● Queue를 두 개 이용하여 Stack을 구현해라.

▶ push()를 구현할 때 첫번째 Queue를 사용하고, pop()을 구현할 때 두번째 Queue를 사용하면, Stack을 구현할 수 있습니다. 일단 첫번째 큐에 enqueue를 하여 데이터를 저장하는 것으로 push()를 구현합니다. 다음으로 pop()을 구현하는 방법은, 첫번째 큐에 저장된 데이터 갯수가 1개 이하가 될 때까지 dequeue()를 한 후에, 추출한 데이터를 두번째 큐에 enqueue()합니다. 결과적으로 가장 나중에, 혹은 최근에 들어온 데이터를 제외한 모든 데이터는 두번째 큐로 옮겨집니다. 그런 다음, 첫번째 큐에 남아 있는 하나의 데이터를 dequeue() 하면 결과적으로 가장 나중에, 혹은 최근에 저장된 데이터를 반환하고, 이는 Last In First Out(LIFO)으로 스택을 구현하게 됩니다. 마지막으로, 첫번째 큐와 두번째 큐의 이름을 swap하면 다음에 진행될 pop()을 동일한 알고리즘으로 진행할 수 있습니다.

```java
import java.util.*;
import java.util.Queue;
// 첫번째 큐에 항목을 추가. 첫번째 큐에 마지막 항목 외 나머지 항목을 두번째 큐로 옮긴다. 첫번째

class MyStack<T> {
    Queue<T> queueFirst, queueSecond;

    // 생성자에서 두 개의 큐를 만든다
    MyStack() {
        queueFirst = new LinkedList<>();
        queueSecond = new LinkedList<>();
    }

    // push: 첫번째 queue에 enqueue하여 데이터를 저장한다
    public void push(T value) {
        queueFirst.add(value);
    }

    // pop: 첫번째 큐에 저장된 데이터의 갯수가 1 이하로 남을 때까지
    // dequeue()를 한 뒤, 추출된 데이터를 두번째 큐에 enqueue()한다.
    // 결과적으로 가장 최근에 들어온 데이터를 제외한 모든 데이터가 두번째 큐로 옮겨진다.
    public T pop() {
        while (queueFirst.size() > 1) {
            queueSecond.add(queueFirst.remove());
        }

        // 다음에 진행될 pop()을 위와 같은 알고리즘으로 진행하기 위해,
        // 첫번째 큐와 두번째 큐의 이름은 swap한다
        Queue<T> tmp = queueFirst;
        queueFirst = queueSecond;
        queueSecond = tmp;

        return queueSecond.remove(); // 가장 최근에 들어온 데이터를 dequeue()한다 — LIFO

        }
    }

    // dequeue: 두번째 stack이 비어 있을 경우, 첫번째 stack이 완전히 빌 때까지
    // 두번째 stack으로 데이터를 옮기고, 두번째 stack에서 데이터를 pop한다.

class MakeStack {
    public static void main(String[] args) {
        MyStack<Integer> stack = new MyStack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
    }
}
```

● 시간복잡도

▶ push()의 경우, 첫번째 큐에 enqueue()를 한번만 하면 되기 때문에, O(1)의 시간복잡도를 갖습니다.

▶ pop()의 경우, 첫번째 큐에 저장되어 있는 n개의 원소중에 n-1개를 두번째 큐로 옮겨야 하므로, O(n)이 됩니다.