Credit Name: Chapter13
Assignment Name: QueueList

How has your program changed from planning to coding to now? Please explain?

At first, the program was planned to show how a queue can be implemented using a linked list. The initial plan included creating a QueueList class with methods like enqueue(), dequeue(), peek(), isEmpty(), and size() to provide basic queue functionalities. A test class was also planned to allow the user to interact with the queue by adding and removing .

1. Forgetting to Initialize rear

Problem: When the first element was added, the rear was not set, which caused a NullPointerException in subsequent operations.

Fix: I updated the enqueue() method to initialize both front and rear when the queue is empty.

```
// Before
public void enqueue(Object item) {
    Node newNode = new Node(item);
    front = newNode;
}
// After
public void enqueue(Object item) {
    Node newNode = new Node(item);
    if (isEmpty()) {
        front = newNode;
        rear = newNode;
    } else {
        rear.setNext(newNode);
        rear = newNode;
    }
}
```

2. Incorrect dequeue() Logic

Problem: When the queue became empty after a dequeue() operation, the rear pointer was not updated, leading to a broken queue state.

Fix: I added a condition to update rear to null when the queue becomes empty.

```
// Before
public Object dequeue() {
    Object item = front.getData();
    front = front.getNext();
    return item;
}
// After
public Object dequeue() {
    Object item = front.getData();
    front = front.getNext();
    if (front == null) {
        rear = null;
    }
    return item;
}
```

3. Missing setNext() in enqueue()

Problem: The new node added during enqueue() was not linked to the existing queue, causing data loss.

Fix: I added rear.setNext(newNode) to link the new node to the existing queue before updating rear.

```java
// Before
public void enqueue(Object item) {
    rear = new Node(item);
}
// After
public void enqueue(Object item) {
    Node newNode = new Node(item);
    if (!isEmpty()) {
        rear.setNext(newNode);
    }
    rear = newNode;
}
```

4. Error Handling in peek()

Problem: The peek() method did not handle empty queues, leading to runtime errors when trying to access the front element.

Fix: I added a condition to check if the queue was empty before returning the front node's data.

```java
// Before
public Object peek() {
    return front.getData();
}
// After
public Object peek() {
    if (isEmpty()) {
        System.out.println("Queue is empty. No front item.");
        return null;
    }
    return front.getData();
}
```

5. Incorrect size() Calculation

Problem: The size() method did not traverse the queue properly, leading to incorrect size calculations.

Fix: I added a line to move the current pointer to the next node during traversal.

```
// Before
public int size() {
    int count = 0;
    Node current = front;
    while (current != null) {
        count++;
    }
    return count;
}
// After
public int size() {
    int count = 0;
    Node current = front;
    while (current != null) {
        count++;
        current = current.getNext();

    return count;
}
```

Now, the program works properly. It handles enqueue and dequeue operations, ensures the queue remains consistent after each operation.These adjustments improved the program's functionality and ensured a better user experience.