

13 July 2023

Double Ended Queue / Deck:

Insertion and Deletion operations are not restricted to one end, rather can be performed on either of the two ends.

A Deque can be categorised into two categories:

>Input Restricted Deque: insertion is restricted to one end but the deletion can take place at either end of the list

>Output Restricted Deque: deletion is restricted but the insertion can take place at either end of the list

The screenshot shows a Java IDE with a file named `Main.java` containing a `Queue` class. The code implements a double-ended queue using an array `items` of size `QUEUE_SIZE`. The `enqueue` method handles insertion at the front or rear, and the `dequeue` method handles removal from either end. Handwritten annotations in yellow and blue ink are present on the code and a diagram.

Handwritten Annotations:

- `QUEUE_SIZE = 5` is written in yellow at the top right.
- A diagram of an array `items` is shown with indices 0 to 4. The values are 10, 10, 10, 20, and an empty space. The first three 10s are crossed out with a blue 'X' and a yellow 'X' respectively. The value 20 is written in blue.
- The word `items` is written in yellow next to the array diagram.
- The variable `front` is written in yellow above the first index (0).
- The variable `rear` is written in yellow above the last index (4).
- The variable `flag` is written in yellow above the `enqueue` method.
- The variable `flag` is written in yellow above the `dequeue` method.

```
8 public Queue() {
9     front = 0;
10    rear = -1;
11    items = new int[QUEUE_SIZE];
12 }
13
14 public void enqueue(int x, int flag) {
15     if (rear == QUEUE_SIZE - 1) {
16         System.out.println("\nQueue Overflow.");
17         return;
18     }
19     if (flag == 0) { // insert at front
20         for (int i = rear; i >= front; i--)
21             items[i + 1] = items[i];
22         items[front] = x;
23         rear++;
24     } else if (flag == 1) { // insert at rear
25         items[++rear] = x;
26     } else {
27         System.out.println("\nInvalid flag");
28         return;
29     }
30 }
31
32 public void dequeue(int flag) {
33     if (rear < front) {
```

Ln 6, Col 44 Spaces: 4 UTF-8 LF {} Java

← → NS 13 July 2023

Main.java

Queue > QUEUE SIZE

```
27     System.out.println("\nInvalid flag");
28     return;
29 }
30
31
32 public void dequeue(int flag) {
33     if (rear < front) {
34         System.out.println("\nQueue Underflow. Cannot remove.");
35         return;
36     }
37     if (flag == 0) { // remove from front
38         System.out.println("\nThe value removed is " + items[front]);
39         for (int i = front; i <= rear; i++)
40             items[i] = items[i + 1];
41         items[rear] = 0; // optional
42         rear--;
43     } else if (flag == 1) { // remove at rear
44         System.out.println("\nThe value removed is " + items[rear]);
45         items[rear] = 0;
46         rear--;
47     } else {
48         System.out.println("\nInvalid flag");
49         return;
50     }
51 }
52
```

items

0	1	2	3	4
10	20	30		

my.newtonschool.co/playground/code/z6oa8427pl1

Example

Input:

2 (No of Times)

5

N = 1 2 3 4 5

6

4

1 2 3 4

7

Output:

5 1 2 3 4

Count of Right Shift

1	2	3	4	5
5	1	2	3	4
4	5	1	2	3
3	4	5	1	2
2	3	4	5	1
1	2	3	4	5
5	1	2	3	4

N

shift Right