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 1:**

**Input**

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

**Output**

[null,null,null,null,-3,null,0,-2]

**Explanation**

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

minStack.getMin(); // return -2

就是让你给Stack加上一个getMin的功能

我们不能通过设置一个int数来记录，

因为假设这个最小值pop掉了，我们无法直接得到第二小的数，除非你把整个stack遍历一遍，但这样效率是n

我们可以创造一个额外stack min

如果说当前push小于等于min的顶那就push

而pop的时候如果当前stack与min的stack的peek相等，那就Pop掉min， 然后再pop 常规

class MinStack {

/\*\* initialize your data structure here. \*/

Stack<Integer> reg=new Stack();

Stack<Integer> small=new Stack();

public void push(int x) {

if (small.isEmpty()||x<=small.peek()) small.push(x);

reg.push(x);

}

public void pop() {

if (reg.peek().equals(small.peek())) small.pop();

reg.pop();

}

public int top() {

return reg.peek();

}

public int getMin() {

return small.peek();

}

}

/\*\*

\* Your MinStack object will be instantiated and called as such:

\* MinStack obj = new MinStack();

\* obj.push(x);

\* obj.pop();

\* int param\_3 = obj.top();

\* int param\_4 = obj.getMin();

\*/