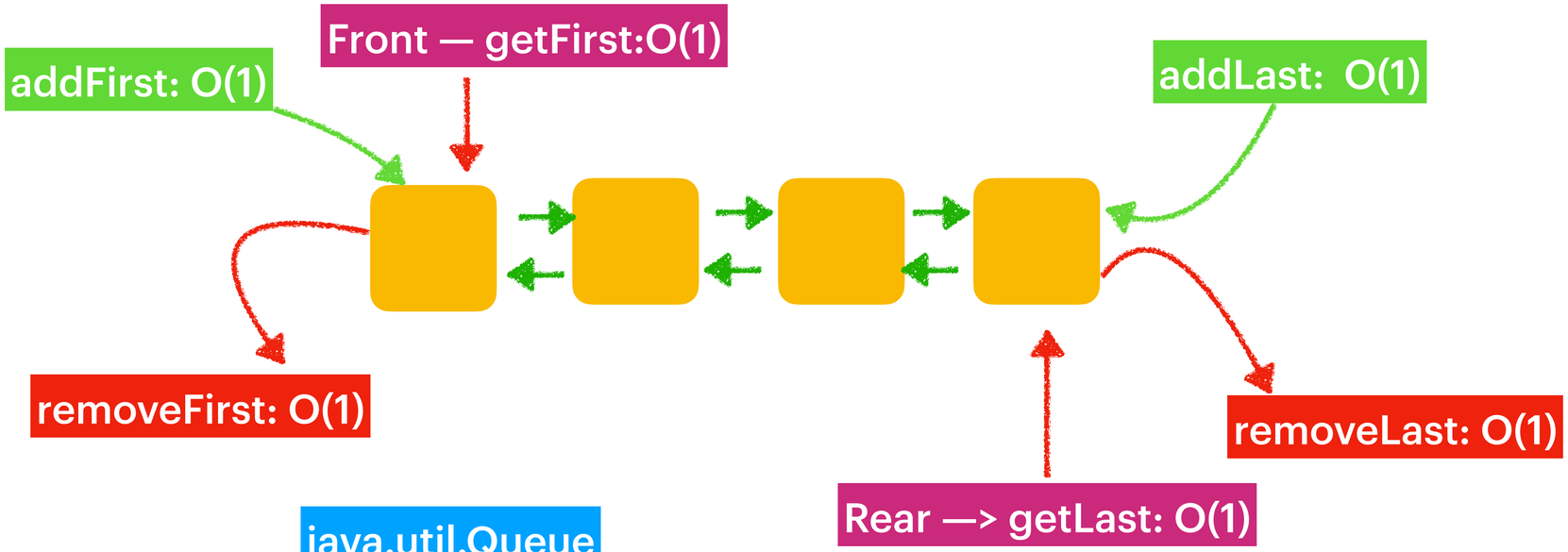
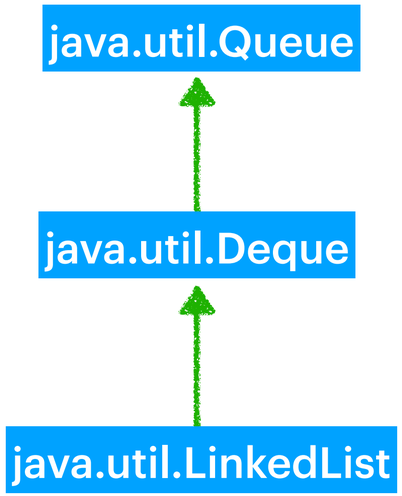
Design Deque [Double Ended Queue]






```
Queue<> queue = new LinkedList() ; [ LinkedList act as queue]

Deque<> deque = new LinkedList(); [ LinkedList act as Deque]
```

java.util.Deque [Methods]

```
public abstract void addFirst(E);
public abstract boolean offerFirst(E);

public abstract void addLast(E);
public abstract boolean offerLast(E);

public abstract E removeFirst();
public abstract E pollFirst();


public abstract E pollLast();
public abstract E removeLast();


public abstract E getFirst();
public abstract E peekFirst();

public abstract E getLast();
public abstract E peekLast();
```

