public class Lab8 {  
 /\*  
  
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).  
 قم بتعزيز تنفيذ ArrayQueue باستخدام أسلوب التدوير () الجديد الذي له دلالات مماثلة للمجموعة، enqueue(dequeue( )).  
 ولكن، يجب أن يكون التنفيذ الخاص بك أكثر كفاءة من إجراء مكالمتين منفصلتين (على سبيل المثال، لأنه ليست هناك حاجة لتعديل الحجم).  
 يتم حفظ العنصر في نفسه واول عنصر لF بعدين نقله  
 public class ArrayQueue<E> implements Queue<E> {  
 private E[] elements;  
 private int front;  
 private int rear;  
 private int size;  
  
 // constructor and other methods  
  
 public void rotate() {  
 if (size > 0) {  
 // حفظ العنصر الأول  
 E firstElement = elements[front];  
  
 // نقل مؤشر front للعنصر التالي  
 front = (front + 1) % elements.length;  
  
 // نقل مؤشر rear للموقع الفارغ الذي سيتم إضافة العنصر إليه  
 rear = (rear + 1) % elements.length;  
  
 // وضع العنصر الأول في الموقع الجديد  
 elements[rear] = firstElement;  
 }  
 }  
}  
2. Implement the clone( ) method for the ArrayQueue class.  
2. قم بتنفيذ طريقة الاستنساخ () لفئة ArrayQueue.  
يتم تعين اري لتنفيد الاري انشاء نسخه للحال ويتم انشاء نسخه من كيو وجعله للنسخه  
 public class ArrayQueue<E> implements Queue<E> {  
 // other class variables and methods  
  
 @Override  
 public ArrayQueue<E> clone() {  
 try {  
  
 @SuppressWarnings("unchecked")  
 ArrayQueue<E> clonedQueue = (ArrayQueue<E>) super.clone();  
 clonedQueue.elements = elements.clone();  
 return clonedQueue;  
 } catch (CloneNotSupportedException e) {  
 throw new AssertionError(); // Should not happen  
  
 }  
 }  
}  
  
3. 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.  
 . قم بتنفيذ طريقة مع سلسلة التوقيع (LinkedQueue Q2) لفئة  
 LinkedQueue التي تأخذ جميع عناصر Q2 وتلحقها بنهاية قائمة الانتظار الأصلية. يجب أن يتم تشغيل العملية في وقت O(1) ويجب أن تؤدي إلى كون  
 Q2 قائمة انتظار فارغة.  
 تحديث واجهه ليكن يتم تحديث Cلثلاث امور  
 public class LinkedQueue<E>  
  
implements Queue<E> {  
 // other class variables and methods  
  
 public void concatenate(LinkedQueue<E> Q2) {  
 if (Q2.isEmpty()) {  
 return;  
 }  
  
 if (isEmpty()) {  
 front = Q2.front;  
 } else {  
 rear.next = Q2.front;  
  
 }  
  
 rear = Q2.rear;  
 size += Q2.size;  
  
 Q2.front = null;  
 Q2.rear = null;  
 Q2.size = 0;  
 }  
}  
  
  
  
4. Use a queue to solve the Josephus Problem.  
4. استخدم قائمة الانتظار لحل مشكلة جوزيفوس.  
يتم انشاء قائمه انتظار ب اسم كيو يتم اضافه جميع جيع اشخاص فور يتم اخراج لنهايه حلقه1  
import java.util.LinkedList;  
  
import java.util.Queue;  
  
public class JosephusProblem {  
 public static int josephus(int n, int k) {  
 Queue<Integer> queue = new LinkedList<>();  
  
 // Enqueue all people from 1 to n  
 for (int i = 1; i <= n; i++) {  
 queue.add(i);  
 }  
  
 while (queue.size() > 1) {  
  
 // Dequeue k-1 people and enqueue them back  
 for (int i = 0; i < k - 1; i++) {  
 int person = queue.remove();  
 queue.add(person);  
 }  
  
 // Dequeue the k-th person (eliminate)  
 queue. Remove();  
 }  
  
5. Use a queue to simulate Round Robin Scheduling.  
5. استخدم قائمة الانتظار لمحاكاة جدولة Round Robin.  
تعريف كل واحد وتعريف قيمه افتراضيه ترجع ترة فارغ حجمها  
public class ArrayQueue<E> implements Queue<E> {  
 private static final int DEFAULT\_CAPACITY = 10;  
 private E[] elements;  
 private int front;  
 private int rear;  
  
 private int size;  
  
 public ArrayQueue() {  
 this(DEFAULT\_CAPACITY);  
 }  
  
 @SuppressWarnings("unchecked")  
 public ArrayQueue(int capacity) {  
 elements = (E[]) new Object[capacity];  
 front = 0;  
 rear = -1;  
  
 size = 0;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 @Override  
 public int size() {  
 return size;  
 }  
  
 @Override  
 public E first() {  
  
 if (isEmpty()) {  
 throw new IllegalStateException("Queue is empty");  
 }  
 return elements[front];  
 }  
  
 @Override  
 public void enqueue(E element) {  
 if (size == elements.length) {  
 throw new IllegalStateException("Queue is  
  
full");  
 }  
 rear = (rear + 1) % elements.length;  
 elements[rear] = element;  
 size++;  
 }  
  
 @Override  
 public E dequeue() {  
 if (isEmpty()) {  
 throw new IllegalStateException("Queue is empty");  
 }  
  
 E removedElement = elements[front];  
 elements[front] = null;  
 front = (front + 1) % elements.length;  
 size--;  
 return removedElement;  
 }  
}  
  
  
  
  
  
  
  
  
  
  
 \*/  
}