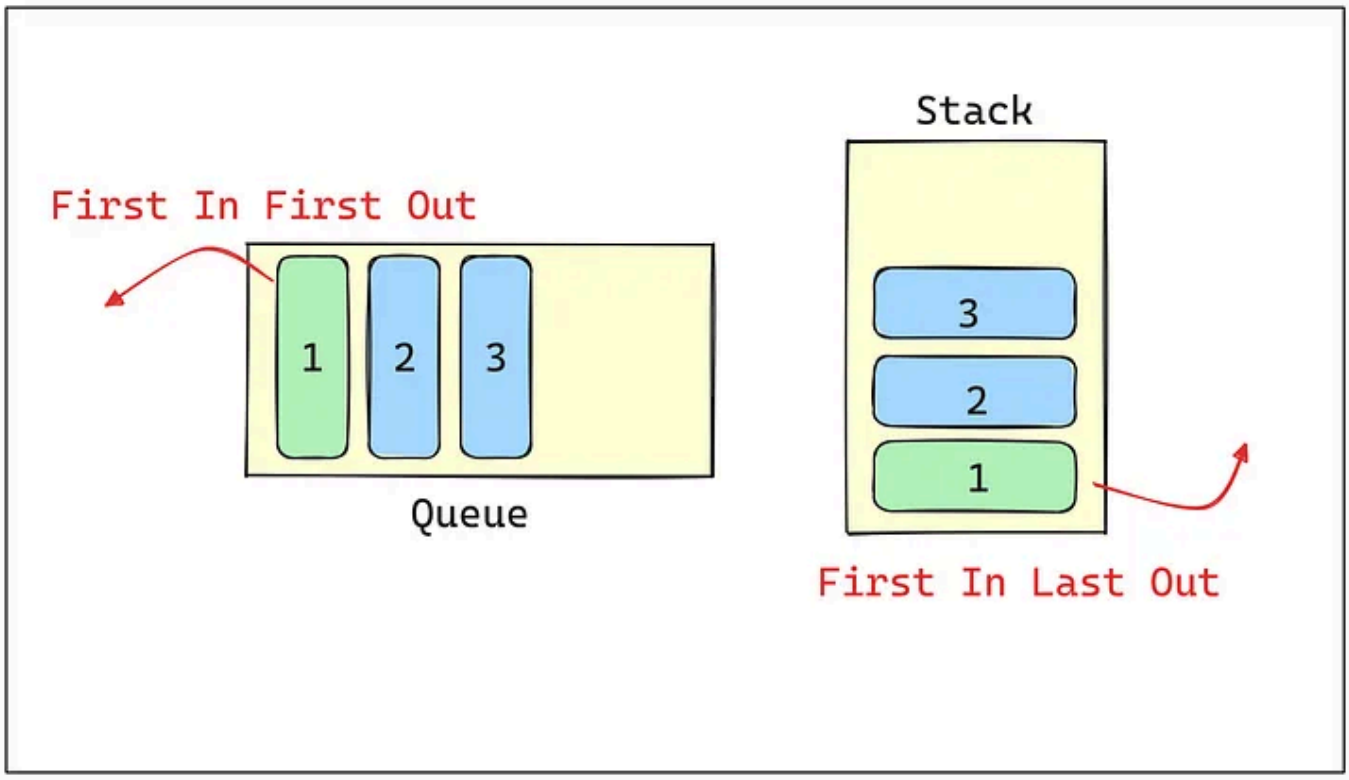
Queue Implementation using two stacks

Vaibhav Chauhan · Follow

4 min read · Jul 25, 2023

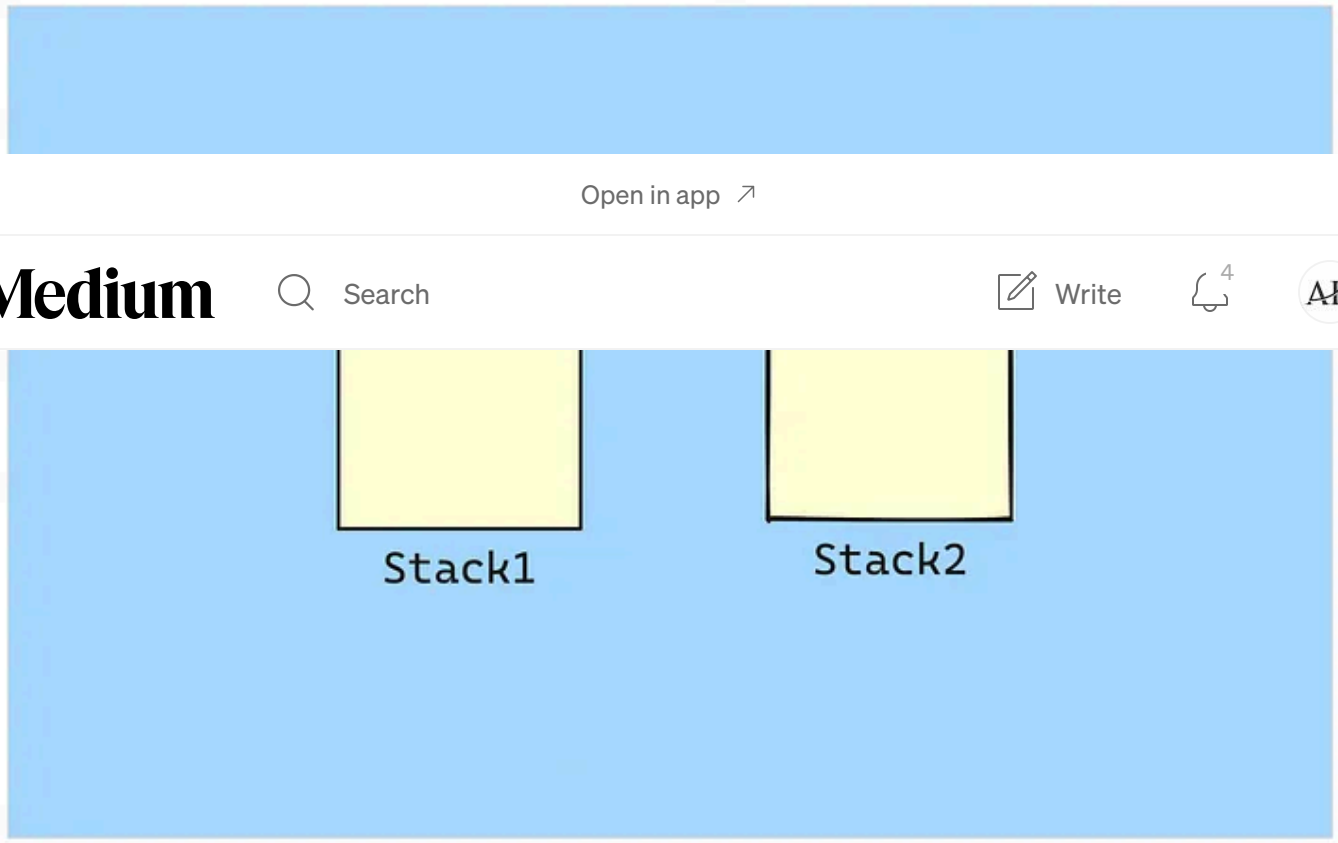
Firstly, we have to understand what principle Queue & Stack follows:

1. Queue:
   The Queue data structure adheres to the “First In First Out” (FIFO) principle. This means that the element that is enqueued (added) first is the one that will be dequeued (removed) first.
2. Stack:
   On the other hand, the Stack data structure follows the “First In Last Out” (FILO) or “Last In First Out” (LIFO) principle. This implies that the last element to be pushed (added) onto the stack is the first one to be popped (removed) from it.