225. Implement Stack using Queues

Easy 3024 882 Add to List Share

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push` , `top` , `pop` , and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a queue, which means that only `push to back` , `peek/pop from front` , `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:

```
Input
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2
myStack.pop(); // return 2
myStack.empty(); // return False
```

Constraints:

- `1 <= x <= 9`
- At most `100` calls will be made to `push` , `pop` , `top` , and `empty` .
- All the calls to `pop` and `top` are valid.

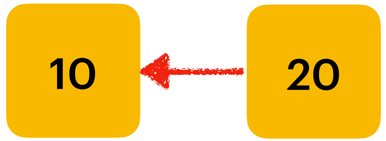
Follow-up: Can you implement the stack using only one queue?

Implement Stack using Single Queue

push(10)

Queue

push(20)



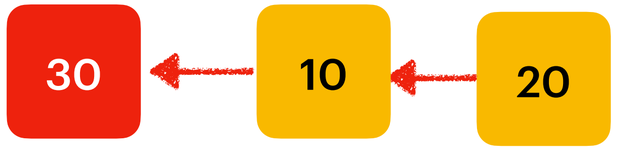
Queue

push(30)

Queue

pop()

Take the currentQueue size: 3
Till size-1 = 3-1 = 2 elements removeFront the add to the Queue. Then remove the size element.



top()

Same as pop() operation but we don't remove the Last element, we just return it.

push(element) : O(1)
pop() : O(n)
top(): O(n)

232. Implement Queue using Stacks

Easy 3678 242 Add to List Share

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (`push` , `peek` , `pop` , and `empty`).

Implement the `MyQueue` class:

- `void push(int x)` Pushes element x to the back of the queue.
- `int pop()` Removes the element from the front of the queue and returns it.
- `int peek()` Returns the element at the front of the queue.
- `boolean empty()` Returns `true` if the queue is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a stack, which means only `push to top` , `peek/pop from top` , `size` , and `is empty` operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Example 1:

```
Input
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 1, 1, false]

Explanation
MyQueue myQueue = new MyQueue();
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2] (leftmost is front of the
queue)
myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is [2]
myQueue.empty(); // return false
```

Constraints:

- `1 <= x <= 9`
- At most `100` calls will be made to `push` , `pop` , `peek` , and `empty` .
- All the calls to `pop` and `peek` are valid.

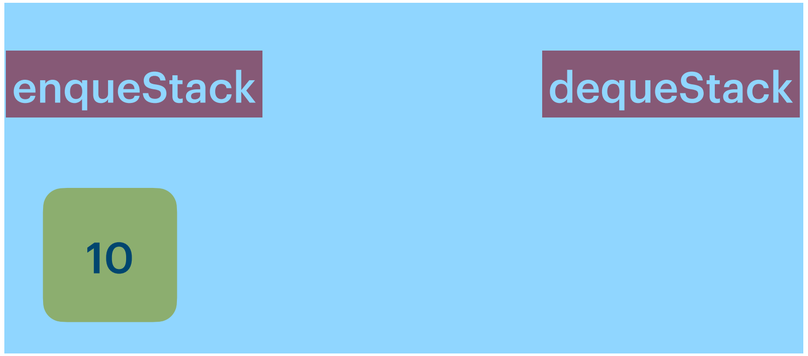
Follow-up: Can you implement the queue such that each operation is **amortized** `O(1)` time complexity? In other words, performing `n` operations will take overall `O(n)` time even if one of those operations may take longer.

