

The Ohio State University

PROJECT 4: SET ON BINARY SEARCH TREES

Daniil Gofman

Ansh Pachauri

SW 2: Dev & Dsgn

Paolo Bucci

Yiyang Chen

Shivam Gupta

September 29, 2023

```
1 import static org.junit.Assert.assertEquals;
2 import static org.junit.Assert.assertTrue;
3
4 import org.junit.Test;
5
6 import components.set.Set;
7
8 /**
9  * JUnit test fixture for {@code Set<String>}s constructor and kernel
10  * methods.
11  * @author Daniil Gofman and Ansh Pachauri
12  *
13  */
14 public abstract class SetTest {
15
16     /**
17      * Invokes the appropriate {@code Set} constructor for the
18      * implementation
19      * under test and returns the result.
20      *
21      * @return the new set
22      * @ensures constructorTest = {}
23      */
24     protected abstract Set<String> constructorTest();
25
26     /**
27      * Invokes the appropriate {@code Set} constructor for the reference
28      * implementation and returns the result.
29      *
30      * @return the new set
31      * @ensures constructorRef = {}
32      */
33     protected abstract Set<String> constructorRef();
34
35     /**
36      * Creates and returns a {@code Set<String>} of the implementation
37      * under
38      * test type with the given entries.
39      *
40      * @param args
41      *         the entries for the set
42      * @return the constructed set
43      * @requires [every entry in args is unique]
44      * @ensures createFromArgsTest = [entries in args]
45      */
46     private Set<String> createFromArgsTest(String... args) {
47         Set<String> set = this.constructorTest();
48         for (String s : args) {
49             assert !set.contains(s);
50         }
51         set.addAll(Arrays.asList(args));
52     }
53 }
```

```
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
56  * implementation
57  * type with the given entries.
58  *
59  * @param args
60  *     the entries for the set
61  * @return the constructed set
62  * @requires [every entry in args is unique]
63  * @ensures createFromArgsRef = [entries in args]
64  */
65 private Set<String> createFromArgsRef(String... args) {
66     Set<String> set = this.constructorRef();
67     for (String s : args) {
68         assert !set.contains(
69             s) : "Violation of: every entry in args is unique";
70         set.add(s);
71     }
72     return set;
73 }
74
75 /**
76  * Test for empty constructor.
77  */
78 @Test
79 public final void testConstructorEmpty() {
80     /*
81     * Set up variables
82     */
83     Set<String> s = this.constructorTest();
84     Set<String> sExpected = this.constructorRef();
85     /*
86     * Assert that values of variables match expectations
87     */
88     assertEquals(sExpected, s);
89 }
90
91 /**
92  * Test for non-empty constructor.
93  */
94 @Test
95 public final void testConstructorNonEmpty() {
96     /*
97     * Set up variables
```

```
97         */
98         Set<String> s = this.createFromArgsTest("Apple", "Banana");
99         Set<String> sExpected = this.createFromArgsRef("Apple", "Banana");
100        /*
101         * Assert that values of variables match expectations
102         */
103        assertEquals(sExpected, s);
104    }
105
106    /**
107     * Test add boundary test case.
108     */
109    @Test
110    public final void testAddEmpty1() {
111        /*
112         * Set up variables
113         */
114        Set<String> s = this.createFromArgsTest();
115        Set<String> sExpected = this.createFromArgsRef("Apple");
116        /*
117         * Call methods under the test.
118         */
119        s.add("Apple");
120        /*
121         * Assert that values of variables match expectations
122         */
123        assertEquals(sExpected, s);
124    }
125
126    /**
127     * Test add boundary test case.
128     */
129    @Test
130    public final void testAddNonEmpty1() {
131        /*
132         * Set up variables
133         */
134        Set<String> s = this.createFromArgsTest("Apple");
135        Set<String> sExpected = this.createFromArgsRef("Apple", "Banana");
136        /*
137         * Call methods under the test.
138         */
139        s.add("Banana");
140        /*
141         * Assert that values of variables match expectations
142         */
143        assertEquals(sExpected, s);
144    }
145
146    /**
```

```
147     * Test add routine test case.
148     */
149     @Test
150     public final void testAddNonEmpty2() {
151         /*
152         * Set up variables
153         */
154         Set<String> s = this.createFromArgsTest("Apple", "Banana");
155         Set<String> sExpected = this.createFromArgsRef("Apple", "Banana",
156             "Orange");
157         /*
158         * Call methods under the test.
159         */
160         s.add("Orange");
161         /*
162         * Assert that values of variables match expectations
163         */
164         assertEquals(sExpected, s);
165     }
166
167     /**
168     * Test add routine test case.
169     */
170     @Test
171     public final void testAddNonEmpty3() {
172         /*
173         * Set up variables
174         */
175         Set<String> s = this.createFromArgsTest("Apple", "Banana",
176             "Orange",
177             "Watermelon", "Kiwi");
178         Set<String> sExpected = this.createFromArgsRef("Apple", "Banana",
179             "Orange", "Watermelon", "Kiwi", "Avocado");
180         /*
181         * Call methods under the test.
182         */
183         s.add("Avocado");
184         /*
185         * Assert that values of variables match expectations
186         */
187         assertEquals(sExpected, s);
188     }
189
190     /**
191     * Test remove boundary test case.
192     */
193     @Test
194     public final void testRemoveEmpty1() {
195         /*
196         * Set up variables
```

```
196         */
197         Set<String> s = this.createFromArgsTest("Apple");
198         Set<String> sExpected = this.createFromArgsRef();
199         /*
200         * Call methods under the test.
201         */
202         String temp = s.remove("Apple");
203         /*
204         * Assert that values of variables match expectations
205         */
206         assertEquals(sExpected, s);
207         assertEquals("Apple", temp);
208     }
209
210     /**
211     * Test remove routine test case.
