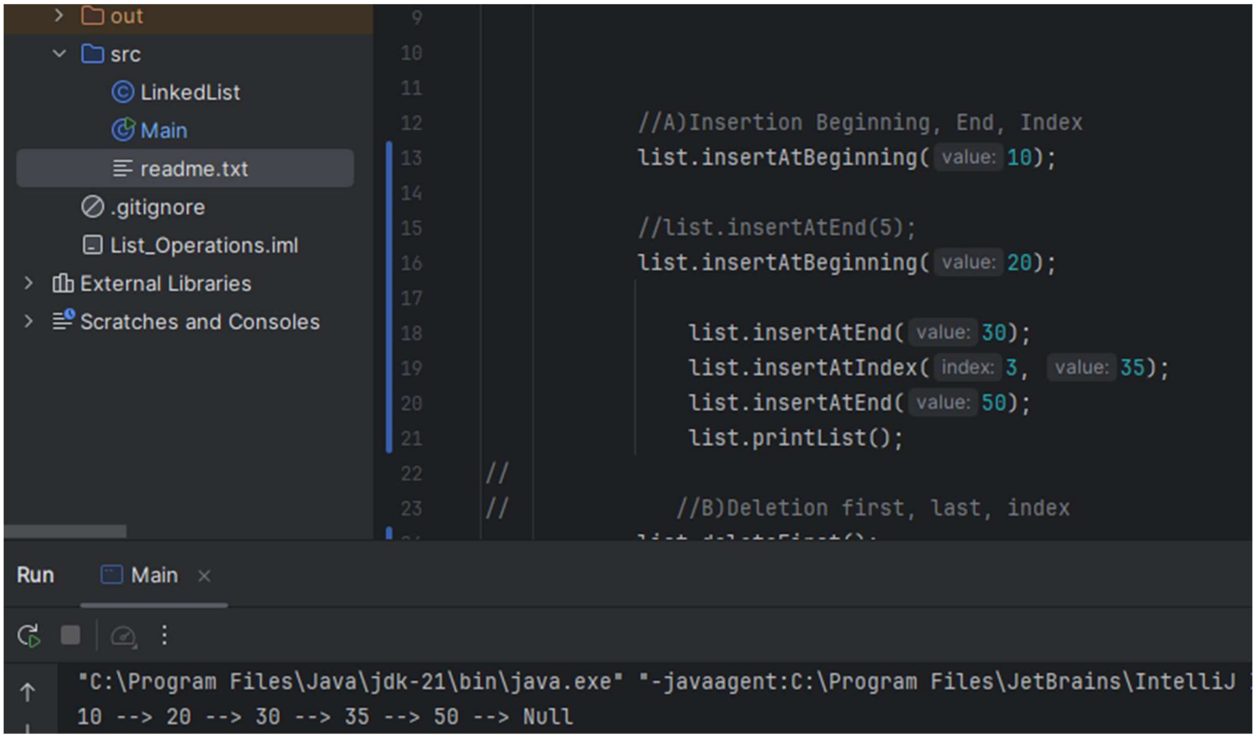
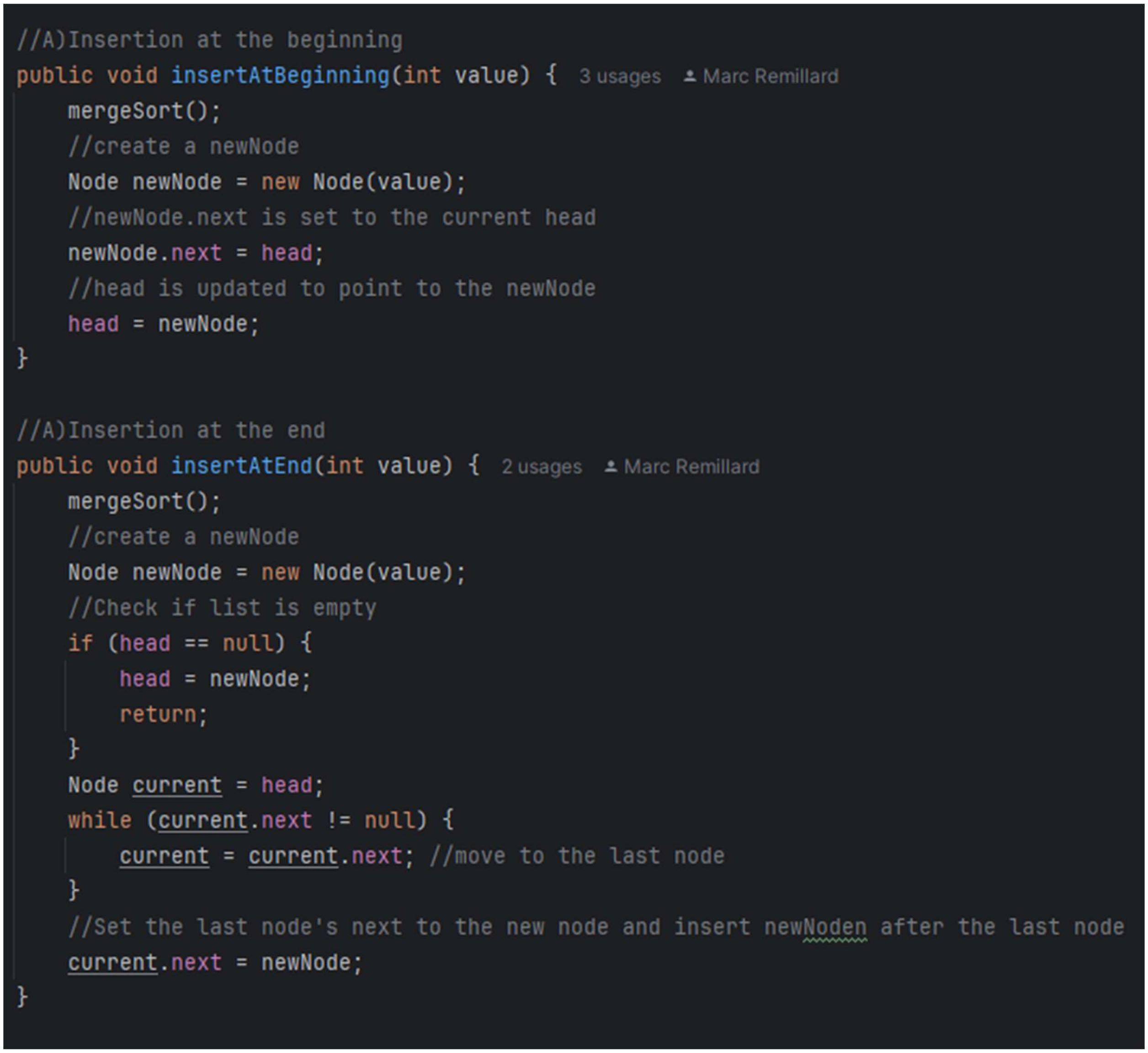
A)

