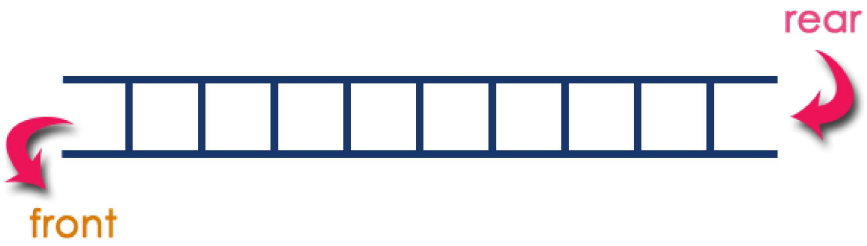
Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Queue

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing elements are performed at two different positions. The insertion is performed at one end and deletion is performed at another end.

In a queue data structure, the insertion operation is performed at a position which is known as **'rear'** or **'tail'** and the deletion operation is performed at a position which is known as **'front'** or **'Head'**. In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle.



Queue Example:

Suppose you are going to a bank and there are people already waiting to be served then you will stand behind them in line wait for your turn. In the mean while the person who came first will be served. This is the best example you can consider for a queue.

Queue ADT

The queue abstract data type is defined by the following structure and operations. A queue is structured, as described above, as an ordered collection of items which are added at one

end, called the **'rear'** or **'tail'** and removed from the other end, called the **'front'** or **'head'**. Queues maintain a ***FIFO*** ordering property.

Primitive Operations:

The queue operations are given below.

Operation	Description
`isEmpty()`	Determines whether the queue is empty. If queue is empty it returns ***true*** otherwise ***false***.
`isFull()`	Determines whether the queue is full. If queue is full it returns ***true*** otherwise ***false***.
`enqueue()`	Adds a new item to the rear of the queue.
`dequeue()`	Removes the item from the front of the queue.
`Front()`	Retrieves and returns the item next in line to be removed without removing it.
`size()`	Returns the number of items currently in the queue.

Representation of Queue using Array

The code below represents a queue and the implementation of primitive operations.

```
public class MyQueue
{
    public static int nSize = 5;
    int[] arr = new int[nSize];
    int head = 0;
    int tail = -1;

    public boolean isEmpty()
    {
        if(tail < 0)
            return true;
        else
            return false;
    }

    public boolean isFull()
    {
        if(tail == nSize-1)
            return true;
        else
            return false;
    }

    public void enqueue(int value)
    {
        if(isFull())
            System.out.println("Queue is full");
```

```
        else
            arr[++tail] = value;
    }

    public int dequeue()
    {
        if(isEmpty())
        {
            System.out.println("Queue is Empty");
            return -9999;
        }
        else
        {
            int value = arr[head];
            moveArray();
            return value;
        }
    }

    public int front()
    {
        if(isEmpty())
        {
            System.out.println("Queue is Empty");
            return -9999;
        }
        else
            return arr[head];
    }

    public int size()
    {
        return (tail+1);
    }

    private void moveArray()
    {
        for(int i = 0; i < tail; i++)
        {
            arr[i] = arr[i+1];
        }
        tail--;
    }
}
```

Code above is the code for queue class. However, to test the MyQueue class we will use the following code in the main class.

```
public class QueueDemo
{
    public static void main(String[] args)
```

```
    {
        MyQueue que = new MyQueue();
        int x = que.dequeue();
        que.enqueue(23);
        que.enqueue(2);
        que.enqueue(73);
        System.out.println(que.front());
        que.enqueue(21);
        que.enqueue(109);
        System.out.println("Queue Size is " + que.size());
        System.out.println("Is Queue Full: " + que.isFull());
        que.enqueue(13);
        x = que.dequeue();
        System.out.println("Dequeued Value Received is " + x);
        x = que.dequeue();
        x = que.dequeue();
        x = que.dequeue();
        x = que.dequeue();
        System.out.println("Dequeued Value Received is " + x);
        x = que.dequeue();
    }
}
```

