Solution 1

```java
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

import java.util.*;

class Program {
  static class LRUCache {
    Map<String, DoublyLinkedListNode> cache = new HashMap<String, DoublyLinkedListNode>();
    int maxSize;
    int currentSize = 0;
    DoublyLinkedList listOfMostRecent = new DoublyLinkedList();

    public LRUCache(int maxSize) {
      this.maxSize = maxSize > 1 ? maxSize : 1;
    }

    // O(1) time | O(1) space
    public void insertKeyValuePair(String key, int value) {
      if (!cache.containsKey(key)) {
        if (currentSize == maxSize) {
          evictLeastRecent();
        } else {
          currentSize++;
        }
        cache.put(key, new DoublyLinkedListNode(key, value));
      } else {
        replaceKey(key, value);
      }
      updateMostRecent(cache.get(key));
    }

    // O(1) time | O(1) space
    public LRUResult getValueFromKey(String key) {
      if (!cache.containsKey(key)) {
        return new LRUResult(false, -1);
      }
      updateMostRecent(cache.get(key));
      return new LRUResult(true, cache.get(key).value);
    }

    // O(1) time | O(1) space
    public String getMostRecentKey() {
      return listOfMostRecent.head.key;
    }

    public void evictLeastRecent() {
      String keyToRemove = listOfMostRecent.tail.key;
      listOfMostRecent.removeTail();
      cache.remove(keyToRemove);
    }

    public void updateMostRecent(DoublyLinkedListNode node) {
      listOfMostRecent.setHeadTo(node);
    }

    public void replaceKey(String key, int value) {
      if (!this.cache.containsKey(key)) {
        return;
      }
      cache.get(key).value = value;
    }
  }

  static class DoublyLinkedList {
    DoublyLinkedListNode head = null;
    DoublyLinkedListNode tail = null;

    public void setHeadTo(DoublyLinkedListNode node) {
      if (head == node) {
        return;
      } else if (head == null) {
        head = node;
        tail = node;
      } else if (head == tail) {
        tail.prev = node;
        head = node;
        head.next = tail;
      } else {
        if (tail == node) {
          removeTail();
        }
        node.removeBindings();
        head.prev = node;
        node.next = head;
        head = node;
      }
    }

    public void removeTail() {
      if (tail == null) {
        return;
      }
      if (tail == head) {
        head = null;
        tail = null;
        return;
      }
      tail = tail.prev;
      tail.next = null;
    }
  }

  static class DoublyLinkedListNode {
    String key;
    int value;
    DoublyLinkedListNode prev = null;
    DoublyLinkedListNode next = null;

    public DoublyLinkedListNode(String key, int value) {
      this.key = key;
      this.value = value;
    }

    public void removeBindings() {
      if (prev != null) {
        prev.next = next;
```

```
      }
      if (next != null) {
        next.prev = prev;
      }
      prev = null;
      next = null;
    }
  }

  static class LRUResult {
    boolean found;
    int value;

    public LRUResult(boolean found, int value) {
      this.found = found;
      this.value = value;
    }
  }
}

```