What is a Queue:

✓ *Observations from above Picture:*
- ✓ *New addition of members happens at end of the queue.*
- ✓ *First person in the queue is the first to get out from queue.*
- ✓ *follows FIFO (First in First Out) method*

✓ *Property of Queue:*
- ✓ *follows FIFO (First in First Out) method*

Why should we learn queue ?

FIFO

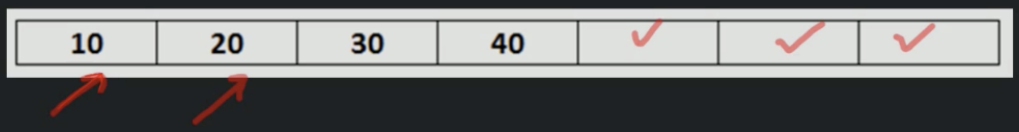
- ✓ When we need to create an application which utilizes 'first incoming data first'.
  - ✓ Example: implementation of 'billing counter'.

Common operations in Queue:

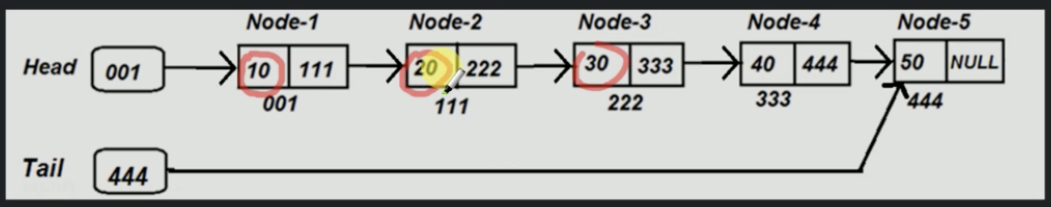
- ✓createQueue()
- ✓enQueue()
- ✓deQueue()
- ✓peekInQueue()
- ✓isEmpty()
- ✓isFull()
- ✓deleteQueue()

Implementation Options of Queue:

- ✓ **Array:**
  - ✓ Linear Queue
  - ✓ Circular Queue



- ✓ **Linked List**
  - ✓ Linear Queue
  - ✓ ~~Circular Queue~~



Creation of Linear Queue (Array Implementation):

createQueue(size)

create a blank array of 'size'

initialize topOfQueue, beginningOfQueue to -1

Time Complexity - Creation of Linear Queue(Array Implementation):

createQueue(size)

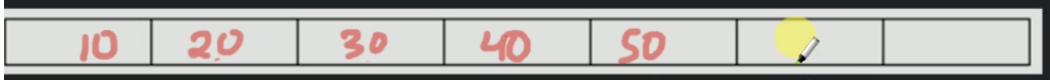
create a blank array of 'size' ---------- *O(1)*

initialize topOfQueue, beginningOfQueue ---------- *O(1)*

Time Complexity – *O(1)*

Space Complexity – *O(n)*

Enqueue operation of Linear Queue(Array Implementation):



60

FIFO

```
enQueue(Value):
    if Queue is full
        return error message
    else
        arr [topOfQueue + 1 ] = Value
        topOfQueue ++
```

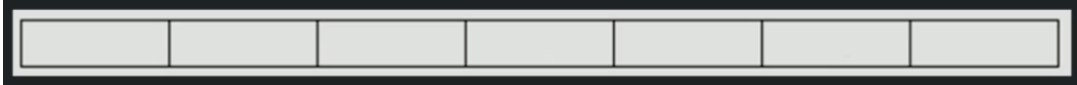
Time Complexity - Enqueue operation of Linear Queue(Array Implementation):

```
enQueue(Value):
    if Queue is full ---------- O(1)
        return error message ---------- O(1)
    else ---------- O(1)
        arr [topOfQueue + 1 ] = Value ---------- O(1)
        topOfQueue ++ ---------- O(1)
```

Time Complexity – O(1)

Space Complexity – O(1)

Dequeue operation of Linear Queue(Array Implementation):



FIFO

```
deQueue():
    if queue is empty
        return error message
    else
        print arr [beginningOfQueue]
        beginningOfQueue ++
        if (beginningOfQueue > endOfQueue) //If last element in the Queue is Dequeued
            beginningOfQueue = topOfQueue = -1
```

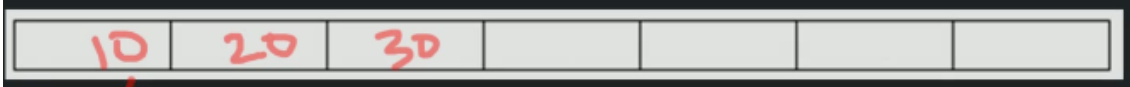
Time Complexity - Dequeue operation of Linear Queue(Array Implementation):

```
deQueue():
    if queue is empty ---------- O(1)
        return error message ---------- O(1)
    else ---------- O(1)
        print first value from queue ---------- O(1)
        beginningOfQueue ++ ---------- O(1)
        if (beginningOfQueue > endOfQueue) ---------- O(1)
            beginningOfQueue = -1 ---------- O(1)
```

Time Complexity – O(1)