212     */
213     @Test
214     public final void testRemoveNonEmpty1() {
215         /*
216         * Set up variables
217         */
218         Set<String> s = this.createFromArgsTest("Apple", "Banana");
219         Set<String> sExpected = this.createFromArgsRef("Apple");
220         /*
221         * Call methods under the test.
222         */
223         String temp2 = s.remove("Banana");
224         /*
225         * Assert that values of variables match expectations
226         */
227         assertEquals(sExpected, s);
228         assertEquals("Banana", temp2);
229     }
230
231     /**
232     * Test remove routine test case.
233     */
234     @Test
235     public final void testRemoveNonEmpty2() {
236         /*
237         * Set up variables
238         */
239         Set<String> s = this.createFromArgsTest("Apple", "Banana",
"Orange",
240         "Watermelon", "Kiwi", "Avocado");
241         Set<String> sExpected = this.createFromArgsRef("Apple", "Orange",
242         "Watermelon", "Kiwi", "Avocado");
243         /*
244         * Call methods under the test.
```

```
245         */
246         String temp = s.remove("Banana");
247         /*
248         * Assert that values of variables match expectations
249         */
250         assertEquals(sExpected, s);
251         assertEquals("Banana", temp);
252     }
253
254     /**
255     * Test size routine case.
256     */
257     @Test
258     public final void testSize() {
259         /*
260         * Set up variables
261         */
262         Set<String> s = this.createFromArgsTest("Pizza", "Steak", "Sushi",
263             "Salmon", "Grilled Pork");
264         Set<String> sExpected = this.createFromArgsRef("Pizza", "Steak",
265             "Sushi", "Salmon", "Grilled Pork");
266         /*
267         * Call method under test
268         */
269         int setSize = s.size();
270         final int expectedSize = 5;
271         /*
272         * Assert that values of variables match expectations
273         */
274         assertEquals(sExpected, s);
275         assertEquals(expectedSize, setSize);
276     }
277
278     /**
279     * Test size routine case.
280     */
281     @Test
282     public final void testSizeNonEmpty() {
283         /*
284         * Set up variables
285         */
286         Set<String> s = this.createFromArgsTest("Pizza", "Steak");
287         Set<String> sExpected = this.createFromArgsRef("Pizza", "Steak");
288         /*
289         * Call method under test
290         */
291         int setSize = s.size();
292         /*
293         * Assert that values of variables match expectations
294         */
```

```
295         assertEquals(2, setSize);
296         assertEquals(sExpected, s);
297     }
298
299     /**
300      * Test size boundary case.
301      */
302     @Test
303     public final void testSizeEmpty() {
304         /*
305          * Set up variables
306          */
307         Set<String> s = this.constructorTest();
308         Set<String> sExpected = this.constructorRef();
309         /*
310          * Call method under test
311          */
312         final int expectedSize = 0;
313         int setSize = s.size();
314         /*
315          * Assert that values of variables match expectations
316          */
317         assertEquals(expectedSize, setSize);
318         assertEquals(sExpected, s);
319     }
320
321     /**
322      * Test contains routine case.
323      */
324     @Test
325     public final void testContainsNonEmpty() {
326         /*
327          * Set up variables and call method under test
328          */
329         Set<String> s = this.createFromArgsTest("Pizza", "Steak",
330             "Mac and Cheese");
331         /*
332          * Call methods under the test.
333          */
334         final boolean expectedValue = false;
335         boolean isIcecream = s.contains("Ice cream");
336         /*
337          * Assert that values of variables match expectations
338          */
339         assertEquals(expectedValue, isIcecream);
340     }
341
342     /**
343      * Test contains boundary case.
344      */
```



```
345     @Test
346     public final void testContainsEmpty() {
347         /*
348          * Set up variables and call method under test
349          */
350         Set<String> s = this.createFromArgsTest();
351         /*
352          * Call methods under the test.
353          */
354         final boolean expectedValue = false;
355         boolean isSteak = s.contains("Steak");
356         /*
357          * Assert that values of variables match expectations
358          */
359         assertEquals(expectedValue, isSteak);
360     }
361
362     /**
363     * Test removeAny routine case.
364     */
365     @Test
366     public final void testRemoveAnyNonEmpty() {
367         /*
368          * Set up variables and call method under test
369          */
370         Set<String> s = this.createFromArgsTest("Pizza", "Steak",
371             "Mac and Cheese");
372         Set<String> sExpected = this.createFromArgsRef("Pizza", "Steak",
373             "Mac and Cheese");
374         /*
375          * Call methods under the test.
376          */
377         String removedValue = s.removeAny();
378         String removedValueExpected = sExpected.remove(removedValue);
379
380         /*
381          * Assert that values of variables match expectations
382          */
383         assertEquals(removedValue, removedValueExpected);
384         assertEquals(s, sExpected);
385     }
386
387     /**
388     * Test removeAny boundary case.
389     */
390     @Test
391     public final void testRemoveAnyLeavingEmpty() {
392         /*
393          * Set up variables and call method under test
394          */
```

```
395     Set<String> s = this.createFromArgsTest("Steak");
396     Set<String> sExpected = this.createFromArgsRef("Steak");
397     /*
398     * Call methods under the test.
399     */
400     String removedValue = s.removeAny();
401     assertTrue(sExpected.contains(removedValue));
402     String removedValueExpected = sExpected.remove(removedValue);
403
404     /*
405     * Assert that values of variables match expectations
406     */
407     assertEquals(removedValue, removedValueExpected);
408     assertEquals(s, sExpected);
409 }
410
411 }
412
```