225 用队列实现栈

Label：栈

请你仅使用两个队列实现一个后入先出（LIFO）的栈，并支持普通队列的全部四种操作（push、top、pop 和 empty）。
实现 MyStack 类：
    void push(int x) 将元素 x 压入栈顶。
    int pop() 移除并返回栈顶元素。
    int top() 返回栈顶元素。
    boolean empty() 如果栈是空的，返回 true ；否则，返回 false 。

- 两个队列 实现 栈

```
class MyStack {

    Queue<Integer> inQueue = null;
    Queue<Integer> outQueue = null;
    /** Initialize your data structure here. */
    public MyStack() {
        inQueue = new LinkedList<>();
        outQueue = new LinkedList<>();
    }

    /** Push element x onto stack. */
    public void push(int x) {
        inQueue.add(x);  // 每次 inQueue 中加入的都是队头元素
        while (!outQueue.isEmpty()) {
            inQueue.add(outQueue.poll());  // 将 outQueue 中元素倒给 inQueue ，相当于把 栈顶 与 原栈 接上
        }
        // 相互交换后，outQueue的队列信息就是栈，而 outQueue 是一个空队列，等待下一轮的 栈顶元素
        Queue temp = inQueue;
        inQueue = outQueue;
        outQueue = temp;
    }

    /** Removes the element on top of the stack and returns that element. */
    public int pop() {
        return outQueue.poll();
    }

    /** Get the top element. */
    public int top() {
        return outQueue.peek();
    }

    /** Returns whether the stack is empty. */
    public boolean empty() {
        return outQueue.isEmpty();
    }
}
```

- 一个队列实现栈

```
class MyStack {
    Queue<Integer> queue;

    /** Initialize your data structure here. */
    public MyStack() {
        queue = new LinkedList<Integer>();
    }

    /** Push element x onto stack. */
    public void push(int x) {
        int n = queue.size();
        queue.offer(x);
        for (int i = 0; i < n; i++) {
            queue.offer(queue.poll());
        }
    }

    /** Removes the element on top of the stack and returns that element. */
    public int pop() {
        return queue.poll();
    }

    /** Get the top element. */
    public int top() {
        return queue.peek();
    }

    /** Returns whether the stack is empty. */
    public boolean empty() {
        return queue.isEmpty();
    }
}
```