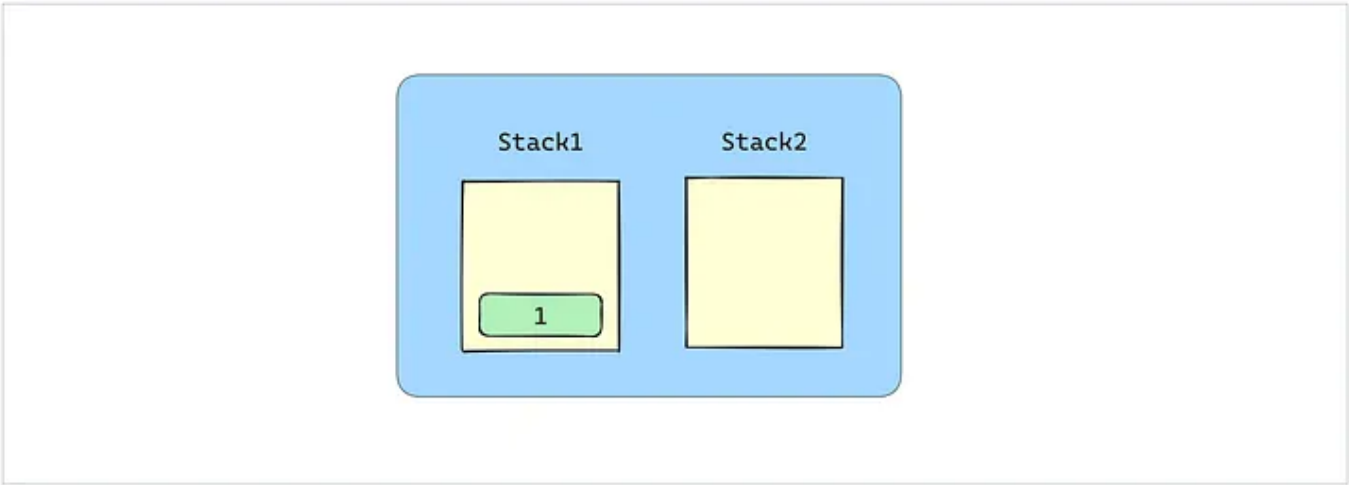


Now we understand the basic principle of Stack and Queue. Let's see how we can implement Queue using Stack.

As we have seen, the rules for stacks and queues are somewhat contradictory. So, in order for our stack to function as a Queue, we need to add another stack to reverse the items when we push a new element.

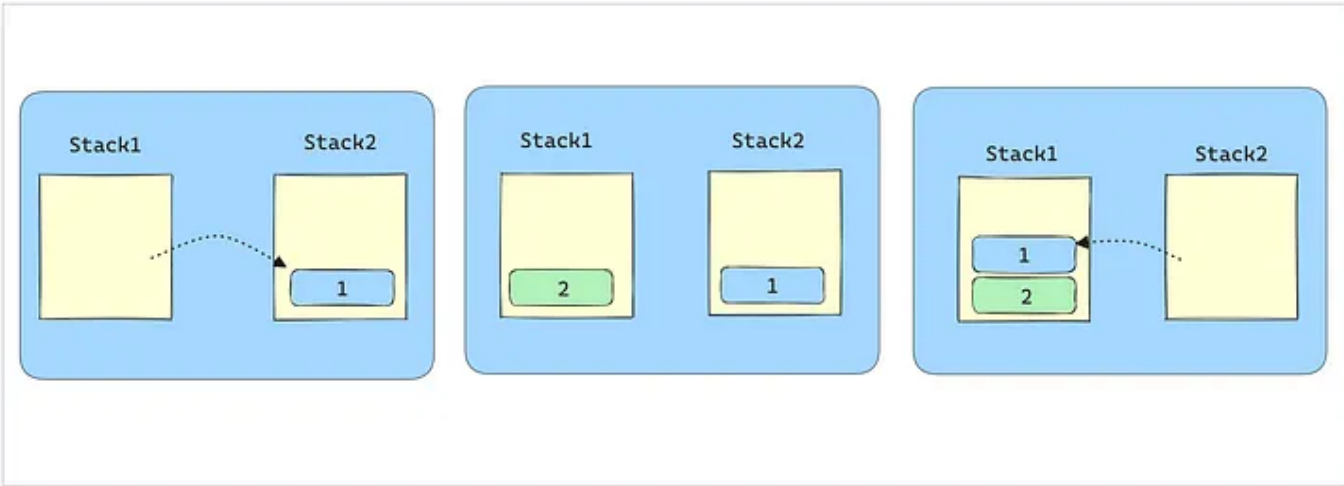


Let's do some Dry-Run, suppose we want to “Push 1" in our stack.

