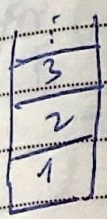
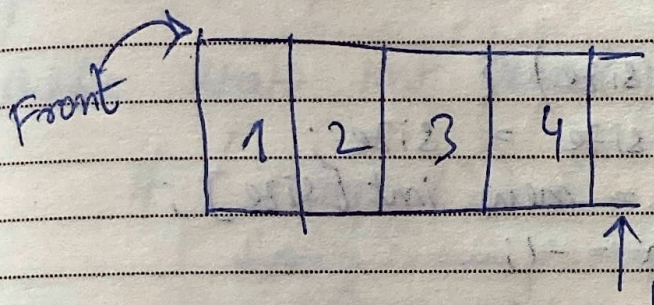
Queue

In stack, we used to add elements one on top of the other. And we saw that it followed a LIFO structure.



However, in Queue we follow a front-Rear data structure. we add elements from the Rear.



When we pop out elements from a Queue, we do that from the front. So, we pop out 1, then 2, ... so on.
Here, in Queue we follow FIFO (First In First Out) structure.

operations

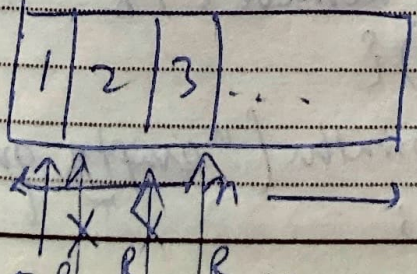
Note: In array, add is O(1). But remove and peek is O(n). So, not optimised to implement Queue by array

Enque → Add → adding elements

Dequeue → Remove → removing elements

Front → Peek → to view value of front elements

Implementation 1 (Queue using Array)



Size = n (fixed)
Initially, Front (F) = -1, R = -1
then F = 0, R = 0, then F = 0, R = 1, and so on.

25 July 2022
Monday
31st Wk • 206-159

Queue using Array

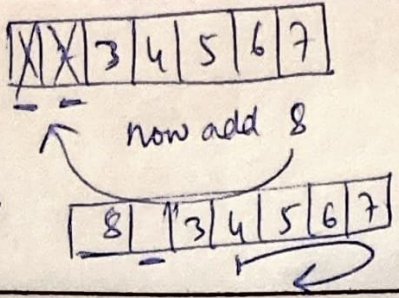
```
Public class QueueB {
    Static class Queue {
        Static int arr[];
        Static int size;
        Static int rear = -1;

        Queue (int size) {
            this.size = size;
            arr = new int[size];
            rear = -1;
        }
        Public Static boolean isEmpty() {
            return (rear == -1);
        }
        Public Static boolean isFull() {
            return (rear == size-1);
        }
        Public Static void add (int data) {
            if (isFull()) {
                System.out.println("overflow");
                return;
            }
            arr[++rear] = data;
        }
        // O(n)
        Public Static int remove() {
            if (isEmpty()) {
                System.out.println("empty queue");
                return -1;
            }
```

```
    int front = arr[0];
    for (int i= 0; i< rear; i++)
    {
      arr[i] = arr[i+1];
    }
      rear --;
      return front;
  }
  Public static int peek() {
      if (isEmpty()) {
        System.out.println("empty queue");
        return -1;
      }
      return arr[0];
    }
  }

  Public static void main (String args[]) {
    Queue q = new Queue(5);
    q.add(1);
    q.add(2);
    q.add(3);

    System.out.println(q.remove());
    System.out.println(q.peek());
  }
}
```



** circular queue using array **

O(1) → add, remove, peek
so, better to use array for circular Queue.

```
Public class QueueB {
    Static class Queue {
        Static int arr[];
        Static int size;
        Static int front = -1;
        Static int rear = -1;

        Queue(int size) {
            this.size = size;
            arr = new int[size];
        }

        Public static boolean isEmpty() {
            return(rear == -1 && front == -1);
        }

        Public static boolean isFull() {
            return ((rear+1) % size == front);
        }

        Public static void add(int data) {
            if (isFull()) {
                System.out.println("overflow");
                return;
            }
            // if it's the 1st element, if not full
            if (front == -1) {
                front = 0;
            }
            rear = (rear+1) % size;
            arr[rear] = data;
        }
```

```
Public static int remove() {
    if (isEmpty()) {
        System.out.println("empty queue");
        return -1;
    }
    int res = arr[front];
    // if only 1 element is present
    if (front == rear) {
        front = rear = -1;
    } else {
        front = (front+1) % size;
    }
    return res;
}
Public static int peek() {
    if (isEmpty()) {
        System.out.println("empty queue");
        return -1;
    }
    return arr[front];
}
Public static void main(String args[]) {
    Queue q = new Queue(3);
    q.add(1);
    q.add(2);
    q.add(3);
    System.out.println(q.remove());
    q.add(4);
    System.out.println(q.remove());
    while(!q.isEmpty()) {
        System.out.println(q.remove());
    }
}
```

