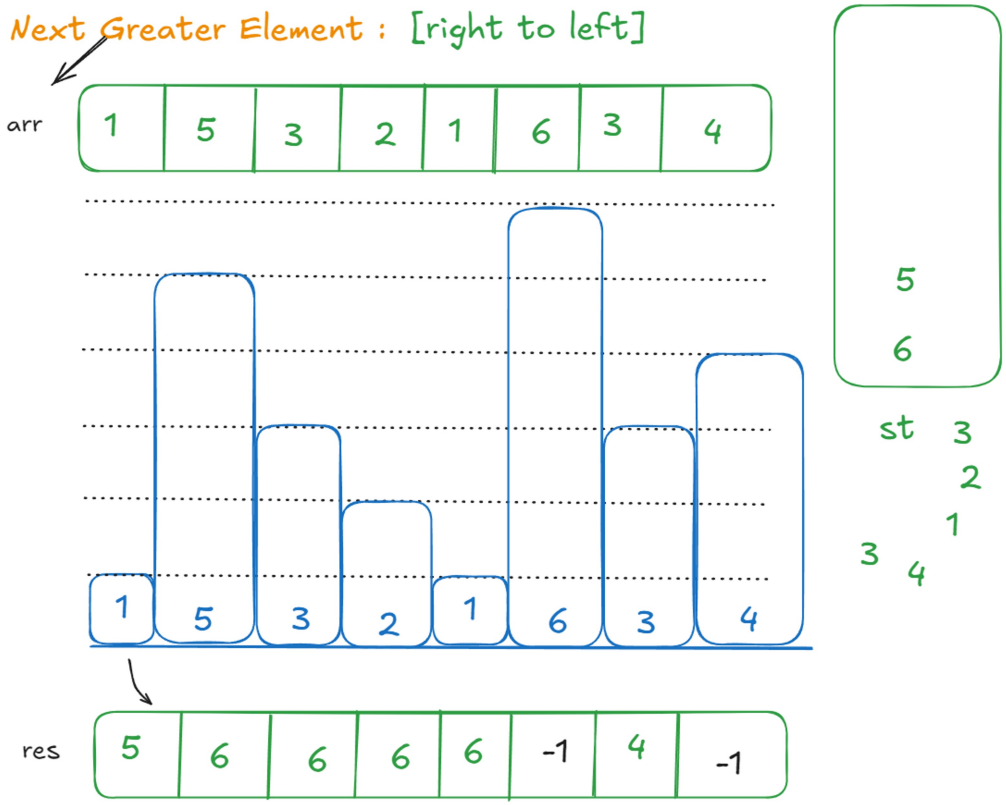
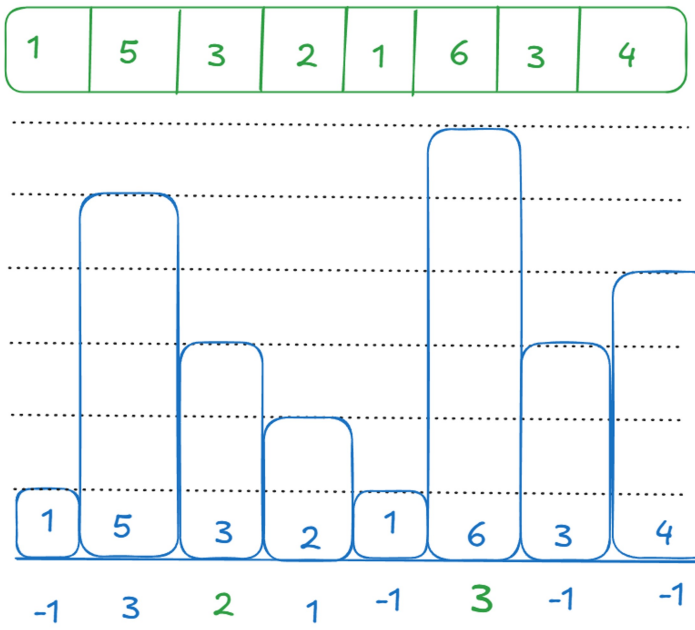
Stack & Queue [Day 17]

