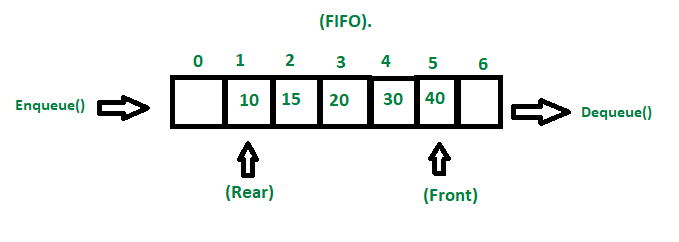
**QUEUE**

**What is queue**

* A queue is a data structure used for storing data (similar to Linked Lists and Stacks).
* In queue, the order in which data arrives is important.
* In general, a queue is a line of people or things waiting to be served in sequential order starting at the beginning of the line or sequence.

**Definition:**

* A *queue* is an ordered list in which insertions are done at one end (*rear*) and deletions are done at other end (*front*).
* The first element to be inserted is the first one to be deleted. Hence, it is called First in First out (FIFO) or Last in Last out (LILO) list.
* Similar to Stacks, special names are given to the two changes that can be made to a queue.
* When an element is inserted in a queue, the concept is called EnQueue.
* When an element is removed from the queue, the concept is called DeQueue.
* DeQueueing an empty queue is called underflow and EnQueuing an element in a full queue is called overflow.

The concept of a queue can be explained by observing a line at a reservation counter. When we enter the line we stand at the end of the line and the person who is at the front of the line is the one who will be served next. He will exit the queue and be served.

**Operations in the queue**

* enQueue(int data): Inserts an element at the end of the queue
* int deQueue(): Removes and returns the element at the front of the queue
* int Front(): Returns the element at the front without removing it
* int QueueSize(): Returns the number of elements stored in the queue
* int IsEmptyQueue(): Indicates whether no elements are stored in the queue or not

**Exceptions**

While executing DeQueue on an empty queue throws an “Empty Queue

Exception” and executing EnQueue on a full queue throws a “Full Queue Exception”.

**Applications**

Following are the some of the applications that use queues.

**Direct Applications**

* Operating systems schedule jobs (with equal priority) in the order of arrival (e.g., a print queue).
* Simulation of real-world queues such as lines at a ticket counter, or any other first-come first-served scenario requires a queue.
* Multiprogramming.
* Asynchronous data transfer (file IO, pipes, sockets).
* Waiting times of customers at call center.
* Determining number of cashiers to have at a supermarket.

**Indirect Applications**

* + Auxiliary data structure for algorithms
  + Component of other data structures

**Basic Operations on Queue:**

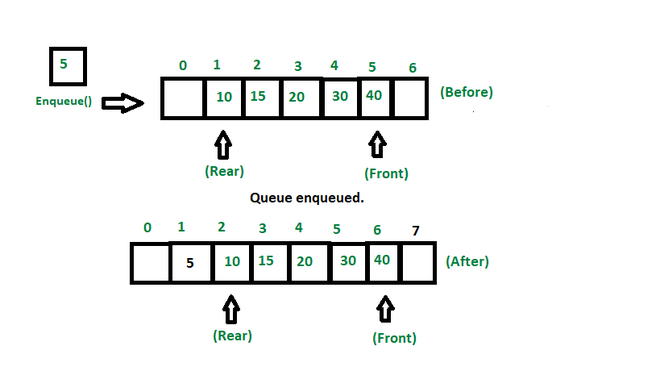
* **enqueue():** Inserts an element at the end of the queue i.e. at the rear end.
* **dequeue():**This operation removes and returns an element that is at the front end of the queue.
* **front():**This operation returns the element at the front end without removing it.
* **rear():**This operation returns the element at the rear end without removing it.
* **isEmpty():**This operation indicates whether the queue is empty or not.
* **isFull():** This operation indicates whether the queue is full or not.
* **size():**This operation returns the size of the queue i.e. the total number of elements it contains.

