# Queues: A Tale of Two Stacks

by saikiran9194



*Check out the resources on the page's right side to learn more about queues. The video tutorial is by Gayle Laakmann McDowell, author of the best-selling interview book Cracking the Coding Interview.*

A queue is an abstract data type that maintains the order in which elements were added to it, allowing the oldest elements to be removed from the front and new elements to be added to the rear. This is called a *First-In-First-Out* (FIFO) data structure because the first element added to the queue (i.e., the one that has been waiting the longest) is always the first one to be removed.

A basic queue has the following operations:

- *Enqueue*: add a new element to the end of the queue.
- *Dequeue*: remove the element from the front of the queue and return it.

In this challenge, you must first implement a queue using *two stacks*. Then process $q$ queries, where each query is one of the following $3$ types:

1. `1 x`: Enqueue element $x$ into the end of the queue.
2. `2`: Dequeue the element at the front of the queue.
3. `3`: Print the element at the front of the queue.

**Input Format**

The first line contains a single integer, $q$, denoting the number of queries.
Each line $i$ of the $q$ subsequent lines contains a single query in the form described in the problem statement above. All three queries start with an integer denoting the query $type$, but only query $1$ is followed by an additional space-separated value, $x$, denoting the value to be enqueued.

**Constraints**

- $1 \leq q \leq 10^5$
- $1 \leq type \leq 3$
- $1 \leq |x| \leq 10^9$
- It is guaranteed that a valid answer always exists for each query of type $3$.

**Output Format**

For each query of type $3$, print the value of the element at the front of the queue on a new line.

**Sample Input**

```
10
1 42
2
1 14
3
1 28
3
1 60
1 78
```

```
2
2
```

**Sample Output**

```
14
14
```

**Explanation**

We perform the following sequence of actions:

1. Enqueue $\mathbf{42}$; $\boldsymbol{queue} = \{\mathbf{42}\}$.
2. Dequeue the value at the head of the queue, $\mathbf{42}$; $\boldsymbol{queue} = \{\}$.
3. Enqueue $\mathbf{14}$; $\boldsymbol{queue} = \{\mathbf{14}\}$.
4. Print the value at the head of the queue, $\mathbf{14}$; $\boldsymbol{queue} = \{\mathbf{14}\}$.
5. Enqueue $\mathbf{28}$; $\boldsymbol{queue} = \{\mathbf{14} \leftarrow \mathbf{28}\}$.
6. Print the value at the head of the queue, $\mathbf{14}$; $\boldsymbol{queue} = \{\mathbf{14} \leftarrow \mathbf{28}\}$.
7. Enqueue $\mathbf{60}$; $\boldsymbol{queue} = \{\mathbf{14} \leftarrow \mathbf{28} \leftarrow \mathbf{60}\}$.
8. Enqueue $\mathbf{78}$; $\boldsymbol{queue} = \{\mathbf{14} \leftarrow \mathbf{28} \leftarrow \mathbf{60} \leftarrow \mathbf{78}\}$.
9. Dequeue the value at the head of the queue, $\mathbf{14}$; $\boldsymbol{queue} = \{\mathbf{28} \leftarrow \mathbf{60} \leftarrow \mathbf{78}\}$.
10. Dequeue the value at the head of the queue, $\mathbf{28}$; $\boldsymbol{queue} = \{\mathbf{60} \leftarrow \mathbf{78}\}$.

```
    Stack<Integer> s1 = new Stack<>();
    Stack<Integer> s2 = new Stack<>();


    //enqueue -- keep pushing onto s1
    public  void enqueue(int val) {

        s1.push(val);
    }

    //dequeue -- remove front
    public  void dequeue () {

        if (s2.isEmpty()){
            prepStack();
        }

        s2.pop();
    }

    //print the front element
    public  int peek() {

        if (s2.isEmpty())
             prepStack();
        return s2.peek();

    }

    //get all elements from s1 to s2 for front operations
    public  void prepStack() {
        while (!s1.isEmpty()) {
            int temp = s1.pop();
            s2.push(temp);
        }
    }
}

public class Solution {

    public static void main(String[] args) {
        MyQueue queue = new MyQueue();

        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();

        for (int i = 0; i < n; i++) {
            int operation = scan.nextInt();
            if (operation == 1) { // enqueue
              queue.enqueue(scan.nextInt());
            } else if (operation == 2) { // dequeue
              queue.dequeue();
            } else if (operation == 3) { // print/peek
              System.out.println(queue.peek());
            }
        }
        scan.close();
    }
}

```

Line: 4 Col: 20

Upload Code as File  Test against custom input  Run Code  Submit Code

Congrats, you solved this challenge!

| | | |
|---|---|---|
| ✔ Test Case #0 | ✔ Test Case #1 | ✔ Test Case #2 |
| ✔ Test Case #3 | ✔ Test Case #4 | ✔ Test Case #5 |
| ✔ Test Case #6 | ✔ Test Case #7 | ✔ Test Case #8 |
| ✔ Test Case #9 | ✔ Test Case #10 | ✔ Test Case #11 |
| ✔ Test Case #12 | ✔ Test Case #13 | ✔ Test Case #14 |
| ✔ Test Case #15 | | |

Next Challenge