CSC508 Data Structures

Topic 6 : Queue

Compiled & edited by: Zahid Zainal

Recap

- Stack Definition
- Stack Operations
- Stack Application
- Stack Implementation

UNIVERSITI TEKNOLOGI MARA

Topic Structure

- Queue Definition
- Queue Operations
- Queue Implementation
- Circular Queue

Learning Outcomes

- At the end of this lesson, students should be able to:
  - Describe queue data structure
  - Explain queue implementation
  - Implement queue operation

Queue Definition

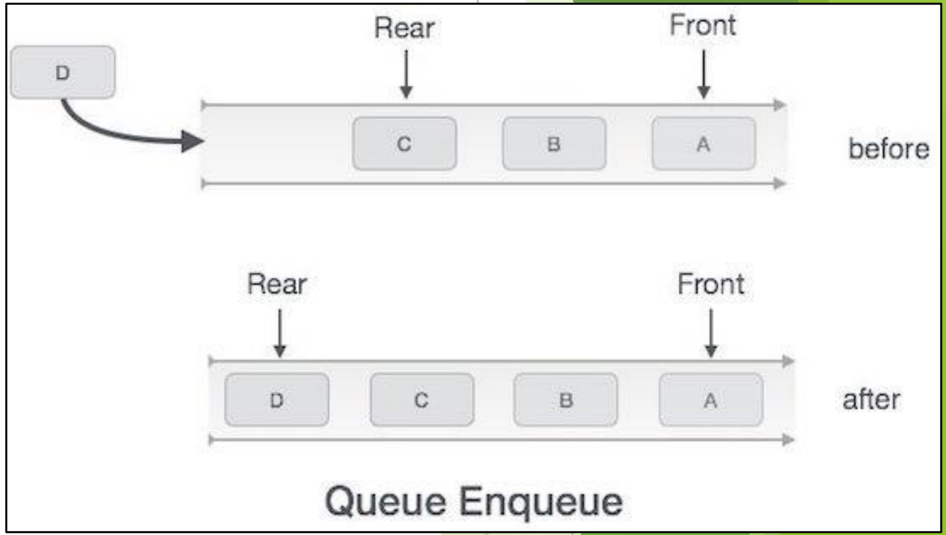
- A queue is a list of homogeneous elements which get
  - Added at one end (the back or rear)
  - Deleted from the other end (the front)
- It is based on first in first out (FIFO) algorithm
  - Middle elements are inaccessible
- Example :
  - Queue management system (QMS) at bank
  - Task scheduling in CPU

First In First Out (FIFO)

- Some real life queues are not exactly FIFO. Removal from a queue may be done by priority rather than by arrival time order (called priority queues)
  - Boarding into a plane: first class, frequent flyers, economic
  - Treatment in a hospital: emergency cases, normal patients
  - Threads with higher priority for critical OS tasks

Queue Operations

- `addQueue()` – Add a new element into the queue. Also called enqueue.
- `deleteQueue ()` – Retrieve and remove an element from the queue. Also called dequeue.
- `isFull()` – Check whether the queue (array) is full.
- `front()` – Check front element
- `rear()` – Check the rear element



https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm

Queue Implementation

- Using Arrays
  - Use a 1D array (static or dynamic), size is fixed
  - Elements are added using a front index
  - Elements are removed using a rear index
- Using Linked Lists
  - Elements are stored inside linked nodes
  - Elements are added at the end of the list (using a tail pointer)
  - Elements are removed from the beginning (using head pointer)

Array-based Queue

queueFront - Keep track of the first element in the queue
queueRear - Keep track on the last element in the queue

```
class MyQueue{
    char maxQueueSize = 100;
    char []newQueue = new int[maxQueueSize];
    int queueFront, queueRear;

    public void MyQueue() {
        queueFront = 0;
        queueRear = -1;
    }
}
```

Initialize queueFront to 0 and queueRear to -1

Array-based Queue (cont.)

```
public boolean isEmpty() {
    return (queueRear - queueFront == -1);
}

public boolean isFull() {
    return (queueRear == maxQueueSize - 1);
}

public char front() {
    return newQueue[queueFront];
}

public char rear() {
    return newQueue[queueRear];
}
```

Array-based Queue (cont.)