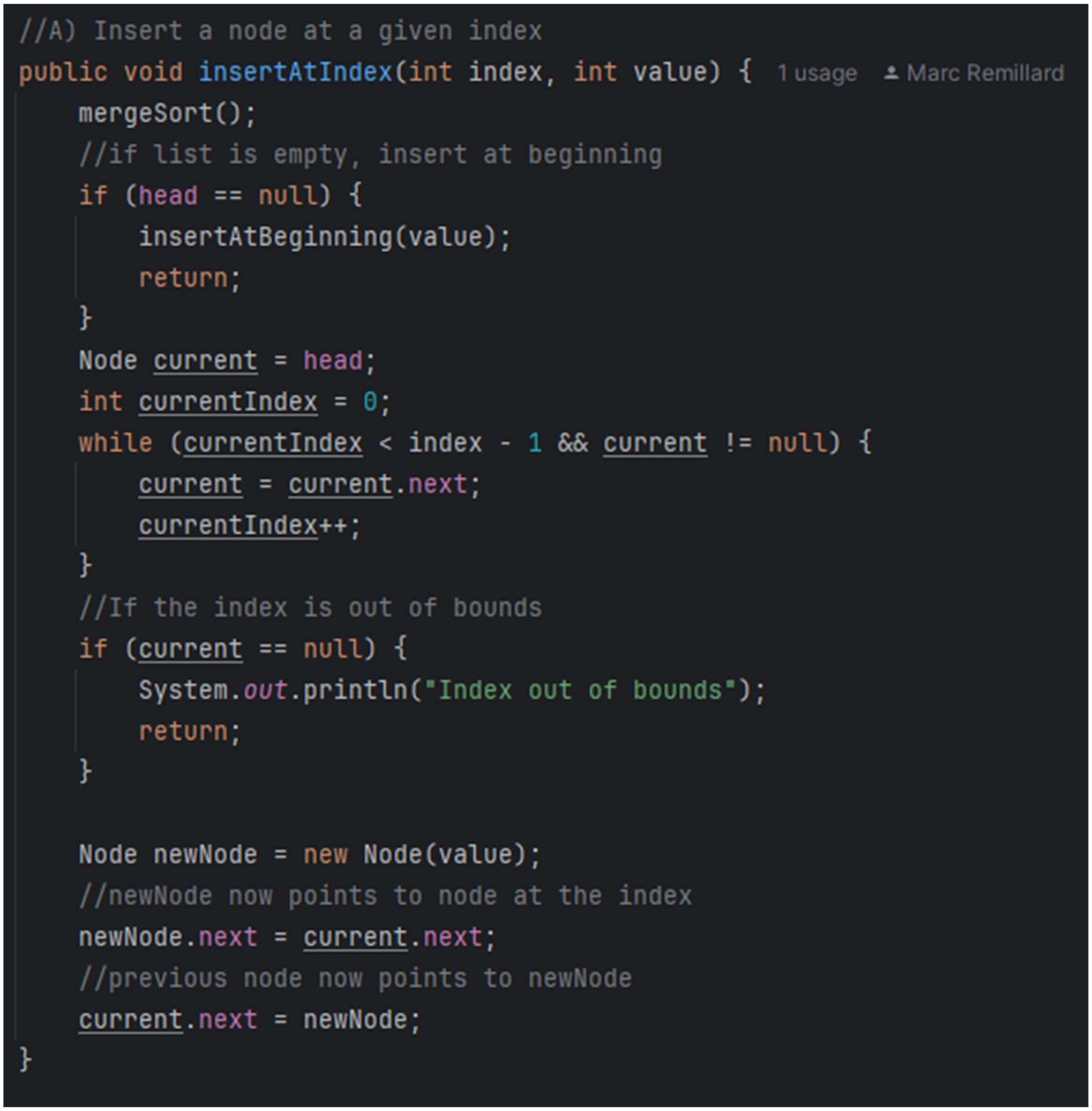


```
        //A)Insertion Beginning, End, Index
        list.insertAtBeginning( value: 10);

        //list.insertAtEnd(5);
        list.insertAtBeginning( value: 20);

            list.insertAtEnd( value: 30);
            list.insertAtIndex( index: 3,  value: 35);
            list.insertAtEnd( value: 50);
            list.printList();
//
//         //B)Deletion first, last, index
        list.deleteFirst();
```

