Least Recently Used(LRU)，即最近最少使用页面置换算法。**选择最长时间没有被引用的页面进行置换**，思想是：如果一个页面很久没有被引用到，那么可以认为在将来该页面也很少被访问到。当发生缺页（CPU要访问的页不在内存中），计算内存中每个页上一次被访问的时间，置换上次使用到当前时间最长的一个页面。

java如何实现？可以使用**双向链表+哈希表**的方式。**HashMap主要是为了判断是否命中缓存。LinkedList用于维护一个按最近一次访问时间排序的页面链表。**链表头结点是最近刚刚访问过的页面，链表尾结点是最久未被访问的页面。访问内存时，若命中缓存，找到响应的页面，将其移动到链表头部，表示该页面是最近刚刚访问的。缺页时，将链表尾部的页面移除，同时新页面放到链表头。

该类有四个方法：

- moveToFirst()：把该元素移动链表的头部
- removeLast()：把链表尾部元素删除。
- get()：同步获取元素,map未命中，返回null。map命中则获取，并调用moveToFirst。
- put()：同步放入元素。如果map未命中，如果链表长度已经超过缓存的大小，移除链表尾部的元素，把元素放入链表的头部和map里。如果map命中，则moveToFirst把该元素移到链表的头。

```
public class LRUCache<K, V> {
    public static void main(String[] args) {
        LRUCache<String,String> lru=new LRUCache<>(10);
        lru.put("C", null);
        lru.put("A", null);
        lru.put("D", null);
        lru.put("B", null);
        lru.put("E", null);
        lru.put("B", null);
        lru.put("A", null);
        lru.put("B", null);
        lru.put("C", null);
        lru.put("D", null);

        System.out.println(lru);
        /*
        out:[D, C, B, A, E]
        */

    }
    //缓存大小，put时候需判断有没超过缓存大小
    private final int cacheSize;
    //用散列表判断是否命中缓存。
    private HashMap<K, V> map = new HashMap<>();
    //用链表维护最近一次访问时间排序的页面链表
    private LinkedList<K> linkCache = new LinkedList();

    LRUCache(int cacheSize) {
        this.cacheSize = cacheSize;
    }

    private synchronized void removeLast() {
        linkCache.removeLast();
    }
```

```
//用syn保证线程安全
    public synchronized void put(K key, V val) {
        if (!map.containsKey(key)) {
            if (map.size() >= cacheSize) {
                removeLast();
            }
            map.put(key, val);
            linkCache.addFirst(key);//put加到链表头
        } else {
            moveToFirst(key);
        }
    }

    public synchronized V get(K key) {
        if (!map.containsKey(key)) {
            return null;
        }
        moveToFirst(key);
        return map.get(key);
    }

    private synchronized void moveToFirst(K key) {
        linkCache.remove(key);
        linkCache.addFirst(key);

    }

    @Override
    public String toString() {
        return linkCache.toString();
    }
}
```