- Enqueue

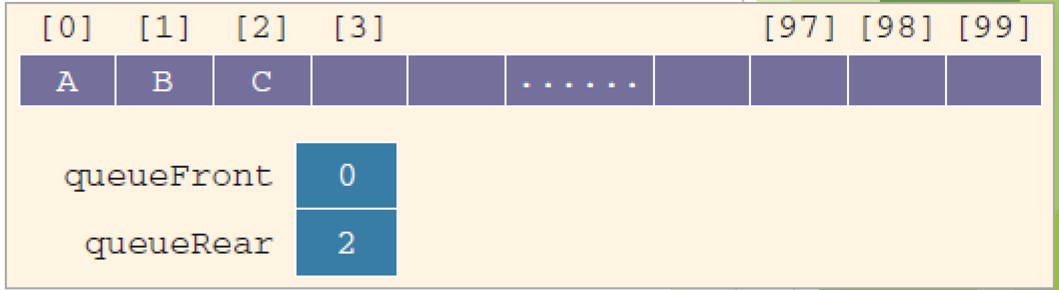
Increment queueRear by 1

```
public void addQueue(char elem) {
    if (!isFull()) {
        newQueue[++queueRear] = elem;
    }else
        System.out.println("Full array");
}
```

Store the element in array at index queueRear

May throw QueueOverflowException

```
q.addQueue('A');
q.addQueue('B');
q.addQueue('C');
```



Array-based Queue (cont.)

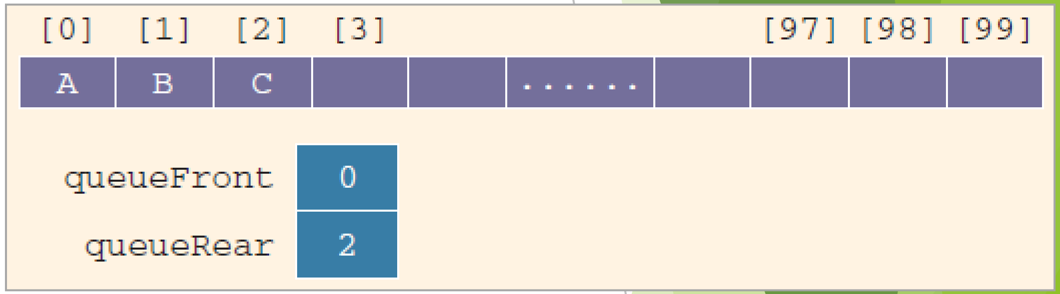
- Dequeue

Return element at array index `queueFront`

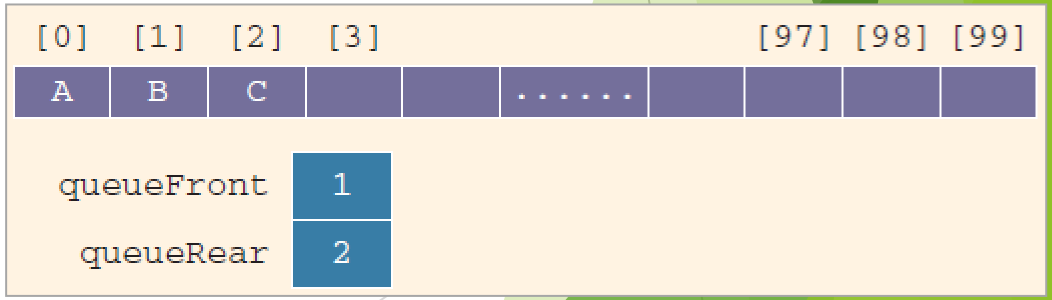
Increment `queueFront by 1`

```java
public char deleteQueue() {
    if (!isEmpty()) {
        return newQueue[queueFront++];
    }else {
        System.out.println("Empty Queue");
        return null;
    }
}
```