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
10 --> 20 --> 30 --> 35 --> 50 --> Null
```

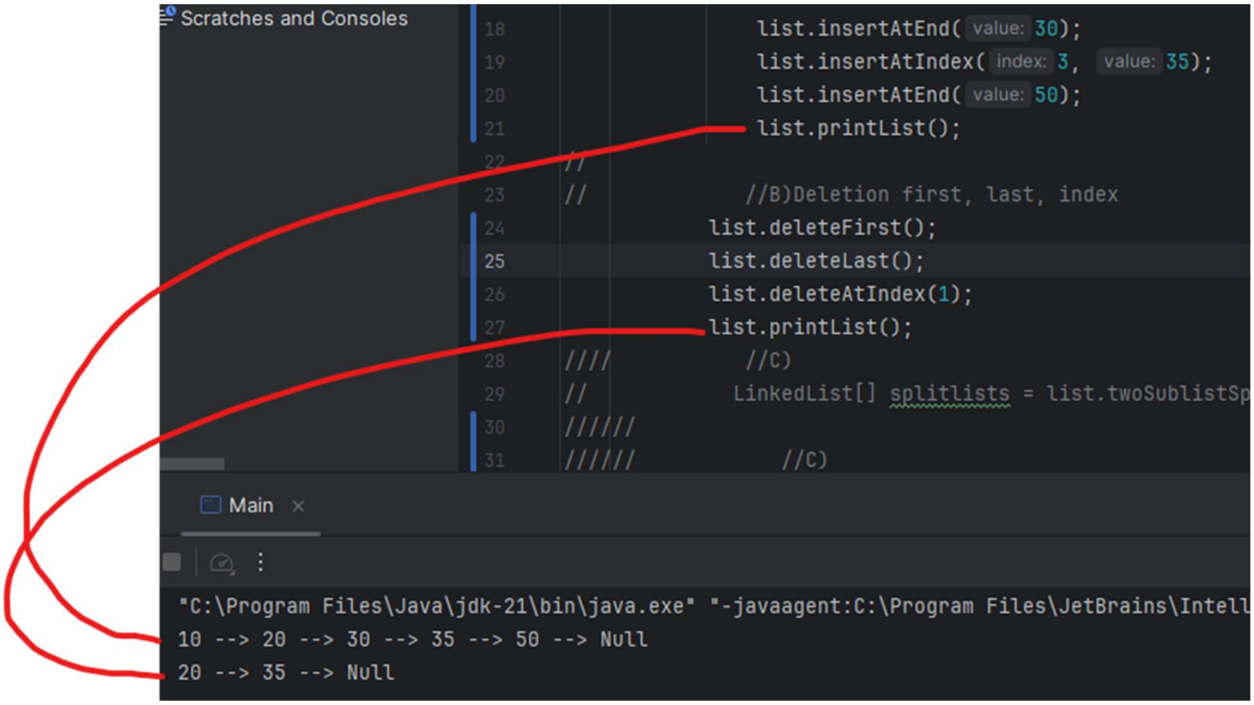


```
//A)Insertion at the beginning
public void insertAtBeginning(int value) {  3 usages   Marc Remillard
    mergeSort();
    //create a newNode
    Node newNode = new Node(value);
    //newNode.next is set to the current head
    newNode.next = head;
    //head is updated to point to the newNode
    head = newNode;
}

