## Topics

1. Create Queue Interface

Certainly! Here's an example of how you can create a Queue interface in Java:

public interface Queue<E> {

boolean isEmpty();

int size();

void enqueue(E element);

E dequeue();

E peek();

}

1. Create Queue Using Array

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

private E[] elements;

private int front;

private int rear;

private int size;

private static final int DEFAULT\_CAPACITY = 10;

public ArrayQueue() {

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

front = 0;

rear = -1;

size = 0;

}

public boolean isEmpty() {

return size == 0;

}

public int size() {

return size;

}

public void enqueue(E element) {

if (size == elements.length) {

resizeArray(size \* 2);

}

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

elements[rear] = element;

size++;

}

public E dequeue() {

if (isEmpty()) {

throw new IllegalStateException("Queue is empty");

}

E element = elements[front];

elements[front] = null; // Dereference the dequeued element

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

size--;

return element;

}

public E peek() {

if (isEmpty()) {

throw new IllegalStateException("Queue is empty");

}

return elements[front];

}

private void resizeArray(int newCapacity) {

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

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

newElements[i] = elements[(front + i) % elements.length];

}

elements = newElements;

front = 0;

rear = size - 1;

}

public static void main(String[] args) {

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

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

System.out.println("Dequeued element: " + queue.dequeue());

System.out.println("Front element: " + queue.peek());

System.out.println("Queue size: " + queue.size());

System.out.println("Is queue empty? " + queue.isEmpty());

}

}

1. Create Queue Using Linked Lists

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

private Node<E> front;

private Node<E> rear;

private int size;

private static class Node<E> {

private E element;

private Node<E> next;

public Node(E element) {

this.element = element;

this.next = null;

}

}

public LinkedListQueue() {

front = null;

rear = null;

size = 0;

}

public boolean isEmpty() {

return size == 0;

}

public int size() {

return size;

}

public void enqueue(E element) {

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

if (isEmpty()) {

front = newNode;

} else {

rear.next = newNode;

}

rear = newNode;

size++;

}

public E dequeue() {

if (isEmpty()) {

throw new IllegalStateException("Queue is empty");

}

E element = front.element;

front = front.next;

if (front == null) {

rear = null;

}

size--;

return element;

}

public E peek() {

if (isEmpty()) {

throw new IllegalStateException("Queue is empty");

}

return front.element;

}

public static void main(String[] args) {

LinkedListQueue<Integer> queue = new LinkedListQueue<>();

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

System.out.println("Dequeued element: " + queue.dequeue());

System.out.println("Front element: " + queue.peek());

System.out.println("Queue size: " + queue.size());

System.out.println("Is queue empty? " + queue.isEmpty());

}

}

1. Implement Basic Methods of Queue

* isEmpty()
* size()
* first()
* enqueue(E e)
* dequeue()

import java.util.LinkedList;

public class MyQueue<E> {

private LinkedList<E> queue;

public MyQueue() {

queue = new LinkedList<>();

}

public boolean isEmpty() {

return queue.isEmpty();

}

public int size() {

return queue.size();

}

public E first() {

if (isEmpty()) {

throw new IllegalStateException("Queue is empty");

}

return queue.peekFirst();

}

public void enqueue(E element) {

queue.addLast(element);

}

public E dequeue() {

if (isEmpty()) {

throw new IllegalStateException("Queue is empty");

}

return queue.removeFirst();

}

public static void main(String[] args) {

MyQueue<Integer> queue = new MyQueue<>();

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

System.out.println("Dequeued element: " + queue.dequeue());

System.out.println("Front element: " + queue.first());

System.out.println("Queue size: " + queue.size());

System.out.println("Is queue empty? " + queue.isEmpty());

}

}

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

// Existing code...

public void rotate() {

if (isEmpty()) {

return; // Nothing to rotate if the queue is empty

}

E element = dequeue();

enqueue(element);

}

// Existing code...

}

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

import java.util.Arrays;

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

private static final int DEFAULT\_CAPACITY = 10;

private E[] elements;

private int front;

private int rear;

@SuppressWarnings("unchecked")

public ArrayQueue() {

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

front = 0;

rear = -1;

}

// Existing code...

@SuppressWarnings("unchecked")

public ArrayQueue<E> clone() {

ArrayQueue<E> clonedQueue = new ArrayQueue<>();

clonedQueue.elements = Arrays.copyOf(elements, elements.length);

clonedQueue.front = front;

clonedQueue.rear = rear;

return clonedQueue;

}

// Existing code...

}

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 class LinkedQueue<E> implements Queue<E> {

private Node<E> front;

private Node<E> rear;

private int size;

private static class Node<E> {

private E element;

private Node<E> next;

public Node(E element) {

this.element = element;

this.next = null;

}

}

public LinkedQueue() {

front = null;

rear = null;

size = 0;

}

// Existing code...

public void concatenate(LinkedQueue<E> Q2) {

if (Q2.isEmpty()) {

return; // Nothing to concatenate if Q2 is empty

}

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;

}

// Existing code...

}

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<>();

// Enqueue all people from 1 to n

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

queue.add(i);

}

while (queue.size() > 1) {

// Count k-1 positions and dequeue

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

int front = queue.remove();

queue.add(front);

}

// Dequeue the k-th position person (eliminated)

queue.remove();

}

// The last person remaining in the queue is the survivor

return queue.remove();

}

public static void main(String[] args) {

int n = 7; // Number of people

int k = 3; // Counting interval

int survivor = josephus(n, k);

System.out.println("The survivor in the Josephus Problem with n = " + n + " and k = " + k + " is: " + survivor);

}

}

1. Use a queue to simulate Round Robin Scheduling.

import java.util.LinkedList;

import java.util.Queue;

public class RoundRobinScheduling {

public static void simulate(int[] processes, int timeQuantum) {

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

// Enqueue all processes

for (int process : processes) {

queue.add(process);

}

while (!queue.isEmpty()) {

int currentProcess = queue.remove();

// Execute the current process for the time quantum

System.out.println("Executing process " + currentProcess + " for time quantum " + timeQuantum);

// Reduce the remaining execution time of the process

int remainingTime = timeQuantum - 1;

if (remainingTime > 0) {

// Enqueue the process at the end of the queue

queue.add(currentProcess);

}

}

}

public static void main(String[] args) {

int[] processes = {1, 2, 3, 4, 5}; // Processes to be scheduled

int timeQuantum = 2; // Time quantum for each process

System.out.println("Round Robin Scheduling Simulation");

System.out.println("Processes: " + java.util.Arrays.toString(processes));

System.out.println("Time Quantum: " + timeQuantum);

System.out.println("------------------------------");

simulate(processes, timeQuantum);

}

}