May throw `QueueUnderflowException`




```java
q.deleteQueue();
```



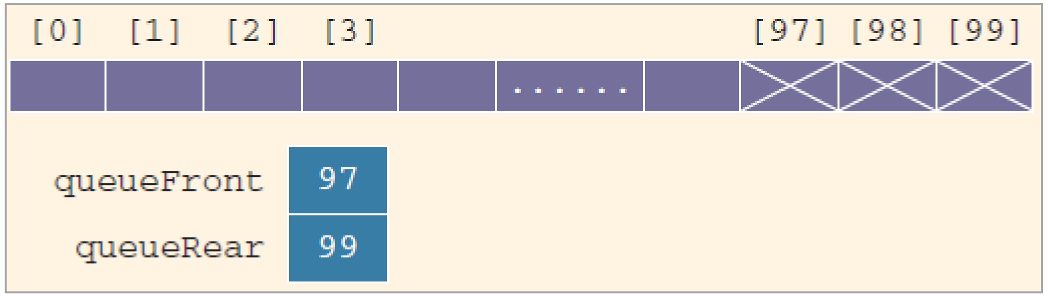
Testing

```
public static void main(String[] args) {
    MyQueue Q1 = new MyQueue();
    Q1.addQueue('F');
    Q1.addQueue('A');
    Q1.addQueue('B');
    Q1.addQueue('E');
    Q1.addQueue('Q');
    Q1.printQueue();
    System.out.println("Remove " + Q1.deleteQueue());
    System.out.println("Remove " + Q1.deleteQueue());
    Q1.printQueue();
    System.out.println("Front - " + Q1.front());
    System.out.println("Rear - " + Q1.rear());
}
```

```
Queue created
F
A
B
E
Q
Remove F
Remove A
B
E
Q
Front - B
Rear - Q
```

Array Implementation: Full Tank!?

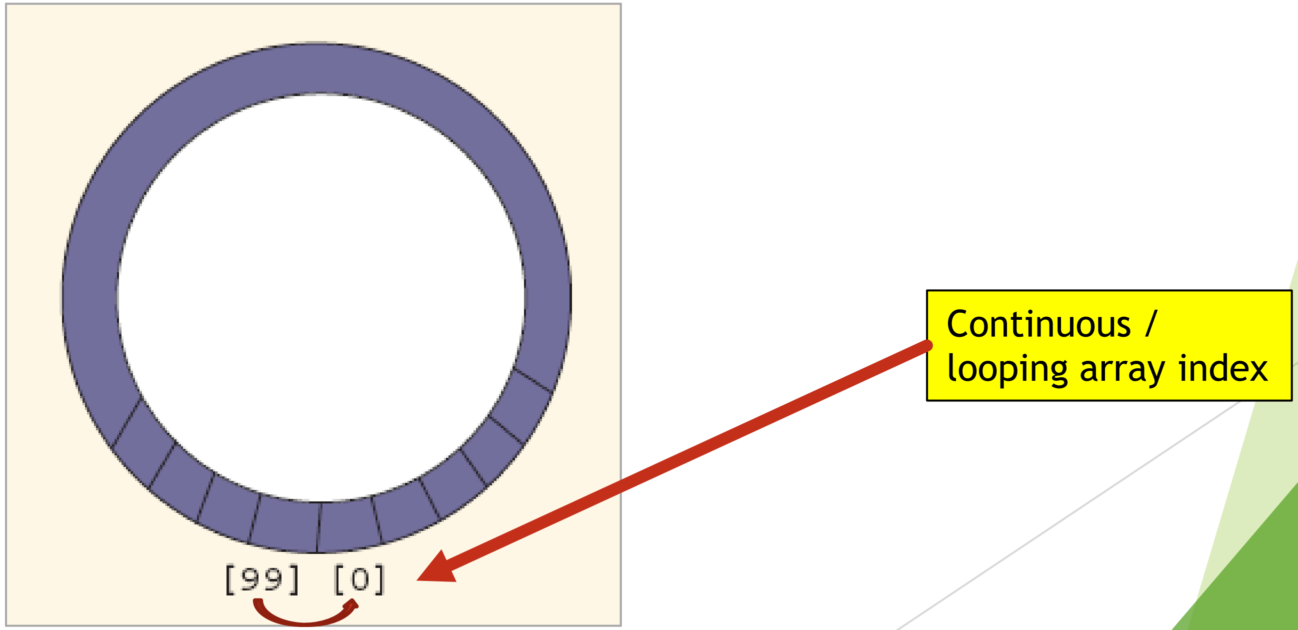
- Will this queue design work? Unlike array which can reuse the memory space after every pop operation, queue cannot
- When queue rear index reaches end of the array, does this mean the queue is full?



- Solution 1: When queue overflows to the rear and front indicates available slots, shift all elements to the beginning of the array
  - Problem: too slow for large queues

Array Implementation: Full Tank!? (cont.)

- Solution 2: Assume that the array is circular array
  - Conceptually, i.e. change indices in a circular way
  - Reusing the unused array index.

