## 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).

import java.util.Arrays;

public class ArrayQueue<T> {

private T[] data; // مصفوفة لتخزين البيانات

private int front; // مؤشر العنصر الأول

private int rear; // مؤشر آخر عنصر

private int size; // عدد العناصر في الطابور

private static final int DEFAULT\_CAPACITY = 10; // الحجم الافتراضي

// \*\* المُنشئ (Constructor) \*\*

@SuppressWarnings("unchecked")

public ArrayQueue(int capacity) {

data = (T[]) new Object[capacity]; // إنشاء مصفوفة بالحجم المحدد

front = 0;

rear = -1;

size = 0;

}

public ArrayQueue() {

this(DEFAULT\_CAPACITY); // استخدام الحجم الافتراضي

}

// \*\* التحقق مما إذا كان الطابور فارغًا \*\*

public boolean isEmpty() {

return size == 0;

}

// \*\* التحقق مما إذا كان الطابور ممتلئًا \*\*

public boolean isFull() {

return size == data.length;

}

// \*\* إدخال عنصر إلى الطابور \*\*

public void enqueue(T item) {

if (isFull()) {

throw new IllegalStateException("الطابور ممتلئ!");

}

rear = (rear + 1) % data.length; // تحديث rear دائريًا

data[rear] = item;

size++;

}

// \*\* إزالة العنصر الأمامي من الطابور وإرجاعه \*\*

public T dequeue() {

if (isEmpty()) {

throw new IllegalStateException("الطابور فارغ!");

}

T item = data[front];

data[front] = null; // إخلاء الخانة

front = (front + 1) % data.length; // تحديث front دائريًا

size--;

return item;

}

// \*\* إرجاع العنصر الأمامي دون حذفه \*\*

public T front() {

if (isEmpty()) {

throw new IllegalStateException("الطابور فارغ!");

}

return data[front];

}

// \*\* تنفيذ `rotate()` بشكل أكثر كفاءة \*\*

public void rotate() {

if (!isEmpty()) {

rear = (rear + 1) % data.length; // تحديث rear ليشير إلى الموضع الجديد

data[rear] = data[front]; // نقل العنصر الأمامي إلى النهاية

data[front] = null; // إخلاء الخانة القديمة

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

// \*\*لاحظ: لم نعدل `size` لأن عدد العناصر لم يتغير\*\*

}

}

// \*\* دالة لطباعة الطابور \*\*

@Override

public String toString() {

return "ArrayQueue" + Arrays.toString(data);

}

// \*\* الدالة الرئيسية لتجربة الكود \*\*

public static void main(String[] args) {

ArrayQueue<Integer> queue = new ArrayQueue<>(5);

// إدخال عناصر إلى الطابور

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.enqueue(40);

queue.enqueue(50);

System.out.println("قبل الدوران:");

System.out.println(queue);

// تنفيذ `rotate()`

queue.rotate();

System.out.println("\nبعد الدوران:");

System.out.println(queue);

}

}

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

import java.util.Arrays;

public class ArrayQueue<T> implements Cloneable {

private T[] data; // مصفوفة لتخزين البيانات

private int front; // مؤشر العنصر الأول

private int rear; // مؤشر آخر عنصر

private int size; // عدد العناصر في الطابور

private static final int DEFAULT\_CAPACITY = 10; // الحجم الافتراضي

// \*\* المُنشئ (Constructor) \*\*

@SuppressWarnings("unchecked")

public ArrayQueue(int capacity) {

data = (T[]) new Object[capacity]; // إنشاء مصفوفة بالحجم المحدد

front = 0;

rear = -1;

size = 0;

}

public ArrayQueue() {

this(DEFAULT\_CAPACITY); // استخدام الحجم الافتراضي

}

// \*\* التحقق مما إذا كان الطابور فارغًا \*\*

public boolean isEmpty() {

return size == 0;

}

// \*\* التحقق مما إذا كان الطابور ممتلئًا \*\*

public boolean isFull() {

return size == data.length;

}

// \*\* إدخال عنصر إلى الطابور \*\*

public void enqueue(T item) {

if (isFull()) {

throw new IllegalStateException("الطابور ممتلئ!");

}

rear = (rear + 1) % data.length; // تحديث rear دائريًا

data[rear] = item;

size++;

}

// \*\* إزالة العنصر الأمامي من الطابور وإرجاعه \*\*

public T dequeue() {

if (isEmpty()) {

throw new IllegalStateException("الطابور فارغ!");

}

T item = data[front];

data[front] = null; // إخلاء الخانة

front = (front + 1) % data.length; // تحديث front دائريًا

size--;

return item;

}

// \*\* د

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.

