## 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()

// Queue Interface

public interface Queue<E> {

boolean isEmpty(); // للتحقق مما إذا كانت قائمة الانتظار فارغة

int size(); // للحصول على حجم قائمة الانتظار

E first(); // للحصول على العنصر الأول دون إزالته

void enqueue(E e); // لإضافة عنصر إلى قائمة الانتظار

E dequeue(); // لإزالة العنصر الأول من قائمة الانتظار

}

// Array Queue Implementation

public class ArrayQueue<E> implements Queue<E> {

private E[] array;

private int front;

private int rear;

private int size;

private static final int DEFAULT\_CAPACITY = 10;

@SuppressWarnings("unchecked")

public ArrayQueue() {

array = (E[]) new Object[DEFAULT\_CAPACITY];

front = 0;

rear = 0;

size = 0;

}

@Override

public boolean isEmpty() {

return size == 0;

}

@Override

public int size() {

return size;

}

@Override

public E first() {

if (isEmpty()) {

return null;

}

return array[front];

}

@Override

public void enqueue(E e) {

if (size == array.length) {

resize(2 \* array.length); // زيادة السعة

}

array[rear] = e;

rear = (rear + 1) % array.length;

size++;

}

@Override

public E dequeue() {

if (isEmpty()) {

return null;

}

E element = array[front];

array[front] = null; // لتفادي التسريبات

front = (front + 1) % array.length;

size--;

return element;

}

private void resize(int newCapacity) {

@SuppressWarnings("unchecked")

E[] newArray = (E[]) new Object[newCapacity];

for (int i = 0; i < size; i++) {

newArray[i] = array[(front + i) % array.length];

}

array = newArray;

front = 0;

rear = size;

}

}

// Linked List Queue Implementation

public class LinkedQueue<E> implements Queue<E> {

private static class Node<E> {

E data;

Node<E> next;

Node(E data) {

this.data = data;

}

}

private Node<E> front;

private Node<E> rear;

private int size;

public LinkedQueue() {

front = null;

rear = null;

size = 0;

}

@Override

public boolean isEmpty() {

return size == 0;

}

@Override

public int size() {

return size;

}

@Override

public E first() {

if (isEmpty()) {

return null;

}

return front.data;

}

@Override

public void enqueue(E e) {

Node<E> newNode = new Node<>(e);

if (isEmpty()) {

front = newNode;

} else {

rear.next = newNode;

}

rear = newNode;

size++;

}

@Override

public E dequeue() {

if (isEmpty()) {

return null;

}

E element = front.data;

front = front.next;

size--;

if (isEmpty()) {

rear = null; // إذا أصبحت فارغة

}

return element;

}

}

// Main Class to Test the Queue Implementations

public class QueueTest {

public static void main(String[] args) {

// اختبار ArrayQueue

System.out.println("Testing ArrayQueue:");

Queue<Integer> arrayQueue = new ArrayQueue<>();

arrayQueue.enqueue(1);

arrayQueue.enqueue(2);

arrayQueue.enqueue(3);

System.out.println("First element: " + arrayQueue.first());

System.out.println("Size: " + arrayQueue.size());

System.out.println("Dequeue: " + arrayQueue.dequeue());

System.out.println("Size after dequeue: " + arrayQueue.size());

// اختبار LinkedQueue

System.out.println("\nTesting LinkedQueue:");

Queue<Integer> linkedQueue = new LinkedQueue<>();

linkedQueue.enqueue(4);

linkedQueue.enqueue(5);

linkedQueue.enqueue(6);

System.out.println("First element: " + linkedQueue.first());

System.out.println("Size: " + linkedQueue.size());

System.out.println("Dequeue: " + linkedQueue.dequeue());

System.out.println("Size after dequeue: " + linkedQueue.size());

}

}

## 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()) {

E temp = dequeue(); // نزيل العنصر الأول

enqueue(temp); // نضيفه إلى النهاية

}

}

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

@Override

public ArrayQueue<E> clone() {

ArrayQueue<E> clone = new ArrayQueue<>(this.capacity);

for (int i = 0; i < size(); i++) {

clone.enqueue(this.array[(front + i) % capacity]);

}

return clone;

}

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()) return; // إذا كانت قائمة الانتظار الثانية فارغة، لا تفعل شيئًا

if (this.isEmpty()) {

this.front = Q2.front; // نقل المرجع إلى العناصر

this.rear = Q2.rear;

} else {

this.rear.next = Q2.front; // ربط نهاية القائمة الأولى ببداية القائمة الثانية

this.rear = Q2.rear; // تحديث نهاية القائمة

}

Q2.front = null; // جعل Q2 فارغة

Q2.rear = null;

}

1. Use a queue to solve the Josephus Problem.

public int josephus(int n, int k) {

Queue<Integer> queue = new LinkedList<>();

for (int i = 1; i <= n; i++) {

queue.add(i);

}

while (queue.size() > 1) {

for (int i = 0; i < k - 1; i++) {

queue.add(queue.poll()); // نقل العنصر إلى النهاية

}

queue.poll(); // إزالة العنصر في الموقع k

}

return queue.poll(); // العنصر المتبقي هو الناجي

}

1. Use a queue to simulate Round Robin Scheduling.

public void roundRobin(List<Process> processes, int timeQuantum) {

Queue<Process> queue = new LinkedList<>(processes);

while (!queue.isEmpty()) {

Process current = queue.poll();

if (current.burstTime > timeQuantum) {

current.burstTime -= timeQuantum;

queue.add(current); // إعادة إدخال العملية إلى نهاية القائمة

} else {

// العملية انتهت

System.out.println("Process " + current.id + " finished.");

}

}

}