JavaScript Implement Stack Using Queues

Challenge

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

> ⓘ Note
>
> Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

> Important
>
> You must use only standard operations of a queue, which means that only `push to back`, `peek/pop from front`, `size` and `is empty` operations are valid.

Example

```
Input:
['MyStack', 'push', 'push', 'top', 'pop', 'empty']
[[], [1], [2], [], [], []]
Output: [null, null, null, 2, 2, false]
Explanation: MyStack myStack = new MyStack();
             myStack.push(1);
             myStack.push(2);
             myStack.top(); // return 2
             myStack.pop(); // return 2
             myStack.empty(); // return False
```

Constraints

- `1 <= x <= 9`
- At most `100` calls will be made to `push`, `pop`, `top`, and `empty`.
- All the calls to `pop` and `top` are valid.

Solution

```
class MyStack {
    constructor() {
        this.queue = [];
    }

    push(x) {
        this.queue.push(x);
    }
```

Solution continues on next page...

```
    pop() {
        for (let i = 1; i < this.queue.length; i++) {
            this.queue.push(this.queue.shift());
        }

        return this.queue.shift();
    }

    top() {
        for (let i = 1; i < this.queue.length; i++) {
            this.queue.push(this.queue.shift());
        }

        let temp = this.queue.shift();

        this.queue.push(temp);

        return temp;
    }

    empty() {
        return this.queue.length === 0;
    }
};
```

Explanation

I've defined a class called `MyStack`. This class represents a stack data structure implemented using an array. This class has several methods for manipulating the stack.

The constructor initializes an empty array called `queue`.

The `push(x)` method adds an element `x` to the end of the `queue` array.

The `pop()` method removes and returns the top element of the stack. It does this by iterating through the `queue` array from index `1` to the end. For each iteration, it removes the first element of the `queue` array and adds it to the end. Finally, it removes and returns the first element of the `queue` array.

The `top()` method returns the top element of the stack without removing it. It follows the same process as the `pop()` method, but instead of removing the first element of the `queue` array at the end, it stores it in a variable called `temp`, adds it back to the end of the `queue` array, and then returns the value of `temp`.

The `empty()` method checks if the `queue` array is empty and returns a boolean value indicating whether it is empty or not.

In summary, this class implements a stack data structure using an array. The `push()` method adds elements to the stack, the `pop()` method removes and returns the top element, the `top()` method returns the top element without removing it, and the `empty()` method checks if the stack is empty.

Author: Trevor Morin