Queue

Queues are data structures that follow the **First In First Out (FIFO)** i.e. the first element that is added to the queue is the first one to be removed.

Elements are always added to the back and removed from the front. Think of it as a line of people waiting for a bus. The person who is at the beginning of the line is the first one to enter the bus.

 New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed **enqueue** and **dequeue**. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item. The picture demonstrates the FIFO access.

The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

|  |
| --- |
| public class QueueUsingArrayMain {    private int capacity;  int queueArr[];  int front;  int rear ;  int currentSize = 0;    public QueueUsingArrayMain(int sizeOfQueue){  this.capacity = sizeOfQueue;  front = 0;  rear = -1;  queueArr = new int[this.capacity];  }    /\*\*  \* this method is used to add element in the queue  \* @param data  \*/  public void enqueue(int data) {  if (isFull()) {  System.out.println("Queue is full!! Can not add more elements");  } else {  rear++;  if(rear == capacity-1){  rear = 0;  }  queueArr[rear] = data;  currentSize++;  System.out.println(data+ " added to the queue");  }  }    /\*\*  \* This method removes an element from the front of the queue  \*/  public void dequeue() {  if (isEmpty()) {  System.out.println("Queue is empty!! Can not dequeue element");  } else {  front++;  if(front == capacity-1){  System.out.println(queueArr[front-1]+" removed from the queue");  front = 0;  } else {  System.out.println(queueArr[front-1]+" removed from the queue");  }  currentSize--;  }  }    /\*\*  \* This method is use to check if element is full or not  \* @return boolean  \*/  public boolean isFull(){  if (currentSize == capacity){  return true;  }  return false;  }    /\*\*  \* This method is use to check if element is empty or not  \* @return  \*/  public boolean isEmpty(){    if (currentSize == 0){  return true;  }  return false;  }    public static void main(String a[]){    QueueUsingArrayMain queue = new QueueUsingArrayMain(5);  queue.enqueue(6);  queue.dequeue();  queue.enqueue(3);  queue.enqueue(99);  queue.enqueue(56);  queue.dequeue();  queue.enqueue(43);  queue.dequeue();  queue.enqueue(89);  queue.enqueue(77);  queue.dequeue();  queue.enqueue(32);  queue.enqueue(232);  }  } |

Queue implementation in java

In this post , we will see how to implement Queue using Array in java.  
**Queue** is abstract data type which demonstrates **First in first out (FIFO)** behavior. We will implement same behavior using Array.  
Although java provides implementation for  all abstract data types such as Stack , Queue and LinkedList but it is always good idea to understand basic data structures and implement them yourself.  
Please note that Array implementation of Queue is not dynamic in nature.  
There are two most important operations of Queue:  
**enQueue :**It is operation when we insert element into the queue.  
**deQueue :** It is operation when we remove element from the queue. Read more at

**Java Program to implement Queue using Array:**

When you run above program, you will get below output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | 6 added to the queue  6 removed from the queue  3 added to the queue  99 added to the queue  56 added to the queue  3 removed from the queue  43 added to the queue  99 removed from the queue  89 added to the queue  77 added to the queue  56 removed from the queue  32 added to the queue  232 added to the queue |