## 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()

// واجهة الطابور

public interface QueueInterface<E> {

boolean isEmpty();

int size();

E first();

void enqueue(E e);

E dequeue();

}

// الطابور باستخدام المصفوفة

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

private ArrayList<E> queue = new ArrayList<>();

public boolean isEmpty() {

return queue.isEmpty();

}

public int size() {

return queue.size();

}

public E first() {

return queue.get(0);

}

public void enqueue(E e) {

queue.add(e);

}

public E dequeue() {

return queue.remove(0);

}

}

// العقدة للقائمة المرتبطة

class Node<E> {

E data;

Node<E> next;

public Node(E data) {

this.data = data;

}

}

// الطابور باستخدام القائمة المرتبطة

public class LinkedListQueue<E> implements QueueInterface<E> {

private Node<E> front, rear;

public boolean isEmpty() {

return front == null;

}

public int size() {

int count = 0;

Node<E> temp = front;

while (temp != null) {

count++;

temp = temp.next;

}

return count;

}

public E first() {

return front.data;

}

public void enqueue(E e) {

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

if (rear == null) {

front = rear = temp;

return;

}

rear.next = temp;

rear = temp;

}

public E dequeue() {

if (isEmpty()) {

return null;

}

E data = front.data;

front = front.next;

if (front == null) {

rear = null;

}

return data;

}

}

## 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).

// الطابور باستخدام المصفوفة مع الدالة rotate()

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

private ArrayList<E> queue = new ArrayList<>();

public boolean isEmpty() {

return queue.isEmpty();

}

public int size() {

return queue.size();

}

public E first() {

return queue.get(0);

}

public void enqueue(E e) {

queue.add(e);

}

public E dequeue() {

return queue.remove(0);

}

// دالة rotate() الجديدة

public void rotate() {

if (!isEmpty()) {

queue.add(queue.remove(0));

}

}

}

Implement the clone( ) method for the ArrayQueue class.

import java.util.ArrayList;

// الطابور باستخدام المصفوفة مع الدالة clone()

public class ArrayQueue<E> implements Cloneable {

private ArrayList<E> queue;

public ArrayQueue() {

this.queue = new ArrayList<>();

}

public boolean isEmpty() {

return queue.isEmpty();

}

public int size() {

return queue.size();

}

public E first() {

return queue.get(0);

}

public void enqueue(E e) {

queue.add(e);

}

public E dequeue() {

return queue.remove(0);

}

// دالة clone() الجديدة

@Override

public ArrayQueue<E> clone() {

try {

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

cloneQueue.queue = (ArrayList<E>) this.queue.clone();

return cloneQueue;

} catch (CloneNotSupportedException e) {

throw new AssertionError();

}

}

}

import java.util.ArrayList;

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.

class Node<E> {

E data;

Node<E> next;

public Node(E data) {

this.data = data;

}

}

class LinkedQueue<E> {

private Node<E> front, rear;

public LinkedQueue() {

this.front = this.rear = null;

}

public boolean isEmpty() {

return this.front == null;

}

public void enqueue(E data) {

Node<E> temp = new Node<>(data);

if (this.rear == null) {

this.front = this.rear = temp;

return;

}

this.rear.next = temp;

this.rear = temp;

}

public E dequeue() {

if (isEmpty()) {

return null;

}

Node<E> temp = this.front;

this.front = this.front.next;

if (this.front == null) {

this.rear = null;

}

return temp.data;

}

public void concatenate(LinkedQueue<E> Q2) {

if (this.rear != null) {

this.rear.next = Q2.front;

} else {

this.front = Q2.front;

}

if (Q2.rear != null) {

this.rear = Q2.rear;

}

Q2.front = Q2.rear = null;

}

}

1. 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<>();

// إضافة جميع الأشخاص إلى الطابور

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

queue.add(i);

}

while (queue.size() > 1) {

// تخطي k-1 أشخاص

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

int frontPerson = queue.poll();

queue.add(frontPerson);

}

// إزالة الشخص k-th

queue.poll();

}

// الشخص الأخير المتبقي هو الناجي

return queue.poll();

}

public static void main(String[] args) {

int n = 7;

int k = 3;

System.out.println("الشخص الذي ينجو في مشكلة جوزيفوس مع " + n + " أشخاص وكل " + k + " شخص محذوف هو: " + josephus(n, k));

}

}

1. Use a queue to simulate Round Robin Scheduling.

import java.util.LinkedList;

import java.util.Queue;

class Process {

int pid; // Process ID

int bt; // Burst Time

public Process(int pid, int bt) {

this.pid = pid;

this.bt = bt;

}

}

public class RoundRobinScheduling {

static void roundRobin(Queue<Process> q, int quantum) {

int time = 0; // Current time

while (!q.isEmpty()) {

Process p = q.poll();

if (p.bt > quantum) {

time += quantum;

System.out.println("Process ID: " + p.pid + ", start time: " + (time - quantum) + ", end time: " + time);

p.bt -= quantum;

q.add(p);

} else {

time += p.bt;

System.out.println("Process ID: " + p.pid + ", start time: " + (time - p.bt) + ", end time: " + time);

}

}

}

public static void main(String[] args) {

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

q.add(new Process(1, 24));

q.add(new Process(2, 3));

q.add(new Process(3, 3));

roundRobin(q, 4);

}

}