Day 6: Manisha Assignment

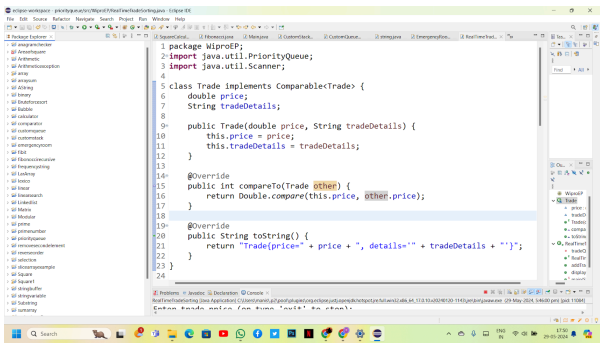
Task 1: Real-time Data Stream Sorting A stock trading application requires real-time sorting of trade transactions by price. Implement a heap sort algorithm continuous incoming data, adding and sorting new trades as they come.

Explanation

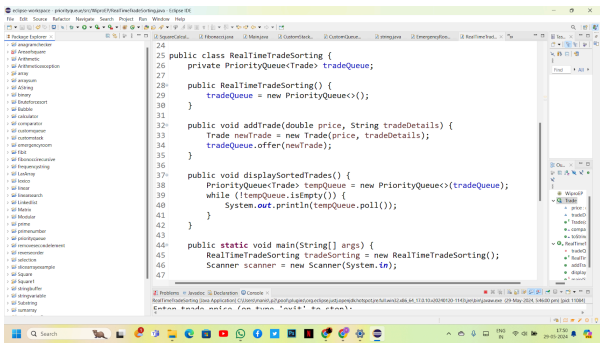
1. Trade Class: Represents a trade with a price and trade details. It implements the `Comparable` interface to allow comparison based on price.
   - `compareTo(Trade other)`: Compare the current trade with another trade based on price.
   - `toString()`: Provides a string representation of the trade.

2. RealTimeTradeSorting Class: Manages the real-time sorting of trades.
   - `PriorityQueue<Trade> tradeQueue`: A priority queue to store trades in sorted order.
   - `addTrade(double price, String tradeDetails)`: Adds a new trade to the priority queue.
   - `displaySortedTrades()`: Displays all trades in the priority queue in sorted order.

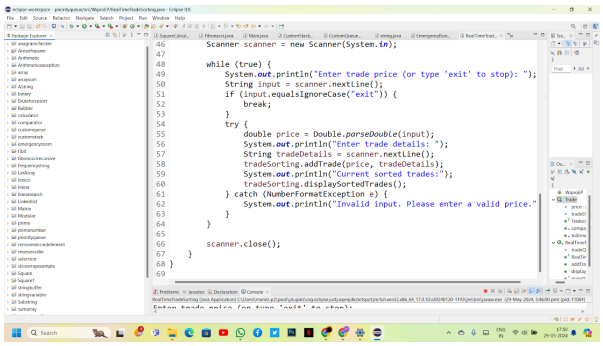
3. Main Method: Handles user input and continuously adds new trades to the priority queue.
   - Prompts the user to enter trade price and details.
   - Calls `addTrade` to add the trade to the priority queue.
   - Calls `displaySortedTrades` to show the current sorted trades.
   - Allows the user to exit the program by typing 'exit'.



