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

First, this is a Stack implementation. But this stack is a little bit special, it needs to keep the minimum element for each step. One variable won't work, because it need to keep min for each step, when push/pop happens, min will change.

What is stack? First in, Last out;
What data structure can be used to implement a traditional stack? ArrayDeque, Stack..

How to store minimum element?

1. Create a new object, for example, int[] e. e[0] is value, e[1] is minValue. benefit: easy to implement. Shortage: waste space

2. Use two stacks, one store value, one store

minValue, keep the same activity on 2 stacks.
benefit: have potential to save space in some senario

3. Enhancement to store minValue.
   When the push value is larger than existing top value in stack, which means it won't change the minValue, we don't need to store the same minValue in stack, we just need to push the new minValue when we got different minValue.
   However, if push value equals to existing top value, we still need to store the same minValue.
   Even more, we can store minValue as an array object int[] min, min[0] is minValue, min[1] is the counter.

Solution 1

```
class MinStack {
  private Deque<int[]> deque;

  public Minstack(){
    deque = new ArrayDeque<int[]>();
  }

  public void push(int x){
    int min = 0;
    if (deque.isEmpty()|| deque.getFirst()[1]> x) {
      min = x;
    } else {
      min = deque.getFirst()[1];
    }
    deque.addFirst(new int[]{x, min});
  }

  public void pop(){
    deque.removeFirst();
  }

  public int top(){
    return deque.getFirst()[0];
  }

  public int getMin(){
    return deque.getFirst()[1];
  }
}
```

Solution 2

```
class MinStack {
  private Deque<Interger> deque;
  private Deque<int[]> minDeque;

  public MinStack(){
    deque = new ArrayDeque<Interger>();
    minDeque = new ArrayDeque<int[]>();
  }

  public void push(int x){
    deque.addFirst(x);

    if (deque.isEmpty() || minDeque.getFirst()[0]> x) {
      minDeque.addFirst(new int[]{x, 1});
    } else {
      minDeque.getFirst[1]++;
    }
  }

  public void pop(){
    int removeValue = deque.removeFirst();
    if (deque.isEmpty) {
      minDeque.removeFirst();
      return;
    }
    if (removeValue = minDeque.getFirst()[0]) {
      minDeque.getFirst()[1]--;
      if (minDeque.getFirst()[1] < 1) {
        minDeque.removeFirst();
      }
    }
  }

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

  public int getMin(){
    return minDeque.getFirst()[0];
  }
}
```

Takeaways

- Use ArrayDeque to implement stack
- Use the same data Structure ato synchronize activity