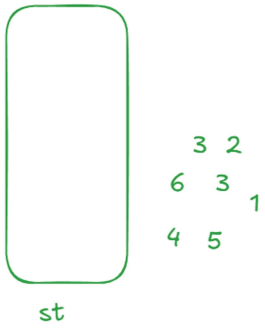
Next Greater Element : [right to left]




```
import java.util.Stack;
public class NextGreaterElement {
    public static int[] nextGreaterElements(int[] arr){
        int n = arr.length;
        int[] res = new int[n];

        Stack<Integer> st = new Stack<>();
        for(int i = n-1; i>=0; i--){
            while(!st.isEmpty() && st.peek() <= arr[i]){
                st.pop();
            }
            if(st.isEmpty()){
                res[i] = -1;
            }else{
                res[i] = st.peek();
            }
            st.push(arr[i]);
        }
        return res;
    }
    public static void main(String[] args) {
        int[] arr = {1,5,3,2,1,6,3,4};
        int[] res = nextGreaterElements(arr);
        for(int val : res){
            System.out.print(val + " ");
        }
        System.out.println();

    }
}
```

Next Smaller Element : [right to left]

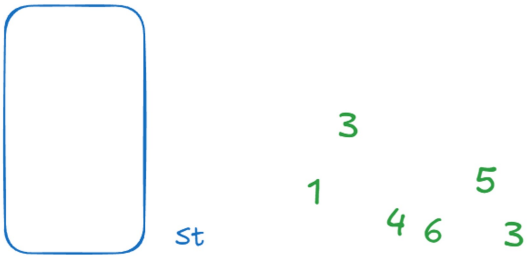




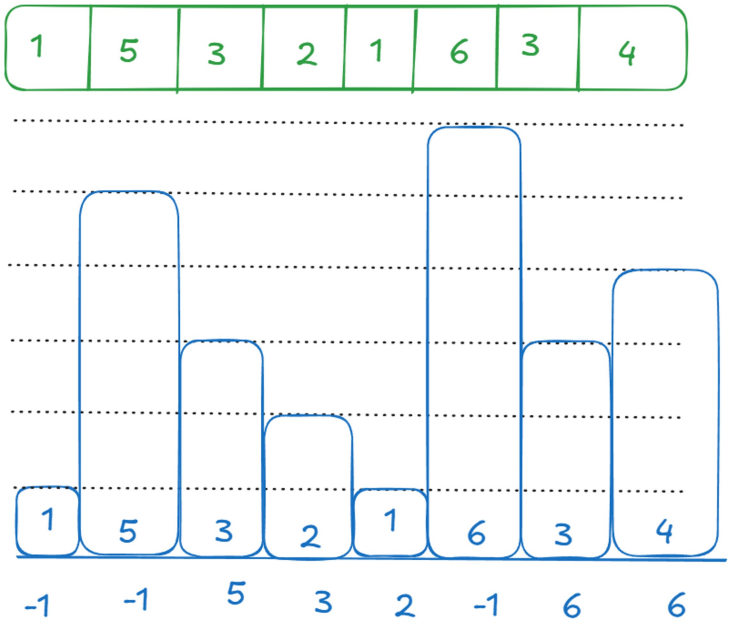

```java
import java.util.Stack;
public class NextSmallerElement {
    public static int[] nextSmallerElements(int[] arr){
        int n = arr.length;
        int[] res = new int[n];

        Stack<Integer> st = new Stack<>();
        for(int i = n-1; i>=0; i--){
            while(!st.isEmpty() && st.peek() >= arr[i]){
                st.pop();
            }
            if(st.isEmpty()){
                res[i] = -1;
            }else{
                res[i] = st.peek();
            }
            st.push(arr[i]);
        }
        return res;
    }
    public static void main(String[] args) {
        int[] arr = {1,5,3,2,1,6,3,4};
        int[] res = nextSmallerElements(arr);
        for(int val : res){
            System.out.print(val + " ");
        }
        System.out.println();

    }
}
```