**Types of Queues:**

* **Simple Queue:**Simple queue also known as a linear queue is the most basic version of a queue. Here, insertion of an element i.e. the Enqueue operation takes place at the rear end and removal of an element i.e. the Dequeue operation takes place at the front end. Here problem is that if we pop some item from front and then rear reach to the capacity of the queue and although there are empty spaces before front means the queue is not full but as per condition in isFull() function, it will show that the queue is full then. To solve this problem we use [circular queue](https://www.geeksforgeeks.org/introduction-and-array-implementation-of-circular-queue/#:~:text=A%20Circular%20Queue%20is%20a,also%20called%20'Ring%20Buffer'.).
* [**Circular Queue**](https://www.geeksforgeeks.org/introduction-and-array-implementation-of-circular-queue/#:~:text=A%20Circular%20Queue%20is%20a,also%20called%20'Ring%20Buffer'.)**:**In a circular queue, the element of the queue act as a circular ring. The working of a circular queue is similar to the linear queue except for the fact that the last element is connected to the first element. Its advantage is that the memory is utilized in a better way. This is because if there is an empty space i.e. if no element is present at a certain position in the queue, then an element can be easily added at that position using modulo capacity(*%n*).
* [**Priority Queue**](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/)**:**This queue is a special type of queue. Its specialty is that it arranges the elements in a queue based on some priority. The priority can be something where the element with the highest value has the priority so it creates a queue with decreasing order of values. The priority can also be such that the element with the lowest value gets the highest priority so in turn it creates a queue with increasing order of values. In pre-define priority queue, C++ gives priority to highest value whereas Java gives priority to lowest value.
* [**Dequeue**](https://www.geeksforgeeks.org/deque-set-1-introduction-applications/)**:**Dequeue is also known as Double Ended Queue. As the name suggests double ended, it means that an element can be inserted or removed from both ends of the queue, unlike the other queues in which it can be done only from one end. Because of this property, it may not obey the First In First Out property.

### ****Enqueue():****

Enqueue() operation in Queue **adds (or stores) an element to the end of the queue**.  
The following steps should be taken to enqueue (insert) data into a queue:

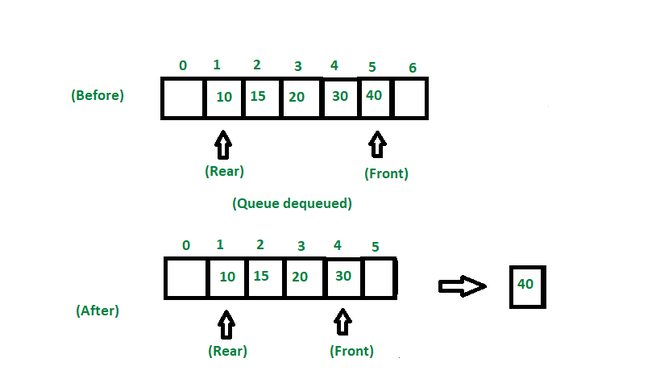
* **Step 1:** Check if the queue is full.
* **Step 2:**If the queue is full, return overflow error and exit.
* **Step 3:** If the queue is not full, increment the rear pointer to point to the next empty space.
* **Step 4:** Add the data element to the queue location, where the rear is pointing.
* **Step 5:** return success.

|  |
| --- |
| void queueEnqueue(int data)  {      // Check queue is full or not      if (capacity == rear) {          System.out.println("\nQueue is full\n");          return;      }        // Insert element at the rear      else {          queue[rear] = data;          rear++;      }      return;  } |

### ****Dequeue():****

Removes (or access) the first element from the queue.  
The following steps are taken to perform the dequeue operation:

* **Step 1:** Check if the queue is empty.
* **Step 2:** If the queue is empty, return the underflow error and exit.
* **Step 3:** If the queue is not empty, access the data where the front is pointing.
* **Step 4:** Increment the front pointer to point to the next available data element.
* **Step 5:** The Return success.

****

|  |
| --- |
| void queueDequeue()  {      // If queue is empty      if (front == rear) {          System.out.println("\nQueue is empty\n");          return;      }        // Shift all the elements from index 2      // till rear to the left by one      else {          for (int i = 0; i < rear - 1; i++) {              queue[i] = queue[i + 1];          }            // decrement rear          rear--;      }      return;  } |

### ****front():****

This operation returns the element at the front end without removing it.

|  |
| --- |
| int front(Queue queue)  {      if (isempty(queue))          return Integer.MIN\_VALUE;      return queue.arr[queue.front];  } |

### ****rear():****

This operation returns the element at the rear end without removing it.

|  |
| --- |
| public static int rear(Queue<Integer> myQueue)  {      if (myQueue.isEmpty()) {          System.out.println("Queue is empty.");          return -1;      }        int rearElement = -1;      while (!myQueue.isEmpty()) {          rearElement = myQueue.poll();      }        return rearElement;  } |

### ****isEmpty():****

This operation returns a boolean value that indicates whether the queue is empty or not.

|  |
| --- |
| boolean isEmpty()  {      if (front == -1)          return true;      else          return false;  } |

### ****isFull():****

This operation returns a Boolean value that indicates whether the queue is full or not.

|  |
| --- |
| boolean isFull()  {      if (front == 0 && rear == MAX\_SIZE - 1) {          return true;      }      return false;  } |