Name: ibrahim Alshaya

ID: 202176470

Task1:

The recursive contains method checks elements by traversing the list recursively, while toString builds the string representation. Testing confirms the recursive methods work well, showing they're effective for linked list operations.

Task1 output:

```
The list is: [Taif Dammam Abha Riyadh Jubail]
It is true that the list contains Dammam.
It is false that the list contains Jeddah.

Process finished with exit code 0
```

Task1 code:

contains method:

```
public boolean contains(T el) {
    return recursiveContains(head, el);
}

2 usages
private boolean recursiveContains(SLLNode<T> current, T el) {
    // Base case: if current is null, the element is not found
    if (current == null)
        return false;

    // If the current node's info matches the element, return true
    if (current.info.equals(el))
        return true;

    // Recursive call to check the rest of the list
    return recursiveContains(current.next, el);
}
```

ToString method:

```java
2 usages
private String recursiveToString(SLLNode<T> current) {
    // Base case: if current is null, return an empty string
    if (current == null)
        return "";

    // If current is the last node, return its info
    if (current.next == null)
        return current.info.toString();

    // Recursive call to append current node's info and continue with the next node
    return current.info.toString() + " " + recursiveToString(current.next);
}
```

Task2:

The recursive approach simplifies the code by recursively shifting elements in the array after dequeuing an element. Moreover, I wrote a test program to verify the functionality of the recursive dequeue method. This showcases the effectiveness of recursion in implementing queue operations concisely.

Task2 output:

```
C:\Users\xiibx\.jdks\openjdk-20.0.2\bin\java.exe "-javaage
The queue is: 60   20   40   30   70
First dequeued element is: 60
Second dequeued element is: 20
After two node deletion the queue is: 40   30   70
Element at queue front is: 40
```

Task2 code:

Original method:

```
public T dequeue() {
    if (isEmpty())
        throw new UnsupportedOperationException("Queue is empty!");

    T temp = queue[front]; // Store the front element

    // Base case: if there's only one element
    if (front == rear) {
        front = rear = -1; // Reset front and rear
    } else {
        // Recursive call to shift elements and update rear
        shiftElements( index: 0);
        rear--;
    }

    return temp; // Return the dequeued element
}
```

Helper method:

```
// Helper method to recursively shift elements in the array
2 usages
private void shiftElements(int index) {
    if (index == rear) {
        queue[index] = null; // Set the last element to null
        return;
    }

    queue[index] = queue[index + 1]; // Shift the element
    shiftElements( index: index + 1); // Recursive call for the next index
}
```