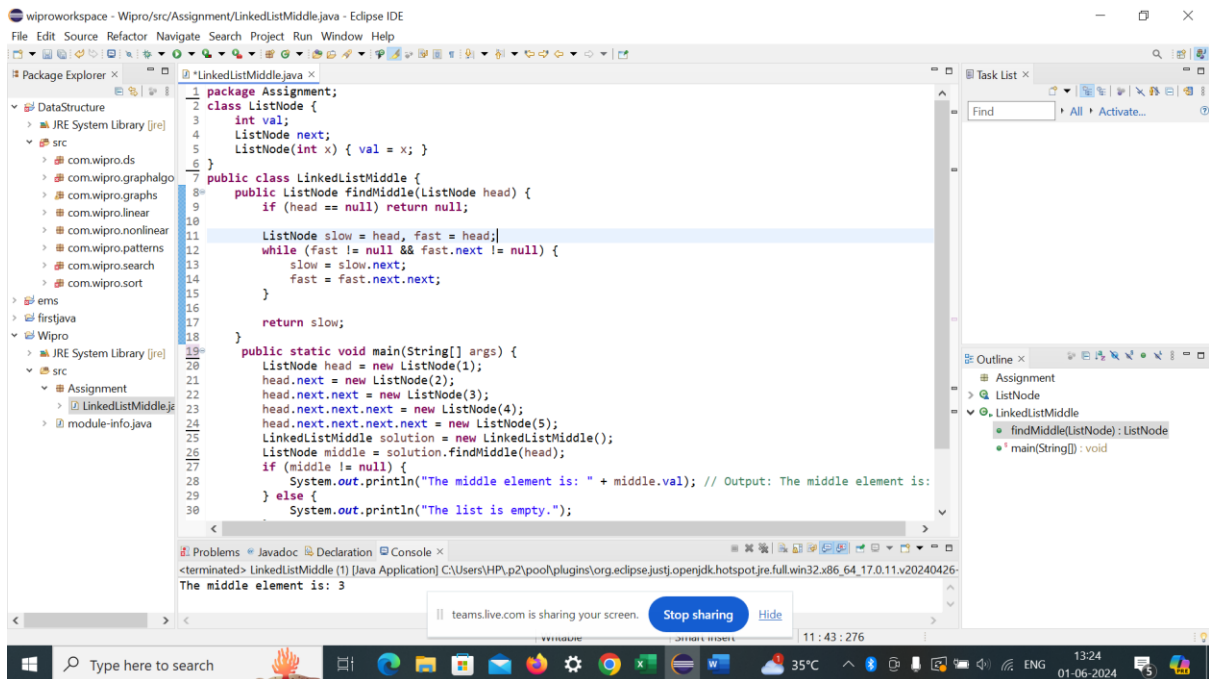


Data Structures

Task 2: Linked List Middle Element Search

You are given a singly linked list. Write a function to find the middle element without using any extra space and only one traversal through the linked list.



