Solution: Min Stack

Let's solve the Min Stack problem using the Custom Data Structure pattern.

We'll cover the following

Statement

Design a custom stack class, **Min Stack**, allowing us to push, pop, and retrieve the minimum value in constant time. Implement the following methods for **Min Stack**:

- **Constructor:** This initializes the **Min Stack** object.
- **Pop():** This removes and returns from the stack the value that was most recently pushed onto it.
- **Push():** This pushes the provided value onto the stack.
- **Min Number():** This returns the minimum value in the stack in $O(1)$ time.

> **Note:** The time complexity of all the methods above should be $O(1)$.

Constraints:

- $-2^{31} \leq \textbf{value} \leq 2^{31} - 1$
- The **Pop()** and **Min Number()** methods will always be called on non-empty stacks.
- At most, 3×10^3 calls will be made to **Push()**, **Pop()**, and **Min Number()**.

Solution

As explained in the problem statement, we need to customize the standard stack data structure so that it supports retrieval of the minimum value in $O(1)$ time. This means our customization should not alter the performance characteristics of the standard **Push()** and **Pop()** operations of a stack, while also meeting the new requirement.

Referencing the statement above, the implementation of such a data structure can be realized with the help of two stacks—min stack and main stack. These stacks are initialized in the **Constructor**.

- The **main stack** holds the actual stack with all the elements.
- The **min stack** is a stack whose top always contains the current minimum value in the stack.

Few points to consider:

- Whenever **Push()** is called, the value is pushed into the main stack, and we check the following conditions:

    - If the min stack is empty or the value being pushed is less than the top value of the min stack, the new value is pushed to the min stack.
    - Otherwise, the current minimum value, present at the top of the min stack is pushed again in the min stack.
- The **Pop()** function removes the top element from the main stack and also from the min stack.
- The **Min Number()** function checks the min stack. If it is not empty, it returns the top of the min stack, which represents the current minimum value present in the stack.

The slide deck below illustrates the key steps of the solution.