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

import org.junit.Test;

import components.sequence.Sequence;
import components.set.Set;

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

    /**
     * Invokes the appropriate {@code Set} constructor and returns
the result.
     *
     * @return the new set
     * @ensures constructorTest = {}
     */
    protected abstract Set<String> constructorTest();

    /**
     * Invokes the appropriate {@code Set} constructor and returns
the result.
     *
     * @return the new set
     * @ensures constructorRef = {}
     */
    protected abstract Set<String> constructorRef();

    /**
     * Creates and returns a {@code Set<String>} of the
implementation under
     * test type with the given entries.
     *
     * @param args
     *            the entries for the set
     * @return the constructed set
     * @requires [every entry in args is unique]
     * @ensures createFromArgsTest = [entries in args]
```

```
41      */
42     private Set<String> createFromArgsTest String... args) {
43         Set<String> set = this.constructorTest();
44         for (String s : args) {
45             assert !set.contains(
46                     s) : "Violation of: every entry in args is
   unique";
47             set.add(s);
48         }
49         return set;
50     }
51 
52     /**
53      * Creates and returns a {@code Set<String>} of the reference
   implementation
54      * type with the given entries.
55      *
56      * @param args
57      *            the entries for the set
58      * @return the constructed set
59      * @requires [every entry in args is unique]
60      * @ensures createFromArgsRef = [entries in args]
61      */
62     private Set<String> createFromArgsRef(String... args) {
63         Set<String> set = this.constructorRef();
64         for (String s : args) {
65             assert !set.contains(
66                     s) : "Violation of: every entry in args is
   unique";
67             set.add(s);
68         }
69         return set;
70     }
71 
72     /**
73      * Routine Test case with set1 = <"2","4","6"> and addedOn =
   <"8">.
74      */
75     @Test
76     public void test1() {
77         /*
78          * Set up variables and call method under test
79          */
80         Set<String> set1 = this.createFromArgsTest("2", "4", "6");
```

```
 81         Set<String> expectedSet1 =
    this.createFromArgsRef("2","4","6","8");
 82         String addedOn = "8";
 83         set1.add(addedOn);
 84 
 85         /*
 86          * Assert that values of variables match expectations
 87          */
 88         assertEquals(expectedSet1, set1);
 89     }
 90 
 91     /**
 92      * Boundary Test case with set1 = <> and addedOn = "1".
 93      */
 94     @Test
 95     public void test2() {
 96         /*
 97          * Set up variables and call method under test
 98          */
 99         Set<String> set1 = this.createFromArgsTest();
100         Set<String> expectedSet1 = this.createFromArgsRef("1");
101         String addedOn = "1";
102         set1.add(addedOn);
103 
104         /*
105          * Assert that values of variables match expectations
106          */
107         assertEquals(expectedSet1, set1);
108     }
109 
110     /**
111      * Challenging Test case with set1 = <"1","2","3"> and addedOn
    = "".
112      */
113     @Test
114     public void test3() {
115         /*
116          * Set up variables and call method under test
117          */
118         Set<String> set1 = this.createFromArgsTest("1", "2", "3");
119         Set<String> expectedSet1 = this.createFromArgsRef("1",
    "9", "2",
120                 "3");
121         String addedOn = "";
```

```
122         set1.add(addedOn);
123 
124         /*
125          * Assert that values of variables match expectations
126          */
127         assertEquals(expectedSet1, set1);
128     }
129 
130     /**
131      * Routine Test case with set1 = <"1","2","3">.
132      */
133     @Test
134     public void test4() {
135         /*
136          * Set up variables and call method under test
137          */
138         Set<String> set1 = this.createFromArgsTest("1", "2", "3");
139         Set<String> expectedSet1 = this.createFromArgsRef("1",
    "2");
140 
141         set1.remove("3");
142 
143         /*
144          * Assert that values of variables match expectations
145          */
146         assertEquals(expectedSet1, set1);
147     }
148 
149     /**
150      * Boundary Test case with set1 = <"1">.
151      */
152     @Test
153     public void test5() {
154         /*
155          * Set up variables and call method under test
156          */
157         Set<String> set1 = this.createFromArgsTest("1");
158         Set<String> expectedSet1 = this.createFromArgsRef();
159 
160         set1.remove("1");
161 
162         /*
163          * Assert that values of variables match expectations
164          */
```

```
165         assertEquals(expectedSet1, set1);
166     }
167
168     /**
169      * Challenging Test case with set1 = <"1","2","3">.
170      */
171     @Test
172     public void test6() {
173         /*
174          * Set up variables and call method under test
175          */
176         Set<String> set1 = this.createFromArgsTest("");
177         Set<String> expectedSet1 = this.createFromArgsRef();
178
179         set1.remove("");
180
181         /*
182          * Assert that values of variables match expectations
183          */
184         assertEquals(expectedSet1, set1);
185     }
186
187     /**
188      * Routine Test case with set1 = <"1","2","3">.
189      */
190     @Test
191     public void test7() {
192         /*
193          * Set up variables and call method under test
194          */
195         Set<String> set1 = this.createFromArgsTest("1", "2", "3");
196         Set<String> expectedSet1 = this.createFromArgsRef("1",
    "2", "3");
197         int lengthOfTest = set1.size();
198         int lengthOfRef = 3;
199
200         /*
201          * Assert that values of variables match expectations
202          */
203         assertEquals(expectedSet1, set1);
204         assertEquals(lengthOfTest, lengthOfRef);
205     }
206
207     /**
```