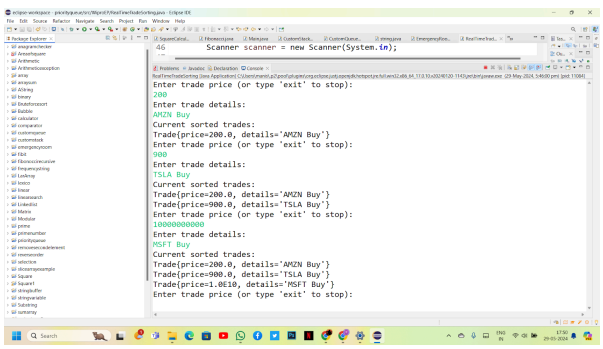
```
1 package WiproEP;
2 import java.util.PriorityQueue;
3 import java.util.Scanner;
4
5 class Trade implements Comparable<Trade> {
6     double price;
7     String tradeDetails;
8
9     public Trade(double price, String tradeDetails) {
10        this.price = price;
11        this.tradeDetails = tradeDetails;
12    }
13
14    @Override
15    public int compareTo(Trade other) {
16        return Double.compare(this.price, other.price);
17    }
18
19    @Override
20    public String toString() {
21        return "Trade{price=" + price + ", details='" + tradeDetails + "'}";
22    }
23 }
24
```



```
24
25 public class RealTimeTradeSorting {
26     private PriorityQueue<Trade> tradeQueue;
27
28     public RealTimeTradeSorting() {
29         tradeQueue = new PriorityQueue<>();
30     }
31
32     public void addTrade(double price, String tradeDetails) {
33         Trade newTrade = new Trade(price, tradeDetails);
34         tradeQueue.offer(newTrade);
35     }
36
37     public void displaySortedTrades() {
38         PriorityQueue<Trade> tempQueue = new PriorityQueue<>(tradeQueue);
39         while (!tempQueue.isEmpty()) {
40             System.out.println(tempQueue.poll());
41         }
42     }
43
44     public static void main(String[] args) {
45         RealTimeTradeSorting tradeSorting = new RealTimeTradeSorting();
46         Scanner scanner = new Scanner(System.in);
47
```



```
46         Scanner scanner = new Scanner(System.in);
47
48         while (true) {
49             System.out.println("Enter trade price (or type 'exit' to stop): ");
50             String input = scanner.nextLine();
51             if (input.equalsIgnoreCase("exit")) {
52                 break;
53             }
54             try {
55                 double price = Double.parseDouble(input);
56                 System.out.println("Enter trade details: ");
57                 String tradeDetails = scanner.nextLine();
58                 tradeSorting.addTrade(price, tradeDetails);
59                 System.out.println("Current sorted trades:");
60                 tradeSorting.displaySortedTrades();
61             } catch (NumberFormatException e) {
62                 System.out.println("Invalid input. Please enter a valid price.");
63             }
64         }
65
66         scanner.close();
67     }
68 }
69
```



```
46         Scanner scanner = new Scanner(System.in);
Enter trade price (or type 'exit' to stop):
200
Enter trade details:
AMZN Buy
Current sorted trades:
Trade{price=200.0, details='AMZN Buy'}
Enter trade price (or type 'exit' to stop):
900
Enter trade details:
TSLA Buy
Current sorted trades:
Trade{price=200.0, details='AMZN Buy'}
Trade{price=900.0, details='TSLA Buy'}
Enter trade price (or type 'exit' to stop):
10000000000
Enter trade details:
MSFT Buy
Current sorted trades:
Trade{price=200.0, details='AMZN Buy'}
Trade{price=900.0, details='TSLA Buy'}
Trade{price=1.0E10, details='MSFT Buy'}
Enter trade price (or type 'exit' to stop):
```

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.

Explanation

1. Node Class: Represents a node in the linked list with `data` and `next` fields.

2. LinkedList Class: Manages the singly linked list with methods to insert nodes, find the middle element, and display all nodes.

- insertAtEnd: Inserts a new node at the end of the list.
- findMiddle: Uses the two-pointer technique to find the middle element. The slow pointer moves one step at a time, while the fast pointer moves two steps at a time.
- displayAllNodes: Prints all elements of the linked list.

