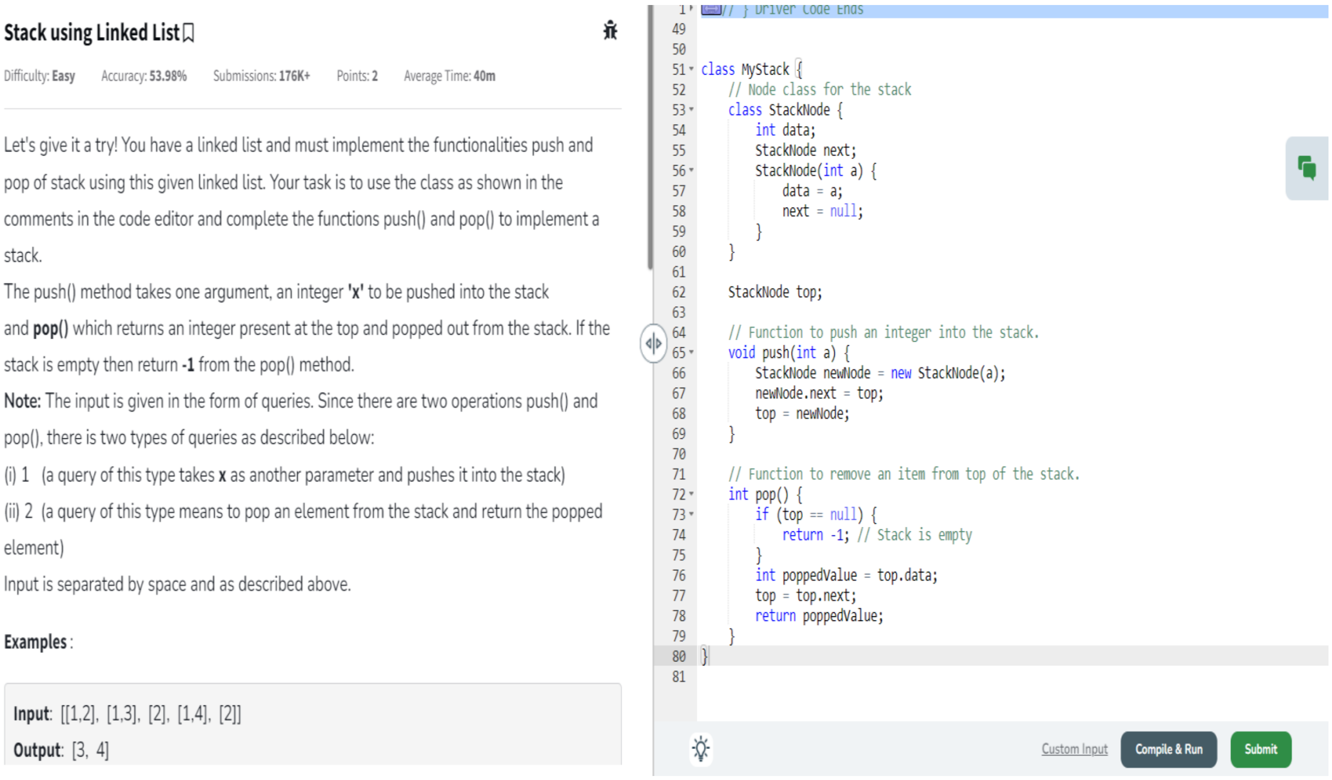
ADVANCED PROGRAMMING LAB – ASSIGNMENT 6

NAME- KESHAV KUMAR

UID- 22BCS14105

Ques 1 Implement Stack using Linked List



Stack using Linked List

Difficulty: **Easy** Accuracy: **53.98%** Submissions: **176K+** Points: **2** Average Time: **40m**

Let's give it a try! You have a linked list and must implement the functionalities push and pop of stack using this given linked list. Your task is to use the class as shown in the comments in the code editor and complete the functions push() and pop() to implement a stack.

The push() method takes one argument, an integer '**x**' to be pushed into the stack and **pop()** which returns an integer present at the top and popped out from the stack. If the stack is empty then return **-1** from the pop() method.

