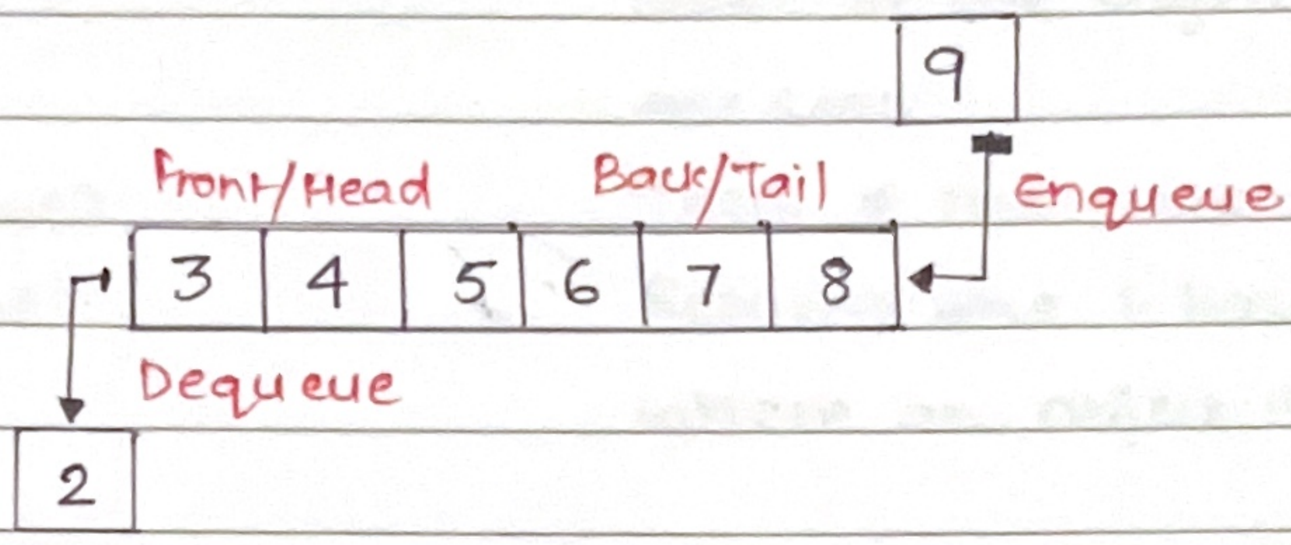
QUEUE

A Queue in Java, part of the java.util package, extends the collection interface and processes elements in First-In-First-Out (FIFO) order. Elements are added at the end and removed from start ensuring sequential processing.



METHOD	DESCRIPTION
add(E e)	Inserts the specified element into the queue. Throws an exception if it fails
offer(E e)	Inserts the specified element into queue. Returns false if it fails.
remove()	Retrives and remove the head of queue. Throws an exception if the queue is empty.
poll()	Retrives and removes the head of the queue, or returns null if the queue is empty.
element()	Retrives but does not remove, the head of queue. Throws an exception if queue is empty
peek()	Retrives but does not remove the head of queue or returns null if the queue is empty.

- Priority Queue

An unbounded priority queue based on priority heap. Elements are ordered according to their natural ordering by a comparator provided at queue construction.

Example:

```
class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new PriorityQueue<>();
        queue.add(15);
        queue.add(10);
        queue.add(20);
        System.out.println(queue.poll()); // 10
    }
}
```

- Deque (Double-ended Queue)

Allows insertion and removal at both ends

Implemented by classes such as "ArrayDequeue" and "LinkedList"

Example

```
class Main {
    public static void main(String[] args) {
        Dequeue
        Queue<Integer> deque = new ArrayDeque<>();
        deque.add(1);
        deque.add(2);
        System.out.println(deque.removeFirst()); // 1
    }
}
```

- ArrayDeque

A resizable-array implementation of the deque interface

more efficent than linkedlist for stack and queue operation

- Circular Queue

A fixed queue where the last position is connected back to first position to form a circle.

Manages a fixed size buffer and uses modulo for indexing.

Implementation

```
public class CircularQueue {
    private int[] data;
    private int head;
    private int tail;
    private int size;
    private int capacity;

    public CircularQueue(int k) {
        capacity = k
        data = new int[k];
        head: 0;
        tail = -1;
        size = 0;
    }

    public boolean enQueue(int value) {
        if (size == capacity) return false;
        tail = (tail + 1) % capacity;
        data[tail] = value;
        return true;
    }

    public boolean deQueue() {
        if (size == 0) return false;
        head = (head + 1) % capacity;
        size--;
        return true;
    }
```

```
public int front(){
    if (size==0) return -1;
    return (data[head]);
}
public int Rear(){
    if (size==0) return -1;
    return data[tail];
}
public boolean isEmpty(){
    return size==0;
}
public boolean isFull(){
    return size==capacity;
}
}
```