The Ohio State University

PROJECT 3: HASHING IMPLEMENTATION OF MAP

Daniil Gofman

Ansh Pachauri

SW 2: Dev & Dsgn

Paolo Bucci

Yiyang Chen

Shivam Gupta

September 20, 2023

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

import org.junit.Test;

import components.map.Map;
import components.map.Map.Pair;

/**
 * JUnit test fixture for {@code Map<String, String>}'s constructor and kernel
 * methods.
 *
 * @author Daniil Gofman
 *
 */
public abstract class MapTest {

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

    /**
     * Invokes the appropriate {@code Map} constructor for the reference
     * implementation and returns the result.
     *
     * @return the new map
     * @ensures constructorRef = {}
     */
    protected abstract Map<String, String> constructorRef();

    /**
     *
     * Creates and returns a {@code Map<String, String>} of the implementation
     * under test type with the given entries.
     *
     * @param args
     *            the (key, value) pairs for the map
     * @return the constructed map
     * @requires <pre>
     * [args.length is even]  and
     * [the 'key' entries in args are unique]
     * </pre>
     * @ensures createFromArgsTest = [pairs in args]
     */
    private Map<String, String> createFromArgsTest(String... args) {
        assert args.length % 2 == 0 : "Violation of: args.length is even";
        Map<String, String> map = this.constructorTest();
        for (int i = 0; i < args.length; i += 2) {
            assert !map.hasKey(args[i]) : ""
                    + "Violation of: the 'key' entries in args are unique";
            map.add(args[i], args[i + 1]);
        }
        return map;
    }

    /**
     *
     * Creates and returns a {@code Map<String, String>} of the reference
```

```
63      * implementation type with the given entries.
64      *
65      * @param args
66      *            the (key, value) pairs for the map
67      * @return the constructed map
68      * @requires <pre>
69      * [args.length is even]  and
70      * [the 'key' entries in args are unique]
71      * </pre>
72      * @ensures createFromArgsRef = [pairs in args]
73      */
74     private Map<String, String> createFromArgsRef(String... args) {
75         assert args.length % 2 == 0 : "Violation of: args.length is even";
76         Map<String, String> map = this.constructorRef();
77         for (int i = 0; i < args.length; i += 2) {
78             assert !map.hasKey(args[i]) : ""
79                     + "Violation of: the 'key' entries in args are unique";
80             map.add(args[i], args[i + 1]);
81         }
82         return map;
83     }
84
85     /**
86      * Test empty constructor.
87      */
88     @Test
89     public final void testEmptyConstructor() {
90         Map<String, String> map = this.constructorTest();
91         Map<String, String> mapExpected = this.constructorRef();
92         assertEquals(map, mapExpected);
93     }
94
95     /**
96      * Test constructor with parameters.
97      */
98     @Test
99     public final void testConstructor() {
100        Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
101                "Toyota", "3", "Chevrolet", "4", "Lexus");
102        Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
103                "2", "Toyota", "3", "Chevrolet", "4", "Lexus");
104        assertEquals(map, mapExpected);
105    }
106
107    /**
108     * Test Add starting with empty containers.
109     */
110    @Test
111    public final void testAddEmpty() {
112        Map<String, String> map = this.constructorTest();
113        Map<String, String> mapExpected = this.constructorRef();
114
115        map.add("1", "Steak");
116        mapExpected.add("1", "Steak");
117        assertEquals(map, mapExpected);
118    }
119
120    /**
121     * Test for Add not empty containers.
122     */
123    @Test
124    public final void testAddNotEmpty() {
```

```
125         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
126                 "Toyota");
127         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
128                 "2", "Toyota", "3", "Chevrolet", "4", "Lexus");
129
130         map.add("3", "Chevrolet");
131         map.add("4", "Lexus");
132
133         assertEquals(map, mapExpected);
134     }
135
136     /**
137      * Test for Add not empty containers adding values to both.
138      */
139     @Test
140     public final void testAddNotEmpty2() {
141         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
142                 "Toyota");
143         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
144                 "2", "Toyota");
145
146         map.add("3", "Chevrolet");
147         map.add("4", "Lexus");
148         mapExpected.add("3", "Chevrolet");
149         mapExpected.add("4", "Lexus");
150
151         assertEquals(map, mapExpected);
152     }
153
154     /**
155      * Test Remove leaving empty with empty and not empty maps.
156      */
157     @Test
158     public final void testRemoveLeaveEmpty() {
159         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
160                 "Toyota");
161         Map<String, String> mapExpected = this.constructorRef();
162         Pair<String, String> firstVal = map.remove("1");
163         Pair<String, String> secondVal = map.remove("2");
164         assertEquals("Tesla", firstVal.value());
165         assertEquals("Toyota", secondVal.value());
166         assertEquals(map, mapExpected);
167     }
168
169     /**
170      * Test Remove leaving empty with two not empty maps.
171      */
172     @Test
173     public final void testRemoveLeaveEmpty2() {
174         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
175                 "Toyota");
176         Map<String, String> mapExpected = this.createFromArgsRef("2", "Toyota");
177         map.remove("1");
178         map.remove("2");
179         mapExpected.remove("2");
180         assertEquals(map, mapExpected);
181     }
182
183     /**
184      * Test Remove leaving not empty.
185      */
186     @Test
```

