## Topics

1. Create Queue Interface
2. Create Queue Using Array
3. Create Queue Using Linked Lists
4. Implement Basic Methods of Queue

* isEmpty()
* size()
* first()
* enqueue(E e)
* dequeue()

## Homework

1. Augment the ArrayQueue implementation with a new rotate( ) method having semantics identical to the combination, enqueue(dequeue( )). But, your implementation should be more efficient than making two separate calls (for example, because there is no need to modify the size).

public void rotate() {  
    if (!isEmpty()) {  
        // Move the front element to the back without modifying size  
        E temp = data[front];  
        front = (front + 1) % [data.length](http://data.length); // Move front pointer  
        data[(front + size - 1) % [data.length](http://data.length)] = temp; // Place at the new rear  
    }  
}

1. Implement the clone( ) method for the ArrayQueue class.

public ArrayQueue<E> clone() {  
    ArrayQueue<E> clonedQueue = new ArrayQueue<>([data.length](http://data.length));  
    [clonedQueue.size](http://clonedQueue.size) = [this.size](http://this.size);  
    [clonedQueue.front](http://clonedQueue.front) = [this.front](http://this.front);  
    for (int i = 0; i < size; i++) {  
        [clonedQueue.data](http://clonedQueue.data)[([clonedQueue.front](http://clonedQueue.front) + i) % [data.length](http://data.length)] = [this.data](http://this.data)[(front + i) % [data.length](http://data.length)];  
    }  
    return clonedQueue;  
}

1. Implement a method with signature concatenate(LinkedQueue Q2) for the LinkedQueue class that takes all elements of Q2 and appends them to the end of the original queue. The operation should run in O(1) time and should result in Q2 being an empty queue.

public void concatenate(LinkedQueue<E> Q2) {  
    if ([Q2.isEmpty](http://Q2.isEmpty)()) return;  
  
    if ([this.isEmpty](http://this.isEmpty)()) {  
        [this.head](http://this.head) = [Q2.head](http://Q2.head);  
        [this.tail](http://this.tail) = [Q2.tail](http://Q2.tail);  
    } else {  
        [this.tail.setNext](http://this.tail.setNext)([Q2.head](http://Q2.head));  
        [this.tail](http://this.tail) = [Q2.tail](http://Q2.tail);  
    }  
  
    [this.size](http://this.size) += [Q2.size](http://Q2.size);  
    [Q2.head](http://Q2.head) = [Q2.tail](http://Q2.tail) = null; // Empty Q2  
    [Q2.size](http://Q2.size) = 0;  
}

1. Use a queue to solve the Josephus Problem.

public static int josephus(int n, int k) {  
    Queue<Integer> queue = new LinkedList<>();  
     
    for (int i = 1; i <= n; i++) {  
        [queue.add](http://queue.add)(i);  
    }  
  
    while ([queue.size](http://queue.size)() > 1) {  
        for (int i = 1; i < k; i++) {  
            [queue.add](http://queue.add)([queue.remove](http://queue.remove)()); // Rotate the queue  
        }  
        [queue.remove](http://queue.remove)(); // Eliminate the k-th person  
    }  
  
    return [queue.remove](http://queue.remove)(); // Last remaining person  
}  
  
public static void main(String[] args) {  
    [System.out.println](http://System.out.println)("Winner: " + josephus(7, 3)); // Example: 7 people, eliminate every 3rd  
}

1. Use a queue to simulate Round Robin Scheduling.

public static void roundRobinScheduling(String[] processes, int[] burstTimes, int timeQuantum) {  
    Queue<Process> queue = new LinkedList<>();  
    for (int i = 0; i < [processes.length](http://processes.length); i++) {  
        [queue.add](http://queue.add)(new Process(processes[i], burstTimes[i]));  
    }  
  
    while (![queue.isEmpty](http://queue.isEmpty)()) {  
        Process current = [queue.remove](http://queue.remove)();  
        if ([current.burstTime](http://current.burstTime) > timeQuantum) {  
            [System.out.println](http://System.out.println)("Executing: " + [current.name](http://current.name) + " for " + timeQuantum + " units");  
            [current.burstTime](http://current.burstTime) -= timeQuantum;  
            [queue.add](http://queue.add)(current); // Re-queue the process  
        } else {  
            [System.out.println](http://System.out.println)("Executing: " + [current.name](http://current.name) + " for " + [current.burstTime](http://current.burstTime) + " units");  
        }  
    }  
}  
  
static class Process {  
    String name;  
    int burstTime;  
  
    Process(String name, int burstTime) {  
        [this.name](http://this.name) = name;  
        [this.burstTime](http://this.burstTime) = burstTime;  
    }  
}  
  
public static void main(String[] args) {  
    String[] processes = {"P1", "P2", "P3"};  
    int[] burstTimes = {10, 5, 8};  
    int timeQuantum = 4;  
  
    roundRobinScheduling(processes, burstTimes, timeQuantum);  
}