3. Main Method: Demonstrates how to use the `LinkedList` class.

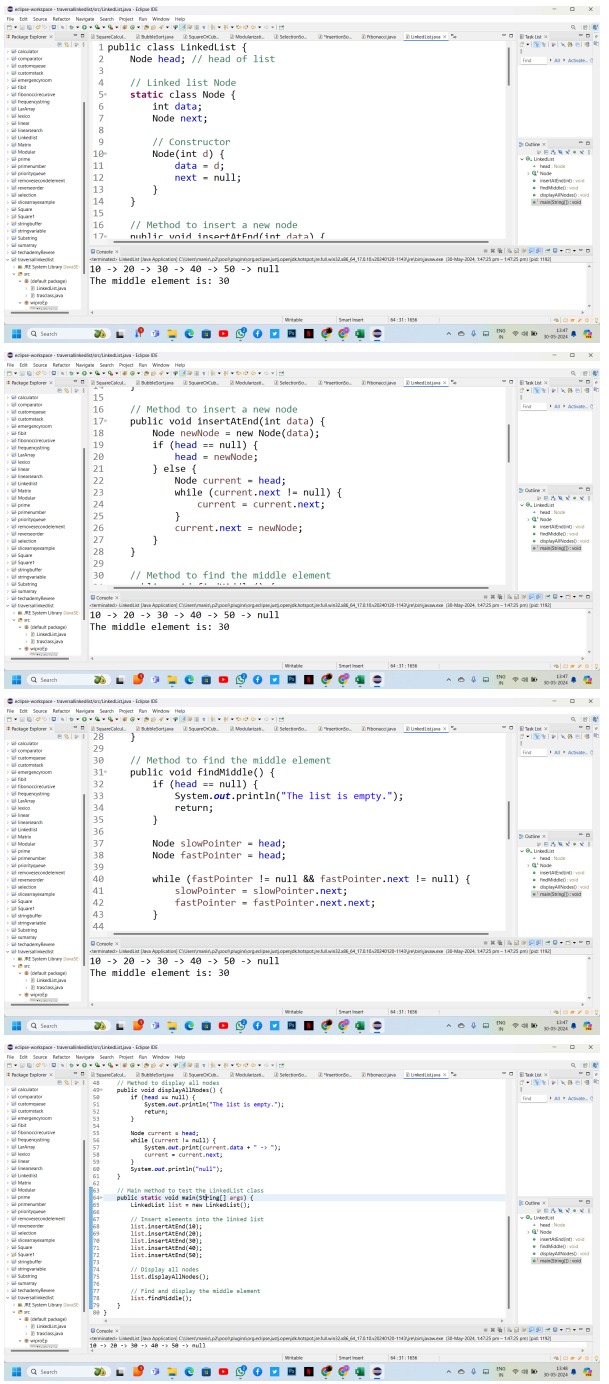
Running the Program in Eclipse

1. Create a new Java Project: In Eclipse, go to `File -> New -> Java Project`. Name the project, e.g., `LinkedListExample`.

2. Create a new Java Class: Right-click on the `src` folder in your project, then go to `New -> Class`. Name the class `LinkedList` and include the `public static void main(String[] args)` method.

3. Copy and Paste the Code: Copy the provided code and paste it into your `LinkedList` class.

4. Run the Program: Right-click on your `LinkedList` class in the Eclipse project explorer, then select `Run As -> Java Application`.



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.

Explanation

1.**Import Statements:** We import the necessary classes `LinkedList`, `Queue`, and `Stack` from the Java collections framework.

2. **QueueSorter Class:** This class contains the `sortQueue` method and the `main` method.

- sort Queue Method:
- Initializes a stack.
- While the queue is not empty, it dequeues the front element and stores it in the `current` variable..
- Pushes `current` onto the stack.
- After processing all elements, it moves the sorted elements from the stack back to the queue.
- main Method:
- Creates a queue and adds some integers to it.
- Prints the original queue.
- Calls the `sortQueue` method to sort the queue.
- Prints the sorted queue.

