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

import org.junit.Test;

import components.set.Set;

/**
 * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
 *
 * @author Qinuo Shi & Yiming Cheng
 *
 */
public abstract class SetTest {

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

    /**
     * Invokes the appropriate {@code Set} constructor for the reference
     * implementation 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]
     */
    private Set<String> createFromArgsTest(String... args) {
        Set<String> set = this.constructorTest();
        for (String s : args) {
            assert !set.contains(
                    s) : "Violation of: every entry in args is unique";
            set.add(s);
        }
        return set;
    }

    /**
     * Creates and returns a {@code Set<String>} of the reference implementation
     * type with the given entries.
     *
     * @param args
```

```
58      *            the entries for the set
59      * @return the constructed set
60      * @requires [every entry in args is unique]
61      * @ensures createFromArgsRef = [entries in args]
62      */
63     private Set<String> createFromArgsRef(String... args) {
64         Set<String> set = this.constructorRef();
65         for (String s : args) {
66             assert !set.contains(
67                     s) : "Violation of: every entry in args is unique";
68             set.add(s);
69         }
70         return set;
71     }
72
73     // TODO - add test cases for constructor, add, remove, removeAny, contains, and size
74
75     /*
76      * Test for constructors
77      */
78     @Test
79     public final void testConstructor() {
80         Set<String> s = this.constructorTest();
81         Set<String> sexpected = this.constructorRef();
82
83         assertEquals(sexpected, s);
84     }
85
86     /*
87      * Test for kernel methods
88      */
89
90     /*
91      * Test for add
92      */
93     @Test
94     public final void testEmpty() {
95         Set<String> s = this.createFromArgsTest();
96         Set<String> sexpected = this.createFromArgsRef("a");
97
98         s.add("a");
99         assertEquals(sexpected, s);
100    }
101
102    @Test
103    public final void testNonEmpty1() {
104        Set<String> s = this.createFromArgsTest("a", "b");
105        Set<String> sexpected = this.createFromArgsRef("a", "b", "c");
106
107        s.add("c");
108        assertEquals(sexpected, s);
109    }
110
111    @Test
112    public final void testNonEmpty2() {
113        Set<String> s = this.createFromArgsTest("a", "b", "c", "d");
114        Set<String> sexpected = this.createFromArgsRef("a", "b", "c", "d", "e");
```

```
115
116         s.add("e");
117         assertEquals(sexpected, s);
118     }
119
120     @Test
121     public final void testNonEmpty3() {
122         Set<String> s = this.createFromArgsTest("Z", "Y", "W");
123         Set<String> sexpected = this.createFromArgsRef("Z", "Y", "W", "X");
124
125         s.add("X");
126         assertEquals(sexpected, s);
127     }
128
129     @Test
130     public final void testNonEmpty4() {
131         Set<String> s = this.createFromArgsTest("d", "c", "a", "b", "e", "f",
132                 "g");
133         Set<String> sexpected = this.createFromArgsRef("d", "c", "a", "b", "e",
134                 "f", "g", "s");
135
136         s.add("s");
137         assertEquals(sexpected, s);
138     }
139
140     /*
141      * Test for remove
142      */
143     @Test
144     public final void testRemoveEmpty() {
145         Set<String> s = this.createFromArgsTest("a");
146         Set<String> sexpected = this.createFromArgsRef();
147         String rexpected = "a";
148
149         String r = s.remove("a");
150         assertEquals(sexpected, s);
151         assertEquals(rexpected, r);
152     }
153
154     @Test
155     public final void testRemoveNonEmpty1() {
156         Set<String> s = this.createFromArgsTest("f", "a", "b", "n", "g");
157         Set<String> sexpected = this.createFromArgsRef("f", "a", "b", "n");
158         String rexpected = "g";
159
160         String r = s.remove("g");
161         assertEquals(sexpected, s);
162         assertEquals(rexpected, r);
163     }
164
165     @Test
166     public final void testRemoveNonEmpty2() {
167         Set<String> s = this.createFromArgsTest("f", "a", "b", "n", "g");
168         Set<String> sexpected = this.createFromArgsRef("f", "a", "n", "g");
169         String rexpected = "b";
170
171         String r = s.remove("b");
```