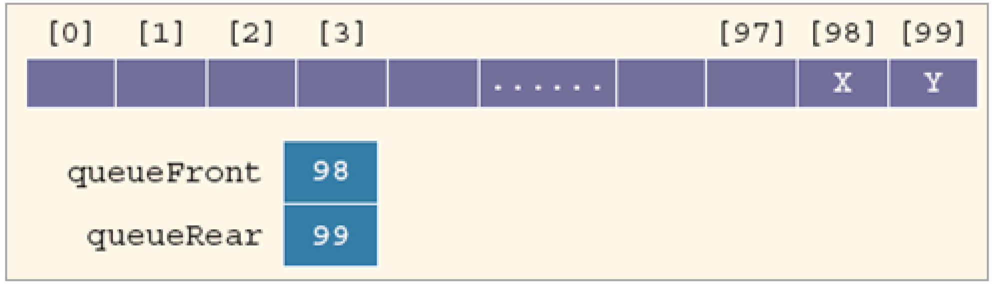


Circular queue

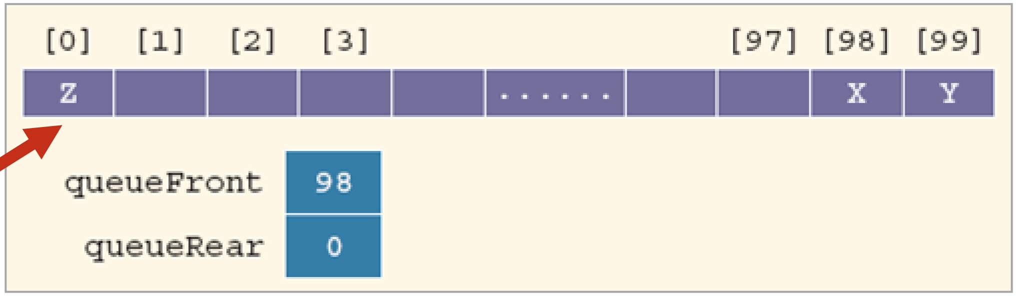
```
queueRear = (queueRear + 1) % maxQueueSize;
```

```
queueFront = (queueFront + 1) % maxQueueSize;
```

Appropriately update the `queueRear` and `queueFront`



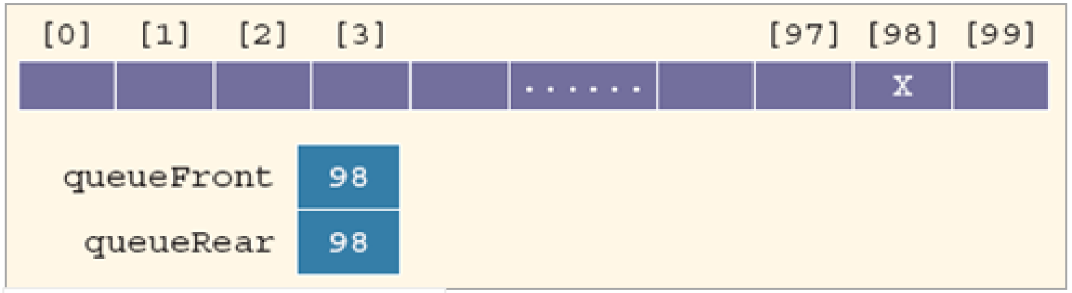

```
q.addQueue('Z');
```



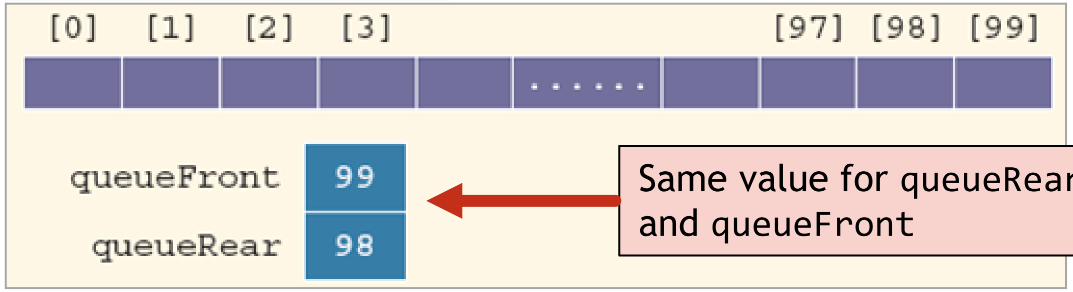
New queue element is added at the first index