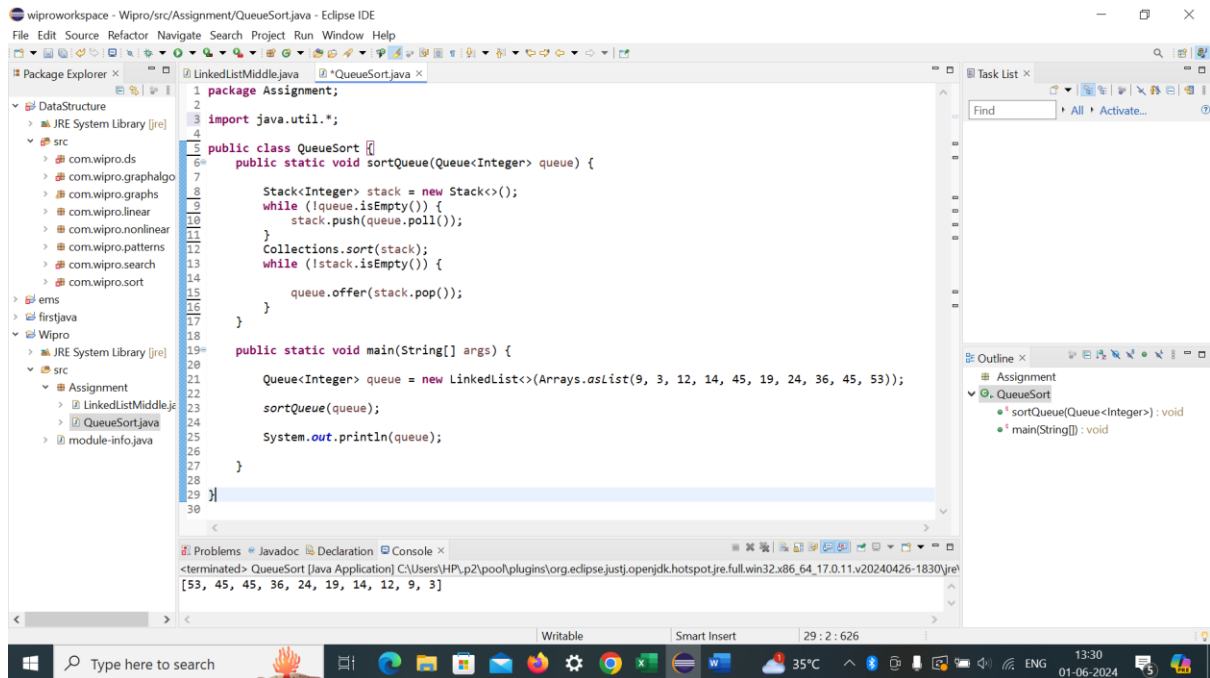
```
package Assignment;
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}
public class LinkedListMiddle {
    public ListNode findMiddle(ListNode head) {
        if (head == null) return null;
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
public static void main(String[] args) {
    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(3);
    head.next.next.next = new ListNode(4);
    head.next.next.next.next = new ListNode(5);
    LinkedListMiddle solution = new LinkedListMiddle();
    ListNode middle = solution.findMiddle(head);
    if (middle != null) {
        System.out.println("The middle element is: " + middle.val); // Output: The middle element is:
    } else {
        System.out.println("The list is empty.");
    }
}
```

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with packages like `com.wipro.ds`, `com.wipro.graphalgo`, `com.wipro.graphs`, `com.wipro.linear`, `com.wipro.nonlinear`, `com.wipro.patterns`, `com.wipro.search`, and `com.wipro.sort`.
- Editor:** Displays the `LinkedListMiddle.java` file with the implementation of the `findMiddle` method and a `main` method for testing.
- Task List:** Shows a list of tasks, including `findMiddle(ListNode) : ListNode` and `main(String[]) : void`.
- Problems:** Shows a message: `<terminated> LinkedListMiddle (1) [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-`.
- Console:** Displays the output: `The middle element is: 3`.

Task 3: Queue Sorting with Limited Space

You have a queue of integers that you need to sort. You can only use additional space equivalent to one stack. Describe the steps you would take to sort the elements in the queue.



The screenshot shows the Eclipse IDE with a Java project named 'wiproworkspace'. The main editor displays the file 'QueueSort.java' with the following code:

```
1 package Assignment;  
2  
3 import java.util.*;  
4  
5 public class QueueSort {  
6     public static void sortQueue(Queue<Integer> queue) {  
7  
8         Stack<Integer> stack = new Stack<>();  
9         while (!queue.isEmpty()) {  
10             stack.push(queue.poll());  
11         }  
12         Collections.sort(stack);  
13         while (!stack.isEmpty()) {  
14             queue.offer(stack.pop());  
15         }  
16     }  
17  
18     public static void main(String[] args) {  
19  
20         Queue<Integer> queue = new LinkedList<>(Arrays.asList(9, 3, 12, 14, 45, 19, 24, 36, 45, 53));  
21  
22         sortQueue(queue);  
23  
24         System.out.println(queue);  
25  
26     }  
27  
28 }  
29 }  
30
```

The Package Explorer on the left shows the project structure, including the 'src' folder and the 'QueueSort.java' file. The Outline view on the right shows the class structure with methods 'sortQueue' and 'main'. The Console view at the bottom shows the output of the program: '[53, 45, 45, 36, 24, 19, 14, 12, 9, 3]'.

Task 4: Stack Sorting In-Place

You must write a function to sort a stack such that the smallest items are on the top. You can use an additional temporary stack, but you may not copy the elements into any other data structure such as an array. The stack supports the following operations: push, pop, peek, and isEmpty.