For linked list also, for add, remove, peek → O(1).

Queue using Linked List

```
Public class QueueB {
    static class Node {
        int data;
        Node next;
        Node (int data) {
            this.data = data;
            next = null;
        }
    }
    static class Queue {
        static Node head = null;
        static Node tail = null;

        Public static ~~void~~ boolean isEmpty () {
            return (head == null && tail == null);
        }

        Public static void add (int data) {
            Node newNode = new Node (data);
            if (isEmpty ()) {
                tail = head = newNode;
            } else {
                tail.next = newNode;
                tail = newNode;
            }
        }

        Public static int remove () {
            if (isEmpty ()) {
                System.out.println ("empty queue");
                return -1;
```

```
int front = head.data;
//single node
if (head == tail) {
    tail = null;
}
head = head.next;
return front;
}

Public static int peek() {
if (isEmpty()) {
    System.out.println("empty queue");
    return -1;
}
return head.data;
}
}

Public static void main (String args[]) {
Queue q = new Queue;
q.add(1);
q.add(2);
q.add(3);
q.add(4);
q.add(5);
while (!q.isEmpty()) {
    System.out.println(q.peek());
    q.remove();
}
}
}
```

** Java collection Framework - Queue **

```
import java.util.*;

Public class QueueB {
    Public Static void main (String args[]){
    // Queue implementation using JCF
    Queue<Integer> q = new ArrayDeque();

    q.add(1);
    q.add(2);
    q.add(3);
    q.add(4);
    q.add(5);

    while (!q.isEmpty()) {
        System.out.println(q.peek());
        q.remove();
    }
    }
}
```

* queue is an interface. The class ArrayDeque, have implements the interface.

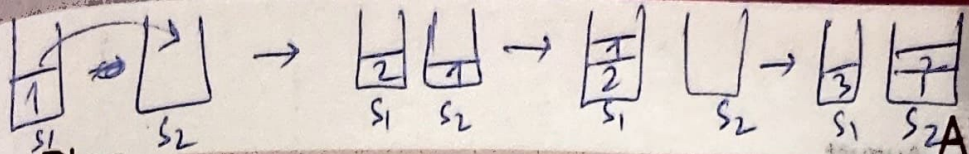
** Queue using 2 Stacks **

Push → O(n)
Pop → O(n)

```
import java.util.*;

Public class QueueB {
    Static class Queue {

Static Stack<Integer> s1 = new Stack<>();
Static Stack<Integer> s2 = new Stack<>();
```



222 6.

Action Plan

AUG '22

AUG

01 Mon

02 Tue

03 Wed

04 Thu

```
Public static boolean isEmpty () {
    return s1.isEmpty();
}
```

05 Fri

06 Sat

07 Sun

08 Mon

```
Public static void add (int data) {
    while (! s1.isEmpty()) {
```

09 Tue

10 Wed

11 Thu

12 Fri

```
        s2.push (s1.pop()); // Pop from s1 & Push to s2
    }
    s1.push (data);
    while (! s2.isEmpty())
    {
        s1.push (s2.pop()); // Pop from s2 & Push to s1
    }
}
```

13 Sat

14 Sun

15 Mon

16 Tue

17 Wed

18 Thu

19 Fri

20 Sat

```
Public static int remove () {
    return s1.pop();
}
```

21 Sun

22 Mon

23 Tue

24 Wed

```
Public static int peek () {
    return s1.peek();
}
```

25 Thu

26 Fri

27 Sat

28 Sun

```
Public static void main (String args[]) {
    Queue q = new Queue();
    q.add(1); q.add(2); q.add(3);
```

29 Mon

30 Tue

31 Wed

```
    while (! q.isEmpty()) {
        System.out.println (q.peek());
        q.remove();
    }
}
```