Description

155. Min Stack

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.
```

Idea

We need somehow maintain the minimum number property. A data structure that is closest to a stack is stack itself, so we can use another stack to maintain the minimum number. Every time we push a number, we push the number in our original dataStack, and push the global minimum number in our minStack. How do we know the global minimum? The Math.min(x, minStack.peek()) is it. When pop out, we pop out both stacks.

Java

```java
class MinStack {

    private Deque<Integer> minStack;
    private Deque<Integer> dataStack;
    public MinStack() {
        minStack = new ArrayDeque<>();
        dataStack = new ArrayDeque<>();
    }

    public void push(int x) {
        dataStack.push(x);
        minStack.push((minStack.isEmpty() || x < minStack.peek()) ?
                       x : minStack.peek());
    }

    public void pop() {
        dataStack.pop();
        minStack.pop();
    }

    public int top() {
        return dataStack.peek();
    }

    public int getMin() {
        return minStack.peek();
    }
}
```

C++

```
class MinStack {
public:
    stack<int> dataStack;
    stack<int> minStack;
    MinStack() {

    }

    void push(int x) {
        dataStack.push(x);
        minStack.push((minStack.empty() || x < minStack.top()) ?
                       x : minStack.top());
    }

    void pop() {
        dataStack.pop();
        minStack.pop();
    }

    int top() {
        return dataStack.top();
    }

    int getMin() {
        return minStack.top();
    }
};
```

In the case the numbers pushed are mostly increasing, we don't have to push the minStack every time.

Java

```
class MinStack {

    private Deque<Integer> minStack;
    private Deque<Integer> dataStack;

    public MinStack() {
        minStack = new ArrayDeque<>();
        dataStack = new ArrayDeque<>();
    }

    public void push(int x) {
        dataStack.push(x);
        if (minStack.isEmpty() || x <= minStack.peek()) {
            minStack.push(x);
        }
    }

    public void pop() {
        if (dataStack.peek().equals(minStack.peek())) {
            minStack.pop();
        }
        dataStack.pop();

    }

    public int top() {
        return dataStack.peek();
    }

    public int getMin() {
        return minStack.peek();
    }
}
```

Summary

- Cache model, maintain a caching data structure.
- OOD