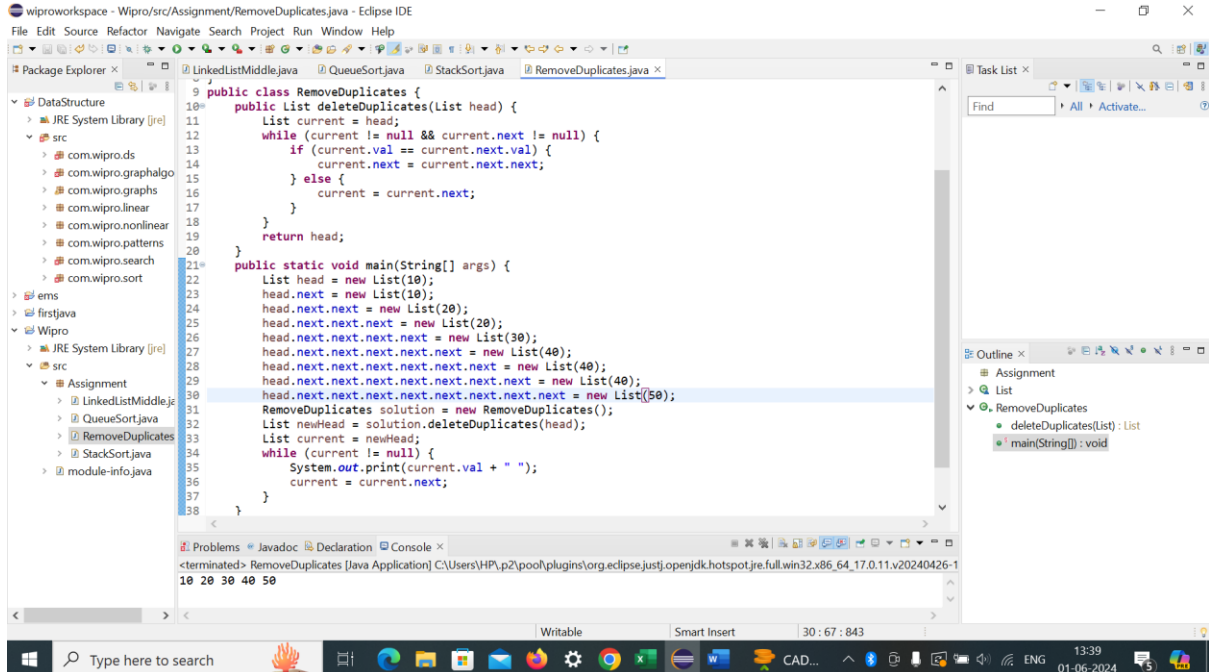
```
1 package Assignment;
2 import java.util.Stack;
3 public class StackSort {
4     public static void sortStack(Stack<Integer> stack) {
5         Stack<Integer> tempStack = new Stack<>();
6         while (!stack.isEmpty()) {
7             int tmp = stack.pop();
8             while (!tempStack.isEmpty() && tempStack.peek() > tmp) {
9                 stack.push(tempStack.pop());
10            }
11            tempStack.push(tmp);
12        }
13        while (!tempStack.isEmpty()) {
14            stack.push(tempStack.pop());
15        }
16    }
17    public static void main(String[] args) {
18        Stack<Integer> stack = new Stack<>();
19        stack.push(8);
20        stack.push(9);
21        stack.push(4);
22        stack.push(12);
23        stack.push(4);
24        stack.push(7);
25        stack.push(1);
26        stack.push(2);
27        stack.push(9);
28        stack.push(6);
29        sortStack(stack);
30        System.out.println(stack);
31    }
32 }
```

Console Output:

```
<terminated> StackSort [Java Application] C:\Users\HP\p2\poo\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\l
[12, 9, 9, 8, 8, 7, 6, 4, 4, 2, 1]
```

Task 5: Removing Duplicates from a Sorted Linked List

A sorted linked list has been constructed with repeated elements. Describe an algorithm to remove all duplicates from the linked list efficiently.