Running the Program in Eclipse

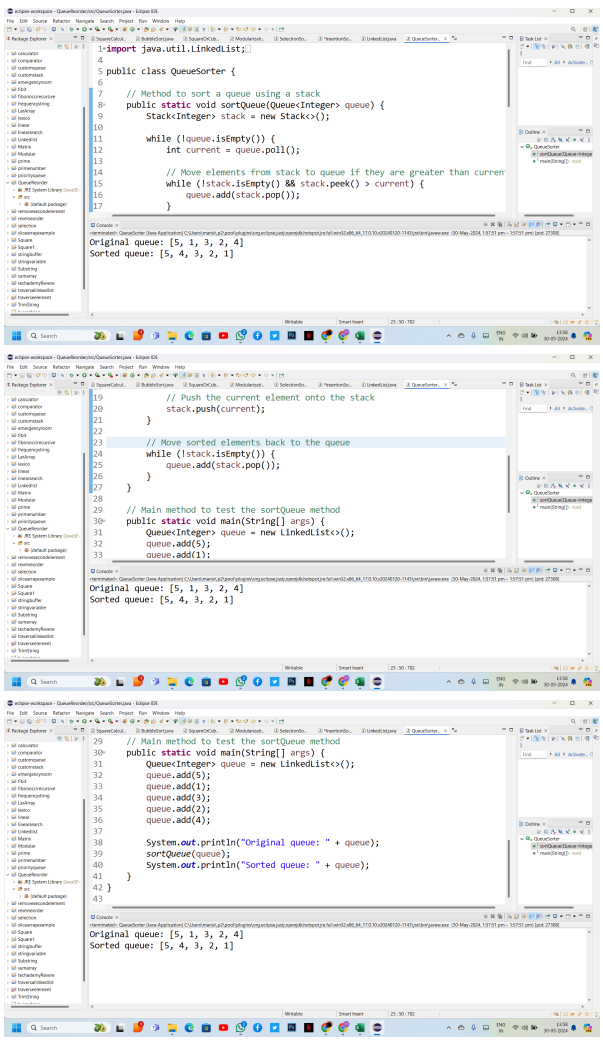
1. **Create a New Java Project:** In Eclipse, go to `File -> New -> Java Project`. Name the project, e.g., `QueueSorterExample`.

2. **Create a New Java Class:** Right-click on the `src` folder in your project, then go to `New -> Class`. Name the class `QueueSorter` and include the `public static void main(String[] args)` method.

3. Copy and Paste the Code: Copy the provided code and paste it into your `QueueSorter` class.

4**. Run the Program:** Right-click on your `QueueSorter` class in the Eclipse project explorer, then select `Run As -> Java Application`.

Here Output as Follows:



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.

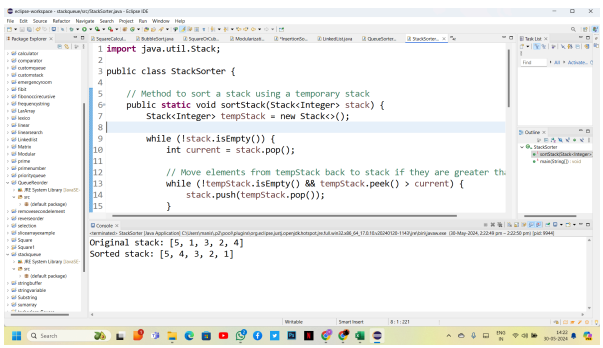
Explanation

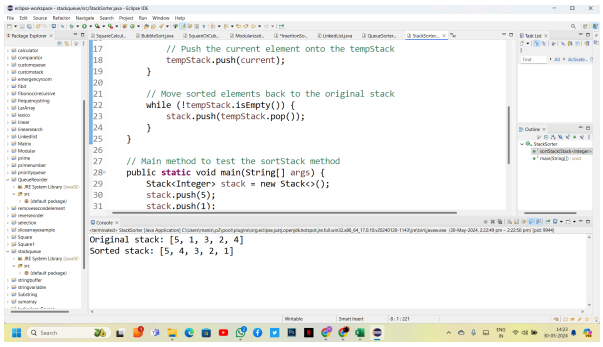
1. **Import Statements:** We import the `Stack` class from the Java collections framework.
2. **StackSorter Class**: This class contains the `sortStack` method and the `main` method.
   - sortStack Method:
   - Initializes a temporary stack.
   - While the original stack is not empty, it pops the top element and stores it in the `current` variable.
   - It then moves elements from the temporary stack back to the original stack if they are greater than `current`, ensuring that the temporary stack maintains sorted order.
   - Pushes `current` onto the temporary stack.