### **1️⃣ تعريف الكلاس والخصائص**

public class ArrayQueue<T> implements Cloneable {

private T[] data; // مصفوفة لتخزين العناصر

private int front; // مؤشر العنصر الأول (أول عنصر في الطابور)

private int rear; // مؤشر آخر عنصر (آخر عنصر مضاف)

private int size; // عدد العناصر الحالية في الطابور

private static final int DEFAULT\_CAPACITY = 10; // الحجم الافتراضي للطابور

@SuppressWarnings("unchecked")

public ArrayQueue(int capacity) {

data = (T[]) new Object[capacity]; // إنشاء مصفوفة بالحجم المحدد

front = 0;

rear = -1;

size = 0;

}

public ArrayQueue() {

this(DEFAULT\_CAPACITY); // استخدام الحجم الافتراضي (10)

}

// التحقق مما إذا كان الطابور فارغًا

public boolean isEmpty() {

return size == 0;

}

// التحقق مما إذا كان الطابور ممتلئًا

public boolean isFull() {

return size == data.length;

}

مثال عملي

public static void main(String[] args) {

ArrayQueue<Integer> queue = new ArrayQueue<>(5);

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.enqueue(40);

queue.enqueue(50);

System.out.println("قبل الحذف: " + Arrays.toString(queue.data));

queue.dequeue();

queue.dequeue();

System.out.println("بعد حذف عنصرين: " + Arrays.toString(queue.data));

queue.enqueue(60);

queue.enqueue(70);

System.out.println("بعد إضافة عنصرين جديدين: " + Arrays.toString(queue.data));

}

**لماذا استخدمنا المصفوفة الدائرية؟**

* **تجنب إهدار الذاكرة** عند حذف العناصر في المصفوفة العادية.
* **تحسين الأداء** باستخدام O(1) لكل من enqueue() و dequeue()

1. Use a queue to solve the Josephus Problem.

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

public class JosephusProblem {

public static int josephus(int n, int k) {

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

// إدراج الأشخاص من 1 إلى n في الطابور

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

queue.add(i);

}

// تنفيذ عملية الحذف حتى يبقى شخص واحد

while (queue.size() > 1) {

// تحريك `k-1` أشخاص إلى نهاية الطابور

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

queue.add(queue.poll()); // نقل أول عنصر إلى نهاية الطابور

}

queue.poll(); // حذف الشخص المستهدف

}

return queue.peek(); // الشخص الأخير هو الفائز

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// إدخال عدد الأشخاص والرقم الذي يتم عنده الحذف

System.out.print("أدخل عدد الأشخاص (n): ");

int n = scanner.nextInt();

System.out.print("أدخل عدد الخطوات (k): ");

int k = scanner.nextInt();

// حساب وحفظ الفائز

int winner = josephus(n, k);

// طباعة النتيجة

System.out.println("الشخص الفائز هو: " + winner);

scanner.close();

}

}

1. Use a queue to simulate Round Robin Scheduling.

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

// تعريف كلاس Process لتمثيل كل عملية

class Process {

String name;

int burstTime; // إجمالي وقت التنفيذ

int remainingTime; // الوقت المتبقي للتنفيذ

public Process(String name, int burstTime) {

this.name = name;

this.burstTime = burstTime;

this.remainingTime = burstTime;

}

}

public class RoundRobinScheduler {

public static void roundRobinScheduling(Queue<Process> queue, int timeQuantum) {

int currentTime = 0;

while (!queue.isEmpty()) {

Process process = queue.poll(); // استخراج أول عملية في الطابور

// تحديد الوقت الذي سيتم تنفيذه

int executionTime = Math.min(process.remainingTime, timeQuantum);

// تحديث الوقت المتبقي

process.remainingTime -= executionTime;

currentTime += executionTime;

// طباعة تفاصيل التنفيذ

System.out.println("Executing " + process.name + " for " + executionTime + " units. Remaining: " + process.remainingTime);

// إذا لم تكتمل العملية، نعيدها إلى نهاية الطابور

if (process.remainingTime > 0) {

queue.add(process);

} else {

System.out.println(process.name + " has completed at time " + currentTime);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

// إدخال عدد العمليات

System.out.print("أدخل عدد العمليات: ");

int n = scanner.nextInt();

// إدخال العمليات

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

System.out.print("أدخل اسم العملية " + i + ": ");

String name = scanner.next();

System.out.print("أدخل وقت تنفيذ العملية " + name + ": ");

int burstTime = scanner.nextInt();

queue.add(new Process(name, burstTime));

}

// إدخال الزمن المخصص لكل عملية (time quantum)

System.out.print("أدخل زمن time quantum: ");

int timeQuantum = scanner.nextInt();

// تنفيذ الجدولة بطريقة Round Robin

System.out.println("\nبدء الجدولة:");

roundRobinScheduling(queue, timeQuantum);

scanner.close();

}

}