

```

1 import java.util.Iterator;
2
3 /**
4  * {@code Map} represented as a {@code Queue} of pairs with implementations of
5  * primary methods.
6  *
7  * @param <K>
8  *     type of {@code Map} domain (key) entries
9  * @param <V>
10 *     type of {@code Map} range (associated value) entries
11 *
12 * @convention <pre>
13 * for all key1, key2: K, value1, value2: V, str1, str2: string of (key, value)
14 *     where (str1 * <(key1, value1)> is prefix of $this.pairsQueue and
15 *           str2 * <(key2, value2)> is prefix of $this.pairsQueue and
16 *           str1 /= str2)
17 *     (key1 /= key2)
18 * </pre>
19 * @correspondence this = entries($this.pairsQueue)
20 */
21 public class Map2<K, V> extends MapSecondary<K, V> {
22     /**
23      * Private members -----
24      */
25
26     /**
27      * Pairs included in {@code this}.
28      */
29     private Queue<Pair<K, V>> pairsQueue;
30
31     /**
32      * Finds pair with first component {@code key} and, if such exists, moves it
33      * to the front of {@code q}.
34      *
35      * @param <K>
36      *     type of {@code Pair} key
37      * @param <V>
38      *     type of {@code Pair} value
39      * @param q
40      *     the {@code Queue} to be searched
41      * @param key
42      *     the key to be searched for
43      * @updates q
44      * @ensures <pre>
45      *     perms(q, #q) and
46      *     if there exists value: V (<(key, value)> is substring of q)
47      *     then there exists value: V (<(key, value)> is prefix of q)
48      * </pre>
49      */
50     private static <K, V> void moveToFront(Queue<Pair<K, V>> q, K key) {
51         assert q != null : "Violation of: q is not null";
52         assert key != null : "Violation of: key is not null";
53
54         Pair<K, V> temp = null;
55
56         for (int i = 0; i < q.length(); i++) {
57             if (q.front().key().equals(key)) {
58                 i = q.length();

```

```

65         } else {
66             temp = q.dequeue();
67             q.enqueue(temp);
68         }
69     }
70
71 }
72
73 /**
74  * Creator of initial representation.
75  */
76 private void createNewRep() {
77     this.pairsQueue = new Queue1L<Pair<K, V>>();
78 }
79
80 /*
81  * Constructors -----
82  */
83
84 /**
85  * No-argument constructor.
86  */
87 public Map2() {
88     this.createNewRep();
89 }
90
91 /*
92  * Standard methods -----
93  */
94
95 @SuppressWarnings("unchecked")
96 @Override
97 public final Map<K, V> newInstance() {
98     try {
99         return this.getClass().getConstructor().newInstance();
100     } catch (ReflectiveOperationException e) {
101         throw new AssertionError(
102             "Cannot construct object of type " + this.getClass());
103     }
104 }
105
106 @Override
107 public final void clear() {
108     this.createNewRep();
109 }
110
111 @Override
112 public final void transferFrom(Map<K, V> source) {
113     assert source != null : "Violation of: source is not null";
114     assert source != this : "Violation of: source is not this";
115     assert source instanceof Map2<?, ?> : ""
116         + "Violation of: source is of dynamic type Map2<?,?>";
117     /*
118      * This cast cannot fail since the assert above would have stopped
119      * execution in that case: source must be of dynamic type Map2<?,?>, and
120      * the ?,? must be K,V or the call would not have compiled.
121      */
122     Map2<K, V> localSource = (Map2<K, V>) source;
123     this.pairsQueue = localSource.pairsQueue;

```

```
124     localSource.createNewRep();
125 }
126
127 /*
128  * Kernel methods -----
129  */
130
131 @Override
132 public final void add(K key, V value) {
133     assert key != null : "Violation of: key is not null";
134     assert value != null : "Violation of: value is not null";
135     assert !this.hasKey(key) : "Violation of: key is not in DOMAIN(this)";
136
137     Pair<K, V> pair = new SimplePair<>(key, value);
138
139     this.pairsQueue.enqueue(pair);
140
141 }
142
143 @Override
144 public final Pair<K, V> remove(K key) {
145     assert key != null : "Violation of: key is not null";
146     assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";
147
148     Pair<K, V> pair = null;
149
150     moveToFront(this.pairsQueue, key);
151
152     pair = this.pairsQueue.dequeue();
153
154     return pair;
155 }
156
157 @Override
158 public final Pair<K, V> removeAny() {
159     assert this.size() > 0 : "Violation of: |this| > 0";
160
161     return this.pairsQueue.dequeue();
162 }
163
164 @Override
165 public final V value(K key) {
166     assert key != null : "Violation of: key is not null";
167     assert this.hasKey(key) : "Violation of: key is in DOMAIN(this)";
168
169     moveToFront(this.pairsQueue, key);
170
171     return this.pairsQueue.front().value();
172 }
173
174 @Override
175 public final boolean hasKey(K key) {
176     assert key != null : "Violation of: key is not null";
177
178     boolean hasKey = false;
179     Pair<K, V> temp = null;
180
181     for (int i = 0; i < this.pairsQueue.length(); i++) {
182         temp = this.pairsQueue.dequeue();
```

```
183
184         if (temp.key().equals(key)) {
185             hasKey = true;
186         }
187
188         this.pairsQueue.enqueue(temp);
189     }
190
191     return hasKey;
192 }
193
194 @Override
195 public final int size() {
196
197     return this.pairsQueue.length();
198 }
199
200 @Override
201 public final Iterator<Pair<K, V>> iterator() {
202     return this.pairsQueue.iterator();
203 }
204
205 }
206
```