Design a max stack data structure that supports the stack operations and supports finding the stack's maximum element.

Implement the MaxStack class:

* MaxStack() Initializes the stack object.
* void push(int x) Pushes element x onto the stack.
* int pop() Removes the element on top of the stack and returns it.
* int top() Gets the element on the top of the stack without removing it.
* int peekMax() Retrieves the maximum element in the stack without removing it.
* int popMax() Retrieves the maximum element in the stack and removes it. If there is more than one maximum element, only remove the **top-most** one.

You must come up with a solution that supports O(1) for each top call and O(logn) for each other call.

**Example 1:**

Input  
["MaxStack", "push", "push", "push", "top", "popMax", "top", "peekMax", "pop", "top"]  
[[], [5], [1], [5], [], [], [], [], [], []]  
Output  
[null, null, null, null, 5, 5, 1, 5, 1, 5]  
  
Explanation  
MaxStack stk = new MaxStack();  
stk.push(5); // [5] the top of the stack and the maximum number is 5.  
stk.push(1); // [5, 1] the top of the stack is 1, but the maximum is 5.  
stk.push(5); // [5, 1, 5] the top of the stack is 5, which is also the maximum, because it is the top most one.  
stk.top(); // return 5, [5, 1, 5] the stack did not change.  
stk.popMax(); // return 5, [5, 1] the stack is changed now, and the top is different from the max.  
stk.top(); // return 1, [5, 1] the stack did not change.  
stk.peekMax(); // return 5, [5, 1] the stack did not change.  
stk.pop(); // return 1, [5] the top of the stack and the max element is now 5.  
stk.top(); // return 5, [5] the stack did not change.

**Constraints:**

* -107 <= x <= 107
* At most 105 calls will be made to push, pop, top, peekMax, and popMax.
* There will be **at least one element** in the stack when pop, top, peekMax, or popMax is called.