Agenda

1) Introduction to queue

2) Queue functions in Java

3) Reverse first K elements of given queue

4) Create N no. using only 1, 2 and 3

5) Adapter (Queue using Stack)

what is queue

It follows **FIFO** : first in first out

How to create and use a queue in Java

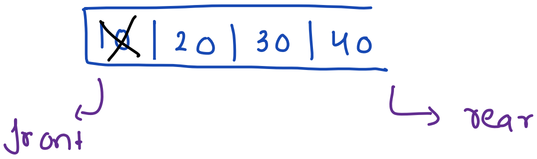
```
Queue < Integer> q  =  new  ArrayDeque <>();
```

↓ name of variable

add → do addition at last.

remove → do removal from front.

```
q.add (10);
q.add (20);
q.add (30);
q.add (40);
```



~~10~~ | 20 | 30 | 40

```
sopln(q.peek());      // front element : 10

sopln (q.remove());   // removing front element and returning it.
```

q.add (x), q.remove(), q.peek() → O(1)

Q.1 Given a queue, reverse first k elements of it.

q :

10	20	30	40	50	60	70	80

K = 3

Expected TC : O(n)

↓ after reversal of first K elements

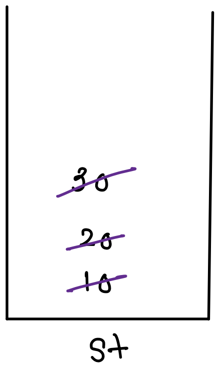
30	20	10	40	50	60	70	80

Step1 : remove K ele from q and add them to stack

q :

~~10~~	~~20~~	~~30~~	40	50	60	70	80

K = 3



Step 2 : pop elements from stack and add them to q.

:

40	50	60	70	80	30	20	10

step 3 : remove n-k elements from q and them to q

~~40~~	~~50~~	~~60~~	~~70~~	~~80~~	30	20	10	40	50	60	70	80

n = 8 K = 3 , n-K = 5

TC : O(n)

SC : O(K)

Q.2 Create N no. in Ascending order using only 1,2 & 3 as digits and return these numbers.

N=4 ans: 1 2 3 11

N = 7 ans: 1 2 3 11 12 13 21

N = 10 ans: 1 2 3 11 12 13 21 22 23 31

~~1~~ ~~2~~ ~~3~~ ~~11~~ ~~12~~ ~~13~~ ~~21~~ ~~22~~ ~~23~~ ~~31~~ ~~32~~ ~~33~~ 111 112 113

121 122 123 131 132 133 211 212 213 221 222 223

231 232 233 311 312 313 321 322 323 331 332 333

(n=10)

q :

~~1~~	~~2~~	~~3~~	~~11~~	~~12~~	~~13~~	~~21~~	~~22~~	~~23~~	~~31~~	32	33

Count = ~~3~~ ~~6~~ ~~9~~
12

ans :

1	2	3	11	12	13	21	22	23	31

(list)

```
// add 1,2,3 in q
// count = 3
while (ans.size() < N) {
    int temp = q.remove();
    ans.add (temp);

    if (count < N) {
        int v1 = temp*10 + 1;
        int v2 = temp*10 + 2;
        int v3 = temp*10 + 3;
        q.add (v1); q.add (v2); q.add (v3);
        count += 3;

    }

}
```

n = 8

q: | ~~1~~ | ~~2~~ | ~~3~~ | ~~11~~ | ~~12~~ | ~~13~~ | ~~21~~ | ~~22~~ | 23

count = ~~3~~ ~~6~~
9

ans: | 1 | 2 | 3 | 11 | 12 | 13 | 21 | 22 |

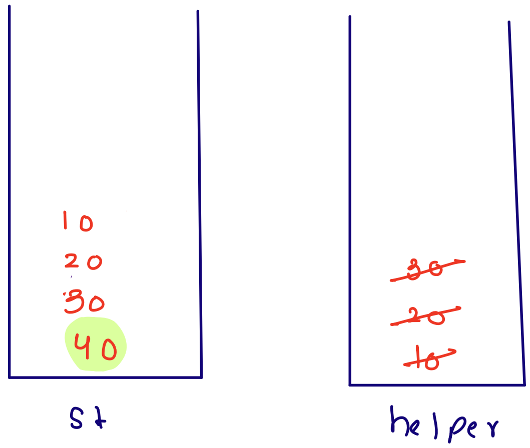
```
//add 1, 2, 3 in q
// count=3
while (ans.size() < N) {
    int temp = q.remove();
    ans.add (temp);

    if( count < N) {
        int v1 = temp*10 + 1;
        int v2 = temp*10 + 2;
        int v3 = temp*10 + 3;
        q.add (v1); q.add (v2); q.add (v3);
        count += 3;

    }

}
```

