Dry Run: Queue (Array Implementation) — Step-by-step Memory Diagram

This dry run follows the provided TestQueue class operations. Each step shows the array state, index labels, and values of 'front' and 'rear'. Explanations are written in a human style as if a tutor walked you through the run.

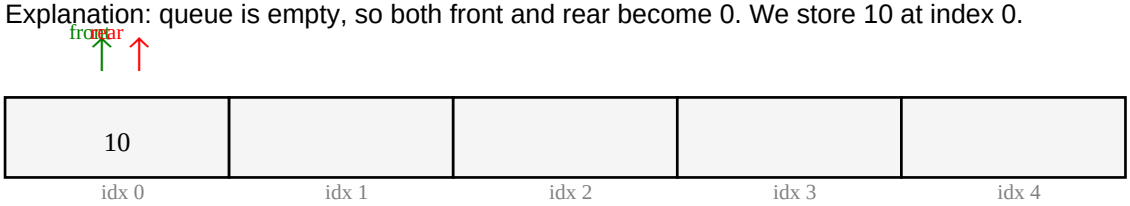
Test setup: Queue q = new Queue(5) (array size = 5)

Initial state (empty queue):

front = -1, rear = -1 — queue is empty

Step 1: q.insert(10)

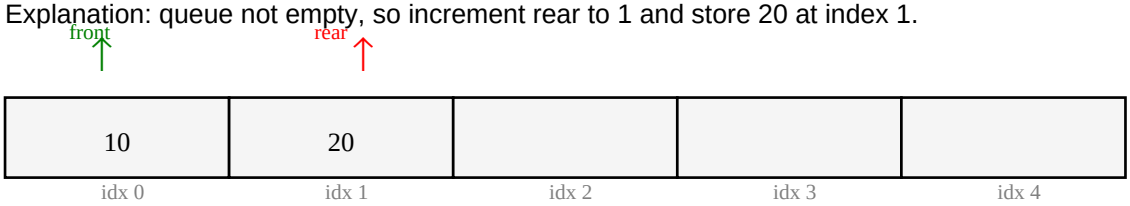
Explanation: queue is empty, so both front and rear become 0. We store 10 at index 0.



After insert: front=0, rear=0

Step 2: q.insert(20)

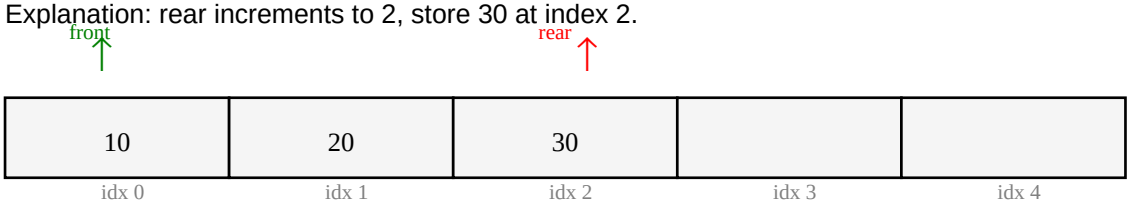
Explanation: queue not empty, so increment rear to 1 and store 20 at index 1.



After insert: front=0, rear=1

Step 3: q.insert(30)

Explanation: rear increments to 2, store 30 at index 2.



After insert: front=0, rear=2

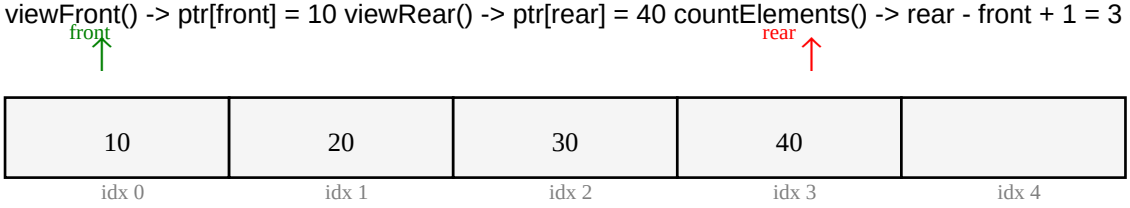
Step 4: q.insert(40)

Explanation: rear increments to 3, store 40 at index 3.

After insert: front=0, rear=3

Step 5: viewFront(), viewRear(), countElements()

Explanation: These are read-only operations — they don't change memory. Current values: viewFront() -> ptr[front] = 10 viewRear() -> ptr[rear] = 40 countElements() -> rear - front + 1 = 3 - 0 + 1 = 4



No change — front=0, rear=3, count=4

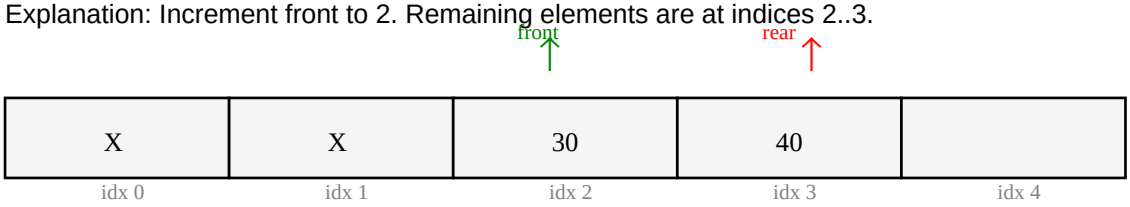
Step 6: q.deleteFront() // removes 10 (FIFO)

Explanation: Remove element at front index 0. We increment front to 1. If front > rear after increment, queue is empty — not the case now.

