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

/**
 * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
 *
 * @author Gabe Azzarita and Ty Fredrick
 *
 */
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
     *            the entries for the set
     * @return the constructed set
     * @requires [every entry in args is unique]
     * @ensures createFromArgsRef = [entries in args]
     */
    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      // Testing empty (default) constructor
74      @Test
75      public final void testForDefaultConstructor() {
76          Set<String> test = this.constructorTest();
77          Set<String> ref = this.constructorRef();
78
79          assertEquals(test, ref);
80      }
81
82      // Testing constructor when passing in an arg
83      @Test
84      public final void testForConstructorEasy() {
85          Set<String> test = this.createFromArgsTest("1");
86          Set<String> ref = this.createFromArgsRef("1");
87
88          assertEquals(test, ref);
89      }
90
91      // Testing constructor when passing in multiple args
92      @Test
93      public final void testForConstructorHard() {
94          Set<String> test = this.createFromArgsTest("January", "February",
95                  "March", "April", "May", "June", "July", "August", "September",
96                  "October", "November", "December");
97          Set<String> ref = this.createFromArgsRef("January", "February", "March",
98                  "April", "May", "June", "July", "August", "September",
99                  "October", "November", "December");
100
101         assertEquals(test, ref);
102     }
103
104     // Testing add with one element
105     @Test
106     public final void testForAddOne() {
107         Set<String> test = this.constructorTest();
108         Set<String> ref = this.createFromArgsRef("purple");
109
110         test.add("purple");
111
112         assertEquals(test, ref);
113     }
114
115     // Testing add with multiple elements easy
116     @Test
117     public final void testForAddTwo() {
118         Set<String> test = this.constructorTest();
119         Set<String> ref = this.createFromArgsRef("Patriots", "Jets");
120
121         test.add("Patriots");
122         test.add("Jets");
```

```
123
124         assertEquals(test, ref);
125     }
126
127     // Testing add with many elements
128     @Test
129     public final void testForAddMultiple() {
130         Set<String> test = this.constructorTest();
131         Set<String> ref = this.createFromArgsRef("A", "B", "C", "1", "2", "3");
132
133         test.add("A");
134         test.add("B");
135         test.add("C");
136         test.add("1");
137         test.add("2");
138         test.add("3");
139
140         assertEquals(test, ref);
141     }
142
143     // Testing remove with one element
144     @Test
145     public final void testForRemoveOne() {
146         Set<String> test = this.createFromArgsTest("green");
147         Set<String> ref = this.constructorRef();
148
149         String rem = test.remove("green");
150
151         // Make sure remove properly returns string
152         assertEquals(rem, "green");
153         assertEquals(test, ref);
154     }
155
156     // Testing remove with multiple elements easy
157     @Test
158     public final void testForRemoveMultiple() {
159         Set<String> test = this.createFromArgsTest("A", "B", "C", "D");
160         Set<String> ref = this.createFromArgsRef("A", "C");
161
162         String rem1 = test.remove("B");
163         String rem2 = test.remove("D");
164
165         // Make sure remove properly returns string
166         assertEquals(rem1, "B");
167         assertEquals(rem2, "D");
168         assertEquals(test, ref);
169     }
170
171     // Testing remove with multiple elements hard
172     @Test
173     public final void testForRemoveHard() {
174         Set<String> test = this.createFromArgsTest("W", "X", "Y", "Z");
175         Set<String> ref = this.constructorRef();
176
177         String rem1 = test.remove("W");
178         String rem2 = test.remove("X");
179         String rem3 = test.remove("Y");
180         String rem4 = test.remove("Z");
181
```

