

```

1 import java.util.Iterator;
2 import java.util.NoSuchElementException;
3
4 import components.map.Map;
5 import components.map.Map2;
6 import components.map.MapSecondary;
7
8 /**
9  * {@code Map} represented as a hash table using {@code Map}s for
   the buckets,
10  * with implementations of primary methods.
11  *
12  * @param <K>
13  *         type of {@code Map} domain (key) entries
14  * @param <V>
15  *         type of {@code Map} range (associated value) entries
16  * @convention <pre>
17  * |$this.hashTable| > 0 and
18  * for all i: integer, pf: PARTIAL_FUNCTION, x: K
19  *     where (0 <= i and i < |$this.hashTable| and
20  *         <pf> = $this.hashTable[i, i+1) and
21  *         x is in DOMAIN(pf))
22  *     ([computed result of x.hashCode()] mod |$this.hashTable| =
   i)) and
23  * for all i: integer
24  *     where (0 <= i and i < |$this.hashTable|)
25  *     ([entry at position i in $this.hashTable is not null]) and
26  * $this.size = sum i: integer, pf: PARTIAL_FUNCTION
27  *     where (0 <= i and i < |$this.hashTable| and
28  *         <pf> = $this.hashTable[i, i+1))
29  *     (|pf|)
30  * </pre>
31  * @correspondence <pre>
32  * this = union i: integer, pf: PARTIAL_FUNCTION
33  *     where (0 <= i and i < |$this.hashTable| and
34  *         <pf> = $this.hashTable[i, i+1))
35  *     (pf)
36  * </pre>
37  *
38  * @author Alex Honigford and Jonny Pater
39  *
40  */
41 public class Map4<K, V> extends MapSecondary<K, V> {
42

```

```
43     /*
44     * Private members
45     */
46
47     /**
48     * Default size of hash table.
49     */
50     private static final int DEFAULT_HASH_TABLE_SIZE = 101;
51
52     /**
53     * Buckets for hashing.
54     */
55     private Map<K, V>[] hashTable;
56
57     /**
58     * Total size of abstract {@code this}.
59     */
60     private int size;
61
62     /**
63     * Computes {@code a} mod {@code b} as % should have been
64     defined to work.
65     *
66     * @param a
67     *         the number being reduced
68     * @param b
69     *         the modulus
70     * @return the result of a mod b, which satisfies  $0 \leq \text{mod} < b$ 
71     * @requires  $b > 0$ 
72     * @ensures <pre>
73     *  $0 \leq \text{mod}$  and  $\text{mod} < b$  and
74     * there exists k: integer ( $a = k * b + \text{mod}$ )
75     * </pre>
76     */
77     private static int mod(int a, int b) {
78         assert b > 0 : "Violation of: b > 0";
79
80         int rem = a % b;
81         if (rem < 0) {
82             rem += b;
83         }
84     }
```

```
84         return rem;
85     }
86
87     /**
88      * Creator of initial representation.
89      *
90      * @param hashTableSize
91      *         the size of the hash table
92      * @requires hashTableSize > 0
93      * @ensures <pre>
94      *   |$this.hashTable| = hashTableSize and
95      *   for all i: integer
96      *     where (0 <= i and i < |$this.hashTable|)
97      *       ($this.hashTable[i, i+1) = <{}>) and
98      *   $this.size = 0
99      * </pre>
100     */
101     @SuppressWarnings("unchecked")
102     private void createNewRep(int hashTableSize) {
103         /*
104          * With "new Map<K, V>[...]" in place of "new Map[...]" it
105          * does not
106          * compile; as shown, it results in a warning about an
107          * unchecked
108          * conversion, though it cannot fail.
109          */
110         this.hashTable = new Map2[hashTableSize];
111         for (int i = 0; i < this.hashTable.length; i++) {
112             this.hashTable[i] = new Map2<K, V>();
113         }
114         this.size = 0;
115     }
116
117     /*
118      * Constructors
119      */
120
121     /**
122      * No-argument constructor.
123      */
124     public Map4() {
```

