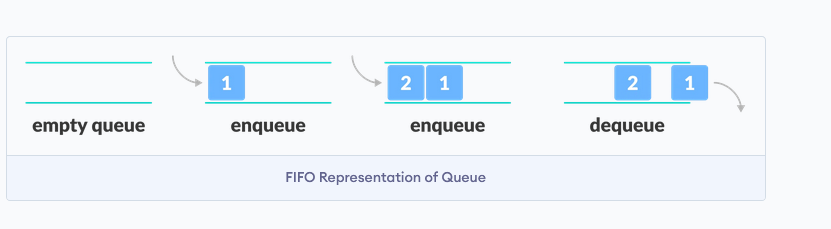
A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

Queue follows the **First In First Out (FIFO)** rule - the item that goes in first is the item that comes out first.



**Basic Operations of Queue**

A queue is an object (an abstract data structure - ADT) that allows the following operations:

* **Enqueue**: Add an element to the end of the queue
* **Dequeue**: Remove an element from the front of the queue
* **IsEmpty**: Check if the queue is empty
* **IsFull**: Check if the queue is full
* **Peek**: Get the value of the front of the queue without removing it

**Working of Queue**

Queue operations work as follows:

* two pointers *FRONT* and *REAR*
* *FRONT* track the first element of the queue
* *REAR* track the last element of the queue
* initially, set value of *FRONT* and *REAR* to -1

### Enqueue Operation

* check if the queue is full
* for the first element, set the value of FRONT to 0
* increase the REAR index by 1
* add the new element in the position pointed to by REAR

### Dequeue Operation

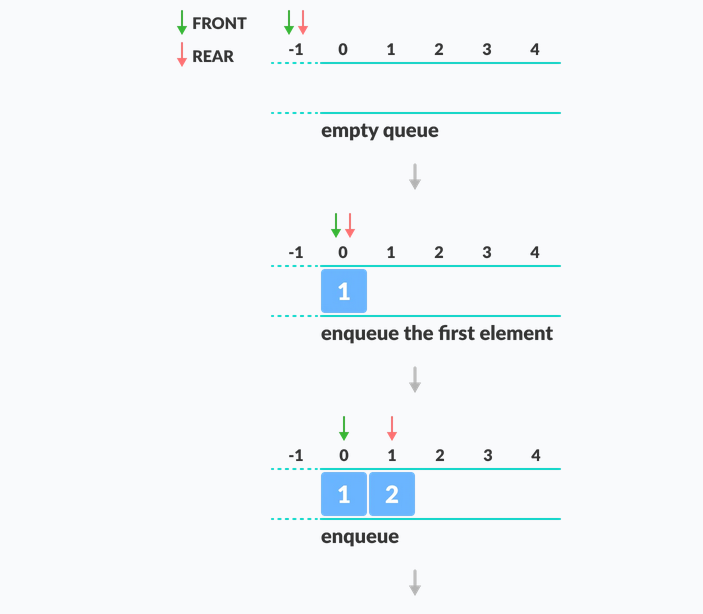
* check if the queue is empty
* return the value pointed by FRONT
* increase the FRONT index by 1
* for the last element, reset the values of FRONT and REAR to -1

### Enqueue Operation

* check if the queue is full
* for the first element, set the value of FRONT to 0
* increase the REAR index by 1
* add the new element in the position pointed to by REAR

### Dequeue Operation

* check if the queue is empty
* return the value pointed by FRONT
* increase the FRONT index by 1
* for the last element, reset the values of FRONT and REAR to -1



public class Queue {

int SIZE = 5;

int items[] = new int[SIZE];

int front, rear;

Queue() {

front = -1;

rear = -1;

}

boolean isFull() {

if (front == 0 && rear == SIZE - 1) {

return true;

}

return false;

}

boolean isEmpty() {

if (front == -1)

return true;

else

return false;

}

void enQueue(int element) {

if (isFull()) {

System.out.println("Queue is full");

} else {

if (front == -1)

front = 0;

rear++;

items[rear] = element;

System.out.println("Inserted " + element);

}

}

int deQueue() {

int element;

if (isEmpty()) {

System.out.println("Queue is empty");

return (-1);

} else {

element = items[front];

if (front >= rear) {

front = -1;

rear = -1;

} /\* Q has only one element, so we reset the queue after deleting it. \*/

else {

front++;

}

System.out.println("Deleted -> " + element);

return (element);

}

}

void display() {

/\* Function to display elements of Queue \*/

int i;

if (isEmpty()) {

System.out.println("Empty Queue");

} else {

System.out.println("\nFront index-> " + front);

System.out.println("Items -> ");

for (i = front; i <= rear; i++)

System.out.print(items[i] + " ");

System.out.println("\nRear index-> " + rear);

}

}

public static void main(String[] args) {

Queue q = new Queue();

// deQueue is not possible on empty queue

q.deQueue();

// enQueue 5 elements

q.enQueue(1);

q.enQueue(2);

q.enQueue(3);

q.enQueue(4);

q.enQueue(5);

// 6th element can't be added to because the queue is full

q.enQueue(6);

q.display();

// deQueue removes element entered first i.e. 1

q.deQueue();

// Now we have just 4 elements

q.display();

}

}