Department of Software Engineering Mehran University of Engineering and Technology, Jamshoro			
Course: SWE224 – Data Structure & Algorithms			
Instructor	Mariam Memon	**Practical/Lab No.**	07
Date		**CLOs**	3
Signature		**Assessment Score**	01

Topic	**Implementation of Queues**
Objectives	To implement Queue ADT and its various methods in Java.

Lab Discussion: Theoretical concepts and Procedural steps

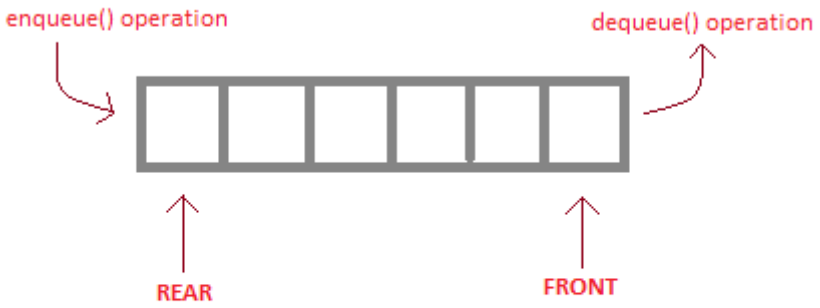
Queue

Queue is an abstract data type or a linear data structure, just like stacks, in which the first element is inserted from one end called the **REAR**(also called **tail**), and the removal of existing element takes place from the other end called as **FRONT**(also called **head**).

This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first.

Which is exactly how queue system works in real world. If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Right? Same is the case with Queue data structure. Data inserted first, will leave the queue first.

The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.



enqueue( ) is the operation for adding an element into Queue.

dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

Applications of Queue

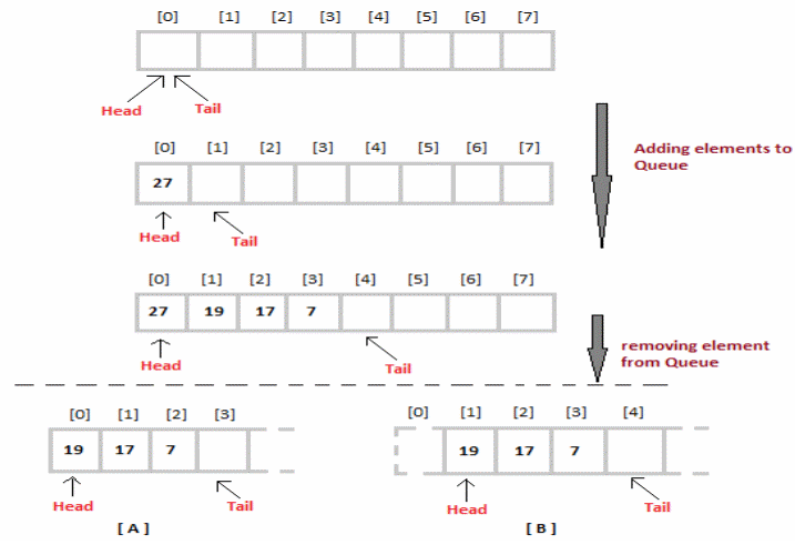
Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios:

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.

Implementation of Queue Data Structure

Queue can be implemented using an Array, Stack or Linked List.

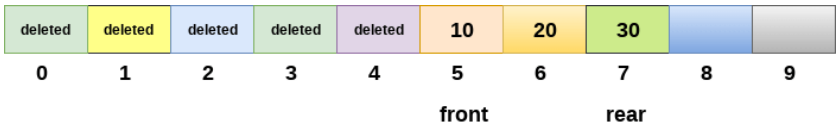
Queue Implementation using Arrays



Drawback of array implementation

Although, the technique of creating a queue is easy, but there are some drawbacks of using this technique to implement a queue.

- **Memory wastage :** The space of the array, which is used to store queue elements, can never be reused to store the elements of that queue because the elements can only be inserted at front end and the value of front might be so high so that, all the space before that, can never be filled.



limitation of array representation of queue

The above figure shows how the memory space is wasted in the array representation of queue. In the above figure, a queue of size 10 having 3 elements, is shown. The value of the front variable is 5, therefore, we can not reinsert the values in the place of already deleted element before the position of front. That much space of the array is wasted and can not be used in the future (for this queue).

