*“Saving the world happens one person at a time. Be at the front of the* ***queue****.”*

*Yaya Toure*

Dear readers, hope you are not bored yet. Are you? But this boredom will not last long, because we will now learn a new data structure today, known as ***Queue***.

Please refer to this [introductory video](https://www.pepcoding.com/resources/online-java-foundation/stacks-and-queues/queues-usage/video) explaining Queue data structure to have a better understanding. After watching the lecture, you can go through the article quickly.

We learnt the Stack data structure before Queue, and both are very similar in types of operations allowed but the way operations are allowed are different.

We will discuss in detail how we can create queue objects using the interface provided in Java Collections, what are the operations allowed in queue, and find some applications of queue data structure in real-world use-cases.

* ***Definition***:

Queue is a ***linear data structure*** which follows a ***FIFO (First In First Out)*** order for doing various operations.

**Q)** What are linear data structures?

**R)** Data structure where data elements are arranged sequentially or linearly, is called a linear data structure. In linear data structure, we can traverse all the data items in a single pass. Until now, all the data structures you have learnt are linear data structures only, whether it is Array, Arraylist or String. Stack is also a linear data structure.

**Q)** What is *FIFO (First In First Out)* order?

**R)** Let us try to get the meaning of FIFO using an example. Consider a ‘queue’ of people standing at an Aadhar Verification Center:



The first person who had joined the queue was the old lady with the stick. And the last person who joined the queue was a girl with a yellow document. Now, the old lady will get her aadhaar verified first, then the businessman, .. and at last the lady with the yellow document.

So, we can say that the first person who first joined the queue was served first, and the last person joining the queue will be served last ( it is nothing but first come first serve). This ordering of elements is known as FIFO (First In First Out).

It is only how elements are removed, i.e. in LIFO order for stacks versus the FIFO order for queues.

* ***Declaration (In Java)***

We can declare a queue using the Queue Interface provided by Java Collections.

Queue<Integer> q;

The above statement will create only a reference to the queue in RAM’s stack memory and this reference will point to NULL.

Now, here comes the catch. How to initialize queue data structure? Can we write the following statement to initialize the queue of integers?

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

Answer is a big ***NO***. For now, please keep in mind that Queue is an ***Interface***. Objects of interfaces cannot be created using the ‘Queue<>()’ statement. It is because there is no body of operations (member functions) in Interface (Queue).

Being an interface the queue needs a concrete class for the declaration. The common classes which can be used to instantiate queue objects are ***LinkedList*** and ***ArrayDeque***.

We will use ArrayDeque in our current sections to learn about queue data structure. Hence, the declaration statement will be:

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

* ***Operations***

These are some of the important operations which are associated with stack data structure:

1. ***add(ele)***: Enqueues/adds an item ele in the queue(to the rear end of the queue).

If the queue is full, then it is said to be a ***Queue Overflow*** condition. Queue overflow can occur when there is no more heap memory, which can be allocated to our queue.

1. ***remove()***: Dequeus/removes an item from the queue and returns the front element which is removed.

The items are popped in the same order in which they are pushed, i.e. the item which is pushed first is also the one which will be removed first.

If the queue is empty, then it is said to be a ***Queue Underflow*** condition. Hence, trying to remove an element from an empty queue (size = 0), will give a run-time error.

1. ***peek()*** (***front()*** in C++): Returns the front element of the queue.

Note: It will give a run-time error, if there is no element present in the queue, i.e. the size of the queue is zero.

1. ***size()***: Returns the number of elements present in the queue.

All these operations: *add, remove, peek* and *size*are ***constant operations***, i.e. the time complexity is ***O(1) per call***.

Hence, if we add n items to the queue (i.e. call add for n times), overall time complexity will be n \* O(1) = O(n), and similarly for any other operation like removing n times from the queue.

Let us take a ***coding exercise*** to learn about the working of these operations. You have to ***predict the output*** of the following code.

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Main

{

public static void main (String[] args) throws java.lang.Exception

{

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

q.add(10);

System.out.println(q + "->" + q.peek() + " " + q.size());

q.add(20);

System.out.println(q + "->" + q.peek() + " " + q.size());

q.add(30);

System.out.println(q + "->" + q.peek() + " " + q.size());

q.add(40);

System.out.println(q + "->" + q.peek() + " " + q.size());

q.remove();

System.out.println(q + "->" + q.peek() + " " + q.size());

q.remove();

System.out.println(q + "->" + q.peek() + " " + q.size());

q.remove();

System.out.println(q + "->" + q.peek() + " " + q.size());