enqueStack

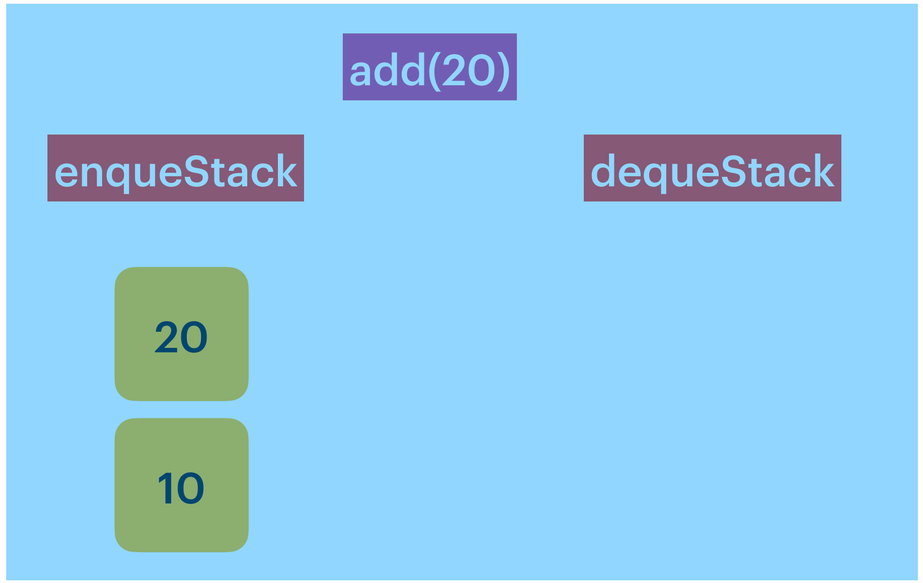
dequeStack

Design Queue using Stack :

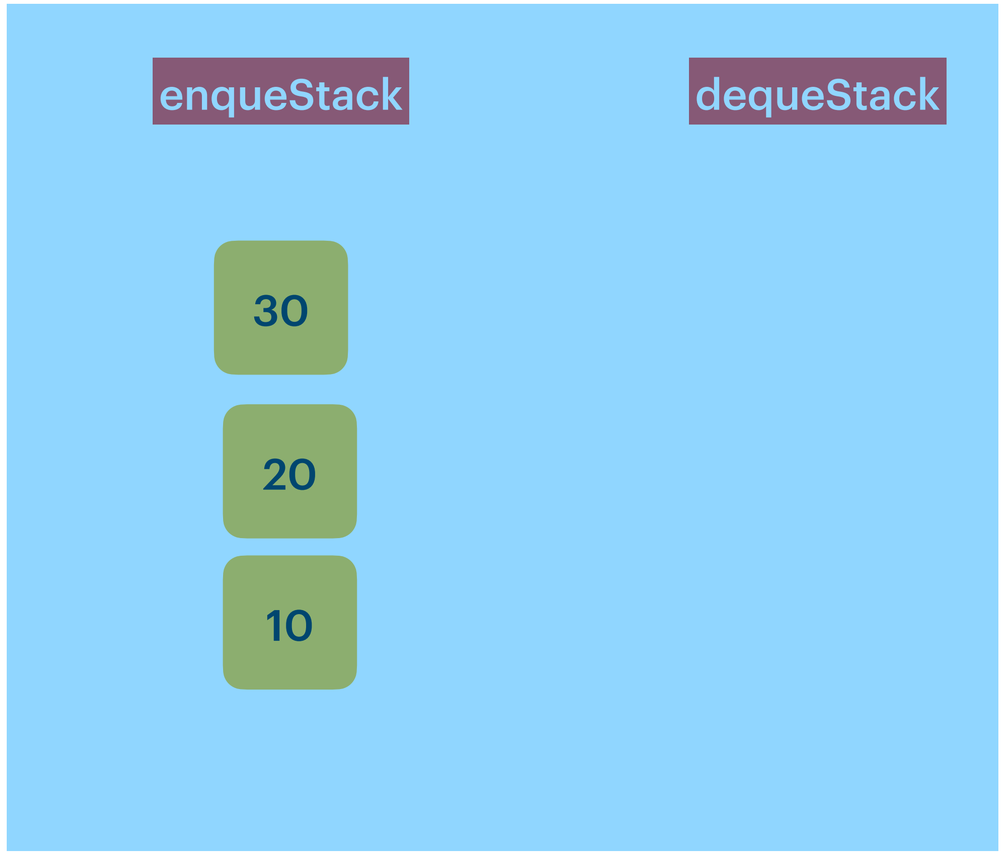
add(10)