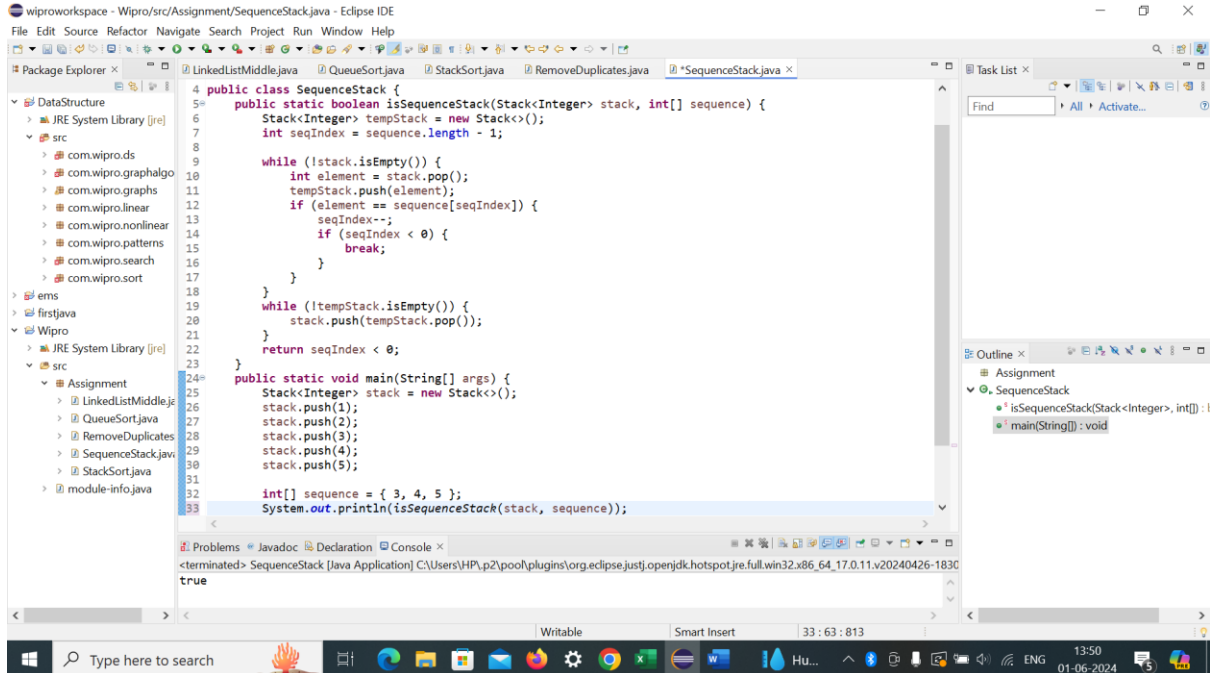
The screenshot shows the Eclipse IDE with a project named 'wiproworkspace'. The package explorer on the left shows a package 'src' containing several files, including 'RemoveDuplicates.java'. The main editor displays the code for 'RemoveDuplicates.java'. The code defines a 'List' class with 'val' and 'next' attributes. The 'deleteDuplicates' method uses a while loop to traverse the list and remove duplicates by skipping nodes with the same value as the previous node. The 'main' method creates a sorted linked list with values 10, 20, 30, 40, and 50, and then calls 'deleteDuplicates' to remove duplicates. The console at the bottom shows the output: '10 20 30 40 50'.

```
9 public class RemoveDuplicates {
10     public List deleteDuplicates(List head) {
11         List current = head;
12         while (current != null && current.next != null) {
13             if (current.val == current.next.val) {
14                 current.next = current.next.next;
15             } else {
16                 current = current.next;
17             }
18         }
19         return head;
20     }
21
22     public static void main(String[] args) {
23         List head = new List(10);
24         head.next = new List(10);
25         head.next.next = new List(20);
26         head.next.next.next = new List(20);
27         head.next.next.next.next = new List(30);
28         head.next.next.next.next.next = new List(40);
29         head.next.next.next.next.next.next = new List(40);
30         head.next.next.next.next.next.next.next = new List(50);
31         RemoveDuplicates solution = new RemoveDuplicates();
32         List newHead = solution.deleteDuplicates(head);
33         List current = newHead;
34         while (current != null) {
35             System.out.print(current.val + " ");
36             current = current.next;
37         }
38     }
39 }
```

Console Output: 10 20 30 40 50

Task 6: Searching for a Sequence in a Stack

Given a stack and a smaller array representing a sequence, write a function that determines if the sequence is present in the stack. Consider the sequence present if, upon popping the elements, all elements of the array appear consecutively in the stack



The screenshot shows the Eclipse IDE with a project named 'wiproworkspace'. The Package Explorer on the left shows a package 'com.wipro.search' containing several Java files, including 'SequenceStack.java'. The main editor displays the code for 'SequenceStack.java'. The code defines a 'Stack<Integer>' and a 'tempStack' to check if a given 'sequence' is present in the stack by popping elements and comparing them to the sequence. The 'main' method demonstrates this with a stack containing [1, 2, 3, 4, 5] and a sequence [3, 4, 5]. The console output shows 'true'.

