*“The significant problems we face cannot be solved by the same level of thinking that created them.”*

*- Albert Einstein*

Dear reader, I welcome you to a series of four problems where you need to implement a queue using stacks and inversely, a stack using queues. I request to solve the following 4 problems in consecutive fashion.

1. **Queue to Stack Adapter**
   1. [**Queue to Stack Adapter - Push Efficient**](https://www.pepcoding.com/resources/online-java-foundation/stacks-and-queues/queue-to-stack-adapter-push-efficient-official/ojquestion)
   2. [**Queue to Stack Adapter - Pop Efficient**](https://www.pepcoding.com/resources/online-java-foundation/stacks-and-queues/queue-to-stack-adapter-pop-efficient/ojquestion)
2. **Stack to Queue Adapter**
   1. [**Stack to Queue Adapter - Push Efficient**](https://www.pepcoding.com/resources/online-java-foundation/stacks-and-queues/stack-to-queue-adapter-add-efficient-official/ojquestion)
   2. [**Stack to Queue Adapter - Pop Efficient**](https://www.pepcoding.com/resources/online-java-foundation/stacks-and-queues/stack-to-queue-adapter-remove-efficient-official/ojquestion)

We are currently solving the part *Queue to Stack Adapter - Push Efficient*.

***Problem Statement***

* You are required to complete the code of the QueueToStackAdapter class we have provided.
* You have only 2 queue data members available- *mainQ* and *helperQ*. (mainQ is to contain data and helperQ is to assist in operations).
* You need to implement a stack using these 2 queues with all the operations of queues.
* You need to complete the code of following operations of stack:
  + ***push***: Should accept new data in LIFO manner.
  + ***pop***: Should remove and return data in LIFO manner. If not available, print “*Stack underflow*" and return -1.
  + ***top***: Should return data in LIFO manner. If no element is available, print "*Stack underflow*" and return -1.
  + ***size***: Should return the number of elements available in the stack.

*Note*: Please do not declare any stack data structure, you must solve this problem by only using the two instances of queue available.

In this ‘*Push Efficient*’ variation, push operation should be as efficient as possible (O(1) per call). In order to achieve ***constant time push operation***, you can take linear time in pop/top operation.

If you are unable to understand the problem, then you can watch the [*question video*](https://www.youtube.com/watch?v=BLrkF4-3zZ4) for better understanding.

***Solution - (O(1) Push, O(n) Pop)***

Let us look at the operations:

* ***push***: As we want push to be constant O(1) operation, hence we will not do anything special, but just enqueue (add) the element in the main queue.
* ***Size***: Size function is also simple, just return the size of main queue mainQ.

Now, let us look at what we should do in pop and top functions, so that we can implement LIFO order (stacks) using the FIFO order in the two queues.

Firstly, just print “*Stack underflow*” and return -1 directly if the main queue (mainQ) size is 0, i.e. there are no elements present.

Now, we are sure that there is at least one element present in the main queue to be popped.

We need to pop (or peek) the element which is inserted at last (to achieve LIFO order) in the queue. But how to get the last (rear) element of the queue, when we are only allowed to get the front element of the queue (using peek operation)?

We will take the help of the auxiliary queue (helperQ) to achieve this task. We will dequeue all the elements (except the last one) from the main queue, one by one, and enqueue them in the auxiliary queue.

Now, what we have achieved is that all the elements, except the last element, are in the same order, as they were before, but in the helper queue.

We will store the last element in a variable top, and pop this element from the main queue.

There will be different lines of code for pop and peek element now:

* For pop operation, don't need to do anything
* For peek operation, push this element also in the auxiliary/helper queue.

Now, we will do the reverse process, i.e. dequeue elements, one by one, from the helper queue, and enqueue them back to the main queue.

Now, return the variable top. Hence, what we achieved by a pair of enqueue-dequeue operations between main and helper queue is, getting the last element from the main queue to pop/peek operation of stack, and making the rest of elements remain in the same order in the end.

Please refer to the [solution video](https://www.youtube.com/watch?v=o_NEL0agPGg) if you find difficulty in understanding the solution completely.

***Pseudo Code/ Algorithm***

1. ***Push***: Add the element in the main queue.
2. ***Size:*** Return the size of the main queue.
3. ***Top***:
   1. If the size of the main queue is 0 (empty), then print “Stack underflow” and return -1.
   2. Dequeue all the elements until the size of the main queue becomes equal to 1 and Enqueue them to the helper queue.
   3. Dequeue the last element and store it in a variable top. Enqueue this element also to the helper queue.
   4. Dequeue all the elements from the helper queue and enqueue them back to the main queue.
   5. Return the *top* element.
4. ***Pop***:
   1. If the size of the main queue is 0 (empty), then print “Stack underflow” and return -1.
   2. Dequeue all the elements until the size of the main queue becomes equal to 1 and Enqueue them to the helper queue.
   3. Dequeue the last element and store it in a variable top.
   4. Dequeue all the elements from the helper queue and enqueue them back to the main queue.
   5. Return the *top* element.

***Implementation (Java)***

How about first trying by yourself without reading the code we provide?

import java.io.\*;

import java.util.\*;

public class Main {

public static class QueueToStackAdapter {

Queue<Integer> mainQ;

Queue<Integer> helperQ;

public QueueToStackAdapter() {

mainQ = new ArrayDeque<>();

helperQ = new ArrayDeque<>();

}

int size() {

return mainQ.size();

}

void push(int val) {

mainQ.add(val);

}

int pop() {

if (size() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

while(mainQ.size() > 1){

helperQ.add(mainQ.remove());

}

int val = mainQ.remove();

while(helperQ.size() > 0){

mainQ.add(helperQ.remove());

}

return val;

}

}

int top() {

if (size() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

while(mainQ.size() > 1){

helperQ.add(mainQ.remove());

}

int val = mainQ.remove();

helperQ.add(val);

while(helperQ.size() > 0){

mainQ.add(helperQ.remove());

}

return val;

}

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

QueueToStackAdapter st = new QueueToStackAdapter();

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("push")) {

int val = Integer.parseInt(str.split(" ")[1]);

st.push(val);

} else if (str.startsWith("pop")) {

int val = st.pop();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("top")) {

int val = st.top();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size")) {

System.out.println(st.size());

}

str = br.readLine();

}

}

}

This code is written and explained by our team in [this video](https://www.youtube.com/watch?v=o_NEL0agPGg). Please refer to it if you are stuck somewhere.

***Time & Space Complexity Analysis***

**Push - O(1)**: We are just adding the element in the main queue.

**Size** **- O(1)**: We are returning the size of the main queue.

**Pop (or Top) - O(n):**

* Firstly, we are dequeuing size elements from the main queue adding them in the helper queue, which takes n \* O(1) = O(n) time.
* Now, we are doing the reverse process (removing n elements from the helper queue and adding them to the main queue), which again takes O(n) time.
* Hence, total time taken will be O(n + n) = O(n).

I hope you enjoyed solving the problem with me. We will come with the next part of the problem ‘Queue to Stack Adapter - Pop Efficient.’ Until then *keep coding*!

Contributor : [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)