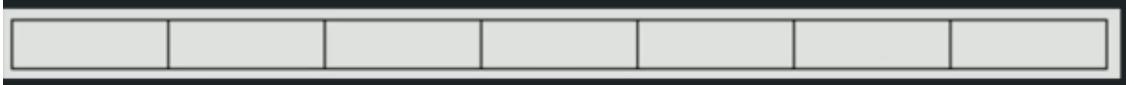
Space Complexity – O(1)

Peek operation of Linear Queue(Array Implementation):




```
peek()
    if queue is empty
        return error message
    else
        print arr [beginningOfQueue]
```

Deleting a Linear Queue(Array Implementation):



deleteQueue():

array = null

Time Complexity - Deleting a Linear Queue(Array Implementation):

deleteQueue():

array = null ---------- O(1)

Time Complexity – O(1)

Space Complexity – O(1)

Time & Space Complexity of Linear Queue(Array Implementation):

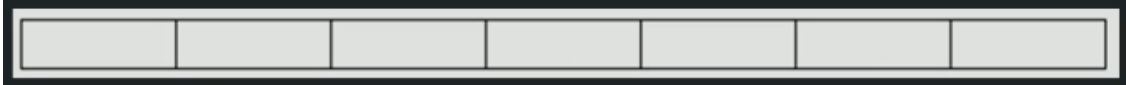
Particulars	Time Complexity	Space Complexity
CreateQueue()	O(1)	O(n)
Enqueue()	O(1)	O(1)
Dequeue()	O(1)	O(1)
Peek()	O(1)	O(1)
isEmpty()	O(1)	O(1)
isFull()	O(1)	O(1)
DeleteQueue()	O(1)	O(1)

CreateQueue space complexity is O(n)when we need define size of array to initialiaze but swift allow us to define array without size so space complexity will O(1)

Why learn Circular Queue ?

✓ deQueue operation causes blank cells Linear Queue(Array Implementation). We need to improve that.

Creation of Circular Queue(Array Implementation):



createQueue(size)

create a blank array of 'size'

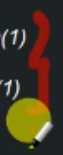
initialize top, start = -1

Time Complexity - Creation of Circular Queue(Array Implementation):

createQueue(size)

create a blank array of 'size' ---------- *O(1)*

initialize top, start = -1 ---------- *O(1)*



Time Complexity – O(1)

Space Complexity – O(n)

Enqueue operation of Circular Queue(Array Implementation):


```
enQueue(Value):
    if (isQueueFull()) Print "Queue overflow error!"
    else
        if (topOfQueue+1 == size) { //if top is already at last cell of array, then reset it to first cell
            topOfQueue=0;
    else
            topOfQueue++;
     arr[topOfQueue] = value;
```

Time Complexity - Enqueue operation of Circular Queue(Array Implementation):

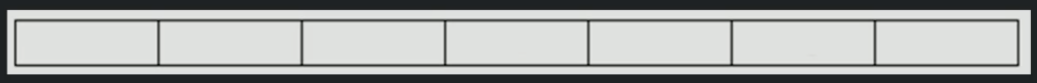
```
enQueue(Value):
    if (isQueueFull()) Print "Queue overflow error!"  ---------- O(1)
    else  ---------- O(1)
        if (topOfQueue+1 == size) ---------- O(1)
            topOfQueue=0; ---------- O(1)
    else  ---------- O(1)
            topOfQueue++; ---------- O(1)
     arr[topOfQueue] = value; ---------- O(1)
```

Progress

Time Complexity – O(1)

Space Complexity – O(1)

Dequeue operation of Circular Queue(Array Implementation):



```
Dequeu()
    if (isQueueEmpty()) Print (Queue underflow error)
    else
        Print (arr[start]);
        if (start == topOfQueue) { //if there is only 1 element in Queue
            start = topOfQueue = -1;
        else if (start+1 == size) { //if start has reached end of array, then start again from 0
            start=0;
        else
            start++
```

Time Complexity - Dequeue operation of Circular Queue(Array Implementation):

```
Dequeu()
    if (isQueueEmpty()) Print (Queue underflow error) ---------- O(1)
    else  ---------- O(1)
        Print (arr[start]); ---------- O(1)
        if (start == topOfQueue)  ---------- O(1)
            start = topOfQueue = -1; ---------- O(1)
        else if (start+1 == size) ---------- O(1)
            start=0; ---------- O(1)
        else  ---------- O(1)
            start++ ---------- O(1)
```

Time Complexity – O(1)

Space Complexity – O(1)

Peek operation of Circular Queue(Array Implementation):


```
peek()
 if (!isQueueEmpty())
       print(arr[start])
 else
    print ("The queue is empty!!")
```

Time & Space Complexity of Circular Queue(Array Implementation):

Particulars	Time Complexity	Space Complexity
CreateQueue()	O(1)	O(n)
Enqueue()	O(1)	O(1)
Dequeue()	O(1)	O(1)
Peek()	O(1)	O(1)
isEmpty()	O(1)	O(1)
isFull()	O(1)	O(1)
DeleteQueue()	O(1)	O(1)