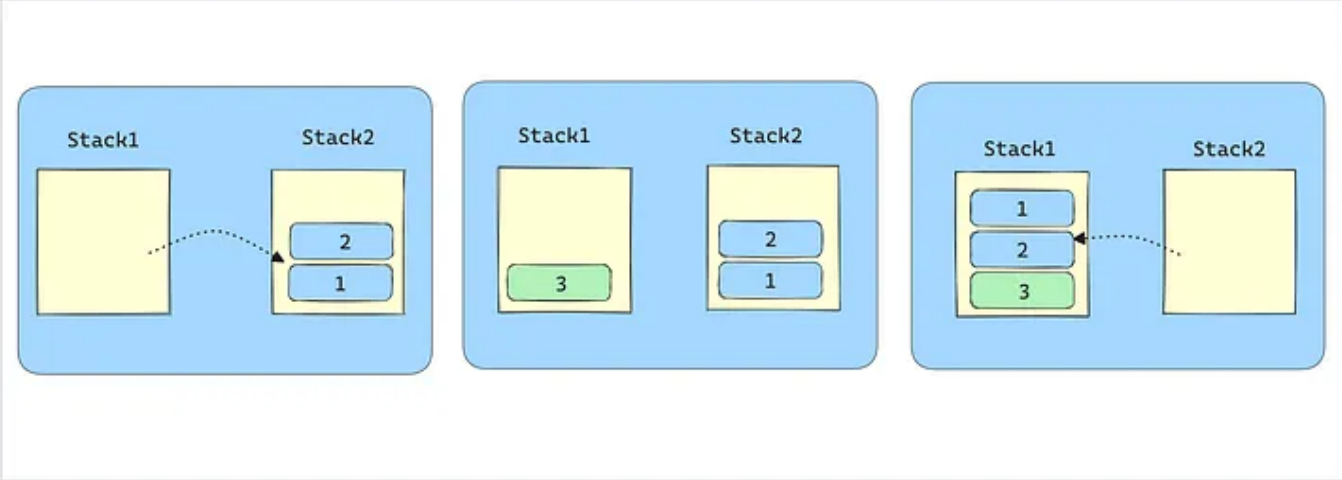


- To make our stack behave like a queue with the first element pushed at the top, we use “Stack2” as an intermediary.

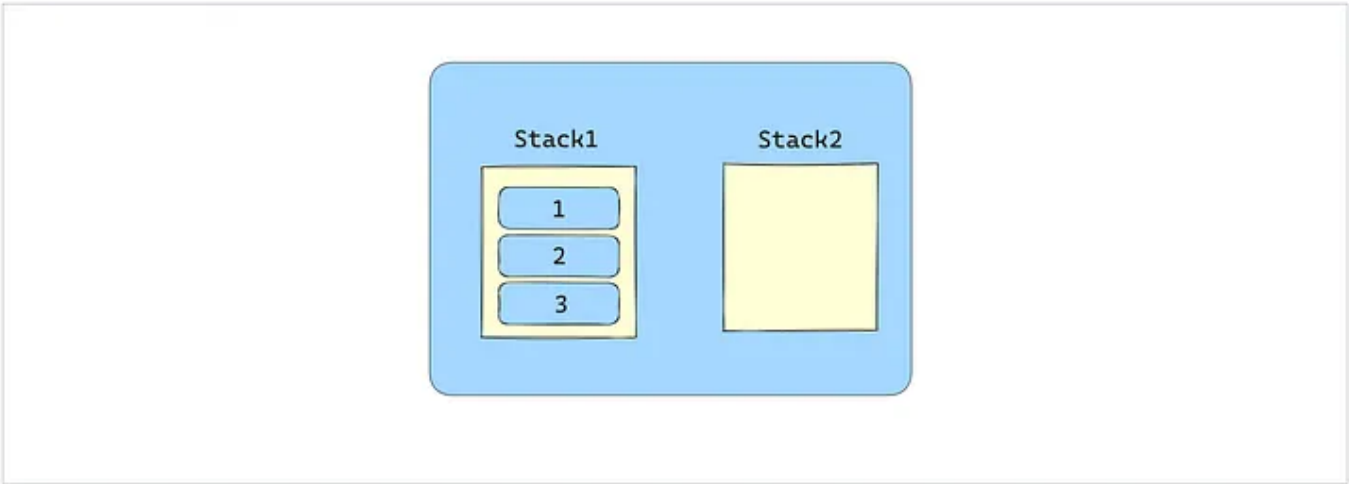
- When we want to push a new element into “Stack1,” we first move all existing elements from “Stack1” to “Stack2.”
- Next, we add the new element to “Stack1.”
- Finally, we move all elements back from “Stack2” to “Stack1” to keep the original sequence intact.