//A)Insertion at the end
public void insertAtEnd(int value) {  2 usages   Marc Remillard
    mergeSort();
    //create a newNode
    Node newNode = new Node(value);
    //Check if list is empty
    if (head == null) {
        head = newNode;
        return;
    }
    Node current = head;
    while (current.next != null) {
        current = current.next; //move to the last node
    }
    //Set the last node's next to the new node and insert newNoden after the last node
    current.next = newNode;
}
```



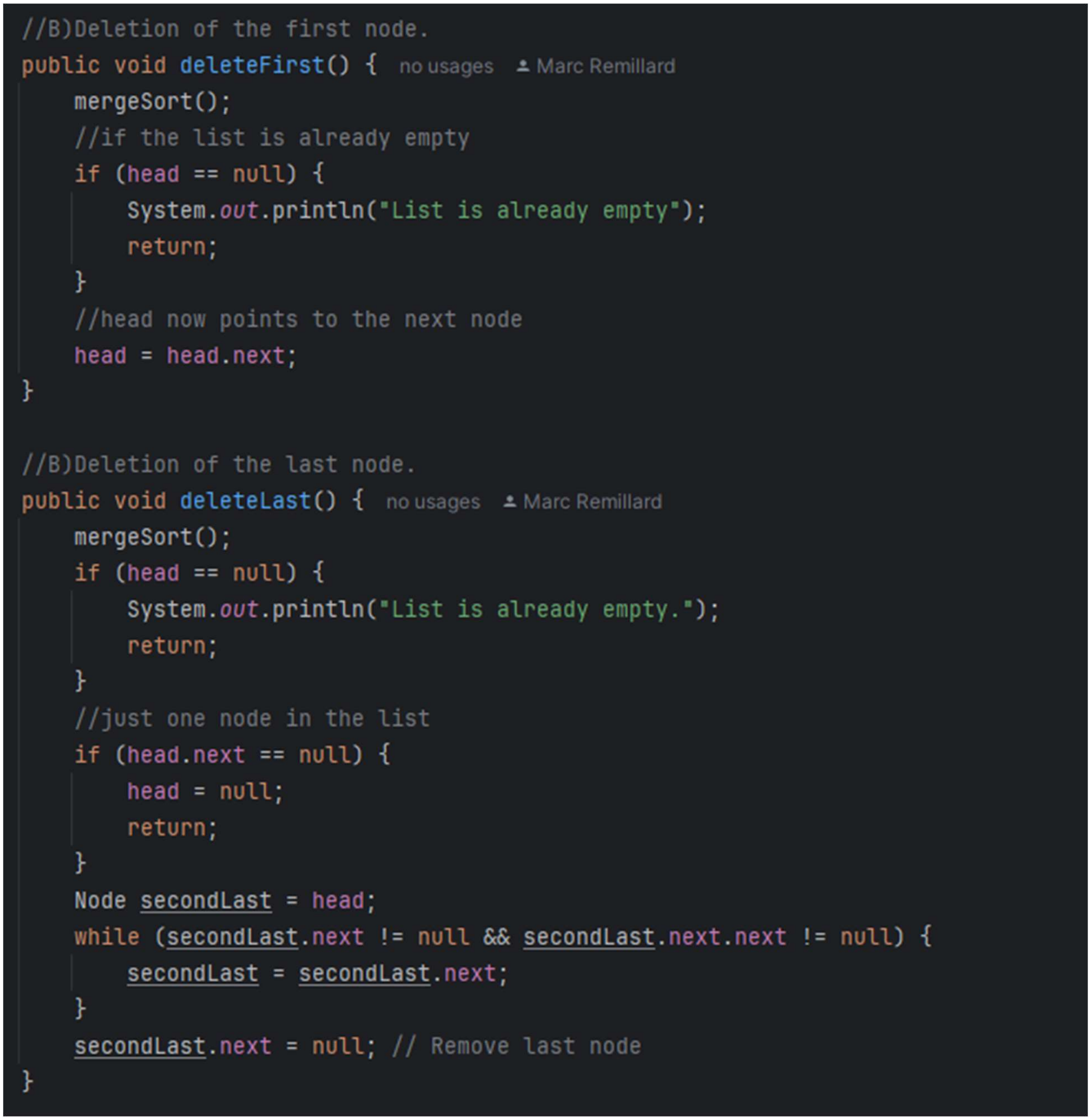
```java
//A) Insert a node at a given index
public void insertAtIndex(int index, int value) {
    mergeSort();
    //if list is empty, insert at beginning
    if (head == null) {
        insertAtBeginning(value);
        return;
    }
    Node current = head;
    int currentIndex = 0;
    while (currentIndex < index - 1 && current != null) {
        current = current.next;
        currentIndex++;
    }
    //If the index is out of bounds
    if (current == null) {
        System.out.println("Index out of bounds");
        return;
    }

    Node newNode = new Node(value);
    //newNode now points to node at the index
    newNode.next = current.next;
    //previous node now points to newNode
    current.next = newNode;
}
```

B)

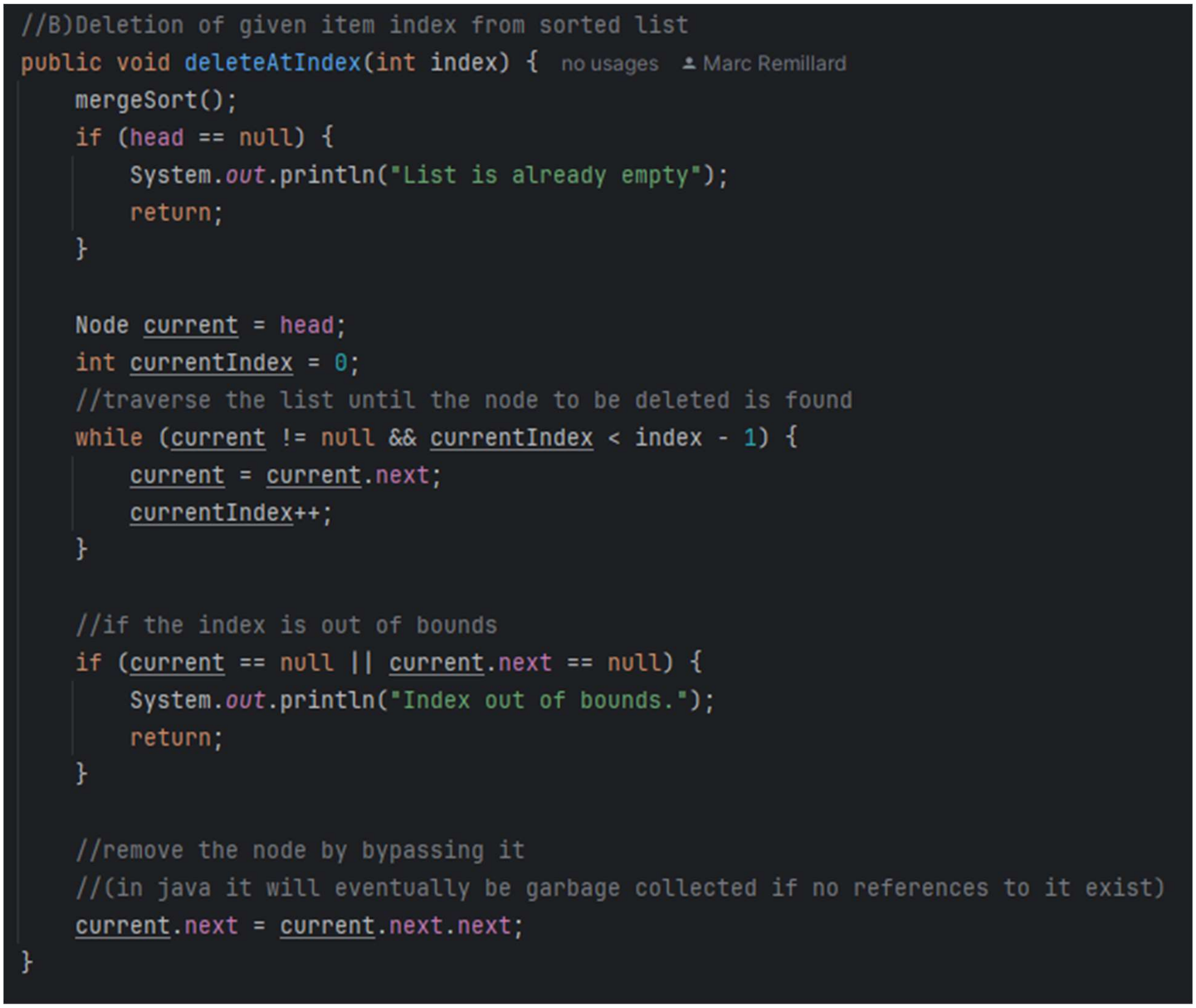