Creation of Queue(Linked List Implementation):

createQueue()

create a blank SingleLinkedList

Time Complexity - Creation of Queue(Linked List Implementation):

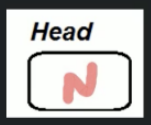
createQueue():

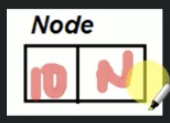
create a blank SingleLinkedList ---------- O(1)

Time Complexity – O(1)

Space Complexity – O(1)

Enqueue operation of Queue(Linked List Implementation):





FIFO


```
enQueue(nodeValue):
    create a node
    node.value = nodeValue
    node.next = null
1.  if tail equals null // if queue is empty
        tail = node
    else  //if queue is not empty
2.      tail.next = node
        tail = node
```

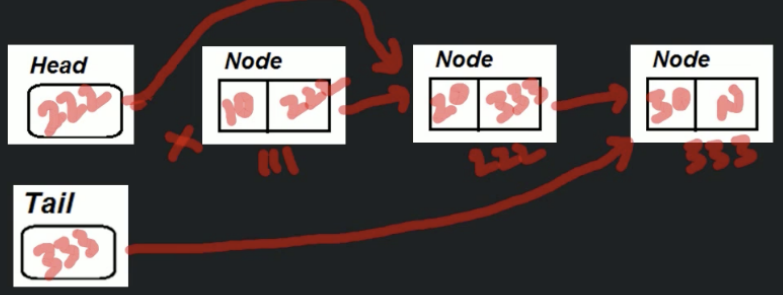
Time Complexity - Enqueue operation of Queue(Linked List Implementation):

```
enQueue(nodeValue):
    create a node ---------- O(1)
    node.value = nodeValue ---------- O(1)
    node.next = null ---------- O(1)
    if tail equals null ---------- O(1)
        tail = node ---------- O(1)
    else ---------- O(1)
        tail.next = node ---------- O(1)
        tail = node ---------- O(1)
```

Time Complexity – O(1)

Space Complexity – O(1)

Dequeue operation of Queue(Linked List Implementation):




```
deQueue():
    if header equals null
        return error message
    tmpNode = header
    header = header.next
    return tmpNode.value
```

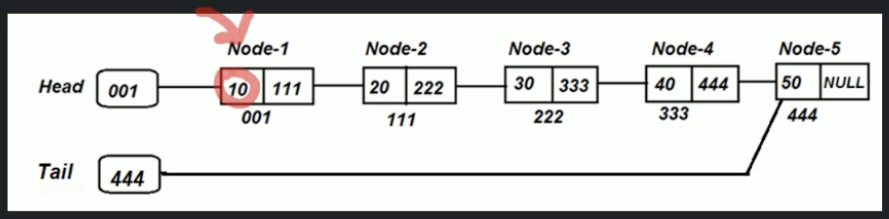
Time Complexity - Dequeue operation of Queue(Linked List Implementation):

```
deQueue():
    if header equals null ---------- O(1)
        return error message ---------- O(1)
    tmpNode = header ---------- O(1)
    header = header.next ---------- O(1)
    return tmpNode.value ---------- O(1)
```

Time Complexity – O(1)

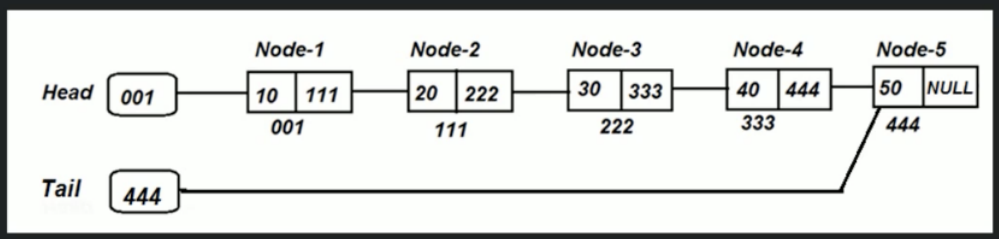
Space Complexity – O(1)

Peek operation of Queue(Linked List Implementation):




```
peek():
    if header equals null
        return error
    else
        return header.value
```

Deleting a Queue(Linked List Implementation):



deleteQueue():

head = tail = null

Time Complexity - Deleting a Queue(Linked List Implementation):

deleteQueue():

head = tail = null ---------- O(1)

Time Complexity – O(1)

Space Complexity – O(1)

Time & Space Complexity of Queue(Linked List Implementation):

Operations	Time Complexity	Space Complexity
CreateQueue()	O(1)	O(1)
Enqueue()	O(1)	O(1)
Dequeue()	O(1)	O(1)
Peek()	O(1)	O(1)
isEmpty()	O(1)	O(1)
~~isFull()~~	~~O(1)~~	~~O(1)~~
DeleteQueue()	O(1)	O(1)

When to Use/Avoid Queue:

✓ **When to Use:**

- ✓ Helps manage the data in particular way (FIFO).
- ✓ Not easily corrupted (No one can easily inset data in middle)

✓ **When to Avoid:**

- ✓ Random access not possible – if we have done some mistake, its costly to rectify.