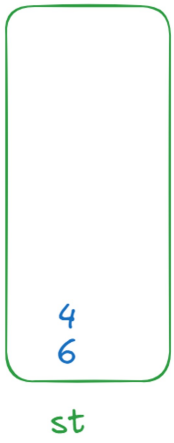
arr

res



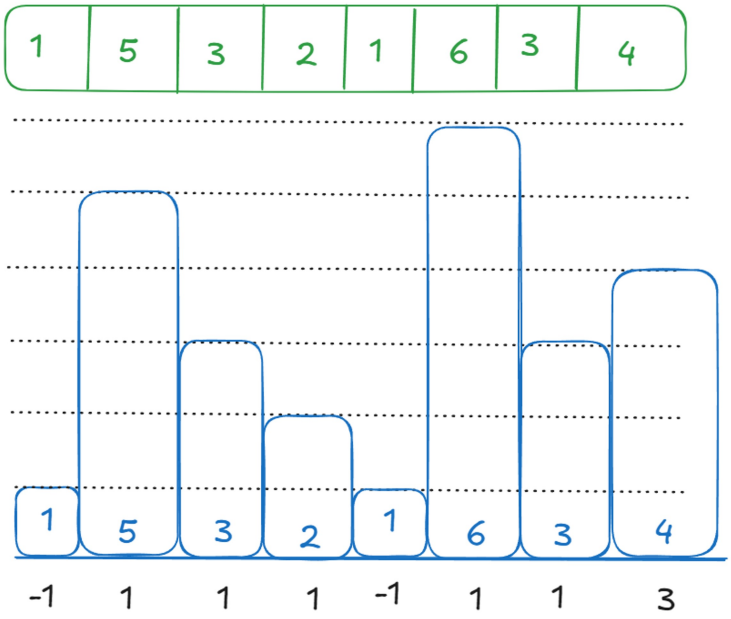
Previous Greater Element :[Left to right]





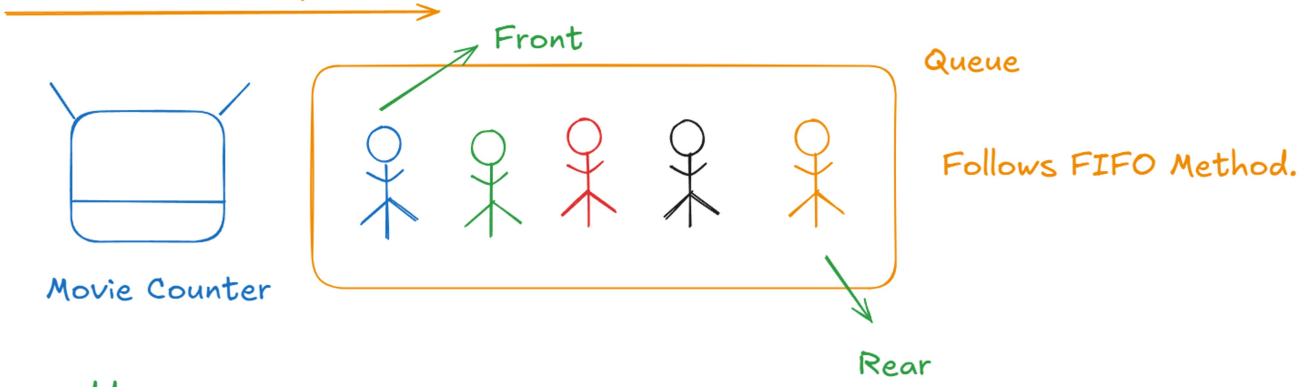

```
import java.util.Stack;
public class PreviousGreaterElement {
    public static int[] prevGreaterElements(int[] arr){
        int n = arr.length;
        int[] res = new int[n];

        Stack<Integer> st = new Stack<>();
        for(int i = 0; i<n; i++){
            while(!st.isEmpty() && st.peek() <= arr[i]){
                st.pop();
            }
            if(st.isEmpty()){
                res[i] = -1;
            }else{
                res[i] = st.peek();
            }
            st.push(arr[i]);
        }
        return res;
    }
    public static void main(String[] args) {
        int[] arr = {1,5,3,2,1,6,3,4};
        int[] res = prevGreaterElements(arr);
        for(int val : res){
            System.out.print(val + " ");
        }
        System.out.println();

    }
}
```