Circular Queue Problem

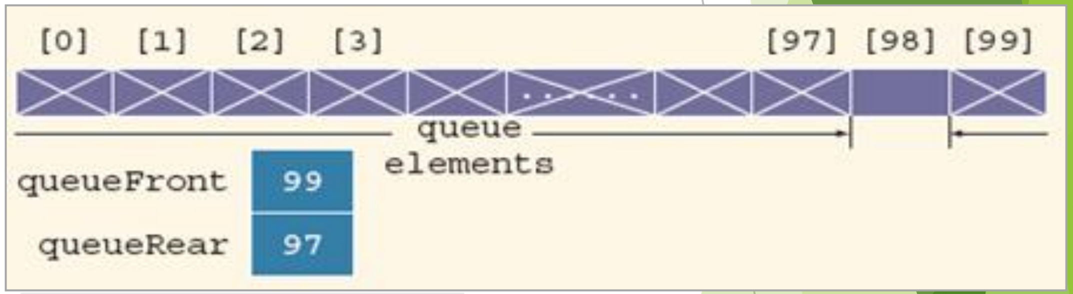
- Two different cases with identical values for `queueFront` and `queueRear`



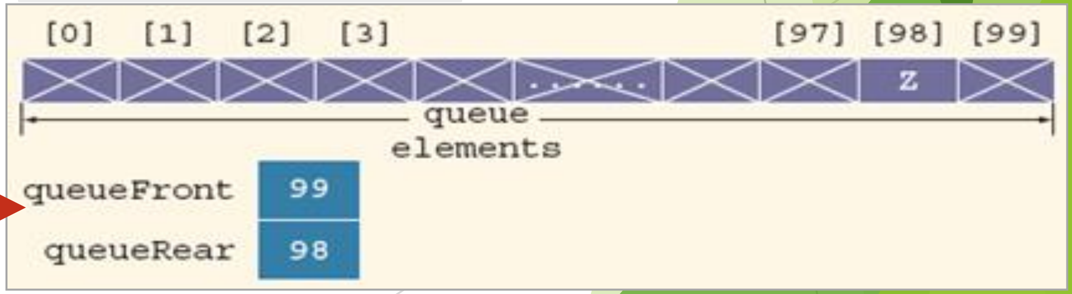
`q.deleteQueue();`



Same value for `queueRear` and `queueFront`



`q.addQueue('Z');`



An empty queue

A full queue

Empty or Full

- Previous **`isEmpty()`** method is not valid.
- Solution 1: Define a boolean variable `lastOpAddQ`
  - In `addQueue` set this variable to `true`
  - In `deleteQueue` set it to `false`
  - Empty iff `(queueRear == queueFront-1) && ! lastOpAddQ`
  - Full iff `(queueRear == queueFront-1) && lastOpAddQ`

Empty or Full (cont)

- Solution 2: use `count` variable
  - Initialized to zero
  - Incremented when a new element is added to the queue
  - Decremented when an element is removed
  - Compare it to zero or `maxQueueSize` to determine queue bring empty or full
  - Very useful if there is frequent need to know the number of elements in the queue

Linked Implementation of Queue

- Array implementation issues
  - Array size is fixed: only a finite number of queue elements can be stored in it
  - The array implementation of the queue requires array to be treated in a special way (circular)

- The linked implementation of a queue simplifies many of the special cases of array implementation
  - In addition, the queue never gets full (in theory)

Linked Implementation of Queue (cont)

- Elements are added at one end and removed from the other
  - We need to know the front of the queue and the rear of the queue
- Elements are added at the end of the list using a `queueRear` pointer (i.e. tail of the list)
  - Similar to `insertLast()` of linked list
- Elements are removed from the beginning using `queueFront` pointer (i.e. head of the list)
  - Similar to `removeFirst()` of linked list

Priority Queue

- FIFO rules of a queue are relaxed.
  - In hospital, patients with severe symptoms are treated first.
- Customers or jobs with higher priority are pushed to front of queue
- To implement:
  - use an ordinary linked list, which keeps the items in order from the highest to lowest priority
  - use a treelike structure

Summary

- A queue is a list of homogeneous elements which elements are added at one end and removed from the other end
- Based on first in first out algorithms (FIFO)
- Queue Implementation : array, linked list
- Array implementation of queue can be improved using circular array
- Priority queue has relaxed FIFO operation

Next Topic…

- Recursion

UNIVERSITI TEKNOLOGI MARA

References

- Carrano, F. & Savitch, W. 2005. *Data Structures and Abstractions with Java, 2nd ed. Prentice-Hall.*
- Malik D.S, & Nair P.S., Data Structures Using Java, Thomson Course Technology, 2003.
- Rada Mihalcea, CSCE 3110 Data Structures and Algorithm Analysis notes, U of North Texas.