---

```
125
126         this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
127     }
128 }
129
130 /**
131  * Constructor resulting in a hash table of size {@code
hashTableSize}.
132  *
133  * @param hashTableSize
134  *         size of hash table
135  * @requires hashTableSize > 0
136  * @ensures this = {}
137  */
138 public Map4(int hashTableSize) {
139     this.createNewRep(hashTableSize);
140 }
141
142
143 /*
144  * Standard methods
145  */
146
147 @SuppressWarnings("unchecked")
148 @Override
149 public final Map<K, V> newInstance() {
150     try {
151         return this.getClass().getConstructor().newInstance();
152     } catch (ReflectiveOperationException e) {
153         throw new AssertionError(
154             "Cannot construct object of type " +
this.getClass());
155     }
156 }
157
158 @Override
159 public final void clear() {
160     this.createNewRep(DEFAULT_HASH_TABLE_SIZE);
161 }
162
163 @Override
164 public final void transferFrom(Map<K, V> source) {
165     assert source != null : "Violation of: source is not
```

```

    null";
166     assert source != this : "Violation of: source is not
    this";
167     assert source instanceof Map4<?, ?> : ""
168         + "Violation of: source is of dynamic type
    Map4<?, ?>";
169     /*
170     * This cast cannot fail since the assert above would have
    stopped
171     * execution in that case: source must be of dynamic type
    Map4<?, ?>, and
172     * the ?,? must be K,V or the call would not have
    compiled.
173     */
174     Map4<K, V> localSource = (Map4<K, V>) source;
175     this.hashTable = localSource.hashTable;
176     this.size = localSource.size;
177     localSource.createNewRep(DEFAULT_HASH_TABLE_SIZE);
178 }
179
180 /*
181  * Kernel methods
    -----
182  */
183
184 @Override
185 public final void add(K key, V value) {
186     assert key != null : "Violation of: key is not null";
187     assert value != null : "Violation of: value is not null";
188     assert !this.hasKey(key) : "Violation of: key is not in
    DOMAIN(this)";
189
190     int i = mod(key.hashCode(), this.hashTable.length);
191     if (!this.hashTable[i].hasKey(key)) {
192         this.hashTable[i].add(key, value);
193     }
194     this.size++;
195 }
196
197 @Override
198 public final Pair<K, V> remove(K key) {
199     assert key != null : "Violation of: key is not null";
200     assert this.hasKey(key) : "Violation of: key is in
    DOMAIN(this)";

```

```
201         int i = mod(key.hashCode(), this.hashTable.length);
202         this.size--;
203         return this.hashTable[i].remove(key);
204     }
205
206     @Override
207     public final Pair<K, V> removeAny() {
208         assert this.size() > 0 : "Violation of: this !=
empty_set";
209         this.size--;
210         int i = 0;
211         while (this.hashTable[i].size() == 0) {
212             i++;
213         }
214         return this.hashTable[i].removeAny();
215     }
216
217     @Override
218     public final V value(K key) {
219         assert key != null : "Violation of: key is not null";
220         assert this.hasKey(key) : "Violation of: key is in
DOMAIN(this)";
221
222         int i = mod(key.hashCode(), this.hashTable.length);
223         return this.hashTable[i].value(key);
224     }
225
226     @Override
227     public final boolean hasKey(K key) {
228         assert key != null : "Violation of: key is not null";
229         int i = mod(key.hashCode(), this.hashTable.length);
230         return this.hashTable[i].hasKey(key);
231     }
232
233     @Override
234     public final int size() {
235         return this.size;
236     }
237
238     @Override
239     public final Iterator<Pair<K, V>> iterator() {
240         return new Map4Iterator();
241     }
242
```

```

243     /**
244     * Implementation of {@code Iterator} interface for {@code
Map4}.
245     */
246     private final class Map4Iterator implements Iterator<Pair<K,
V>> {
247
248         /**
249         * Number of elements seen already (i.e., |~this.seen|).
250         */
251         private int numberSeen;
252
253         /**
254         * Bucket from which current bucket iterator comes.
255         */
256         private int currentBucket;
257
258         /**
259         * Bucket iterator from which next element will come.
260         */
261         private Iterator<Pair<K, V>> bucketIterator;
262
263         /**
264         * No-argument constructor.
265         */
266         Map4Iterator() {
267             this.numberSeen = 0;
268             this.currentBucket = 0;
269             this.bucketIterator =
Map4.this.hashTable[0].iterator();
270         }
271
272         @Override
273         public boolean hasNext() {
274             return this.numberSeen < Map4.this.size;
275         }
276
277         @Override
278         public Pair<K, V> next() {
279             assert this.hasNext() : "Violation of: ~this.unseen /=
<>";
280             if (!this.hasNext()) {
281                 /*
282                 * Exception is supposed to be thrown in this

```

```
case, but with
283         * assertion-checking enabled it cannot happen
because of assert
284         * above.
285         */
286         throw new NoSuchElementException();
287     }
288     this.numberSeen++;
289     while (!this.bucketIterator.hasNext()) {
290         this.currentBucket++;
291         this.bucketIterator =
Map4.this.hashTable[this.currentBucket]
292             .iterator();
293     }
294     return this.bucketIterator.next();
295 }
296
297 @Override
298 public void remove() {
299     throw new UnsupportedOperationException(
300         "remove operation not supported");
301 }
302
303 }
304
305 }
306
```