Queue

* It follows FIFO
* **Operations** 🡪 O(1)

Add()

Enqueue

We add element at rear

Remove()

Dequeue

We remove element from rear

Peek()

Display element at front

* **Implementation**

**Array :** we usually don’t use it in practical examples as it has fixed size

Also remove time complexity is O(n)

Remove:

Int front = arr[0];

For(int i=0 ; i<rear; i++){ arr[i] = arr[i+1];}

Rear = rear --;

Return front;

Add()

Rear = rear +1;

Arr[rear] = data;



rear

front

Circular Queue : (using array)

Array:

Rear updation formula is **rear = (rear + 1)%size**

Front updation formula is **front = (front + 1)%size**

Isfull **: (rear +1 ) % size == front**



Rear

Front

Queue Using linked list :

public static int remove(){

            if(isEmpty()){

                return -1;

            }

            int front = head.data;

            if(head == tail){

                head = tail = null;

            }else{

                head = head.next;

            }return front;

        }

To insert element we simply add it at tail

By pointing tail next to newNode and then make new node to tail

To remove element we simply point head to its next



 public static void add(int data){

            Node newNode = new Node(data);

            if(isEmpty()){

                head = tail = newNode;

            }

            tail.next = newNode;

            tail = newNode;

        }

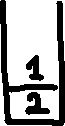
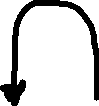
Queue using **Java Collection Framework**

In java Queue is a interface and not a class hence to use Queue in **JCF**

We can either use LinkedList Class or ArrayDeque

Queue<Integer> q = new LinkedList<>();

**Queue using 2 stacks**



1

2

To push another elemnt into stack 1 stack 1 need to be empty

Steps :

Push in stack 1.

If stack 1 not empty push all stack 1 to stack 2

Push element into empty stack 1

Then again push all elements from stack 2 to 1

Stack 2

Stack 1

Stack 2

Stack 1

Stack 2

Stack 1

1

1

Find the first non repeating element in the stream (String) using Queue :

aabccxb🡪 given string 🡪 output : x

Dry run :



a

a

Also as we go through string we add each character visited to

Queue. If the count of character goes above 2 then we pop that

Character from queue.

Here we try to maintain array for each alphabet as alphabet is recognized we increase the count in array by 1

Int count[] = new count[25];

25

0

z

a

b

x

x

b

c

c

b

b

b

b

-1

a

a

a

Que: Interleave two queues (assume given queues has even size)

Given queue : 1 2 3 4 5 6 7 8 9 10

Output : 1 6 2 7 3 8 4 9 5 10

To do so we 1) try to find size of queue

 public static void interLeave(Queue<Integer> q){

        Queue<Integer> firstHalf = new LinkedList<>();

        int size = q.size();

        for(int i=0; i<size/2; i++){

            firstHalf.add(q.remove());

        }

        while (!firstHalf.isEmpty()) {

            q.add(firstHalf.remove());

            q.add(q.remove());

        }

    }

2) we divide queue in two parts

3)first half of queue is stored in

new queue

4)second half is as it is in the

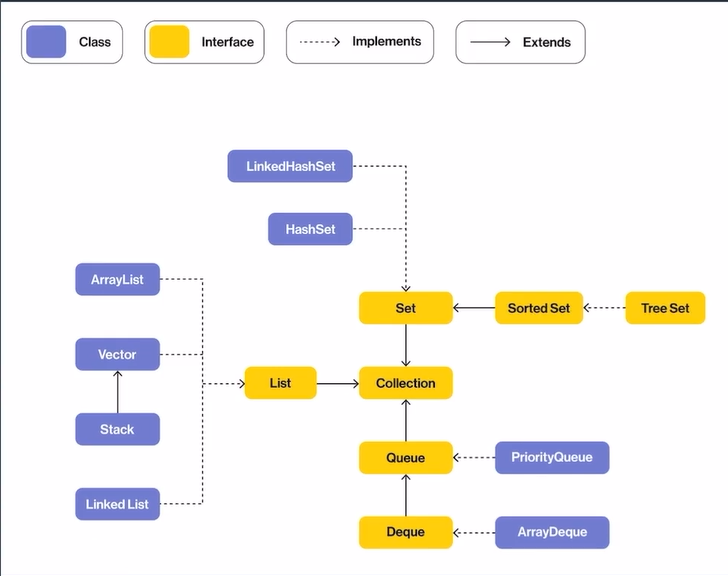
original queue

**Deque : double ended queue**

We can add or remove elements from both the end front and rear

**Operations** :

**addFirst() addLast() removeFirst() removeLast() getFirst() getLast()**

****