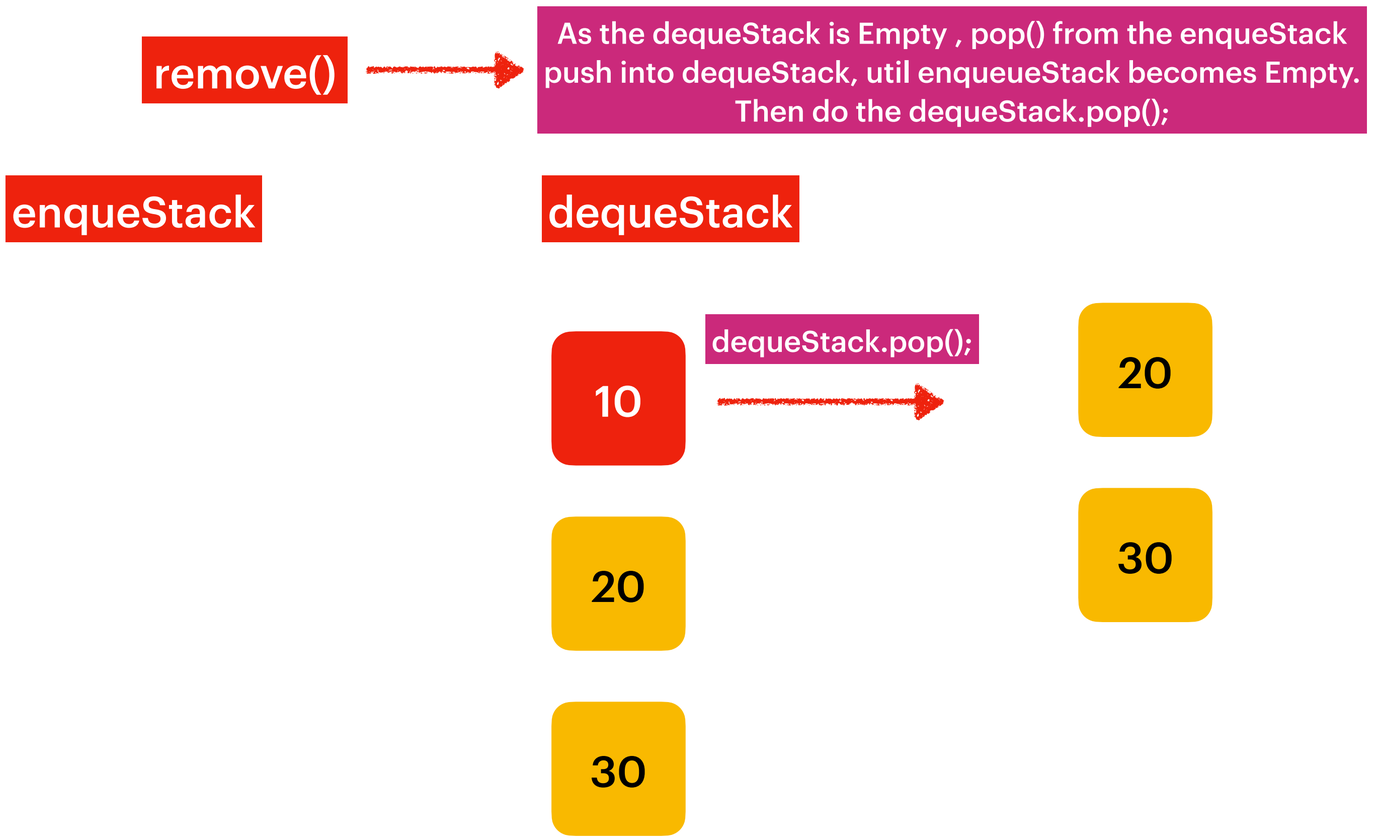
add(20)



