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

import org.junit.Test;

import components.sequence.Sequence;

/**
 * JUnit test fixture for {@code Sequence<String>}'s constructor and kernel
 * methods.
 *
 * @author Put your name here
 *
 */
public abstract class SequenceTest {

    /**
     * Invokes the appropriate {@code Sequence} constructor for the
     * implementation under test and returns the result.
     *
     * @return the new sequence
     * @ensures constructorTest = <>
     */
    protected abstract Sequence<String> constructorTest();

    /**
     * Invokes the appropriate {@code Sequence} constructor for the reference
     * implementation and returns the result.
     *
     * @return the new sequence
     * @ensures constructorRef = <>
     */
    protected abstract Sequence<String> constructorRef();

    /**
     *
     * Creates and returns a {@code Sequence<String>} of the implementation
     * under test type with the given entries.
     *
     * @param args
     *            the entries for the sequence
```

```
41      * @return the constructed sequence
42      * @ensures createFromArgsTest = [entries in args]
43      */
44     private Sequence<String> createFromArgsTest(String... args) {
45         Sequence<String> sequence = this.constructorTest();
46         for (String s : args) {
47             sequence.add(sequence.length(), s);
48         }
49         return sequence;
50     }
51
52     /**
53      *
54      * Creates and returns a {@code Sequence<String>} of the
   reference
55      * implementation type with the given entries.
56      *
57      * @param args
58      *            the entries for the sequence
59      * @return the constructed sequence
60      * @ensures createFromArgsRef = [entries in args]
61      */
62     private Sequence<String> createFromArgsRef(String... args) {
63         Sequence<String> sequence = this.constructorRef();
64         for (String s : args) {
65             sequence.add(sequence.length(), s);
66         }
67         return sequence;
68     }
69
70     /**
71      * Routine Test case with seq1 = <"2","4","6"> and addedOn =
   "8" "6">.
72      */
73     @Test
74     public void test1() {
75         /*
76          * Set up variables and call method under test
77          */
78         Sequence<String> seq1 = this.createFromArgsTest("2", "4",
   "6");
79         Sequence<String> expectedSeq1 =
   this.createFromArgsRef("8", "2", "4",
80                 "6");
```

```
 81         String addedOn = "8";
 82         seq1.add(0, addedOn);
 83 
 84         /*
 85          * Assert that values of variables match expectations
 86          */
 87         assertEquals(expectedSeq1, seq1);
 88     }
 89 
 90     /**
 91      * Boundary Test case with seq1 = <> and addedOn = "".
 92      */
 93     @Test
 94     public void test2() {
 95         /*
 96          * Set up variables and call method under test
 97          */
 98         Sequence<String> seq1 = this.createFromArgsTest();
 99         Sequence<String> expectedSeq1 =
    this.createFromArgsRef("2", "4", "6");
100         String addedOn = "";
101         seq1.add(0, addedOn);
102 
103         /*
104          * Assert that values of variables match expectations
105          */
106         assertEquals(expectedSeq1, seq1);
107     }
108 
109     /**
110      * Challenging Test case with seq1 = <"1","2","3"> and addedOn
    = "9".
111      */
112     @Test
113     public void test3() {
114         /*
115          * Set up variables and call method under test
116          */
117         Sequence<String> seq1 = this.createFromArgsTest("1", "2",
    "3");
118         Sequence<String> expectedSeq1 =
    this.createFromArgsRef("1", "9", "2",
119                 "3");
120         String addedOn = "9";
```

```
121         seq1.add(seq1.length() - 2, addedOn);
122 
123         /*
124          * Assert that values of variables match expectations
125          */
126         assertEquals(expectedSeq1, seq1);
127     }
128 
129     /**
130      * Routine Test case with seq1 = <"1","2","3">.
131      */
132     @Test
133     public void test4() {
134         /*
135          * Set up variables and call method under test
136          */
137         Sequence<String> seq1 = this.createFromArgsTest("1", "2",
    "3");
138         Sequence<String> expectedSeq1 =
    this.createFromArgsRef("1", "2");
139 
140         seq1.remove(2);
141 
142         /*
143          * Assert that values of variables match expectations
144          */
145         assertEquals(expectedSeq1, seq1);
146     }
147 
148     /**
149      * Boundary Test case with seq1 = <"1">.
150      */
151     @Test
152     public void test5() {
153         /*
154          * Set up variables and call method under test
155          */
156         Sequence<String> seq1 = this.createFromArgsTest("1");
157         Sequence<String> expectedSeq1 = this.createFromArgsRef();
158 
159         seq1.remove(0);
160 
161         /*
162          * Assert that values of variables match expectations
```

```
163          */
164         assertEquals(expectedSeq1, seq1);
165     }
166 
167     /**
168      * Challenging Test case with seq1 = <"1","2","3">.
169      */
170     @Test
171     public void test6() {
172         /*
173          * Set up variables and call method under test
174          */
175         Sequence<String> seq1 = this.createFromArgsTest("1", "2",
    "3");
176         Sequence<String> expectedSeq1 =
    this.createFromArgsRef("2", "3");
177 
178         seq1.remove(seq1.length() - seq1.length());
179 
180         /*
181          * Assert that values of variables match expectations
182          */
183         assertEquals(expectedSeq1, seq1);
184     }
185 
186     /**
187      * Routine Test case with seq1 = <"1","2","3">.
188      */
189     @Test
190     public void test7() {
191         /*
192          * Set up variables and call method under test
193          */
194         Sequence<String> seq1 = this.createFromArgsTest("1", "2",
    "3");
195         Sequence<String> expectedSeq1 =
    this.createFromArgsRef("1", "2", "3");
196         int lengthOfTest = seq1.length();
197         int lengthOfRef = 3
198 
199         /*
200          * Assert that values of variables match expectations
201          */
202         assertEquals(expectedSeq1, seq1);
```

```
203         assertEquals(lengthOfTest, lengthOfRef);
204     }
205 
206     /**
207      * Boundary Test case with seq1 = <>.
208      */
209     @Test
210     public void test8() {
211         /*
212          * Set up variables and call method under test
213          */
214         Sequence<String> seq1 = this.createFromArgsTest();
215         Sequence<String> expectedSeq1 = this.createFromArgsRef();
216         int lengthOfTest = seq1.length();
217         int lengthOfRef = 0;
218 
219         /*
220          * Assert that values of variables match expectations
221          */
222         assertEquals(expectedSeq1, seq1);
223         assertEquals(lengthOfTest, lengthOfRef);
224     }
225 
226     /**
227      * Challenging Test case with seq1 = <"">.
228      */
229     @Test
230     public void test9() {
231         /*
232          * Set up variables and call method under test
233          */
234         Sequence<String> seq1 = this.createFromArgsTest("");
235         Sequence<String> expectedSeq1 =
    this.createFromArgsRef("");
236         int lengthOfTest = seq1.length();
237         int lengthOfRef = 1;
238 
239         /*
240          * Assert that values of variables match expectations
241          */
242         assertEquals(expectedSeq1, seq1);
243         assertEquals(lengthOfTest, lengthOfRef);
244     }
245 
```

```
246 }
```