Queue Resize

```
public class CircularQueue {
    private int[] queue;
    private int front;
    private int rear;
    private int size;

    public CircularQueue(int initialCapacity) {
        queue = new int[initialCapacity];
        front = 0;
        rear = 0;
        size = 0;
    }

    private void resize() {
        int newCapacity = queue.length * 2;
        int[] newArray = new int[newCapacity];

        for (int i = 0; i < size; i++) {
            newArray[i] = queue[(front + i) % queue.length];
        }

        queue = newArray;
        front = 0;
        rear = size;
    }

    public void enqueue(int value) {
        if (size == queue.length) {
            resize();
        }
        queue[rear] = value;
        rear = (rear + 1) % queue.length;
        size++;
    }

    // Additional methods (dequeue, etc.) would be implemented here
}
```

Circular LinkedList

```
public class CircularLinkedListQueue {
    private Node tail; // Using tail to keep track of the end of the queue
    private int size; // To keep track of the size of the queue

    // Constructor
    public CircularLinkedListQueue() {
        this.tail = null;
        this.size = 0;
    }

    // Node class
    private class Node {
        int data;
        Node next;

        public Node(int data) {
```

```
        this.data = data;
        this.next = null;
    }
}

// Method to add an element to the queue
public void enqueue(int data) {
    Node newNode = new Node(data);
    if (tail == null) {
        tail = newNode;
        tail.next = tail; // Point to itself, making the list circular
    } else {
        newNode.next = tail.next; // New node points to the head
        tail.next = newNode; // Old tail points to new node
        tail = newNode; // New node becomes the new tail
    }
    size++;
}

// Method to remove an element from the queue
public int dequeue() {
    if (tail == null) {
        throw new IllegalStateException("Queue is empty");
    }

    Node head = tail.next; // The head is the element next to tail
    if (tail == tail.next) { // Only one element in the queue
        tail = null; // Queue is now empty
    } else {
```

```java
        tail.next = head.next; // Tail points to the second element
    }
    size--;
    return head.data;
}

// Method to check if the queue is empty
public boolean isEmpty() {
    return tail == null;
}

// Method to get the size of the queue
public int size() {
    return size;
}

// Method to print the elements of the queue
public void printQueue() {
    if (tail == null) {
        System.out.println("Queue is empty");
        return;
    }
    Node temp = tail.next;
    do {
        System.out.print(temp.data + " ");
        temp = temp.next;
    } while (temp != tail.next);
    System.out.println();
}
```

```
}

// Example usage:
public class Main {
    public static void main(String[] args) {
        CircularLinkedListQueue queue = new CircularLinkedListQueue();
        queue.enqueue(1);
        queue.enqueue(2);
        queue.enqueue(3);
        queue.printQueue(); // Prints: 1 2 3

        queue.dequeue();
        queue.printQueue(); // Prints: 2 3

        System.out.println("Queue size: " + queue.size()); // Prints: 2
    }
}
```