```
                list.insertAtEnd( value: 30);
                list.insertAtIndex( index: 3,  value: 35);
                list.insertAtEnd( value: 50);
                list.printList();
//
//             //B)Deletion first, last, index
            list.deleteFirst();
            list.deleteLast();
            list.deleteAtIndex(1);
            list.printList();
////           //C)
//            LinkedList[] splitlists = list.twoSublistSp
//////
//////            //C)
```

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Intell
10 --> 20 --> 30 --> 35 --> 50 --> Null
20 --> 35 --> Null
```

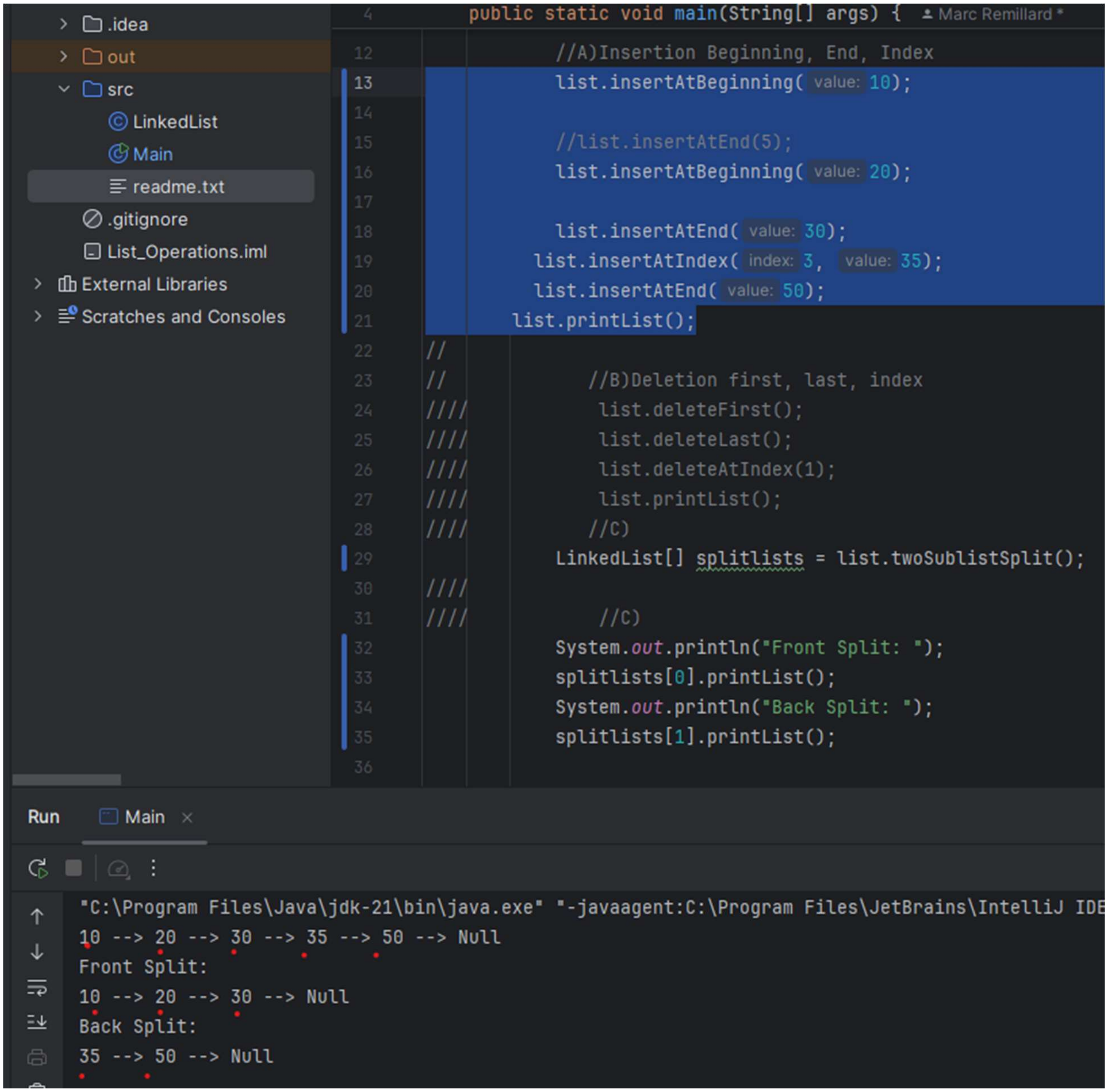


```
//B)Deletion of the first node.
public void deleteFirst() {  no usages  Marc Remillard
    mergeSort();
    //if the list is already empty
    if (head == null) {
        System.out.println("List is already empty");
        return;
    }
    //head now points to the next node
    head = head.next;
}

//B)Deletion of the last node.
public void deleteLast() {  no usages  Marc Remillard
    mergeSort();
    if (head == null) {
        System.out.println("List is already empty.");
        return;
    }
    //just one node in the list
    if (head.next == null) {
        head = null;
        return;
    }
    Node secondLast = head;
    while (secondLast.next != null && secondLast.next.next != null) {
        secondLast = secondLast.next;
    }
    secondLast.next = null; // Remove last node
}
```



