716.Max Stack

Design a max stack that supports push, pop, top, peekMax and popMax.

1. push(x) -- Push element x onto stack.
2. pop() -- Remove the element on top of the stack and return it.
3. top() -- Get the element on the top.
4. peekMax() -- Retrieve the maximum element in the stack.
5. popMax() -- Retrieve the maximum element in the stack, and remove it. If you find more than one maximum elements, only remove the top-most one.

Example 1:

MaxStack stack = new MaxStack();

stack.push(5);

stack.push(1);

stack.push(5);

stack.top(); -> 5

stack.popMax(); -> 5

stack.top(); -> 1

stack.peekMax(); -> 5

stack.pop(); -> 1

stack.top(); -> 5

Note:

1. -1e7 <= x <= 1e7
2. Number of operations won't exceed 10000.
3. The last four operations won't be called when stack is empty.

For `peekMax`, we remember the largest value we've seen on the side. For example if we add `[2, 1, 5, 3, 9]` in stack , we'll store `[2, 2, 5, 5, 9]` in maxStack. This works seamlessly

with `pop` operations, and also it's easy to compute: it's just the maximum of the element we are adding and the previous maximum.

For `popMax`, we know what the current maximum (`peekMax`) is. We can pop until we find that maximum, then push the popped elements back on the stack.

- Time Complexity: O(N) for the `popMax` operation, and O(1) for the other operations, where N is the number of operations performed.
- Space Complexity: O(N), the maximum size of the stack.

```
class MaxStack {
    Stack<Integer> stack;
    Stack<Integer> maxStack;
    public MaxStack() {
        stack = new Stack();
        maxStack = new Stack();
    }
    public void push(int x) {
        int max = maxStack.isEmpty() ? x : maxStack.peek();
        maxStack.push(max > x ? max : x);
        stack.push(x);
    }
    public int pop() {
        maxStack.pop();
        return stack.pop();
    }
    public int top() {
        return stack.peek();
    }
    public int peekMax() {
        return maxStack.peek();
    }
    public int popMax() {
        int max = peekMax();
        Stack<Integer> buffer = new Stack();
        while (top() != max) buffer.push(pop());
        pop();
        while (!buffer.isEmpty()) push(buffer.pop());
        return max;
    }
}
```