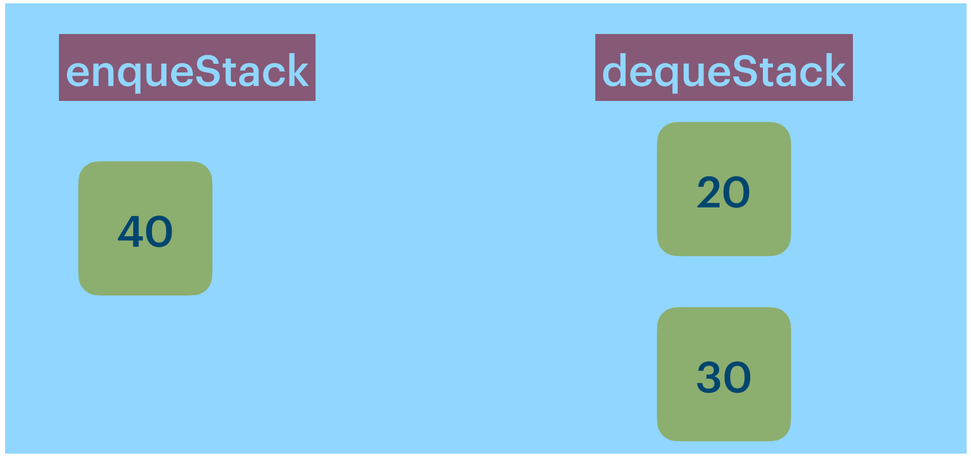
add(30)



add() : O(1)

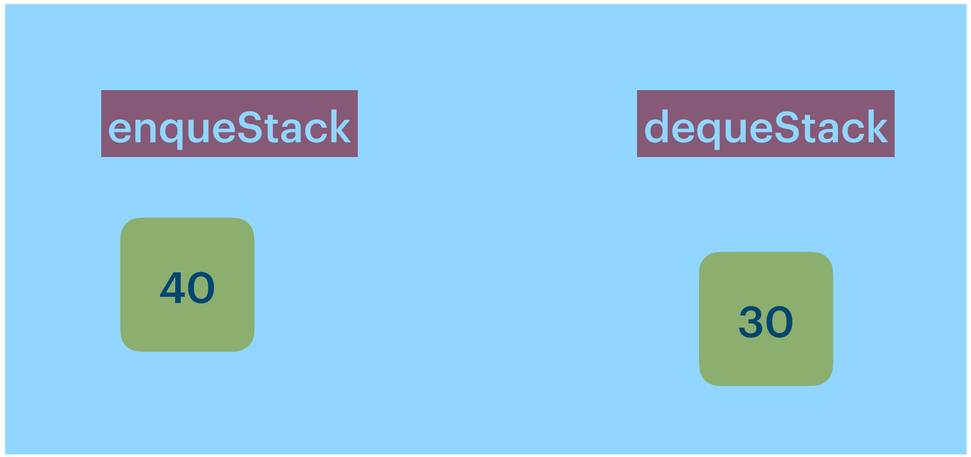

remove()
As the dequeStack is Empty , pop() from the enqueStack push into dequeStack, util enqueueStack becomes Empty. Then do the dequeStack.pop();
enqueStack
dequeStack
10
20
30
dequeStack.pop();
20
30

add(40)



remove() → As the dequeStack is not Empty remove the top()

On And Average remove() : O(1)



On And Average peek()
[Front]: O(1) dequeStack.top();