Adapter

Q. Implement Queue functions using stack (remove efficient)
(as datamember)

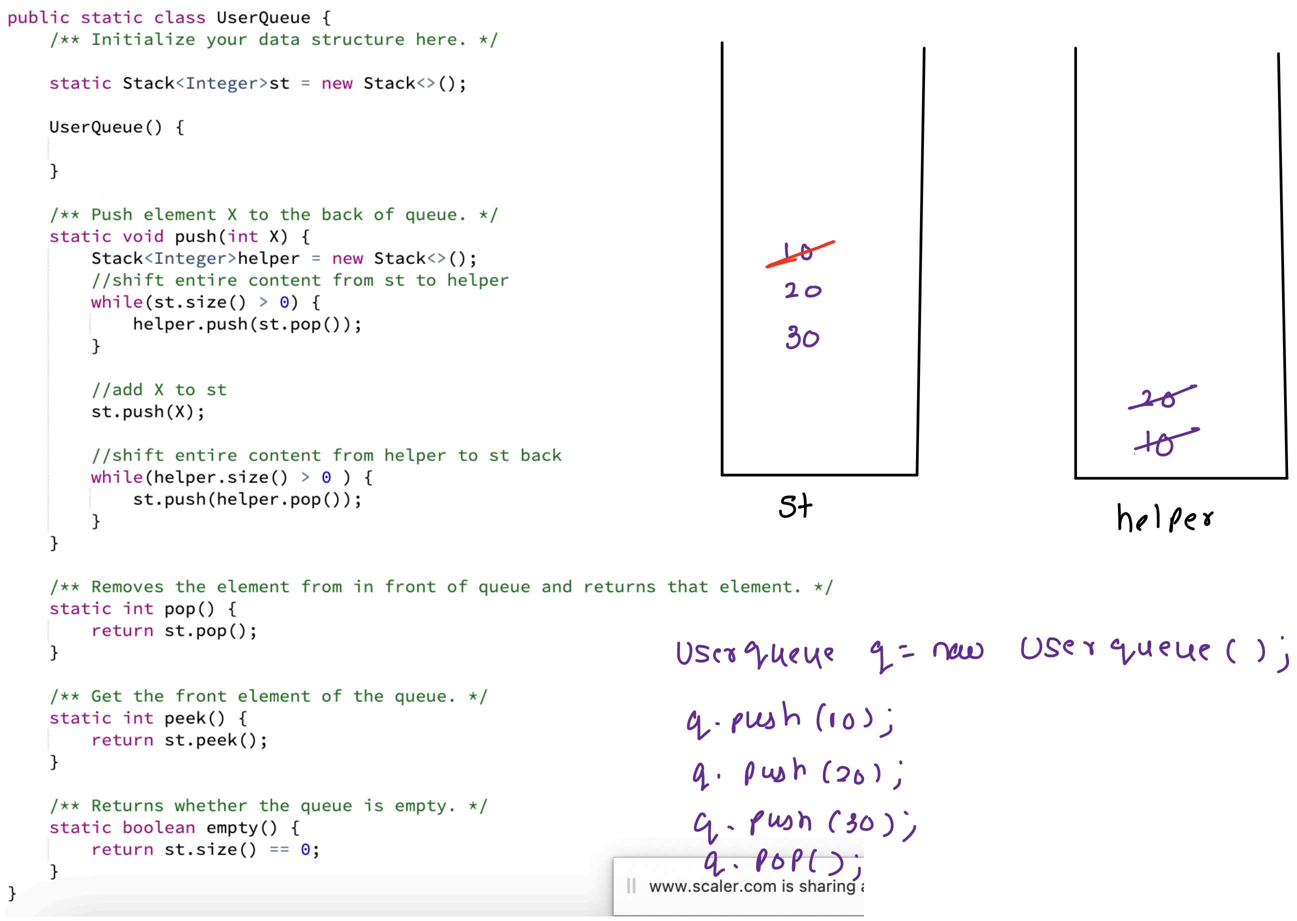
```
class Adapter {

    void add (int x)

    int remove ( )  ⟶ O(1)

    int peek ( )  ⟶ O(1)

}
```

```
Adapter q = new Adapter();

q.add(10);
q.add (20);
q.add (30);
→ q.add (40);

sopln(q.remove());
```



add (x)
- → shift entire content from st to helper
- → push x in st
- → shift entire content from helper to st

remove
- st.pop()



```
public static class UserQueue {
    /** Initialize your data structure here. */

    static Stack<Integer>st = new Stack<>();

    UserQueue() {

    }

    /** Push element X to the back of queue. */
    static void push(int X) {
        Stack<Integer>helper = new Stack<>();
        //shift entire content from st to helper
        while(st.size() > 0) {
            helper.push(st.pop());
        }

        //add X to st
        st.push(X);

        //shift entire content from helper to st back
        while(helper.size() > 0 ) {
            st.push(helper.pop());
        }
    }

    /** Removes the element from in front of queue and returns that element. */
    static int pop() {
        return st.pop();
    }

    /** Get the front element of the queue. */
    static int peek() {
        return st.peek();
    }

    /** Returns whether the queue is empty. */
    static boolean empty() {
        return st.size() == 0;
    }
}
```