Previous Smaller Element : [Left to right]




```java
import java.util.Stack;
public class PreviousSmallerElement {
    public static int[] prevSmallerElements(int[] arr){
        int n = arr.length;
        int[] res = new int[n];

        Stack<Integer> st = new Stack<>();
        for(int i = 0; i<n; i++){
            while(!st.isEmpty() && st.peek() >= arr[i]){
                st.pop();
            }
            if(st.isEmpty()){
                res[i] = -1;
            }else{
                res[i] = st.peek();
            }
            st.push(arr[i]);
        }
        return res;
    }
    public static void main(String[] args) {
        int[] arr = {1,5,3,2,1,6,3,4};
        int[] res = prevSmallerElements(arr);
        for(int val : res){
            System.out.print(val + " ");
        }
        System.out.println();

    }
}
```

Introduction To Queue



Follows FIFO Method.

- add
- remove
- peek
- size
- IsEmpty

Queue

- Enqueue [add]
- Dequeue [remove]
- peek [top element]
- size
- isEmpty

```
Queue<String> q = new Queue<>();  (X)
```

Creating a Queue :

```
Queue<String> q = new LinkedList<>();
Queue<String> q = new PriorityQueue<>();
Queue<String> q = new ArrayDeque<>();
```

Add a Element : [Enqueue]

`q.add("Krishna");` ⟶ error

`q.offer("Krishna");` ⟶ boolean

Removing a Element : [Dequeue]

q.remove("Krishna"); ⟶ error

q.poll("Krishna"); ⟶ null

Peek (Retrieving the head without removing):

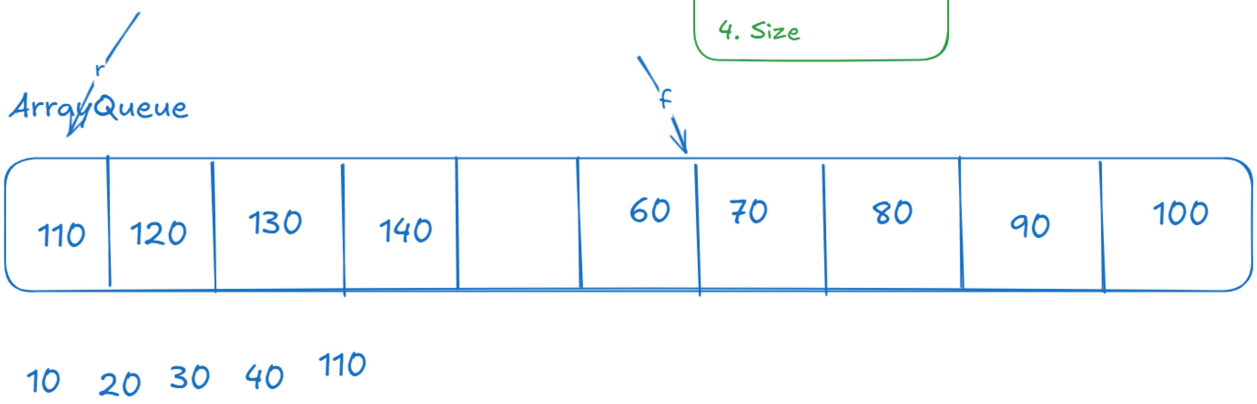
q.element(); ⟶ error
q.peek(); ⟶ null

q.isEmpty() : checks whether your queue is empty or not & returns the result in boolean.

q.size() : return an integer value that is the size of the queue.