Note: The input is given in the form of queries. Since there are two operations push() and pop(), there is two types of queries as described below:

(i) 1 (a query of this type takes **x** as another parameter and pushes it into the stack)

(ii) 2 (a query of this type means to pop an element from the stack and return the popped element)

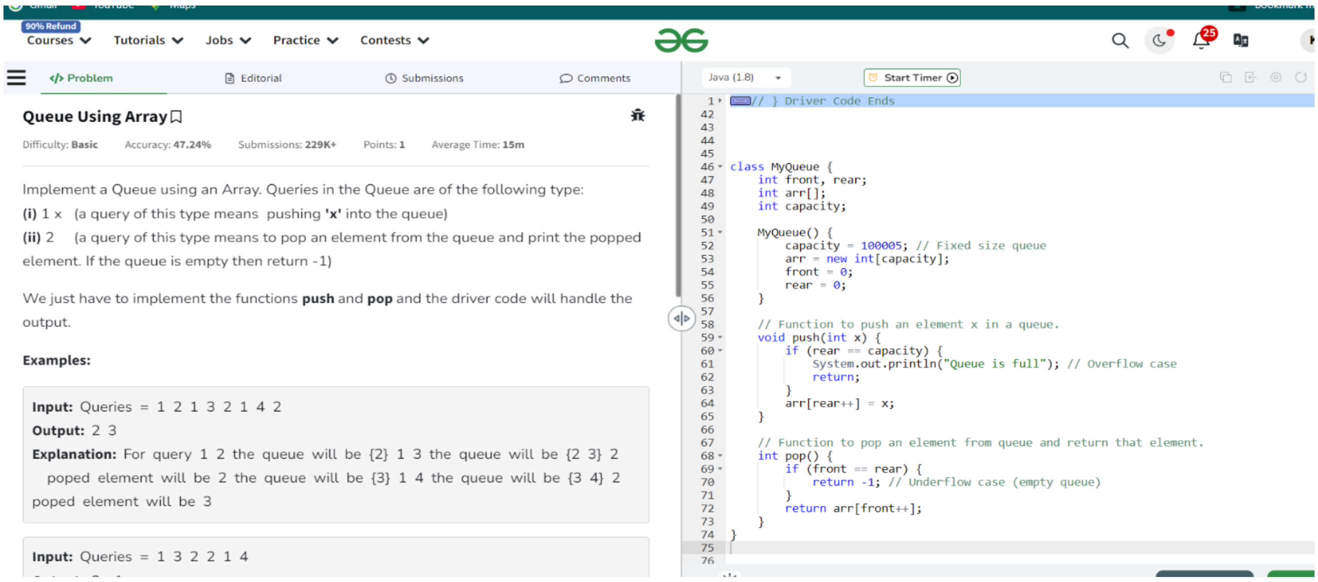
Input is separated by space and as described above.

Examples:

Input: [[1,2], [1,3], [2], [1,4], [2]]
Output: [3, 4]

```java
// } Driver Code Ends


class MyStack {
    // Node class for the stack
    class StackNode {
        int data;
        StackNode next;
        StackNode(int a) {
            data = a;
            next = null;
        }
    }

    StackNode top;

    // Function to push an integer into the stack.
    void push(int a) {
        StackNode newNode = new StackNode(a);
        newNode.next = top;
        top = newNode;
    }

    // Function to remove an item from top of the stack.
    int pop() {
        if (top == null) {
            return -1; // Stack is empty
        }
        int poppedValue = top.data;
        top = top.next;
        return poppedValue;
    }
}
```

Ques 2



Queue Using Array

Difficulty: **Basic** Accuracy: **47.24%** Submissions: **229K+** Points: **1** Average Time: **15m**

Implement a Queue using an Array. Queries in the Queue are of the following type:
(i) 1 x (a query of this type means pushing **'x'** into the queue)
(ii) 2 (a query of this type means to pop an element from the queue and print the popped element. If the queue is empty then return -1)

We just have to implement the functions **push** and **pop** and the driver code will handle the output.