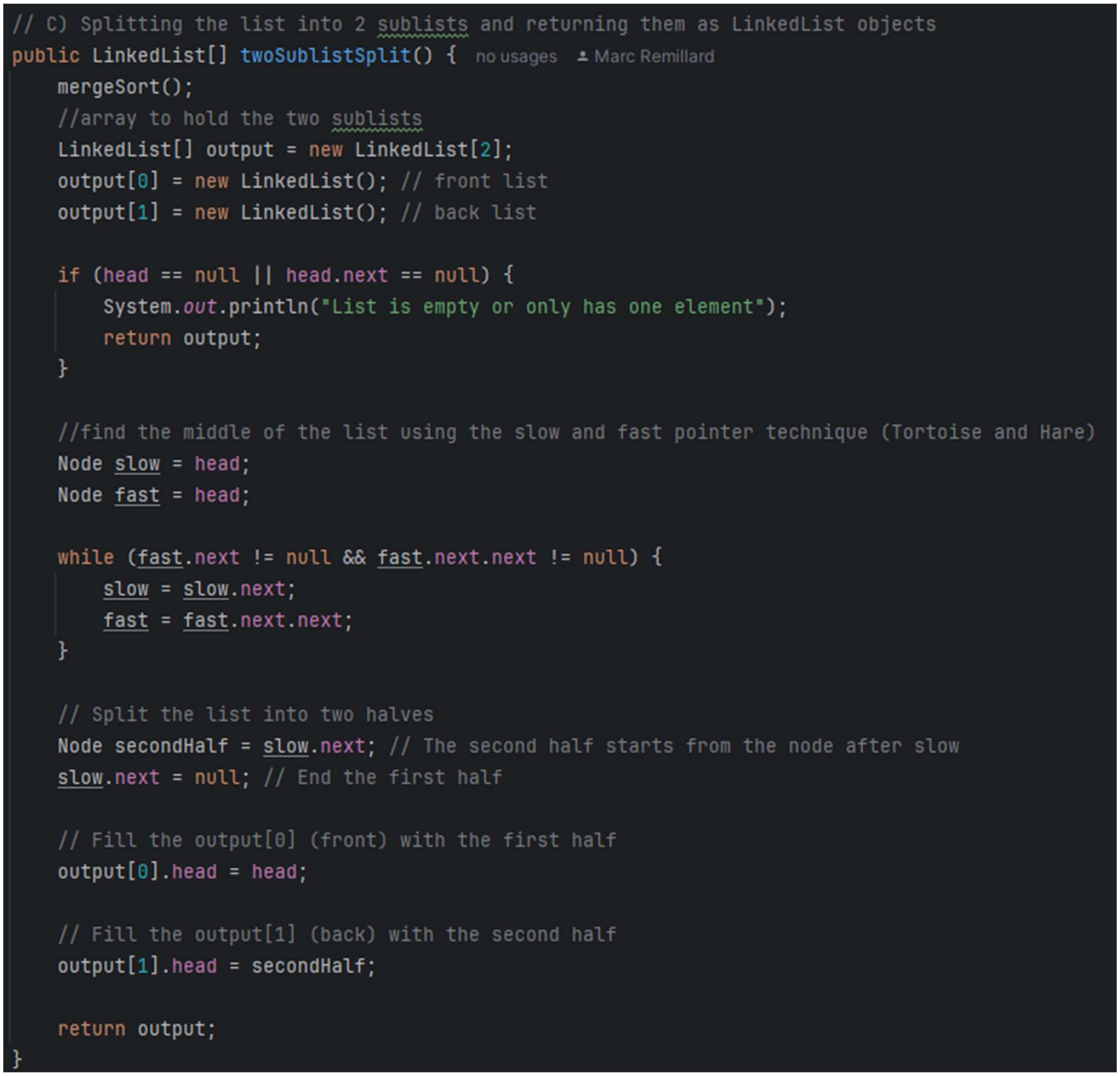
```
//B)Deletion of given item index from sorted list
public void deleteAtIndex(int index) {  no usages  Marc Remillard
    mergeSort();
    if (head == null) {
        System.out.println("List is already empty");
        return;
    }

    Node current = head;
    int currentIndex = 0;
    //traverse the list until the node to be deleted is found
    while (current != null && currentIndex < index - 1) {
        current = current.next;
        currentIndex++;
    }

    //if the index is out of bounds
    if (current == null || current.next == null) {
        System.out.println("Index out of bounds.");
        return;
    }

    //remove the node by bypassing it
    //(in java it will eventually be garbage collected if no references to it exist)
    current.next = current.next.next;
}
```

C)