Output:

```
Queue is Empty
23
Queue Size is 5
Is Queue Full: true
Queue is full
Dequeued Value Received is 23
Dequeued Value Received is 109
Queue is Empty
```

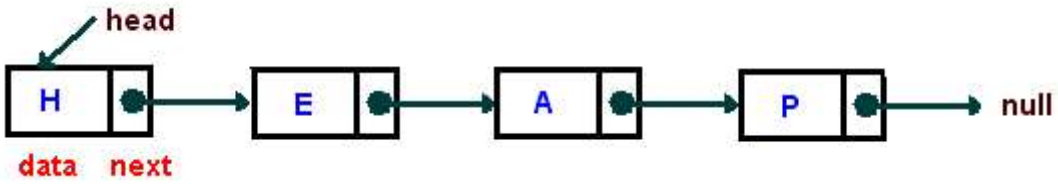
Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Queue Using Linked List (Dynamic Queue)

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing elements are performed at two different positions. The insertion is performed at one end and deletion is performed at another end. In queue data structure, the insertion and deletion operations are performed based on ***First In First Out (FIFO)*** principle.

On the other hand, a linked list is a linear data structure where each element is a separate object or ***node***.



We have already discussed both the data structures in detail in our previous lectures. Today we will use the dynamic property (***variable size***) of ***linked list*** to create the ***dynamic Queue*** unlike stack we created using ***array***.

Code for Dynamic Queue

The code created for ***dynamic queue*** class is also ***generic*** so you can utilize this queue for any type of data you want. The code is given below:

```
public class DyQueue <T>
{
    class Node
    {
        T data;
        Node link;
    }

    Node head = null;
    Node tail = null;

    public boolean isEmpty()
    {
```

```
        if(head == null)
            return true;
        else
            return false;
    }

    public void enqueue(T value)
    {
        Node n = new Node();
        n.data = value;
        n.link = null;

        if(head == null)
        {
            tail = n;
            head = n;
        }
        else
        {
            tail.link = n;
            tail = n;
        }
    }

    public T dequeue()
    {
        if(isEmpty())
        {
            System.out.println("Queue is Empty");
            return null;
        }
        else
        {
            T value = head.data;
            head = head.link;
            return value;
        }
    }

    public void peek()
    {
        if(!isEmpty())
            System.out.println("Next Value = " +head.data);
    }

    public void print()
    {
        Node temp = head;
        System.out.println("\nHEAD                    TAIL");
        System.out.println("-------------------------");
        if(!isEmpty())
        {
```

```
            while(temp != null)
            {
                System.out.print(" | " +temp.data);
                temp = temp.link;
            }
        }
        System.out.println("\n-------------------------");
    }
}
```

Code above is the code for Dynamic Queue. However, to demonstrate the working of dynamic Queue class we will use the following code in the main class.

```
public class DyQueueDemo
{
    public static void main(String[] args)
    {
        DyQueue<Integer> q1 = new DyQueue<>();
        DyQueue<Character> q2 = new DyQueue<>();

        q1.dequeue();
        q1.enqueue(10);
        q1.enqueue(20);
        q1.enqueue(30);
        q1.peek();
        q1.print();
        System.out.println("Dequeued: "+ q1.dequeue());
        System.out.println("Dequeued: "+ q1.dequeue());
        q1.print();

        q2.dequeue();
        q2.enqueue('A');
        q2.enqueue('B');
        q2.enqueue('C');
        q2.peek();
        q2.print();
        System.out.println("Dequeued: "+ q2.dequeue());
        System.out.println("Dequeued: "+ q2.dequeue());
        q2.print();
    }
}
```

Output:

```
Queue is Empty
Next Value = 10

HEAD                    TAIL
-------------------------
 | 10 | 20 | 30
-------------------------
```

```
Dequeued: 10
Dequeued: 20

HEAD                    TAIL
-------------------------
 | 30
-------------------------
Queue is Empty
Next Value = A

HEAD                    TAIL
-------------------------
 | A | B | C
-------------------------
Dequeued: A
Dequeued: B

HEAD                    TAIL
-------------------------
 | C
-------------------------
```