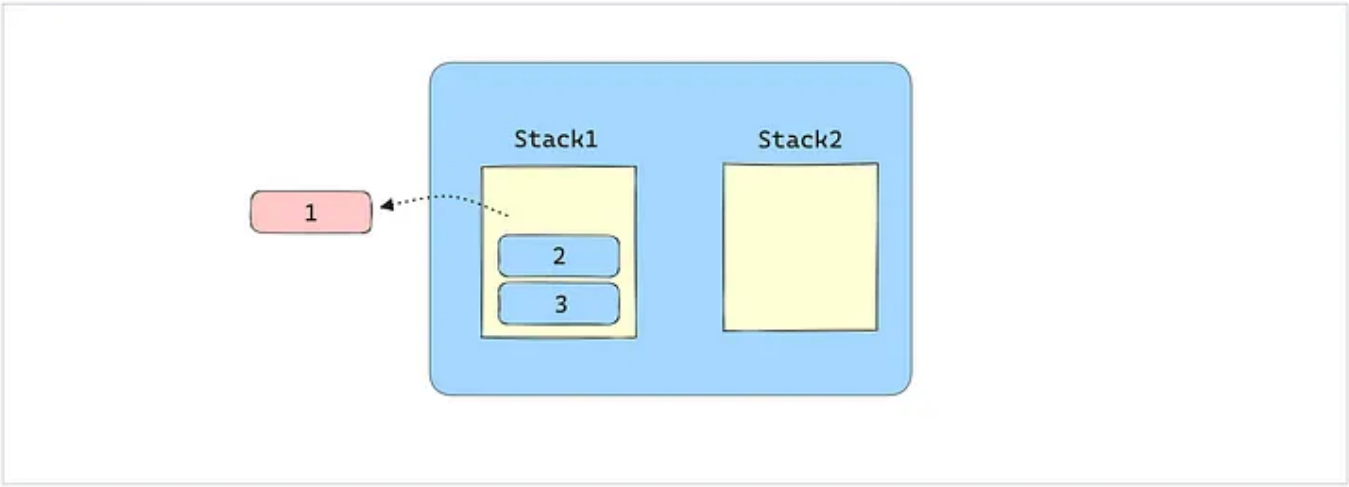
For better understanding, let’s push 3 in our Stack.



So our current state of Stack1 looks like this :



Now, if we call the pop function, it will pop the first element that we inserted in our stack. Exactly the same as Queue will do.



Code in Java:

```
import java.util.Stack;
import java.util.Scanner;

class StackQueue {
    Stack<Integer> s1 = new Stack<Integer>();
    Stack<Integer> s2 = new Stack<Integer>();

    // Function to push an element in the queue using 2 stacks.
```

```
    void push(int x) {
        if (s1.isEmpty()) {  // if stack1 is empty. push that first element in i
            s1.push(x);
        } else {
            while (!s1.isEmpty()) {    //if stack1 is not empty, so make it empt
                s2.push(s1.pop());     // And push all the element from stack1 t
            }

            s1.push(x);               // Then push the new item into stack1.

            while (!s2.isEmpty()) {   // after pushing new item into stack1. Now
                s1.push(s2.pop());    //back all the element from stack2 to stac
            }
        }
    }

    // Function to pop an element from the queue using 2 stacks.
    int pop() {
        if (s1.isEmpty()) {
            return -1;
        }
        return s1.pop();
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        StackQueue queue = new StackQueue();

        System.out.print("Enter the number of elements to push: ");
        int n = scanner.nextInt();

        System.out.println("Enter the elements to push:");
        for (int i = 0; i < n; i++) {
            int element = scanner.nextInt();
            queue.push(element);
        }

        System.out.println("Elements popped from the queue:");
        while (true) {
            int poppedElement = queue.pop();
            if (poppedElement == -1) {
                break;
            }
            System.out.println(poppedElement);
        }
    }
}
```

Overall, the push operation has a time complexity of O(n), while the pop operation has a time complexity of O(1). The space complexity is O(n) due to the two stacks used to implement the queue.

Stack Queue Datastrucutre Algorithms Implementation

Written by Vaibhav Chauhan

Follow

0 Followers

Hi there! I'm Vaibhav Kumar Chauhan, a passionate Software Developer & I'm always excited about exploring new technologies and learning more every day.

Recommended from Medium