CS 315 E #2: Covers: Chapter 8 [Queues], Chapter 10 [Algorithm Efficiency and Sorting] and Chapter 11 [Trees]

**Queues:** Chapter 8 : pp 409- - 424)

* Be Able to understand the difference between implementing a queue as a circular linked list or as a “circular” array.
* Be able to read an apply client code that uses queues.

**Provided Code**:

|  |
| --- |
| public class QueueReferenceBased implements QueueInterface {  private Node lastNode;  …..  } |

|  |
| --- |
| public class QueueArrayBased implements QueueInterface {  private final int MAX\_QUEUE = 8; // maximum size of queue  private Object[] items;  private int front, back, count;  …..} |

|  |
| --- |
| public class Node {  Object item;  Node next;  // constructors, accessors,  // and mutators …  } // end class Node |

**Key to Review Problems on Queues:**

**1.** Using the following client code, show the contents of the queue which is **implemented as a Circular** Array with a capacity of 8 elements, and the output:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| // Client Code  QueueArrayBased A = new QueueArrayBased();  int num1, num2;  for (int i = 4; i <= 6; i++){  A.enqueue(i);  }  // Show Queue Content  for (int i= 1; i<= 3; i++ ){  num1 = (Integer) A.dequeue();  num2 = (Integer)A.dequeue( );  A.enqueue(num1 + num2);  A. enqueue(num2 - num1);  // SHOW content of Queue  }    while (!A.isEmpty()) {  System.out.print(A.dequeue( ) + " ");  } | **Content of A**  front = 0, back = 2   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | | **4** | **5** | **6** |  |  |  |  |  |   front = 2, back = 4   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | |  |  | **6** | **9** | **1** |  |  |  |   front = 4, back = 6   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | |  |  |  |  | **1** | **15** | **3** |  |   front = 6 , back = 0   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | | 14 |  |  |  |  |  | 3 | 16 | | **Output:**  3 16 14 |

2. Given the following Queue implemented as a circular linked list

|  |
| --- |
|  |

Draw a picture and provide the implementation code **for each independent operation**. Ie Use the same picture above for both!

|  |  |  |
| --- | --- | --- |
| **Client Code:**  QueueReferenceBased NumQueue = new QueueReferenceBased(); | Picture of Queue AFTER client code is executed: | Implementation Code needed |
| NumQueue. enqueue(10); | 10  4  7  1  2  lastNode | Node newNode = new Node((Integer) 10);  newNode.next = lastNode.next;  lastNode.next = newNode;  lastNode = newNode; |
| int A = (Integer) NumQueue. dequeue(); | 4  7  1  lastNode | 2  10  Node firstNode = lastNode.next;  lastNode.next = firstNode.next;  return firstNode.item;  **Note: the Integer value 2 would be returned!** |

3. Given the following queue which was implemented as a Circular Array with a capacity of 8:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| front = 7  Back  Front  back = 1  count = 3   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | | 8 | 12 |  |  |  |  |  | 4 | |

Draw a picture and provide the implementation code **for each independent operation**. Ie Use the same picture above for both!

|  |  |  |
| --- | --- | --- |
| **Client Code:**  QueueArrayBased NumQueue = new QueueArrayBased (); | Picture of Queue AFTER client code is executed. | Implementation Code needed |
| NumQueue. Enqueue(10); | front = 7  back = 2  count = 4   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | | 8 | 12 | 10 |  |  |  |  | 4 | | back = (back+1) % 8;  items[2] = 10;  count = count + 1; |
| int A = (Integer) NumQueue. dequeue(); | front = 0  back = 1  count = 2   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | | 8 | 12 |  |  |  |  |  |  | | Object item = items[7];  front = (front+1) % 8;  count = count - 1;  return item; |