Queue Implementation using Array :

1. Front
2. Rear
3. Capacity
4. Size

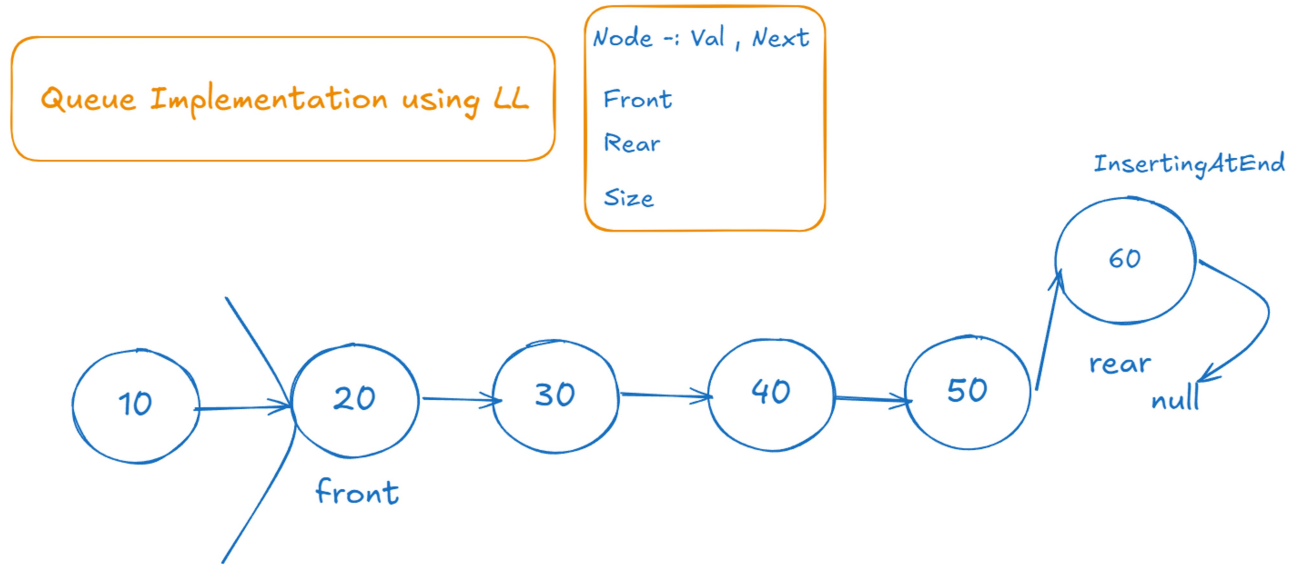


10 20 30 40 110

```
private int[] queue;
private int front;
private int rear;
private int size;
private int capacity;
```

```
private int size;
private int capacity;

public ArrayQueue(int capacity){
    this.capacity = capacity;
    queue = new int[capacity];
    front = 0;
    rear = -1;
    size = 0;
}
```

```
public class ArrayQueue {
    private int[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

    public ArrayQueue(int capacity){
        this.capacity = capacity;
        queue = new int[capacity];
        front = 0;
        rear = -1;
        size = 0;
    }
    public void enqueue(int element){
        if(isFull()){
            System.out.println("Queue is full , Can't add a new Element");
            return;
        }
        rear = (rear + 1) % capacity; // Circular Queue
        queue[rear] = element;
        size++;
    }
    public int dequeue(){
        if(isEmpty()){
            System.out.println("Queue is Empty, Can't dequeue");
            return -1;
        }
        int element = queue[front];
        front = (front + 1) % capacity; // Circular Queue
        size--;
        return element;
    }
    public int peek(){
        if(isEmpty()){
            System.out.println("Queue is Empty, Can't peek");
            return -1;
        }
        return queue[front];
    }
    public boolean isEmpty(){
        return size == 0;
    }
    public boolean isFull(){
        return size == capacity;
    }
    public int getSize(){
        return size;
    }

    public static void main(String[] args) {
        ArrayQueue queue = new ArrayQueue(5); // int[] q = new int[5];
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.enqueue(40);
        queue.enqueue(50);
        System.out.println(queue.peek());
        System.out.println(queue.isFull());
        queue.enqueue(60);


    }
}
```




```
class Node{
    int data;
    Node next;

    public Node(int data){
        this.data = data;
        this.next = null;
    }
}
public class LinkedListQueue {
   private Node front , rear;
   private int size;

   public LinkedListQueue(){
    this.front = null;
    this.rear = null;
    this.size = 0;
   }
   public void enqueue(int data){
     Node newNode = new Node(data);
     if(isEmpty()){
        front = rear = newNode;
     }else{
        rear.next = newNode;
        rear = newNode;
     }
     size++;
   }

   public int dequeue(){
    if(isEmpty()){
        System.out.println("Queue is Empty , Can't Dequeue");
        return -1;
    }
    int dequeuedValue = front.data;
    front = front.next;

    if(front == null){
        rear = null;
    }
    size--;
    return dequeuedValue;
   }

   public int peek(){
        if(isEmpty()){
            System.out.println("Queue is Empty , Can't Peek");
            return -1;
        }
        return front.data;
   }

   public boolean isEmpty(){
```

```
        return front.data;
    }

    public boolean isEmpty(){
        return front == null;
    }

    public int getSize(){
        return size;
    }



    public static void main(String[] args) {
        LinkedListQueue queue = new LinkedListQueue();
    }
}
```

