LeetCode 155

Description

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:
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); --> Returns -3.
minStack.pop();
minStack.top(); --> Returns 0.
minStack.getMin(); --> Returns -2.

Thought

If only one stack, problem is getting the min, so think about using two stack, one performed as normal stack, other to store the min.

Two stack needs to store equal amount of numbers: first stack store each new number, while the second one compare the new number with current min, and store whichever is smaller.

Solution

```
class minStack {
// 1.use deque instead of stack to implement, because stack extends vector and dequ
// 2.private variable to encapsulate
  private Deque< Integer > stack;
  private Deque< Integer > minStack;

  //init the data structure
  public MinStack() {
    stack = new ArrayDeque<>();
    minStack = new ArrayDeque<>();
  }

  public void push(int x) {
    stack.addFirst(x);
    if (minStack.isEmpty() || minStack.getFirst() > x) {
      minStack.addFirst(x);
    } else {
      minStack.addFirst(minStack.getFirst());
    }
  }

  public void pop() {
    stack.removeFirst();
    MinStack.removeFirst();
  }

  public int top() {
    return stack.getFirst();
  }

  public int getMin() {
    return minStack.getFirst();
  }

}
```

Takeaways

- Use a data structure to save min or max value
- The data structure need to be compatible with stack