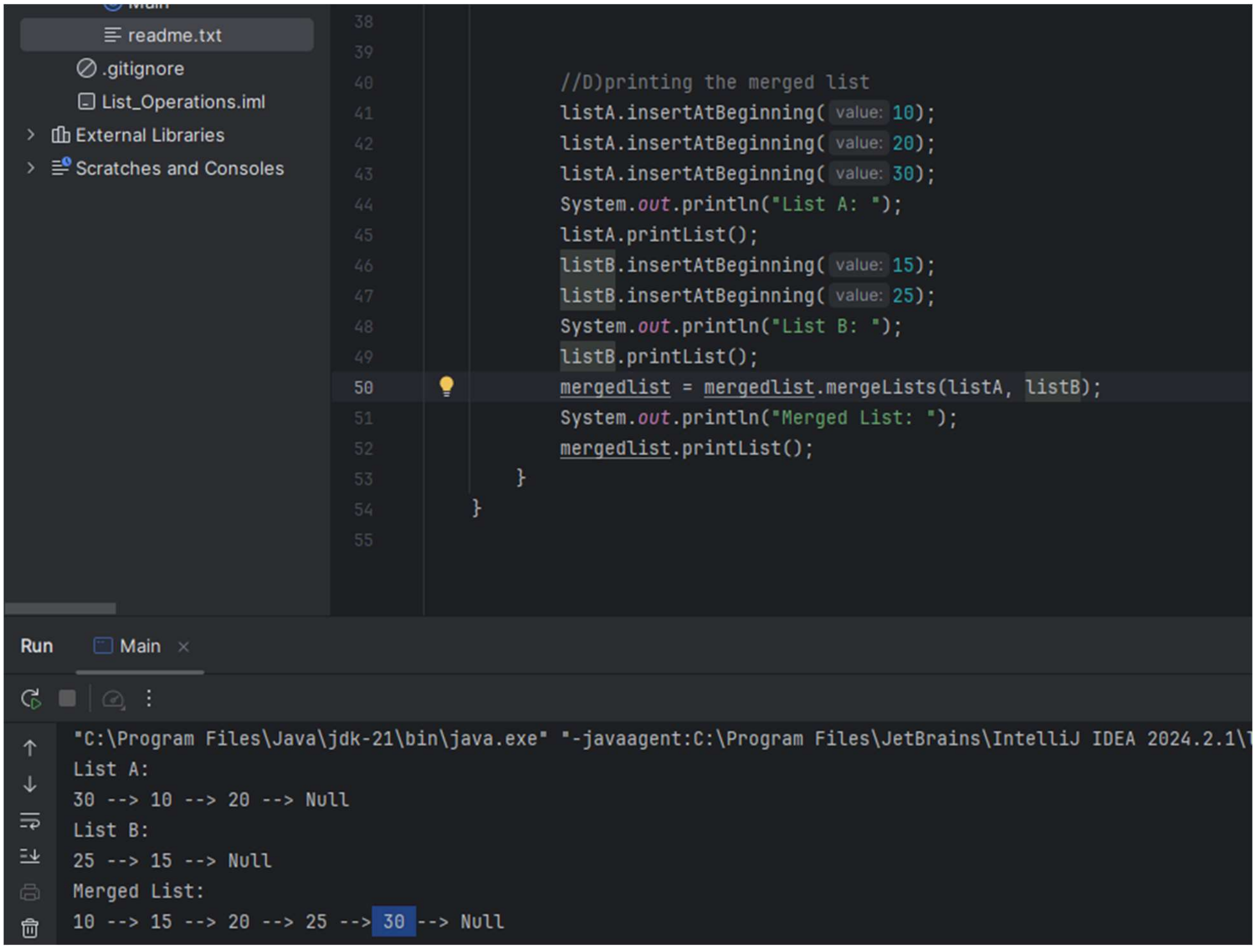
```
    public static void main(String[] args) {
            //A)Insertion Beginning, End, Index
            list.insertAtBeginning( value: 10);

            //list.insertAtEnd(5);
            list.insertAtBeginning( value: 20);

            list.insertAtEnd( value: 30);
          list.insertAtIndex( index: 3,  value: 35);
          list.insertAtEnd( value: 50);
        list.printList();
//
//             //B)Deletion first, last, index
////            list.deleteFirst();
////            list.deleteLast();
////            list.deleteAtIndex(1);
////            list.printList();
////           //C)
            LinkedList[] splitlists = list.twoSublistSplit();
////
////            //C)
            System.out.println("Front Split: ");
            splitlists[0].printList();
            System.out.println("Back Split: ");
            splitlists[1].printList();
```