```
208      * Boundary Test case with set1 = <>.
209      */
210     @Test
211     public void test8() {
212         /*
213          * Set up variables and call method under test
214          */
215         Set<String> set1 = this.createFromArgsTest();
216         Set<String> expectedSet1 = this.createFromArgsRef();
217         int lengthOfTest = set1.size();
218         int lengthOfRef = 0;
219 
220         /*
221          * Assert that values of variables match expectations
222          */
223         assertEquals(expectedSet1, set1);
224         assertEquals(lengthOfTest, lengthOfRef);
225     }
226 
227     /**
228      * Challenging Test case with set1 = <"">.
229      */
230     @Test
231     public void test9() {
232         /*
233          * Set up variables and call method under test
234          */
235         Set<String> set1 = this.createFromArgsTest("");
236         Set<String> expectedSet1 = this.createFromArgsRef("");
237         int lengthOfTest = set1.size();
238         int lengthOfRef = 1;
239 
240         /*
241          * Assert that values of variables match expectations
242          */
243         assertEquals(expectedSet1, set1);
244         assertEquals(lengthOfTest, lengthOfRef);
245     }
246 
247     /**
248      * Routine Test case with set1 = <"1","2","3"> and test = "2"
249      */
250     @Test
251     public void test10() {
```

```
252         /*
253          * Set up variables and call method under test
254          */
255         Set<String> set1 = this.createFromArgsTest("1","2","3");
256         Set<String> expectedSet1 =
    this.createFromArgsRef("1","2","3");
257         String test = "2";
258         boolean containsOfTest = set1.contains(test);
259         boolean containsOfRef = true;
260 
261         /*
262          * Assert that values of variables match expectations
263          */
264         assertEquals(expectedSet1, set1);
265         assertEquals(containsOfTest, containsOfRef);
266     }
267 
268     /**
269      * Boundary Test case with set1 = <> and test = "2"
270      */
271     @Test
272     public void test11() {
273         /*
274          * Set up variables and call method under test
275          */
276         Set<String> set1 = this.createFromArgsTest();
277         Set<String> expectedSet1 = this.createFromArgsRef();
278         String test = "2";
279         boolean containsOfTest = set1.contains(test);
280         boolean containsOfRef = false;
281 
282         /*
283          * Assert that values of variables match expectations
284          */
285         assertEquals(expectedSet1, set1);
286         assertEquals(containsOfTest, containsOfRef);
287     }
288 
289     /**
290      * Challenging Test case with set1 = <""> and test = ""
291      */
292     @Test
293     public void test12() {
294         /*
```

```
295          * Set up variables and call method under test
296          */
297         Set<String> set1 = this.createFromArgsTest("");
298         Set<String> expectedSet1 = this.createFromArgsRef("");
299         String test = "";
300         boolean containsOfTest = set1.contains(test);
301         boolean containsOfRef = true;
302 
303         /*
304          * Assert that values of variables match expectations
305          */
306         assertEquals(expectedSet1, set1);
307         assertEquals(containsOfTest, containsOfRef);
308     }
309 
310     /**
311      * Routine Test case with set1 = <"1","2","3">
312      */
313     @Test
314     public void test13() {
315         /*
316          * Set up variables and call method under test
317          */
318         //Setup
319         Set<String> set1 = this.createFromArgsTest("1","2","3");
320         Set<String> expectedSet1 =
    this.createFromArgsRef("1","2","3");
321 
322         //Call
323         String capture = set1.removeAny();
324 
325         //Evaluation
326         assertEquals(true, expectedSet1.contains(capture));
327         expectedSet1.remove(capture);
328         assertEquals(expectedSet1, set1);
329     }
330 
331     /**
332      * Boundary Test case with set1 = <"1">
333      */
334     @Test
335     public void test14() {
336         /*
337          * Set up variables and call method under test
```

```
338          */
339         //Setup
340         Set<String> set1 = this.createFromArgsTest("1");
341         Set<String> expectedSet1 = this.createFromArgsRef("1");
342 
343         //Call
344         String capture = set1.removeAny();
345 
346         //Evaluation
347         assertEquals(true, expectedSet1.contains(capture));
348         expectedSet1.remove(capture);
349         assertEquals(expectedSet1, set1);
350     }
351 
352     /**
353      * Challenging Test case with set1 = <"1">
354      */
355     @Test
356     public void test15() {
357         /*
358          * Set up variables and call method under test
359          */
360         //Setup
361         Set<String> set1 = this.createFromArgsTest("");
362         Set<String> expectedSet1 = this.createFromArgsRef("");
363 
364         //Call
365         String capture = set1.removeAny();
366 
367         //Evaluation
368         assertEquals(true, expectedSet1.contains(capture));
369         expectedSet1.remove(capture);
370         assertEquals(expectedSet1, set1);
371     }
372 
373 
374 }
```