```
187     public final void testRemoveLeaveNotEmpty() {
188         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
189                 "Toyota");
190         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
191                 "2", "Toyota");
192         map.remove("1");
193         mapExpected.remove("1");
194         assertEquals(map, mapExpected);
195     }
196 
197     /**
198      * Test RemoveAny leaving empty.
199      */
200     @Test
201     public final void testRemoveAnyLeavingEmpty() {
202         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
203                 "Toyota");
204         Map<String, String> mapExpected = this.constructorRef();
205         map.removeAny();
206         map.removeAny();
207         assertEquals(map, mapExpected);
208     }
209 
210     /**
211      * Test RemoveAny leaving empty.
212      */
213     @Test
214     public final void testRemoveAnyLeavingEmpty2() {
215         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
216                 "Toyota");
217         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla");
218         map.removeAny();
219         map.removeAny();
220         mapExpected.removeAny();
221         assertEquals(map, mapExpected);
222     }
223 
224     /**
225      * Test RemoveAny leaving empty, compare values.
226      */
227     @Test
228     public final void testRemoveAnyNotEmpty() {
229         Map<String, String> map = this.createFromArgsTest("1", "Tesla");
230         Pair<String, String> result = map.removeAny();
231         String expectedRes = "Tesla";
232         assertEquals(result.value(), expectedRes);
233     }
234 
235     /**
236      * Test Value with two maps.
237      */
238     @Test
239     public final void testValueTwoMaps() {
240         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
241                 "Toyota");
242         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
243                 "2", "Toyota");
244 
245         assertEquals(map, mapExpected);
246         assertEquals(map.value("1"), mapExpected.value("1"));
247         assertEquals(map.value("2"), mapExpected.value("2"));
248     }
```

```
249
250     /**
251      * Test Value with two maps different size.
252      */
253     @Test
254     public final void testValueTwoMaps2() {
255         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
256                 "Toyota", "3", "Chevrolet", "4", "Lexus");
257         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
258                 "2", "Toyota", "3", "Chevrolet", "4", "Lexus");
259
260         assertEquals(map, mapExpected);
261         assertEquals(map.value("1"), mapExpected.value("1"));
262         assertEquals(map.value("2"), mapExpected.value("2"));
263         assertEquals(map.value("3"), mapExpected.value("3"));
264         assertEquals(map.value("4"), mapExpected.value("4"));
265     }
266
267     /**
268      * Test HasKey with two maps.
269      */
270     @Test
271     public final void testHasKey() {
272         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
273                 "Toyota");
274         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
275                 "2", "Toyota");
276
277         assertEquals(map, mapExpected);
278         assertEquals(map.hasKey("1"), mapExpected.hasKey("1"));
279         assertEquals(map.hasKey("2"), mapExpected.hasKey("2"));
280     }
281
282     /**
283      * Test HasKey maps of different size.
284      */
285     @Test
286     public final void testHasKey2() {
287         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
288                 "Toyota", "3", "Chevrolet", "4", "Lexus");
289         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
290                 "2", "Toyota", "3", "Chevrolet", "4", "Lexus");
291
292         assertEquals(map, mapExpected);
293         assertEquals(map.hasKey("1"), mapExpected.hasKey("1"));
294         assertEquals(map.hasKey("2"), mapExpected.hasKey("2"));
295         assertEquals(map.hasKey("3"), mapExpected.hasKey("3"));
296         assertEquals(map.hasKey("4"), mapExpected.hasKey("4"));
297     }
298
299     /**
300      * Test Size with not empty maps.
301      */
302     @Test
303     public final void testSizeNotEmpty() {
304         Map<String, String> map = this.createFromArgsTest("1", "Tesla", "2",
305                 "Toyota", "3", "Chevrolet", "4", "Lexus");
306         Map<String, String> mapExpected = this.createFromArgsRef("1", "Tesla",
307                 "2", "Toyota", "3", "Chevrolet", "4", "Lexus");
308
309         assertEquals(map.size(), mapExpected.size());
310     }
```

```
311
312     /**
313      * Test Size with empty maps.
314      */
315     @Test
316     public final void testSizeEmpty() {
317         Map<String, String> map = this.constructorTest();
318         Map<String, String> mapExpected = this.constructorRef();
319
320         assertEquals(map.size(), mapExpected.size());
321     }
322 }
323
```