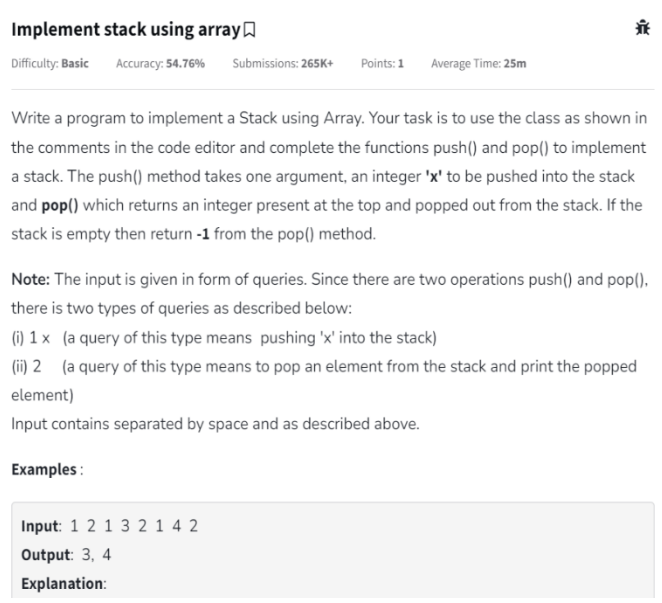
Examples:

Input: Queries = 1 2 1 3 2 1 4 2
Output: 2 3
Explanation: For query 1 2 the queue will be {2} 1 3 the queue will be {2 3} 2 poped element will be 2 the queue will be {3} 1 4 the queue will be {3 4} 2 poped element will be 3

Input: Queries = 1 3 2 2 1 4

```java
// } Driver Code Ends




class MyQueue {
    int front, rear;
    int arr[];
    int capacity;

    MyQueue() {
        capacity = 100005; // Fixed size queue
        arr = new int[capacity];
        front = 0;
        rear = 0;
    }

    // Function to push an element x in a queue.
    void push(int x) {
        if (rear == capacity) {
            System.out.println("Queue is full"); // Overflow case
            return;
        }
        arr[rear++] = x;
    }

    // Function to pop an element from queue and return that element.
    int pop() {
        if (front == rear) {
            return -1; // Underflow case (empty queue)
        }
        return arr[front++];
    }
}
```

Ques 3



Implement stack using array

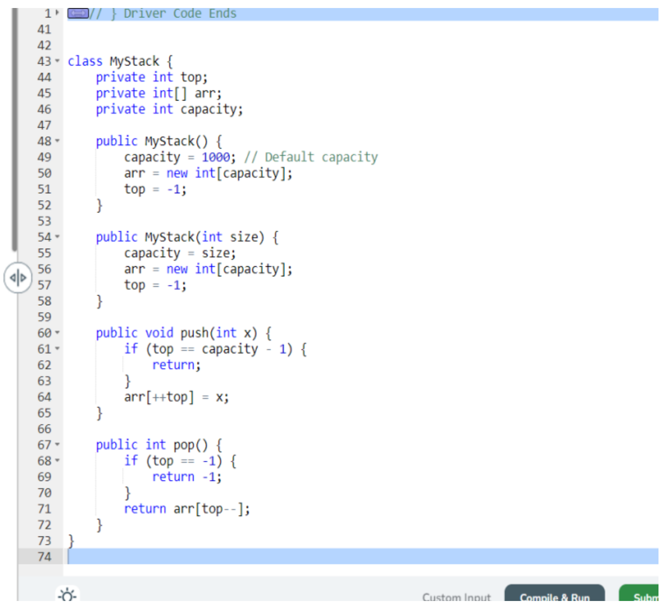
Difficulty: **Basic** Accuracy: **54.76%** Submissions: **265K+** Points: **1** Average Time: **25m**

Write a program to implement a Stack using Array. Your task is to use the class as shown in the comments in the code editor and complete the functions push() and pop() to implement a stack. The push() method takes one argument, an integer **'x'** to be pushed into the stack and **pop()** which returns an integer present at the top and popped out from the stack. If the stack is empty then return **-1** from the pop() method.

Note: The input is given in form of queries. Since there are two operations push() and pop(), there is two types of queries as described below:
(i) 1 x (a query of this type means pushing 'x' into the stack)
(ii) 2 (a query of this type means to pop an element from the stack and print the popped element)
Input contains separated by space and as described above.