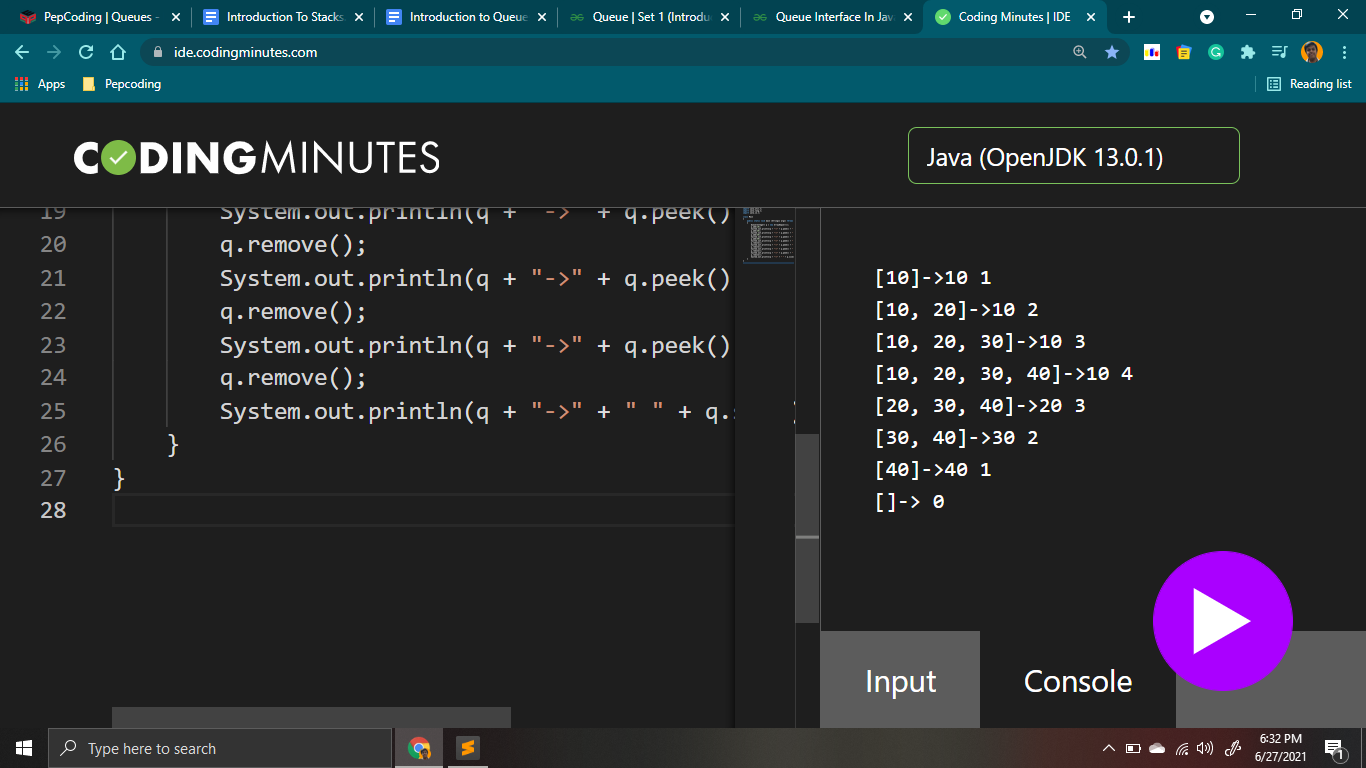
q.remove();

System.out.println(q + "->" + " " + q.size());

}

}

Output:



* ***Applications***
  + Queue can be used for various CPU scheduling algorithms like First Come First Serve (FCFS) scheduling algorithm, etc.
  + Queue is used in networking fields like in routers/switches, mail queues, etc.
  + Queue data structure is used for BFS tree traversals.

In this section, we will apply our knowledge of queues to solve various problems including some of the applications stated above.

**Q)** What aresimilarities and differences between ***stack and queue*** data structures?

***Similarities*** -

* Both are linear data structures.
* Elements can be inserted from one end only. (not taking deque into account).
* Both stack and queue can be implemented using either arrays or linked lists.
* Stacks can be implemented using 2 queues and queues can be implemented using 2 stacks.

***Differences*** -

* Stack follows LIFO (or FILO) order, whereas queue follows FIFO order.
* Insertion & Deletion are from the same end in stack, whereas they are from opposite ends in queue data structure.
* Add & Remove operations are called push and pop for stack, whereas they are called queue & dequeue for queue.
* Stack is used for DFS traversal whereas queue is used for BFS traversal.

I will see you in the next problem: ‘[Normal Queue](https://www.pepcoding.com/resources/online-java-foundation/stacks-and-queues/normal-queue-official/ojquestion)’ where we will try to implement the queue using array data structure. Before moving on to this problem, try to play with queues (provided by java collection) in our *IDE*, explore the basic operations discussed, and have fun!

Contributor : [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)