After delete: front=1, rear=3 — element at idx 0 logically removed

Step 7: q.deleteFront() // removes 20

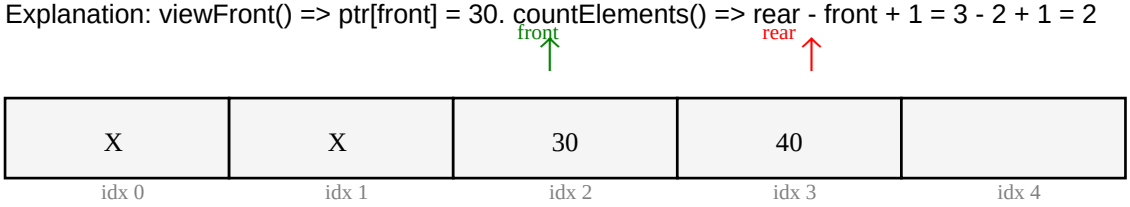
Explanation: Increment front to 2. Remaining elements are at indices 2..3.



After delete: front=2, rear=3 — elements at idx 0,1 removed

Step 8: viewFront() and countElements()

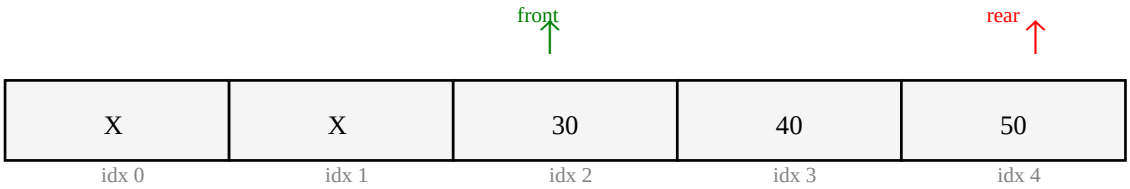
Explanation: viewFront() => ptr[front] = 30. countElements() => rear - front + 1 = 3 - 2 + 1 = 2



viewFront()=30, count=2

Step 9: q.insert(50)

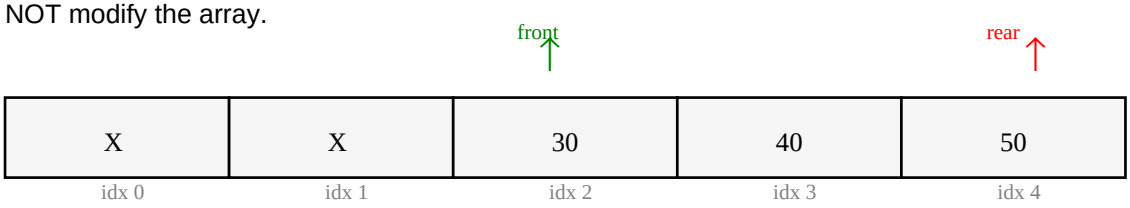
Explanation: queue not full, increment rear to 4 and store 50 at index 4.



After insert: front=2, rear=4

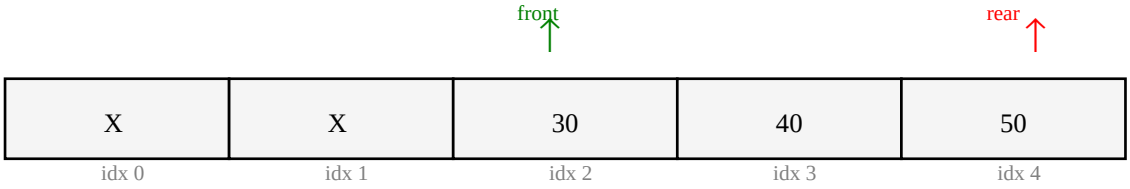
Step 10: q.insert(60) // attempt to insert beyond capacity

Explanation: Now rear is at index 4 (ptr.length - 1). isFull() returns true. So we throw QUEUE_OVERFLOW and do NOT modify the array.



Attempted insert -> isFull() true -> throw QUEUE_OVERFLOW

Step 11: Final state for q (unchanged after overflow attempt)



front=2, rear=4, count = rear-front+1 = 3

Second test: Underflow check (q2 = new Queue(3))

We call q2.deleteFront() immediately on an empty queue.

q2 initial: front=-1, rear=-1 (empty)

Since isEmpty() is true, deleteFront() throws QUEUE_UNDERFLOW. No memory change occurs.

Attempted delete -> QUEUE_UNDERFLOW thrown

Notes & Takeaways

1. front points to the index of the next element to be removed (dequeue).

2. rear points to the index of the last inserted element (enqueue).

3. Enqueue rules:
- If queue empty, set front=rear=0.
- Else increment rear and store element at ptr[rear].

4. Dequeue rules:
- Increment front.
- If front > rear after increment, reset front=rear=-1 to mark empty queue.

5. Count formula: rear - front + 1.

6. This is a simple linear array queue. After several dequeues, lower indices may remain unused. For efficient reuse of array space, implement a circular queue.

7. Exceptions used: QUEUE_OVERFLOW (when rear == ptr.length-1 and insert called), QUEUE_UNDERFLOW (when delete/view called on empty queue).