implement-queue-using-stacks

```
package algorithm.struct;

import java.util.Stack;

/**
 * https://leetcode.com/problems/implement-queue-using-stacks/

 Implement the following operations of a queue using stacks.

 push(x) -- Push element x to the back of queue.
 pop() -- Removes the element from in front of queue.
 peek() -- Get the front element.
 empty() -- Return whether the queue is empty.
 Notes:
 You must use only standard operations of a stack -- which means only push
 Depending on your language, stack may not be supported natively. You may s
 You may assume that all operations are valid (for example, no pop or peek

 * @author xiaobaoqiu  Date: 16-6-2 Time:  下午10:17
 */
public class ImplementQueueUsingStacks {
    public static void main(String[] args) {
        MyQueue queue = new MyQueue();
        queue.push(1);
        queue.push(2);
        System.out.println(queue.peek());
        queue.pop();
        System.out.println(queue.peek());
        queue.pop();
        queue.push(3);
        System.out.println(queue.peek());
    }

    /**
     * 134 ms
     * Your runtime beats 3.24% of java submissions
     */
    static class MyQueue {

        private Stack<Integer> in = new Stack<Integer>();
        private Stack<Integer> out = new Stack<Integer>();

        // Push element x to the back of queue.
        public void push(int x) {
            in.push(x);
        }

        // Removes the element from in front of queue.
        public void pop() {
            if (out.isEmpty()) {
                transfer();
            }

            out.pop();
        }

        // Get the front element.
        public int peek() {
            if (out.isEmpty()) {
                transfer();
            }

            return out.peek();
        }

        // Return whether the queue is empty.
        public boolean empty() {
            return in.isEmpty() && out.isEmpty();
        }

        private void transfer() {
            while(!in.isEmpty()) {
                out.push(in.peek());
                in.pop();
            }
        }
    }
}
```

implement-stack-using-queues

```
package algorithm.struct;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

/**
 * https://leetcode.com/problems/implement-stack-using-queues/
 * <p/>
 * Implement the following operations of a stack using queues.
 * <p/>
 * push(x) -- Push element x onto stack.
 * pop() -- Removes the element on top of the stack.
 * top() -- Get the top element.
 * empty() -- Return whether the stack is empty.
 * Notes:
 * You must use only standard operations of a queue --
 * which means only push to back, peek/pop from front, size, and is empty c
 * <p/>
 * Depending on your language, queue may not be supported natively.
 * You may simulate a queue by using a list or deque (double-ended queue),
 * as long as you use only standard operations of a queue.
 * <p/>
 * You may assume that all operations are valid
 * (for example, no pop or top operations will be called on an empty stack)
 * <p/>
 * Update (2015-06-11):
 * The class name of the Java function had been updated to MyStack instead
 *
 * @author xiaobaoqiu  Date: 16-7-1 Time:  上午12:57
 */
public class ImplementStackUsingQueues {

    public static void main(String[] args) {
        MyStack stack = new MyStack();
        stack.push(1);
        stack.push(2);
        System.out.println(stack.top());
        System.out.println(stack.top());
        stack.empty();
    }

    /**
     * 105 ms
     * Your runtime beats 87.95% of java submissions.
     */
    static class MyStack {
        private Queue<Integer> left = new LinkedList<Integer>();
        private Queue<Integer> right = new LinkedList<Integer>();
        private boolean isLeft = true;

        // Push element x onto stack.
        public void push(int x) {
            if (isLeft) left.offer(x);
            else right.offer(x);
        }

        // Removes the element on top of the stack.
        public void pop() {
            if (isLeft) {
                tranform(left, right);
                isLeft = false;
                left.poll();
            } else {
                tranform(right, left);
                isLeft = true;
                right.poll();
            }
        }

        // Get the top element.
        public int top() {
            if (isLeft) {
                tranform(left, right);
                return left.peek();
            } else {
                tranform(right, left);
                return right.peek();
            }
        }

        // Return whether the stack is empty.
        public boolean empty() {
            return left.isEmpty() && right.isEmpty();
        }

        /**
         * tranform
         */
        private void tranform(Queue<Integer> from, Queue<Integer> to) {
            if (from.size() == 1) return;

            while (from.size() > 1) {
                to.offer(from.poll());
            }
        }
    }
}
```

min-stack

```java
package algorithm.struct;

/**
 * https://leetcode.com/problems/min-stack/
 *
 Design a stack that supports push, pop, top,
 and retrieving the minimum element in constant time.

 push(x) -- Push element x onto stack.
 pop() -- Removes the element on top of the stack.
 top() -- Get the top element.
 getMin() -- Retrieve the minimum element in the stack.

 Example:
 MinStack minStack = new MinStack();
 minStack.push(-2);
 minStack.push(0);
 minStack.push(-3);
 minStack.getMin();   --> Returns -3.
 minStack.pop();
 minStack.top();      --> Returns 0.
 minStack.getMin();   --> Returns -2.

 * @author xiaobaoqiu  Date: 16-7-11 Time:  下午10:45
 */
public class MinStack {
    public static void main(String[] args) {

    }
//
//    public class MinStack {
//
//        /** initialize your data structure here. */
//        public MinStack() {
//
//        }
//
//        public void push(int x) {
//
//        }
//
//        public void pop() {
//
//        }
//
//        public int top() {
//
//        }
//
//        public int getMin() {
//
//        }
//    }
}
```