
540         /**
541          * Call Method.
542          */
543         int remainder = q.divideBy10();
544
545         /**
546          * Assert Values.
547          */
548         assertEquals(0, remainder);
549         assertEquals(qExpected, q);
550     }
551
552     /**
553      * test with a Remainder Option while having q have an absurd amount of
554      * digits.
555      */
556     @Test
557     public void testDivideBy10WithARemainderWithAbsurdDigits() {
558
559         /**
560          * Variables.
561          */
562         NaturalNumber q = this.constructorTest(987654329);
563         NaturalNumber qExpected = this.constructorRef(98765432);
564
565         /**
566          * Call Method.
567          */
568         int remainder = q.divideBy10();
569
570         /**
571          * Assert Values.
572          */
573         assertEquals(9, remainder);
574         assertEquals(qExpected, q);
575     }
576
577     /**
578      * test with no Remainder Option while having q have an absurd amount of
579      * digits.
580      */
581     @Test
582     public void testDivideBy10WithNoRemainderWithAbsurdDigits() {
583
584         /**
585          * Variables.
586          */
587         NaturalNumber q = this.constructorTest(987654320);
588         NaturalNumber qExpected = this.constructorRef(98765432);
589
590         /**
```

```
591          * Call Method.
592          */
593         int remainder = q.divideBy10();
594 
595         /**
596          * Assert Values.
597          */
598         assertEquals(0, remainder);
599         assertEquals(qExpected, q);
600     }
601 
602     /**
603      * Test isZero with 0.
604      */
605     @Test
606     public final void testIsZeroWithZero() {
607         /**
608          * Create variables.
609          */
610         NaturalNumber n = this.constructorTest(0);
611         NaturalNumber nExpected = this.constructorRef(0);
612 
613         /**
614          * Call method.
615          */
616         boolean zero = n.isZero();
617 
618         /**
619          * Evaluate test.
620          */
621         assertEquals(zero, true);
622         assertEquals(n, nExpected);
623 
624     }
625 
626     /**
627      * Test isZero with a number that is not zero.
628      */
629     @Test
630     public final void testIsZeroWithoutZero() {
631         /**
632          * Create variables.
633          */
634         NaturalNumber n = this.constructorTest(347);
635         NaturalNumber nExpected = this.constructorRef(347);
636 
637         /**
638          * Call method.
639          */
640         boolean zero = n.isZero();
641 
642         /**
643          * Evaluate test.
644          */
645         assertEquals(zero, false);
646         assertEquals(n, nExpected);
647 
648     }
649 
```

```
650     /**
651      * Test isZero with a null NaturalNumber.
652      */
653     @Test
654     public final void testIsZeroWithNull() {
655         /**
656          * Create variables.
657          */
658         NaturalNumber n = this.constructorTest();
659         NaturalNumber nExpected = this.constructorRef();
660
661         /**
662          * Call method.
663          */
664         boolean zero = n.isZero();
665
666         /**
667          * Evaluate test.
668          */
669         assertEquals(zero, true);
670         assertEquals(n, nExpected);
671
672     }
673 }
674
```