

Solution 1

```
1 // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3 import java.util.*;
4
5 class Program {
6     static class LRUCache {
7         Map<String, DoublyLinkedListNode> cache = new HashMap<String, DoublyLinkedListNode>();
8         int maxSize;
9         int currentSize = 0;
10        DoublyLinkedList listOfMostRecent = new DoublyLinkedList();
11
12        public LRUCache(int maxSize) {
13            this.maxSize = maxSize > 1 ? maxSize : 1;
14        }
15
16        // O(1) time | O(1) space
17        public void insertKeyValuePair(String key, int value) {
18            if (!cache.containsKey(key)) {
19                if (currentSize == maxSize) {
20                    evictLeastRecent();
21                } else {
22                    currentSize++;
23                }
24                cache.put(key, new DoublyLinkedListNode(key, value));
25            } else {
26                replaceKey(key, value);
27            }
28            updateMostRecent(cache.get(key));
29        }
30
31        // O(1) time | O(1) space
32        public LRUResult getValueFromKey(String key) {
33            if (!cache.containsKey(key)) {
34                return new LRUResult(false, -1);
35            }
36            updateMostRecent(cache.get(key));
37            return new LRUResult(true, cache.get(key).value);
38        }
39
40        // O(1) time | O(1) space
41        public String getMostRecentKey() {
42            return listOfMostRecent.head.key;
43        }
44
45        public void evictLeastRecent() {
46            String keyToRemove = listOfMostRecent.tail.key;
47            listOfMostRecent.removeTail();
48            cache.remove(keyToRemove);
49        }
50
51        public void updateMostRecent(DoublyLinkedListNode node) {
52            listOfMostRecent.setHeadTo(node);
53        }
54
55        public void replaceKey(String key, int value) {
56            if (!this.cache.containsKey(key)) {
57                return;
58            }
59            cache.get(key).value = value;
60        }
61    }
62
63    static class DoublyLinkedList {
64        DoublyLinkedListNode head = null;
65        DoublyLinkedListNode tail = null;
66
67        public void setHeadTo(DoublyLinkedListNode node) {
68            if (head == node) {
69                return;
70            } else if (head == null) {
71                head = node;
72                tail = node;
73            } else if (head == tail) {
74                tail.prev = node;
75                head = node;
76                head.next = tail;
77            } else {
78                if (tail == node) {
79                    removeTail();
80                }
81                node.removeBindings();
82                head.prev = node;
83                node.next = head;
84                head = node;
85            }
86        }
87
88        public void removeTail() {
89            if (tail == null) {
90                return;
91            }
92            if (tail == head) {
93                head = null;
94                tail = null;
95                return;
96            }
97            tail = tail.prev;
98            tail.next = null;
99        }
100    }
101
102    static class DoublyLinkedListNode {
103        String key;
104        int value;
105        DoublyLinkedListNode prev = null;
106        DoublyLinkedListNode next = null;
107
108        public DoublyLinkedListNode(String key, int value) {
109            this.key = key;
110            this.value = value;
111        }
112
113        public void removeBindings() {
114            if (prev != null) {
115                prev.next = next;
```

```
116     }
117     if (next != null) {
118         next.prev = prev;
119     }
120     prev = null;
121     next = null;
122 }
123 }
124
125 static class LRUResult {
126     boolean found;
127     int value;
128
129     public LRUResult(boolean found, int value) {
130         this.found = found;
131         this.value = value;
132     }
133 }
134 }
135
```