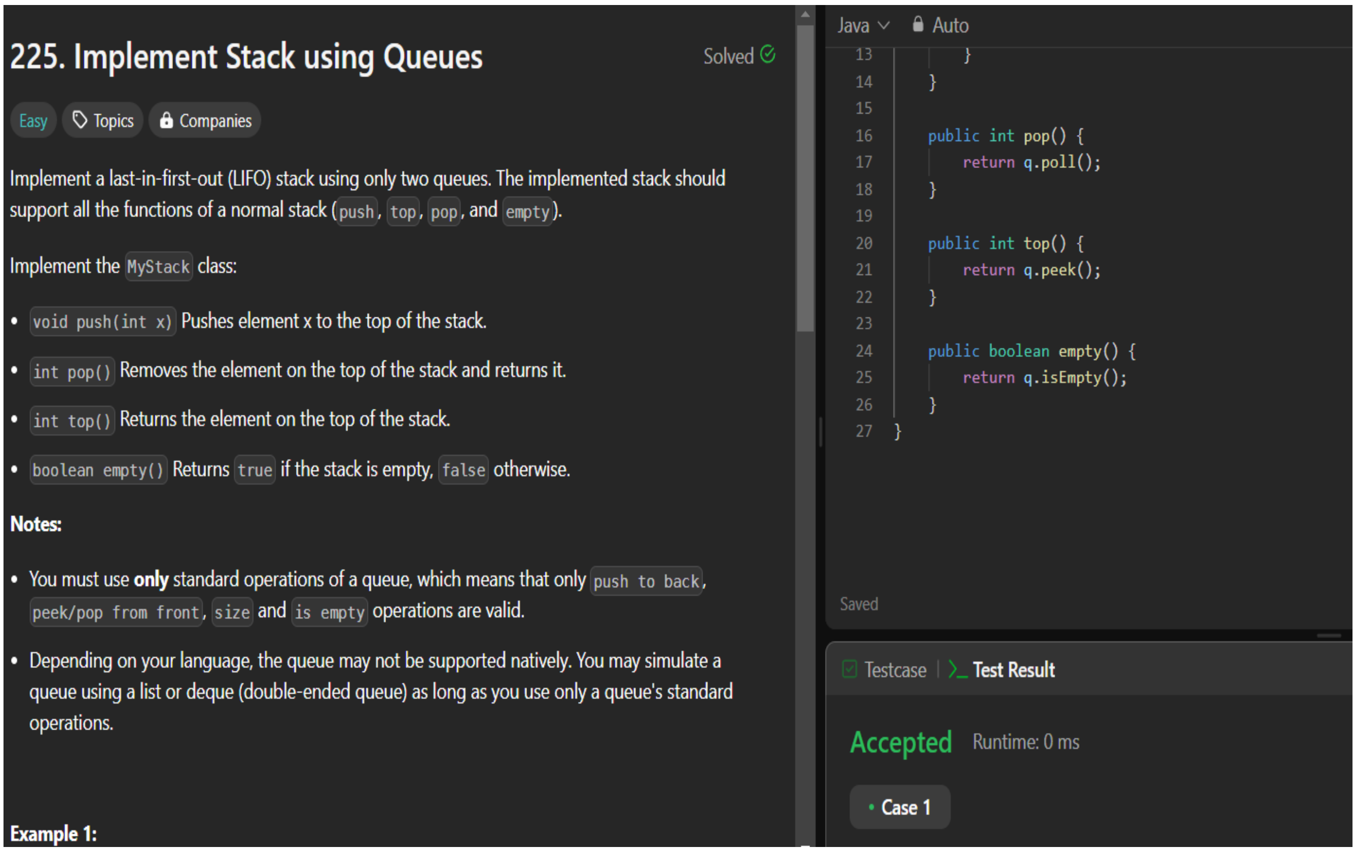
Examples :

Input: 1 2 1 3 2 1 4 2
Output: 3, 4
Explanation:



```java
// } Driver Code Ends


class MyStack {
    private int top;
    private int[] arr;
    private int capacity;

    public MyStack() {
        capacity = 1000; // Default capacity
        arr = new int[capacity];
        top = -1;
    }

    public MyStack(int size) {
        capacity = size;
        arr = new int[capacity];
        top = -1;
    }

    public void push(int x) {
        if (top == capacity - 1) {
            return;
        }
        arr[++top] = x;
    }

    public int pop() {
        if (top == -1) {
            return -1;
        }
        return arr[top--];
    }
}
```

Ques 4



225. Implement Stack using Queues

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.