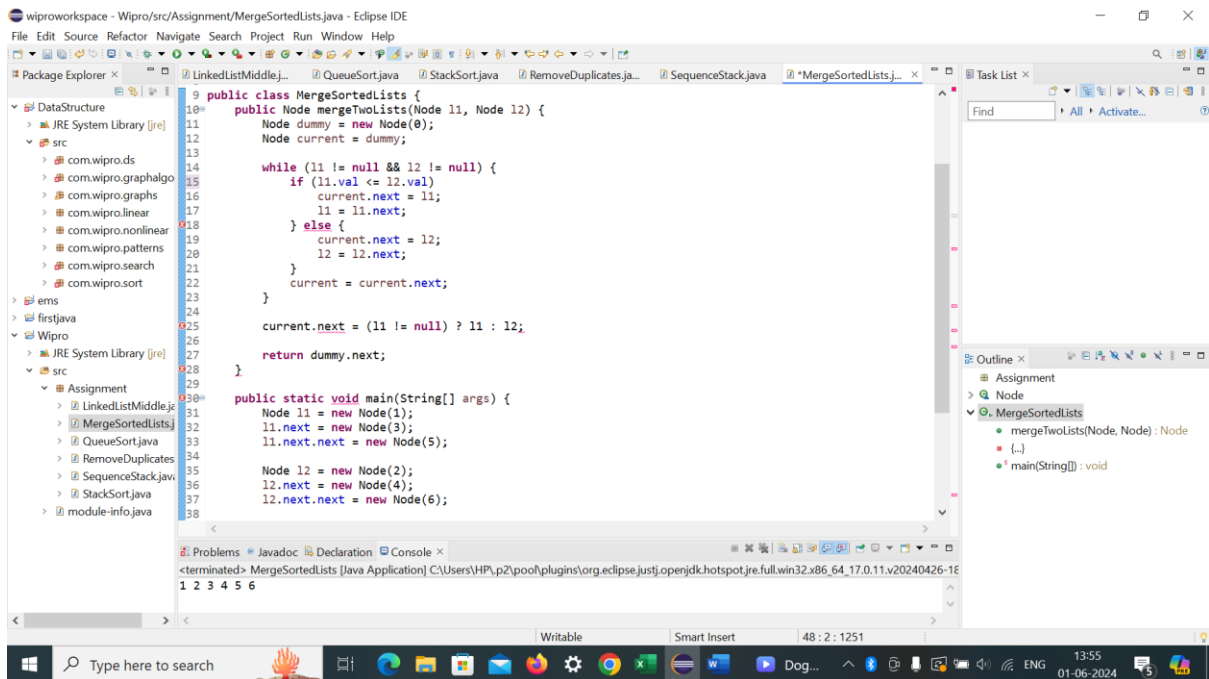
```
4 public class SequenceStack {
5     public static boolean isSequenceStack(Stack<Integer> stack, int[] sequence) {
6         Stack<Integer> tempStack = new Stack<>();
7         int seqIndex = sequence.length - 1;
8
9         while (!stack.isEmpty()) {
10             int element = stack.pop();
11             tempStack.push(element);
12             if (element == sequence[seqIndex]) {
13                 seqIndex--;
14                 if (seqIndex < 0) {
15                     break;
16                 }
17             }
18         }
19         while (!tempStack.isEmpty()) {
20             stack.push(tempStack.pop());
21         }
22         return seqIndex < 0;
23     }
24     public static void main(String[] args) {
25         Stack<Integer> stack = new Stack<>();
26         stack.push(1);
27         stack.push(2);
28         stack.push(3);
29         stack.push(4);
30         stack.push(5);
31
32         int[] sequence = { 3, 4, 5 };
33         System.out.println(isSequenceStack(stack, sequence));
34     }
35 }
```

Console Output:

```
<terminated> SequenceStack [Java Application] C:\Users\HP\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830
true
```

Task 7: Merging Two Sorted Linked Lists

You are provided with the heads of two sorted linked lists. The lists are sorted in ascending order. Create a merged linked list in ascending order from the two input lists without using any extra space (i.e., do not create any new nodes).

