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

/**
 * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
 *
 * @author Charan Nanduri and Evan Frisbie
 *
 */
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) {
```

```
47              assert !set.contains(
48                      s) : "Violation of: every entry in args is
  unique";
49              set.add(s);
50          }
51          return set;
52      }
53
54      /**
55       * Creates and returns a {@code Set<String>} of the reference
  implementation
56       * type with the given entries.
57       *
58       * @param args
59       *            the entries for the set
60       * @return the constructed set
61       * @requires [every entry in args is unique]
62       * @ensures createFromArgsRef = [entries in args]
63       */
64      private Set<String> createFromArgsRef(String... args) {
65          Set<String> set = this.constructorRef();
66          for (String s : args) {
67              assert !set.contains(
68                      s) : "Violation of: every entry in args is
  unique";
69              set.add(s);
70          }
71          return set;
72      }
73
74      // TODO - add test cases for constructor, add, remove,
  removeAny, contains, and size
75      //
76      @Test
77      public void testConstructorNoArgs() {
78          Set<String> s = this.constructorTest();
79          Set<String> sExp = this.constructorRef();
80          assertEquals(sExp, s);
81      }
82
83      //add tests
84
85      @Test
86      public void testAddToEmptyElement() {
87          Set<String> s = this.createFromArgsTest();
```

```
 88         Set<String> sExp = this.createFromArgsRef();
 89         s.add("red");
 90         sExp.add("red");
 91         assertEquals(sExp, s);
 92     }
 93 
 94     @Test
 95     public void testAddToExistingElement() {
 96         Set<String> s = this.createFromArgsTest("yellow", "red",
    "grey");
 97         Set<String> sExp = this.createFromArgsRef("yellow", "red",
    "grey");
 98         s.add("blue");
 99         sExp.add("blue");
100         assertEquals(sExp, s);
101     }
102 
103     //remove tests*
104     @Test
105     public final void testRemoveOne() {
106         Set<String> s = this.createFromArgsTest("red");
107         Set<String> sExp = this.createFromArgsRef("red");
108         String str = s.remove("red");
109         String strExp = sExp.remove("red");
110         assertEquals(sExp, s);
111         assertEquals(strExp, str);
112     }
113 
114     @Test
115     public final void testRemoveMultiple() {
116         Set<String> s = this.createFromArgsTest("yellow", "red",
    "grey",
117                 "blue");
118         Set<String> sExp = this.createFromArgsRef("yellow", "red",
    "grey",
119                 "blue");
120         String str = s.remove("red");
121         String strExp = sExp.remove("red");
122         assertEquals(sExp, s);
123         assertEquals(strExp, str);
124     }
125 
126     @Test
127     public final void testRemoveEnd() {
128         Set<String> s = this.createFromArgsTest("yellow", "red",
```

```
    "grey",
129                 "blue");
130         Set<String> sExp = this.createFromArgsRef("yellow", "red",
    "grey",
131                 "blue");
132         String str = s.remove("blue");
133         String strExp = sExp.remove("blue");
134         assertEquals(sExp, s);
135         assertEquals(strExp, str);
136     }
137
138     @Test
139     public final void testRemoveMiddle() {
140         Set<String> s = this.createFromArgsTest("yellow", "red",
    "grey",
141                 "blue");
142         Set<String> sExp = this.createFromArgsRef("yellow", "red",
    "grey",
143                 "blue");
144         String str = s.remove("grey");
145         String strExp = sExp.remove("grey");
146         assertEquals(sExp, s);
147         assertEquals(strExp, str);
148     }
149
150     // removeAny tests
151     @Test
152     public final void testRemoveAny() {
153         Set<String> s = this.createFromArgsTest("red", "grey");
154         Set<String> sExp = this.createFromArgsRef("red", "grey");
155         String str = s.removeAny();
156         assertTrue(sExp.contains(str));
157         String strExp = sExp.remove(str);
158         assertEquals(sExp, s);
159         assertEquals(strExp, str);
160     }
161
162     @Test
163     public final void testRemoveAnyMakingEmptySet() {
164         Set<String> s = this.createFromArgsTest("red");
165         Set<String> sExp = this.createFromArgsRef("red");
166         String str = s.removeAny();
167         assertTrue(sExp.contains(str));
168         String strExp = sExp.removeAny();
169         assertEquals(sExp, s);
```

```
170         assertEquals(strExp, str);
171     }
172 
173     // contains tests
174 
175     @Test
176     public final void testContainsEmpty() {
177         Set<String> s = this.createFromArgsTest();
178         Set<String> sExp = this.createFromArgsRef();
179         String str = "red";
180         assertEquals(sExp.contains(str), s.contains(str));
181         assertEquals(sExp, s);
182     }
183 
184     @Test
185     public final void testContainsBoolTrue() {
186         Set<String> s = this.createFromArgsTest("red", "grey");
187         Set<String> sExp = this.createFromArgsRef("red", "grey");
188         String str = "red";
189         assertEquals(sExp.contains(str), s.contains(str));
190         assertEquals(sExp, s);
191     }
192 
193     @Test
194     public final void testContainsBoolFalse() {
195         Set<String> s = this.createFromArgsTest("red", "grey");
196         Set<String> sExp = this.createFromArgsRef("red", "grey");
197         String str = "blue";
198         assertEquals(sExp.contains(str), s.contains(str));
199         assertEquals(sExp, s);
200     }
201 
202     //size test
203 
204     @Test
205     public final void testSizeZero() {
206         Set<String> s = this.constructorTest();
207         Set<String> sExp = this.constructorRef();
208         assertEquals(s.size(), sExp.size());
209         assertEquals(sExp, s);
210     }
211 
212     @Test
213     public final void testSizeOne() {
214         Set<String> s = this.createFromArgsTest("red");
```

```
215         Set<String> sExp = this.createFromArgsRef("red");
216         assertEquals(s.size(), sExp.size());
217         assertEquals(sExp, s);
218     }
219 
220     @Test
221     public final void testSizeMultiple() {
222         Set<String> s = this.createFromArgsTest("yellow", "red",
    "grey",
223                 "blue");
224         Set<String> sExp = this.createFromArgsRef("yellow", "red",
    "grey",
225                 "blue");
226         assertEquals(s.size(), sExp.size());
227         assertEquals(sExp, s);
228     }
229 
230 }
231 
```