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDE
10 --> 20 --> 30 --> 35 --> 50 --> Null
Front Split:
10 --> 20 --> 30 --> Null
Back Split:
35 --> 50 --> Null
```



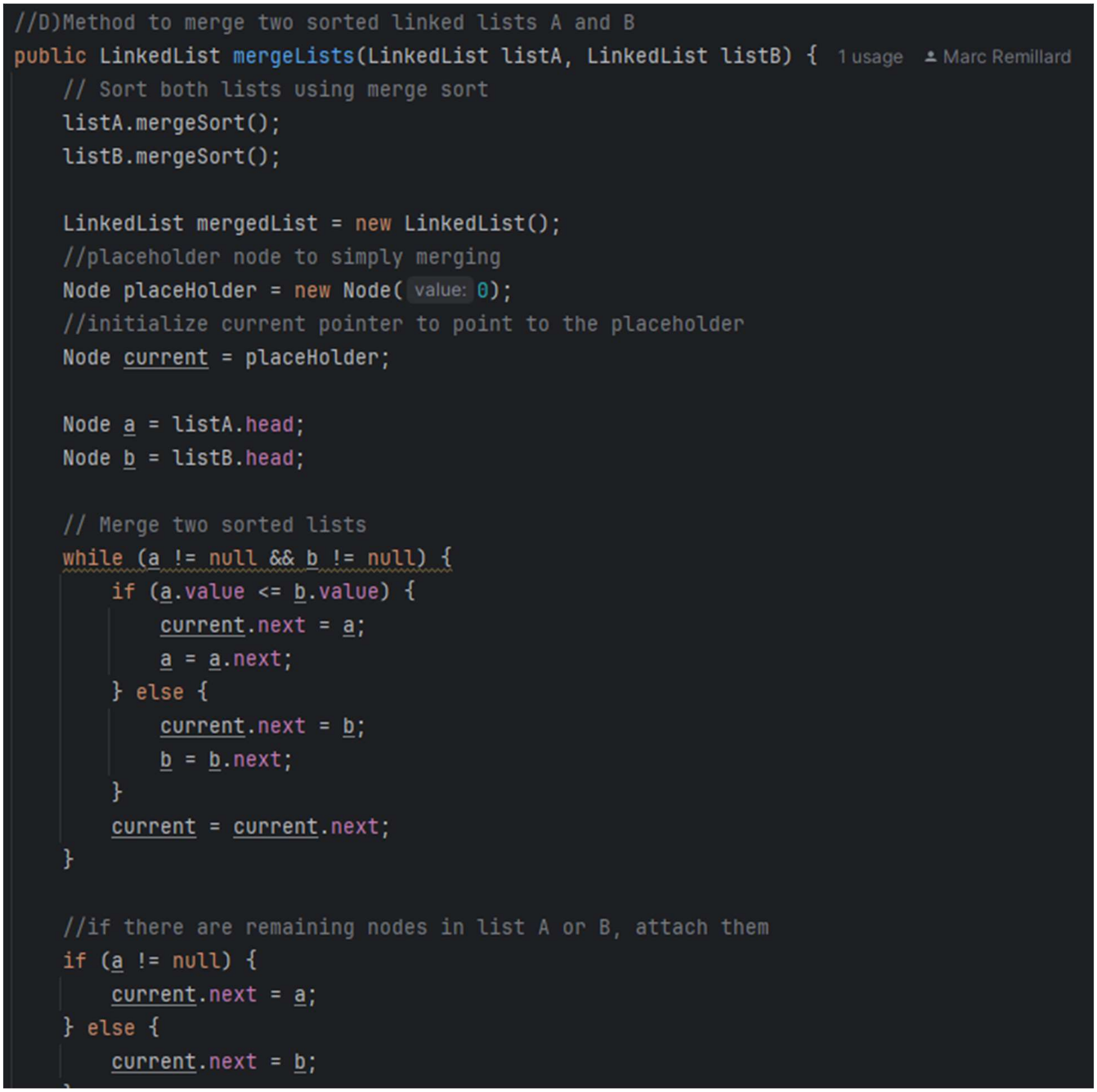
```java
// C) Splitting the list into 2 sublists and returning them as LinkedList objects
public LinkedList[] twoSublistSplit() {
    mergeSort();
    //array to hold the two sublists
    LinkedList[] output = new LinkedList[2];
    output[0] = new LinkedList(); // front list
    output[1] = new LinkedList(); // back list

    if (head == null || head.next == null) {
        System.out.println("List is empty or only has one element");
        return output;
    }

    //find the middle of the list using the slow and fast pointer technique (Tortoise and Hare)
    Node slow = head;
    Node fast = head;

    while (fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    // Split the list into two halves
    Node secondHalf = slow.next; // The second half starts from the node after slow
    slow.next = null; // End the first half

    // Fill the output[0] (front) with the first half
    output[0].head = head;

    // Fill the output[1] (back) with the second half
    output[1].head = secondHalf;

    return output;
}
```

D)



```
        //D)printing the merged list
        listA.insertAtBeginning( value: 10);
        listA.insertAtBeginning( value: 20);
        listA.insertAtBeginning( value: 30);
        System.out.println("List A: ");
        listA.printList();
        listB.insertAtBeginning( value: 15);
        listB.insertAtBeginning( value: 25);
        System.out.println("List B: ");
        listB.printList();
        mergedlist = mergedlist.mergeLists(listA, listB);
        System.out.println("Merged List: ");
        mergedlist.printList();
    }
}
```

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.2.1\
List A:
30 --> 10 --> 20 --> Null
List B:
25 --> 15 --> Null
Merged List:
10 --> 15 --> 20 --> 25 --> 30 --> Null
```



```
//D)Method to merge two sorted linked lists A and B
public LinkedList mergeLists(LinkedList listA, LinkedList listB) {  1 usage  Marc Remillard
    // Sort both lists using merge sort
    listA.mergeSort();
    listB.mergeSort();

    LinkedList mergedList = new LinkedList();
    //placeholder node to simply merging
    Node placeHolder = new Node( value: 0);
    //initialize current pointer to point to the placeholder
    Node current = placeHolder;

    Node a = listA.head;
    Node b = listB.head;

    // Merge two sorted lists
    while (a != null && b != null) {
        if (a.value <= b.value) {
            current.next = a;
            a = a.next;
        } else {
            current.next = b;
            b = b.next;
        }
        current = current.next;
    }

    //if there are remaining nodes in list A or B, attach them
    if (a != null) {
        current.next = a;
    } else {
        current.next = b;
```