```
182         // Make sure remove properly returns string
183         assertEquals(rem1, "W");
184         assertEquals(rem2, "X");
185         assertEquals(rem3, "Y");
186         assertEquals(rem4, "Z");
187         assertEquals(test, ref);
188     }
189 
190     // Testing removeAny with one element
191     @Test
192     public final void testForRemoveAnyOne() {
193         Set<String> test = this.createFromArgsTest("Bengals");
194         Set<String> ref = this.constructorRef();
195 
196         String rem1 = test.removeAny();
197 
198         // Make sure remove properly returns string
199         assertEquals(rem1, "Bengals");
200         assertEquals(test, ref);
201     }
202 
203     // Testing removeAny with two elements in Set
204     @Test
205     public final void testForRemoveAnyTwo() {
206         Set<String> test = this.createFromArgsTest("Salt", "Pepper", "Cumin");
207         Set<String> ref = this.createFromArgsRef("Salt", "Pepper", "Cumin");
208 
209         String remTest = test.removeAny();
210         String remExp = ref.remove(remTest);
211 
212         // Make sure removeAny returns string and check Sets for equality
213         assertEquals(remTest, remExp);
214         assertEquals(test, ref);
215     }
216 
217     // Testing multiple removeAny calls in big set
218     @Test
219     public final void testForRemoveAnyHard() {
220         Set<String> test = this.createFromArgsTest("pink", "red", "blue",
221                 "yellow", "orange", "green", "purple");
222         Set<String> ref = this.createFromArgsRef("pink", "red", "blue",
223                 "yellow", "orange", "green", "purple");
224 
225         String remTest1 = test.removeAny();
226         String remTest2 = test.removeAny();
227         String remExp1 = ref.remove(remTest1);
228         String remExp2 = ref.remove(remTest2);
229 
230         // Make sure removeAny returns string and check Sets for equality
231         assertEquals(remTest1, remExp1);
232         assertEquals(remTest2, remExp2);
233         assertEquals(test, ref);
234     }
235 
236     // Testing contains with empty Set
237     @Test
238     public final void testForContainsFalseNoElement() {
239         Set<String> test = this.constructorTest();
240 
```

```
241         assertEquals(test.contains("hello"), false);
242     }
243
244     // Testing contains with one element that is not in set
245     @Test
246     public final void testForContainsFalseOneElement() {
247         Set<String> test = this.createFromArgsTest("red");
248
249         assertEquals(test.contains("orange"), false);
250     }
251
252     // Testing contains with multiple element that are not in set
253     @Test
254     public final void testForContainsFalseMultiple() {
255         Set<String> test = this.createFromArgsTest("apple", "banana", "cherry",
256                 "date", "fig", "grape", "honeydew", "kiwi", "lemon");
257         Set<String> ref = this.createFromArgsRef("apple", "banana", "cherry",
258                 "date", "fig", "grape", "honeydew", "kiwi", "lemon");
259
260         assertEquals(test.contains("mango"), false);
261         assertEquals(test.contains("watermelon"), false);
262         assertEquals(test.contains("papaya"), false);
263
264         assertEquals(test, ref);
265     }
266
267     // Testing contains with one element that is in set
268     @Test
269     public final void testForContainsTrueOneElement() {
270         Set<String> test = this.createFromArgsTest("red");
271
272         assertEquals(test.contains("red"), true);
273     }
274
275     // Testing contains with multiple element that are not in set
276     @Test
277     public final void testForContainsTrueMultiple() {
278         Set<String> test = this.createFromArgsTest("apple", "banana", "cherry",
279                 "date", "fig", "grape", "honeydew", "kiwi", "lemon");
280         Set<String> ref = this.createFromArgsRef("apple", "banana", "cherry",
281                 "date", "fig", "grape", "honeydew", "kiwi", "lemon");
282
283         assertEquals(test.contains("banana"), true);
284         assertEquals(test.contains("cherry"), true);
285         assertEquals(test.contains("honeydew"), true);
286
287         assertEquals(test, ref);
288     }
289
290     // Testing size when set is empty
291     @Test
292     public final void testForSizeZero() {
293         Set<String> test = this.constructorTest();
294
295         assertEquals(test.size(), 0);
296     }
297
298     // Testing size when set has one element
299     @Test
```

```
300     public final void testForSizeOne() {
301         Set<String> test = this.createFromArgsTest("computer");
302 
303         assertEquals(test.size(), 1);
304     }
305 
306     // Testing size when set has multiple elements
307     @Test
308     public final void testForSizeMultiple() {
309         Set<String> test = this.createFromArgsTest("Joy", "Sadness", "Anger",
310                 "Fear", "Surprise", "Disgust", "Love", "Excitement", "Anxiety",
311                 "Guilt", "Envy", "Jealousy");
312 
313         final int twelve = 12;
314 
315         assertEquals(test.size(), twelve);
316     }
317 }
318 
```