- **Deciding the array size**

On of the most common problem with array implementation is the size of the array which requires to be declared in advance. Due to the fact that, the queue can be extended at runtime depending upon the problem, the extension in the array size is a time taking process and almost impossible to be performed at runtime since a lot of reallocations take place. Due to this reason, we can declare the array large enough so that we can store queue elements as enough as possible but the main problem with this declaration is that, most of the array slots (nearly half) can never be reused. It will again lead to memory wastage.

Queue Implementation in Java using Linked Structures

Due to the drawbacks discussed in the previous section of this tutorial, the array implementation can not be used for the large scale applications where the queues are implemented. One of the alternative of array implementation is linked list implementation of queue.

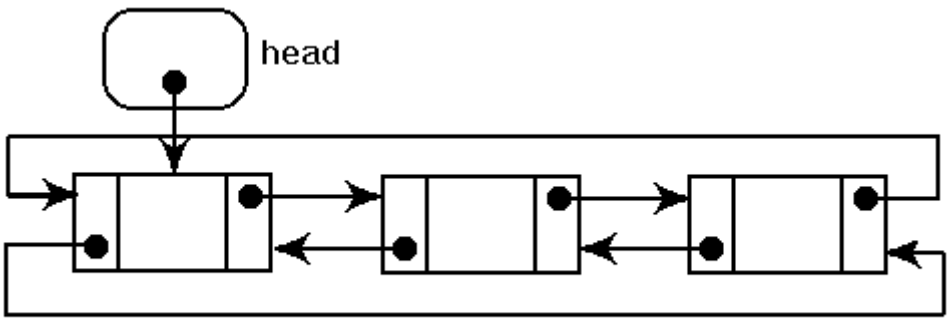
The storage requirement of linked representation of a queue with n elements is o(n) while the time requirement for operations is o(1).

In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. Each element of the queue points to its immediate next element in the memory.

In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.

Insertion and deletions are performed at rear and front end respectively. If front and rear both are NULL, it indicates that the queue is empty.

The linked representation of queue is shown in the following figure.



```
public interface Queue
{
	public void add(Object object);
	public Object first();
	public Object remove();
	public int size();
}

public class LinkedQueue implements Queue
{
	private Node head = new Node(null);
	private int size;

	private static class Node
	{
		Object object;
		Node prev = this, next = this;
		Node(Object object)
		{
			this.object = object;
		}
		Node(Object object, Node prev, Node next)
		{
			this.object = object;
			this.prev = prev;
			this.next = next;
		}
	}

	public void add(Object object)
	{
		head.prev = head.prev.next = new Node(object, head.prev, head);
		++size;
	}

	public Object first()
	{
		if(size==0) throw new IllegalStateException("Queue is empty");
		return head.next.object;
	}

	public Object remove()
	{
		if(size==0) throw new IllegalStateException("Queue is empty");
```

```
                Object object = head.next.object;
                head.next = head.next.next;
                head.next.prev = head;
                --size;
                return object;
        }

        public int size()
        {
                return size;
        }

        public boolean isEmpty()
        {
                return size==0;
        }
}
```

Lab Tasks

Task 1: Implement an ArrayQueue.

Task 2: Write and test the following methods in both ArrayQueue and LinkedQueue class:
1. toString()
2. equals()
3. clone()
4. removeBottomElement()
5. reverse()
6. toStack()

Task 3: Implement and test LinkedQueue toLinkedQueue() method in ArrayQueue class.

Task 4: Implement & test ArrayQueue toArrayQueue() & Object[] toArray() method in LinkedQueue class.

Task 5: Implement a DeQueue.
Deque or Double Ended Queue is a type of queue in which insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow FIFO rule.

Lab Rubrics for Evaluation

Rubrics	Proficient	Adequate	Poor
Programming Algorithms	Completely accurate and efficient design of algorithms **(0.4)**	Accurate but inefficient algorithms **(0.2)**	Unable to design algorithms for the given problems **(0.0)**
Originality	Submitted work shows large amount of original thought **(0.3)**	Submitted work shows some amount of original thought **(0.2)**	Submitted work shows no original thought **(0.0)**
Troubleshooting	Easily traced and corrected faults **(0.3)**	Traced and corrected faults with some guidance **(0.2)**	No troubleshooting skills **(0.0)**