

```
1 import static org.junit.Assert.assertEquals;
2
3 import org.junit.Test;
4
5 import components.sequence.Sequence;
6 import components.stack.Stack;
7
8 /**
9  * JUnit test fixture for {@code Sequence<String>}'s constructor and kernel
10 * methods.
11 *
12 * @author Yunlong Zhang
13 *
14 */
15 public abstract class SequenceTest {
16
17     final String a = "a", b = "b", c = "c", d = "d", e = "e", f = "f", j = "j";
18
19     /**
20      * Invokes the appropriate {@code Sequence} constructor for the
21      * implementation under test and returns the result.
22      *
23      * @return the new sequence
24      * @ensures constructorTest = <>
25      */
26     protected abstract Sequence<String> constructorTest();
27
28     /**
29      * Invokes the appropriate {@code Sequence} constructor for the reference
30      * implementation and returns the result.
31      *
32      * @return the new sequence
33      * @ensures constructorRef = <>
34      */
35     protected abstract Sequence<String> constructorRef();
36
37     /**
38      *
39      * Creates and returns a {@code Sequence<String>} of the implementation
40      * under test type with the given entries.
41      *
42      * @param args
43      *         the entries for the sequence
44      * @return the constructed sequence
45      * @ensures createFromArgsTest = [entries in args]
46      */
47     private Sequence<String> createFromArgsTest(String... args) {
48         Sequence<String> sequence = this.constructorTest();
49         for (String s : args) {
50             sequence.add(sequence.length(), s);
51         }
52         return sequence;
53     }
54
55     /**
56      *
57      * Creates and returns a {@code Sequence<String>} of the reference
58      * implementation type with the given entries.
59      *
```

```

60     * @param args
61     *         the entries for the sequence
62     * @return the constructed sequence
63     * @ensures createFromArgsRef = [entries in args]
64     */
65     private Sequence<String> createFromArgsRef(String... args) {
66         Sequence<String> sequence = this.constructorRef();
67         for (String s : args) {
68             sequence.add(sequence.length(), s);
69         }
70         return sequence;
71     }
72
73     // TODO - add test cases for constructor, add, remove, and length
74     @Test
75     public void testLength() {
76         Sequence<String> s1 = this.createFromArgsRef(this.a, this.b, this.c,
77             this.d, this.e, this.f);
78         int result = s1.length();
79         assertEquals(6, result);
80     }
81
82     @Test
83     public void testRemove() {
84         Sequence<String> s1 = this.createFromArgsRef(this.a, this.b, this.c,
85             this.d, this.e);
86         Sequence<String> s2 = this.createFromArgsRef(this.c, this.d, this.e);
87         s1.remove(0);
88         s2.remove(1);
89         assertEquals(s2, s1);
90     }
91
92     @Test
93     public void testAdd() {
94         Sequence<String> s1 = this.createFromArgsRef(this.a, this.b, this.c,
95             this.d, this.e);
96         Sequence<String> s2 = this.createFromArgsRef(this.a, this.b, this.c,
97             this.d, this.e, this.f, this.j);
98         s1.add(5, this.f);
99         s1.add(6, this.j);
100        assertEquals(s2, s1);
101    }
102
103    /**
104     * Shifts entries between {@code leftStack} and {@code rightStack}, keeping
105     * reverse of the former concatenated with the latter fixed, and resulting
106     * in length of the former equal to {@code newLeftLength}.
107     *
108     * @param <T>
109     *         type of {@code Stack} entries
110     * @param leftStack
111     *         the left {@code Stack}
112     * @param rightStack
113     *         the right {@code Stack}
114     * @param newLeftLength
115     *         desired new length of {@code leftStack}
116     * @updates leftStack, rightStack
117     * @requires <pre>
118     * 0 <= newLeftLength and

```

```
119     * newLeftLength <= |leftStack| + |rightStack|
120     * </pre>
121     * @ensures <pre>
122     *   rev(leftStack) * rightStack = rev(#leftStack) * #rightStack  and
123     *   |leftStack| = newLeftLength}
124     * </pre>
125     */
126     private static <T> void setLengthOfLeftStack(Stack<T> leftStack,
127         Stack<T> rightStack, int newLeftLength) {
128
129         int left = leftStack.length(), right = rightStack.length();
130
131         int num = left - newLeftLength;
132
133         if (num >= 0) {
134             for (int i = 0; i < num; i++) {
135                 T temp = leftStack.pop();
136                 rightStack.push(temp);
137             }
138         } else if (num < 0) {
139             num = -num;
140             for (int i = 0; i > num; i--) {
141                 T temp = rightStack.pop();
142                 leftStack.push(temp);
143             }
144         }
145     }
146 }
147
148 }
```