Notes:

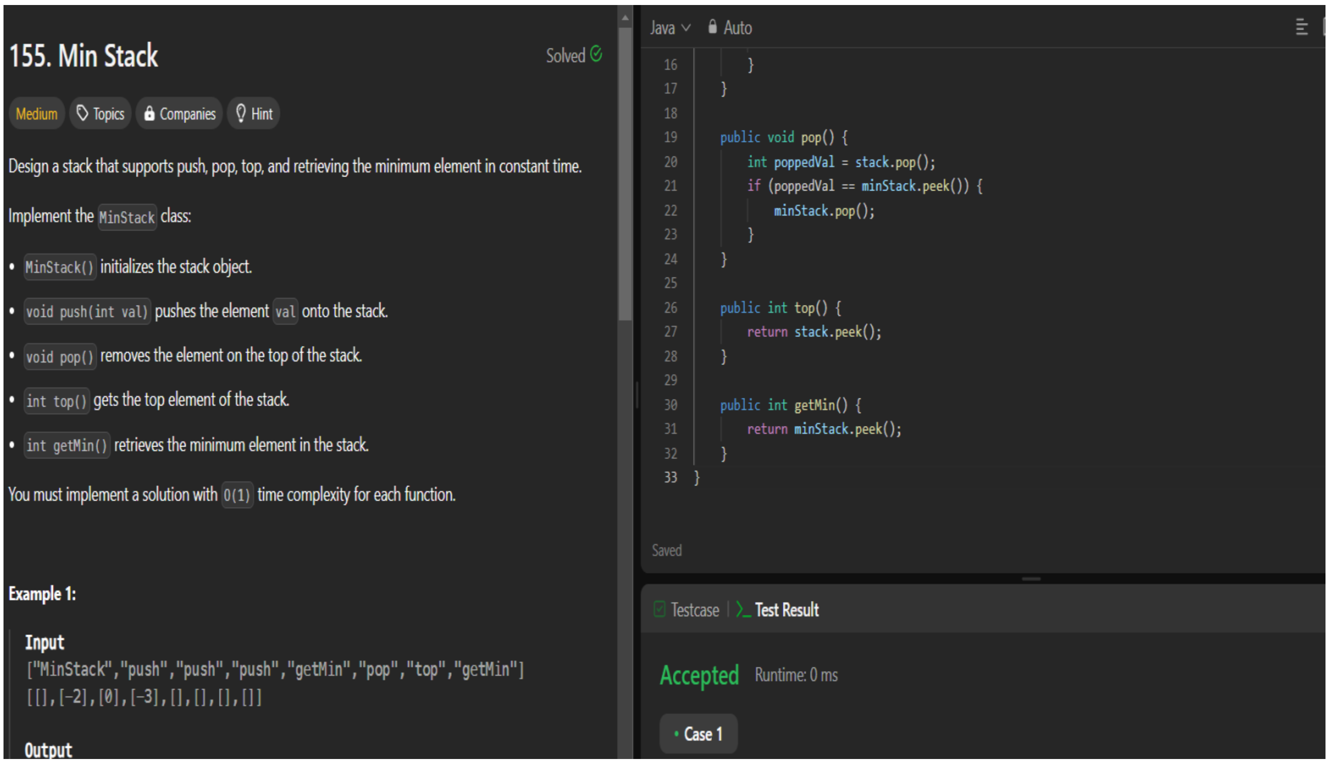
- You must use **only** standard operations of a queue, which means that only `push to back`, `peek/pop from front`, `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:

```java
        }
    }

    public int pop() {
        return q.poll();
    }

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

    public boolean empty() {
        return q.isEmpty();
    }
}
```

Accepted Runtime: 0 ms

Ques 5



155. Min Stack

Solved

Medium | Topics | Companies | Hint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

