~\OneDrive - VietNam National University - HCM INTERNATIONAL UNIVERSITY\Desktop\DSA\DSA LAB NEW\Lab 3 Stacks & Queues\ITITSB22029_DoMinhDuy_Lab3\QueueApp\QueueSimulation.java

```java
// Queue.java
// demonstrates queue
// to run this program: C>java QueueApp

// Write a method to display the queue array and the front and rear indices. Explain how wraparound
// works.
// Write a method to display the queue (loop from 1 to nItems and use a temporary front for
// wraparound).
// Display the aray, the queue, and the front and rear indices.
// Insert fewer items or remove fewer items and investigate what happens when the queue is empty or
// full.
// Extend the insert and remove methods to deal with a full and empty queue.
// Add processing time to the queue. Create a new remove method that removes item N after N calls to
// the method.
// Simulate a queue of customers each one served for a random amount of time. Investigate how
// simulation is affected by:
//  the size of the queue
//  the range of time for wich each customer is served
//  the rate at which customers arrive at the queue
////////////////////////////////////////////////////////////////
import java.util.Random;

class Queue {
   private int maxSize;
   private long[] queArray;
   private int front;
   private int rear;
   private int nItems;
   private int processCounter = 0;
   private int removeAfterN = 5; // Number of calls to remove before removing an item

   // Constructor
   public Queue(int s) {
      maxSize = s;
      queArray = new long[maxSize];
      front = 0;
      rear = -1;
      nItems = 0;
   }

   // Insert an item into the queue
   public void insert(long value) {
      if (isFull()) {
         System.out.println("Queue is full! Cannot insert.");
         return;
      }

      if (rear == maxSize - 1) {
         rear = -1; // wraparound
      }

      queArray[++rear] = value;
      nItems++;
   }

   // Remove an item from the queue
   public long remove() {
      if (isEmpty()) {
         System.out.println("Queue is empty! Cannot remove.");
         return -1;
      }

      long temp = queArray[front++];
      if (front == maxSize) {
         front = 0; // wraparound
      }
      nItems--;
      return temp;
   }

   // Remove after N calls (simulate processing time)
   public long processAndRemove() {
      processCounter++;
      if (processCounter == removeAfterN) {
         processCounter = 0;
         return remove();
      } else {
         System.out.println("Processing item without removing.");
         return -1;
      }
   }

   // Display the entire queue array and front/rear indices
   public void displayArray() {
      System.out.print("Array: ");
      for (int i = 0; i < maxSize; i++) {
         System.out.print(queArray[i] + " ");
      }
      System.out.println();
      System.out.println("Front index: " + front + ", Rear index: " + rear);
   }

   // Display the actual queue from front to rear (handle wraparound)
   public void displayQueue() {
      System.out.print("Queue: ");
      int tempFront = front;
      for (int i = 0; i < nItems; i++) {
         System.out.print(queArray[tempFront] + " ");
         tempFront++;
         if (tempFront == maxSize) {
            tempFront = 0; // wraparound
         }
      }
      System.out.println();
   }

   // Check if the queue is empty
   public boolean isEmpty() {
      return nItems == 0;
   }

   // Check if the queue is full
   public boolean isFull() {
      return nItems == maxSize;
   }

   // Get the current size of the queue
   public int size() {
      return nItems;
   }
}

public class QueueSimulation {

   public static void simulateQueue(int queueSize, int serviceTimeRange, int arrivalRate) {
      Queue queue = new Queue(queueSize);
      Random rand = new Random();

      // Simulating the arrival of customers and service time
      for (int i = 0; i < 15; i++) { // Simulate 15 time units
         // Randomly decide if a customer arrives
         if (rand.nextInt(100) < arrivalRate) { // Arrival rate percentage chance
            long customer = rand.nextInt(100); // Customer with a random ID (0-99)
            System.out.println("Customer " + customer + " arrived.");
            queue.insert(customer);
         }

         // Simulate processing a customer (service time)
         int serviceTime = rand.nextInt(serviceTimeRange) + 1; // Random service time between 1 and serviceTimeRange
         for (int j = 0; j < serviceTime; j++) {
            queue.processAndRemove();
         }

```

```java
            // Display the state of the queue at each time step
            queue.displayQueue();
        }
    }

    public static void main(String[] args) {
        // Initialize the queue simulation with:
        // Queue size: 5
        // Service time range: 3 time units
        // Arrival rate: 70% chance for a new customer per time unit
        simulateQueue(5, 3, 70);
    }
}

/////////////////////////////////////////////////////////////////

```