```java
package stacksqueues;

/* We can use Java inbuilt Deque as a double
ended queue to store the cache keys, with
the descending time of reference from front
to back and a set container to check presence
of a key. But remove a key from the Deque using
remove(), it takes O(N) time. This can be
optimized by storing a reference (iterator) to
each key in a hash map. */
import java.util.Deque;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Iterator;

 class LRUCache {

    // store keys of cache
    private Deque<Integer> doublyQueue;

    // store references of key in cache
    private HashSet<Integer> hashSet;

    // maximum capacity of cache
    private final int CACHE_SIZE;

    LRUCache(int capacity) {
        doublyQueue = new LinkedList<>();
        hashSet = new HashSet<>();
        CACHE_SIZE = capacity;
    }

    /* Refer the page within the LRU cache */
    public void refer(int page) {
        if (!hashSet.contains(page)) {
            if (doublyQueue.size() == CACHE_SIZE) {
                int last = doublyQueue.removeLast();
                hashSet.remove(last);
            }
        }
        else {/* The found page may not be always the last element, even if it's an
            intermediate element that needs to be removed and added to the start
            of the Queue */
            doublyQueue.remove(page);
        }
        doublyQueue.push(page);
        hashSet.add(page);
    }

    // display contents of cache
    public void display() {
        Iterator<Integer> itr = doublyQueue.iterator();
        while (itr.hasNext()) {
            System.out.print(itr.next() + " ");
        }
    }

    public static void main(String[] args) {
        LRUCache cache = new LRUCache(4);
        cache.refer(1);
        cache.refer(2);
        cache.refer(3);
        cache.refer(1);
        cache.refer(4);
        cache.refer(5);
        cache.refer(2);
        cache.refer(2);
        cache.refer(1);
        cache.display();
    }
}
// This code is contributed by Niraj Kumar
```