Input
```
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
```

Output

Java | Auto

```java
        }
    }

    public void pop() {
        int poppedVal = stack.pop();
        if (poppedVal == minStack.peek()) {
            minStack.pop();
        }
    }

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

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

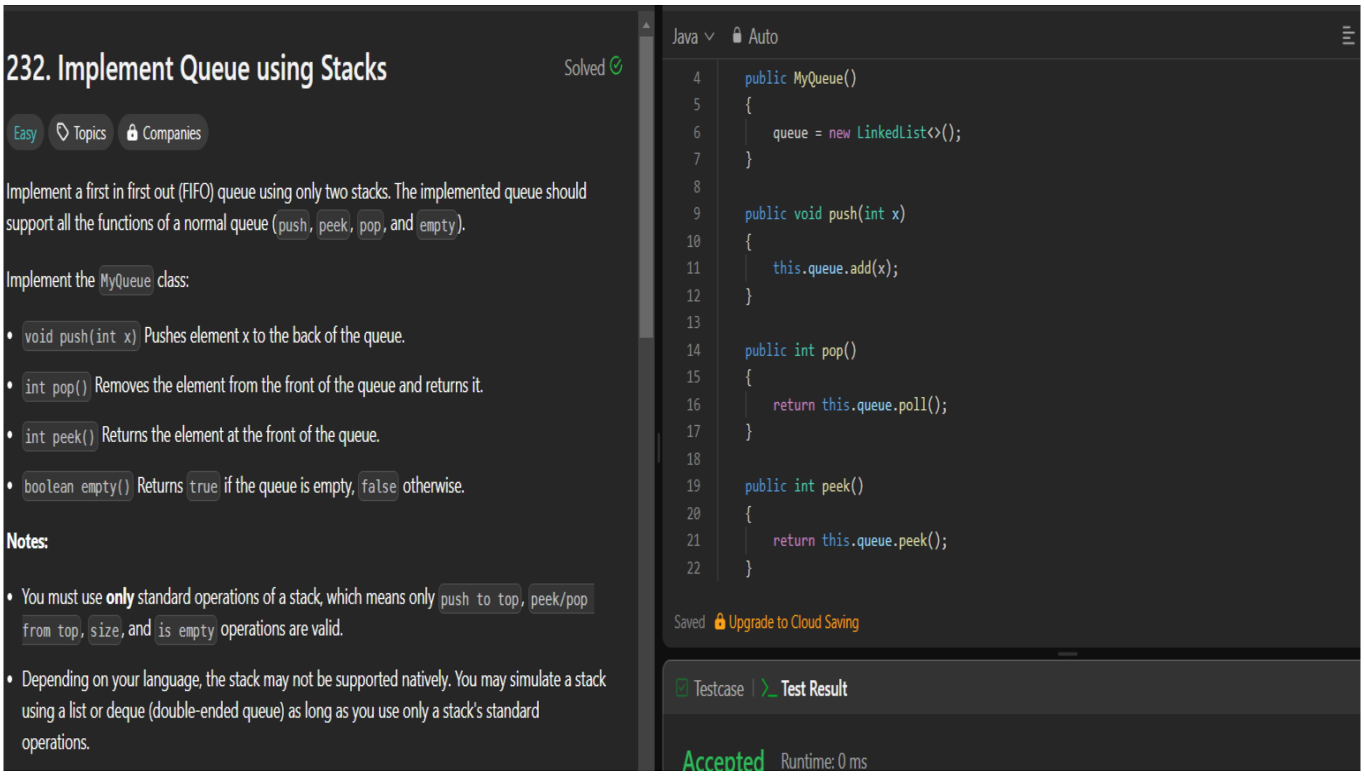
Saved

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1

Ques 6



232. Implement Queue using Stacks

Solved

Easy | Topics | Companies

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (`push`, `peek`, `pop`, and `empty`).

Implement the `MyQueue` class:

- `void push(int x)` Pushes element x to the back of the queue.
- `int pop()` Removes the element from the front of the queue and returns it.
- `int peek()` Returns the element at the front of the queue.
- `boolean empty()` Returns `true` if the queue is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a stack, which means only `push to top`, `peek/pop from top`, `size`, and `is empty` operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Java | Auto

```java
    public MyQueue()
    {
        queue = new LinkedList<>();
    }

    public void push(int x)
    {
        this.queue.add(x);
    }

    public int pop()
    {
        return this.queue.poll();
    }

    public int peek()
    {
        return this.queue.peek();
    }
```

Saved Upgrade to Cloud Saving

Testcase | Test Result

Accepted Runtime: 0 ms