225. Implement Stack using Queues

Solved

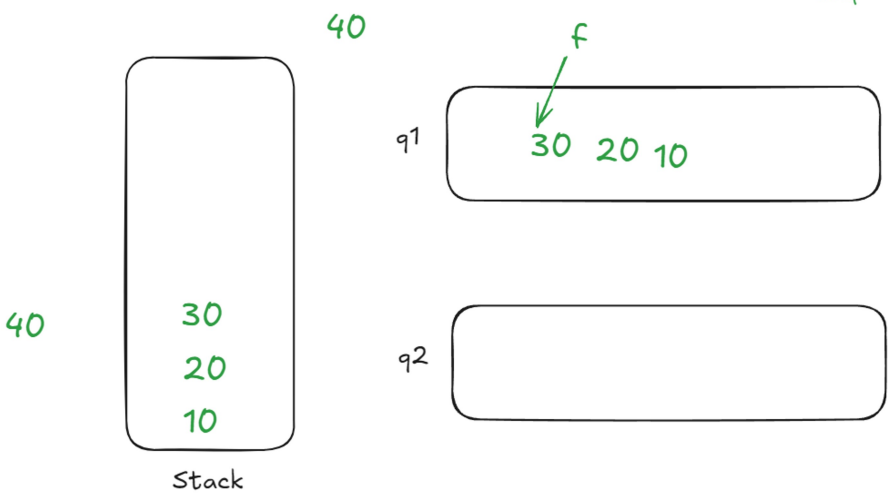
Easy | Topics | Companies

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.

Step 1 : Push the value to the temp queue [q2].
Step 2 : while(!q1.isEmpty){
q2.add(q1.poll());
}
step 3 : Swap q1 - q2



st.push(10)
st.push(20)
st.push(30)
st.push(40)
st.pop()
st.peek()

temp queue = q2

```
private Queue<Integer> q1;
private Queue<Integer> q2;
public MyStack() {
    q1 = new LinkedList<>();
    q2 = new LinkedList<>();
}
```

```
public void push(int x) {
    q2.offer(x);
    while(!q1.isEmpty()){
        q2.offer(q1.poll());
    }

    Queue<Integer> temp = q1;
    q1 = q2;
    q2 = temp;
}
```

q2

q1

```
public int pop() {
    return q1.poll();
}

public int top() {
    return q1.peek();
}

public boolean empty() {
    return q1.isEmpty();
}
```