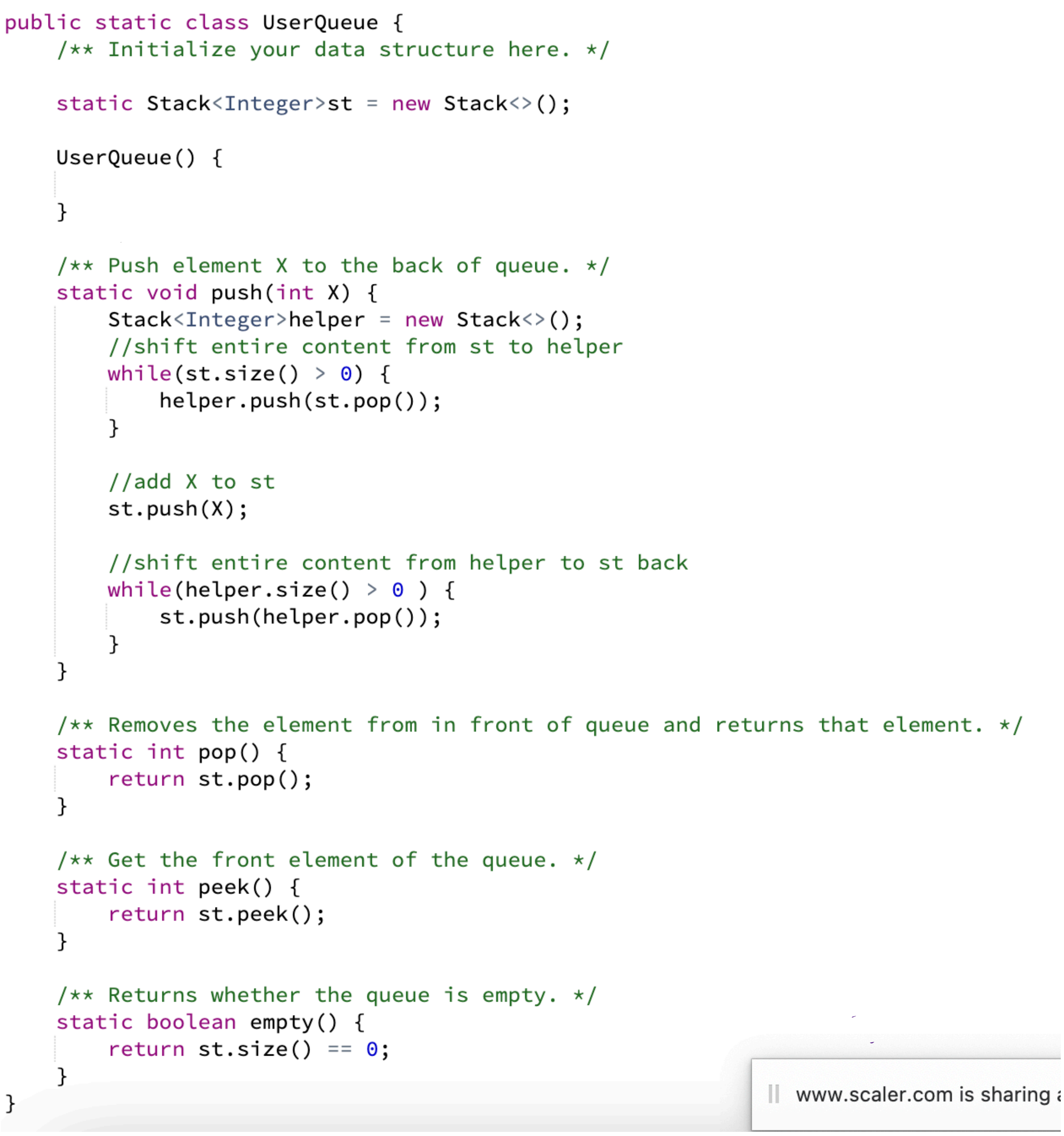
Usrqueue q = new User queue ( );

q. push (10);

q. push (20);

q. push (30);

q. pop( );



```java
public static class UserQueue {
    /** Initialize your data structure here. */

    static Stack<Integer>st = new Stack<>();

    UserQueue() {

    }

    /** Push element X to the back of queue. */
    static void push(int X) {
        Stack<Integer>helper = new Stack<>();
        //shift entire content from st to helper
        while(st.size() > 0) {
            helper.push(st.pop());
        }

        //add X to st
        st.push(X);

        //shift entire content from helper to st back
        while(helper.size() > 0 ) {
            st.push(helper.pop());
        }
    }

    /** Removes the element from in front of queue and returns that element. */
    static int pop() {
        return st.pop();
    }

    /** Get the front element of the queue. */
    static int peek() {
        return st.peek();
    }

    /** Returns whether the queue is empty. */
    static boolean empty() {
        return st.size() == 0;
    }
}
```