A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, the first element is removed first and the last element is removed at last. This interface is dedicated to storing all the elements where the order of the elements matter.

**Methods of Queue**

**add(): Inserts the specified element into the queue. If the task is successful, add() returns true, if not it throws an exception.**

**offer(): Inserts the specified element into the queue. If the task is successful, offer() returns true, if not it returns false.**

**element(): Returns the head of the queue. Throws an exception if the queue is empty.**

**peek(): Returns the head of the queue. Returns null if the queue is empty.**

**remove(): Returns and removes the head of the queue. Throws an exception if the queue is empty.**

**poll(): Returns and removes the head of the queue. Returns null if the queue is empty.**



**FIFO Principle of Queue:**

**A Queue is like a line waiting to purchase tickets, where the first person in line is the first person served. (i.e. First come first serve).**

**Position of the entry in a queue ready to be served, that is, the first entry that will be removed from the queue, is called the front of the queue(sometimes, head of the queue), similarly, the position of the last entry in the queue, that is, the one most recently added, is called the rear (or the tail) of the queue. See the below figure.**



**Characteristics of Queue:**

**Queue can handle multiple data.**

**We can access both ends.**

**They are fast and flexible.**

**/\* Java Program to implement a stack**

**using two queue \*/**

**import java.util.\*;**

**class Stack {**

**Queue<Integer> q1 = new LinkedList<>(),**

**q2 = new LinkedList<>();**

**void remove()**

**{**

**if (q1.isEmpty())**

**return;**

**// Leave one element in q1 and**

**// push others in q2.**

**while (q1.size() != 1) {**

**q2.add(q1.peek());**

**q1.remove();**

**}**

**// Pop the only left element**

**// from q1**

**q1.remove();**

**// swap the names of two queues**

**Queue<Integer> q = q1;**

**q1 = q2;**

**q2 = q;**

**}**

**void add(int x) { q1.add(x); }**

**int top()**

**{**

**if (q1.isEmpty())**

**return -1;**

**while (q1.size() != 1) {**

**q2.add(q1.peek());**

**q1.remove();**

**}**

**// last pushed element**

**int temp = q1.peek();**

**// to empty the auxiliary queue after**

**// last operation**

**q1.remove();**

**// push last element to q2**

**q2.add(temp);**

**// swap the two queues names**

**Queue<Integer> q = q1;**

**q1 = q2;**

**q2 = q;**

**return temp;**

**}**

**int size() { return q1.size(); }**

**// Driver code**

**public static void main(String[] args)**

**{**

**Stack s = new Stack();**

**s.add(1);**

**s.add(2);**

**s.add(3);**

**System.out.println("current size: " + s.size());**

**System.out.println(s.top());**

**s.remove();**

**System.out.println(s.top());**

**s.remove();**

**System.out.println(s.top());**

**System.out.println("current size: " + s.size());**

**}**

**}**

**ArrayList Provides constant time for search operation, so it is better to use it in searching**

**LinkedList provides contant time for add and remove operations . so its better to use this for manipulation.**

**1 2 3 4 5 6 7 8 9 10**

**21 22 23 24 25 26 27 28 29 30**

**Circular Queue: Circular Queue is just a variation of the linear queue in which front and rear-end are connected to each other to optimize the space wastage of the Linear queue and make it efficient.**



**Linear Queue: A Linear Queue is generally referred to as Queue. It is a linear data structure that follows the FIFO (First In First Out) order. A real-life example of a queue is any queue of customers waiting to buy a product from a shop where the customer that came first is served first. In Queue all deletions (dequeue) are made at the front and all insertions (enqueue) are made at the rear end.**



**Reversing queue using another queue**

**package** com.queue.ds;

//Java implementation of the above approach

**import** java.util.\*;

**class** ReverseQueue

{

//Function to return the reversed queue

**static** Queue<Integer> reverse(Queue<Integer> q)

{

// Size of queue

**int** s = q.size();

// Second queue

Queue<Integer> ans = **new** LinkedList<>();

**for** (**int** i = 0; i < s; i++)

{

// Get the last element to the

// front of queue

**for** (**int** j = 0; j < q.size() - 1; j++)

{

**int** x = q.peek();

q.remove();

q.add(x);

}

// Get the last element and

// add it to the new queue

ans.add(q.peek());

q.remove();

}

**return** ans;

}

//Driver Code

**public** **static** **void** main(String[] args)

{

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

// Insert elements

q.add(1);

q.add(2);

q.add(3);

q.add(4);

q.add(5);

q = *reverse*(q);

// Print the queue

**while** (!q.isEmpty())

{

System.***out***.print(q.peek() + " ");

q.remove();

}

}

}