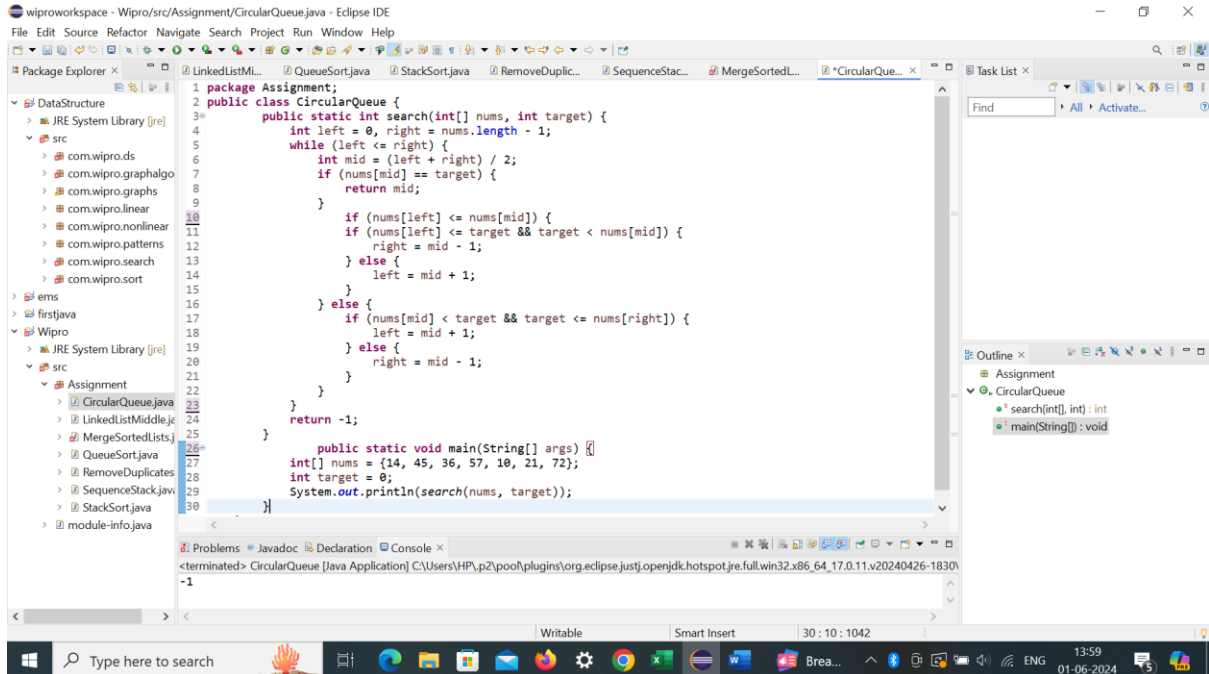


```
9 public class MergeSortedLists {
10     public Node mergeTwoLists(Node l1, Node l2) {
11         Node dummy = new Node(0);
12         Node current = dummy;
13
14         while (l1 != null && l2 != null) {
15             if (l1.val <= l2.val) {
16                 current.next = l1;
17                 l1 = l1.next;
18             } else {
19                 current.next = l2;
20                 l2 = l2.next;
21             }
22             current = current.next;
23         }
24         current.next = (l1 != null) ? l1 : l2;
25         return dummy.next;
26     }
27
28     public static void main(String[] args) {
29         Node l1 = new Node(1);
30         l1.next = new Node(3);
31         l1.next.next = new Node(5);
32
33         Node l2 = new Node(2);
34         l2.next = new Node(4);
35         l2.next.next = new Node(6);
36
37         // ... (rest of the main method code)
38     }
39 }
```

1 2 3 4 5 6

Task 8: Circular Queue Binary Search

Consider a circular queue (implemented using a fixed-size array) where the elements are sorted but have been rotated at an unknown index. Describe an approach to perform a binary search for a given element within this circular queue.



The screenshot shows the Eclipse IDE with a Java project named 'wiproworkspace'. The package explorer on the left shows the project structure, including a 'src' folder with several Java files. The main editor displays the 'CircularQueue.java' file, which contains the following code:

```
1 package Assignment;
2 public class CircularQueue {
3     public static int search(int[] nums, int target) {
4         int left = 0, right = nums.length - 1;
5         while (left <= right) {
6             int mid = (left + right) / 2;
7             if (nums[mid] == target) {
8                 return mid;
9             }
10            if (nums[left] <= nums[mid]) {
11                if (nums[left] <= target && target < nums[mid]) {
12                    right = mid - 1;
13                } else {
14                    left = mid + 1;
15                }
16            } else {
17                if (nums[mid] < target && target <= nums[right]) {
18                    left = mid + 1;
19                } else {
20                    right = mid - 1;
21                }
22            }
23        }
24        return -1;
25    }
26    public static void main(String[] args) {
27        int[] nums = {14, 45, 36, 57, 10, 21, 72};
28        int target = 0;
29        System.out.println(search(nums, target));
30    }
31 }
```

The console at the bottom shows the output of the program: "14". The task list on the right shows the current task: "CircularQueue".