push(10)
push(20)
push(30)
pop()
Peek() → 20

30

Change Status

Completed

Attendance Code: C0740371

232. Implement Queue using Stacks

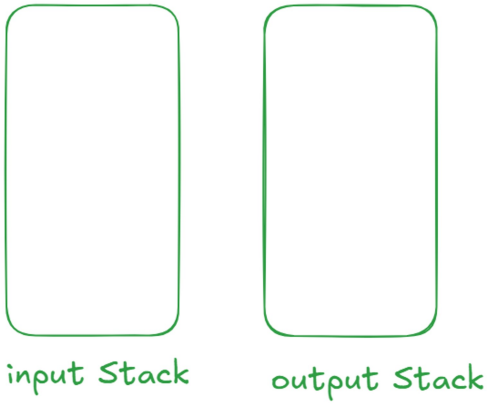
Solved

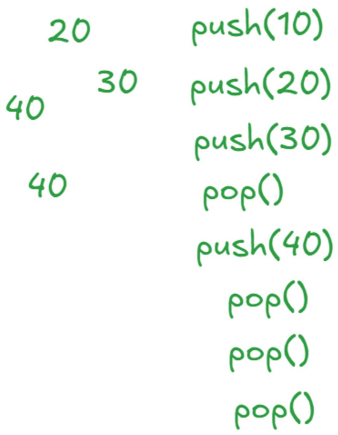
Easy | Topics | Companies

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`).

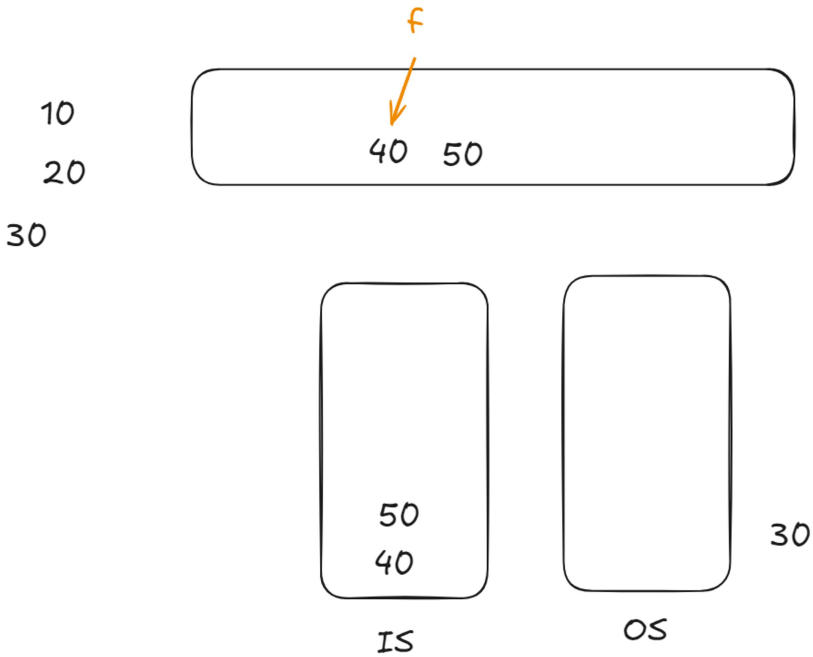
Implement the `MyQueue` class:

- `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.






```
class MyQueue {
    private Stack<Integer> inputStack;
    private Stack<Integer> outputStack;

    public MyQueue() {
        inputStack = new Stack<>();
        outputStack = new Stack<>();
    }

    public void push(int x) {
        inputStack.push(x);
    }

    public int pop() {
        if(outputStack.isEmpty()){
            while(!inputStack.isEmpty()){
                outputStack.push(inputStack.pop());
            }
        }
        return outputStack.pop();
    }

    public int peek() {
        if(outputStack.isEmpty()){
            while(!inputStack.isEmpty()){
                outputStack.push(inputStack.pop());
            }
        }
        return outputStack.peek();
    }

    public boolean empty() {
```



```
        return outputStack.peek();
    }

    public boolean empty() {
        return inputStack.isEmpty() && outputStack.isEmpty();
    }
}

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue obj = new MyQueue();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.peek();
 * boolean param_4 = obj.empty();
 */
```