Topics

* Create Queue Interface
* Create Queue Using Array
* Create Queue Using Linked Lists
* Implement Basic Methods of Queue
* isEmpty()
* size()
* first()
* enqueue(E e)
* dequeue()

Homework

* 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 class ArrayQueue<E> {

private E[] data;

private int front = 0;

private int size = 0;

@SuppressWarnings("unchecked")

public ArrayQueue(int capacity) {

data = (E[]) new Object[capacity];

}

public boolean isEmpty() { return size == 0; }

public int size() { return size; }

public E first() { return isEmpty() ? null : data[front]; }

public void enqueue(E e) {

int avail = (front + size) % data.length;

data[avail] = e;

size++;

}

public E dequeue() {

if (isEmpty()) return null;

E answer = data[front];

data[front] = null;

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

size--;

return answer;

}

// الدالة الجديدة rotate()

public void rotate() {

if (!isEmpty()) {

front = (front + 1) % data.length; // تحريك مؤشر البداية

}

}

}

----------------------------------------------------------------------------------------

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

public class ArrayQueue<E> implements Cloneable {

private E[] data;

private int front = 0;

private int size = 0;

@SuppressWarnings("unchecked")

public ArrayQueue(int capacity) {

data = (E[]) new Object[capacity];

}

@Override

public ArrayQueue<E> clone() {

try {

ArrayQueue<E> cloned = (ArrayQueue<E>) super.clone();

cloned.data = data.clone(); // نسخ المصفوفة

return cloned;

} catch (CloneNotSupportedException e) {

throw new AssertionError();

}

}

}

--------------------------------------------------------------------------------------

* 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 class LinkedQueue<E> {

private static class Node<E> {

E element;

Node<E> next;

public Node(E e, Node<E> n) {

element = e;

next = n;

}

}

private Node<E> front = null;

private Node<E> rear = null;

private int size = 0;

public boolean isEmpty() { return size == 0; }

public int size() { return size; }

public void enqueue(E e) {

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

if (isEmpty()) front = newNode;

else rear.next = newNode;

rear = newNode;

size++;

}

public E dequeue() {

if (isEmpty()) return null;

E answer = front.element;

front = front.next;

size--;

if (isEmpty()) rear = null;

return answer;

}

// دالة الدمج

public void concatenate(LinkedQueue<E> Q2) {

if (Q2.isEmpty()) return; // لا يوجد شيء لدمجه

if (this.isEmpty()) { // إذا كان الطابور الأصلي فارغًا، اجعل Q2 هو الطابور الجديد

this.front = Q2.front;

this.rear = Q2.rear;

} else {

this.rear.next = Q2.front; // ربط نهاية الطابور الأصلي ببداية Q2

this.rear = Q2.rear; // تحديث المؤشر rear ليشير إلى نهاية Q2

}

this.size += Q2.size;

Q2.front = Q2.rear = null; // إفراغ Q2

Q2.size = 0;

}

}

----------------------------------------------------------------------------------------

* Use a queue to solve the Josephus Problem.

import java.util.LinkedList;

import java.util.Queue;

public class JosephusProblem {

public static 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.remove()); // تدوير الطابور

}

queue.remove(); // قتل الشخص

}

return queue.peek();

}

public static void main(String[] args) {

int N = 7, k = 3;

System.out.println("The survivor is: " + josephus(N, k)); // الناتج: 4

}

}

----------------------------------------------------------------------------------------

* Use a queue to simulate Round Robin Scheduling.

import java.util.LinkedList;

import java.util.Queue;

class Process {

String name;

int time;

public Process(String name, int time) {

this.name = name;

this.time = time;

}

}

public class RoundRobin {

public static void simulateRoundRobin(Queue<Process> queue, int quantum) {

while (!queue.isEmpty()) {

Process p = queue.poll();

if (p.time > quantum) {

System.out.println(p.name + " executed for " + quantum + " units.");

p.time -= quantum;

queue.add(p);

} else {

System.out.println(p.name + " finished execution.");

}

}

}

public static void main(String[] args) {

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

queue.add(new Process("P1", 10));

queue.add(new Process("P2", 5));

queue.add(new Process("P3", 8));

simulateRoundRobin(queue, 4);

}

}