```
172         assertEquals(sexpected, s);
173         assertEquals(rexpected, r);
174     }
175 
176     @Test
177     public final void testRemoveNonEmpty3() {
178         Set<String> s = this.createFromArgsTest("c", "b", "a");
179         Set<String> sexpected = this.createFromArgsRef("c", "b");
180         String rexpected = "a";
181 
182         String r = s.remove("a");
183         assertEquals(sexpected, s);
184         assertEquals(rexpected, r);
185     }
186 
187     @Test
188     public final void testRemoveNonEmpty4() {
189         Set<String> s = this.createFromArgsTest("w", "y", "z");
190         Set<String> sexpected = this.createFromArgsRef("y", "z");
191         String rexpected = "w";
192 
193         String r = s.remove("w");
194         assertEquals(sexpected, s);
195         assertEquals(rexpected, r);
196     }
197 
198     @Test
199     public final void testRemoveNonEmpty5() {
200         Set<String> s = this.createFromArgsTest("m", "g", "u", "e", "f", "t");
201         Set<String> sexpected = this.createFromArgsRef("m", "g", "e", "f", "t");
202         String rexpected = "u";
203 
204         String r = s.remove("u");
205         assertEquals(sexpected, s);
206         assertEquals(rexpected, r);
207     }
208 
209     @Test
210     public final void testRemoveNonEmpty6() {
211         Set<String> s = this.createFromArgsTest("t", "s", "r", "q");
212         Set<String> sexpected = this.createFromArgsRef("s", "r", "q");
213         String rexpected = "t";
214 
215         String r = s.remove("t");
216         assertEquals(sexpected, s);
217         assertEquals(rexpected, r);
218     }
219 
220     /*
221      * Test for removeAny
222      */
223     @Test
224     public void testRemoveAnyFromConstructorWithElements1() {
225         Set<String> s = this.createFromArgsTest("a");
226         Set<String> sExpected = this.createFromArgsRef("a");
227 
228         String sremoveAny = s.removeAny();
```

```
229         String sExpectedremoveAny = sExpected.remove(sremoveAny);
230         assertEquals(sExpectedremoveAny, sremoveAny);
231         assertEquals(sExpected, s);
232     }
233 
234     @Test
235     public void testRemoveAnyFromConstructorWithElements2() {
236         Set<String> s = this.createFromArgsTest("a", "b", "c", "d");
237         Set<String> sExpected = this.createFromArgsRef("a", "b", "c", "d");
238 
239         String sremoveAny = s.removeAny();
240         String sExpectedremoveAny = sExpected.remove(sremoveAny);
241         assertEquals(sExpectedremoveAny, sremoveAny);
242         assertEquals(sExpected, s);
243     }
244 
245     /*
246      * Test for contain
247      */
248     @Test
249     public final void testContainsEmpty() {
250         Set<String> s = this.createFromArgsTest();
251         Set<String> sexpected = this.createFromArgsRef();
252 
253         assertEquals(false, s.contains("a"));
254         assertEquals(sexpected, s);
255     }
256 
257     @Test
258     public final void testTrue() {
259         Set<String> s = this.createFromArgsTest("a", "b", "c", "d");
260         Set<String> sexpected = this.createFromArgsRef("a", "b", "c", "d");
261 
262         assertEquals(true, s.contains("b"));
263         assertEquals(sexpected, s);
264     }
265 
266     @Test
267     public final void testFalse() {
268         Set<String> s = this.createFromArgsTest("a", "b", "c", "d");
269         Set<String> sexpected = this.createFromArgsRef("a", "b", "c", "d");
270 
271         assertEquals(false, s.contains("e"));
272         assertEquals(sexpected, s);
273     }
274 
275     /*
276      * Test for size
277      */
278     @Test
279     public final void testSizeEmpty() {
280         Set<String> s = this.createFromArgsTest();
281         Set<String> sexpected = this.createFromArgsRef();
282 
283         assertEquals(0, s.size());
284         assertEquals(sexpected, s);
285     }
```

```
286
287     @Test
288     public final void testSizeOne() {
289         Set<String> s = this.createFromArgsTest("a");
290         Set<String> sexpected = this.createFromArgsRef("a");
291
292         assertEquals(1, s.size());
293         assertEquals(sexpected, s);
294     }
295
296     @Test
297     public final void testSizeTwo() {
298         Set<String> s = this.createFromArgsTest("a", "b", "c", "d");
299         Set<String> sexpected = this.createFromArgsRef("a", "b", "c", "d");
300
301         assertEquals(4, s.size());
302         assertEquals(sexpected, s);
303     }
304
305 }
306
```