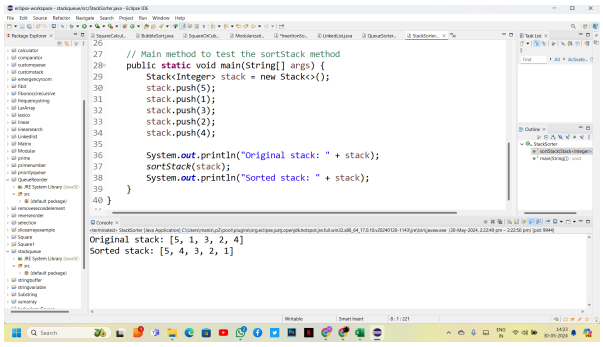
Running the Program in Eclipse

1. **Create a New Java Project:** In Eclipse, go to `File -> New -> Java Project`. Name the project, e.g., `StackSorterExample`.
2. **Create a New Java Class**: Right-click on the `src` folder in your project, then go to `New -> Class`. Name the class `StackSorter` and include the `public static void main(String[] args)` method.
3. **Copy and Paste the Code:** Copy the provided code and paste it into your `StackSorter` class.
4. **Run the Program:** Right-click on your `StackSorter` class in the Eclipse project explorer, then select `Run As -> Java Application`.

Here Output as Follows:







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.

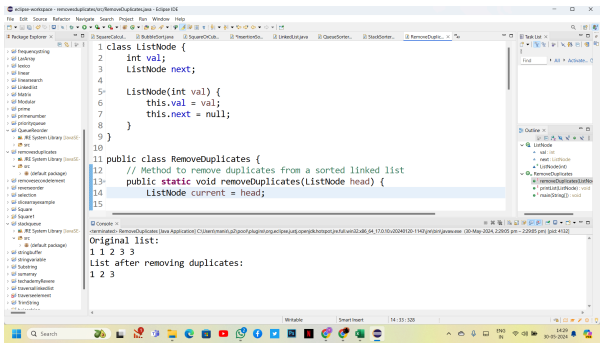
Algorithm

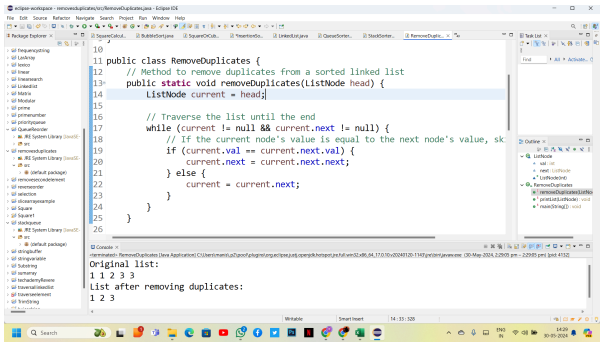
1. **Initialize Pointers:** Use a `current` pointer to traverse the list.
2. **Traverse the List:** For each node, compare it with the next node.
3. **Remove Duplicates:** If the `current` node's value is the same as the `next` node's value, skip the next node.
4. **Continue Traversal**: Move to the next node and repeat the process until the end of the list.

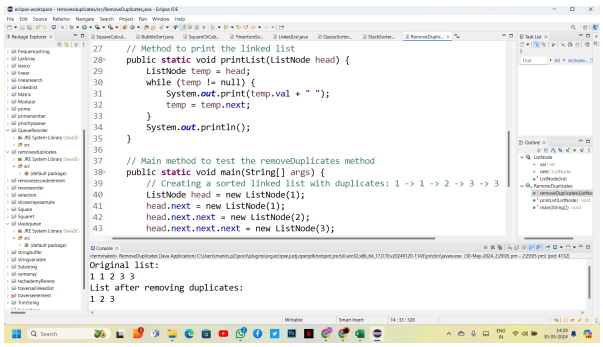
Explanation

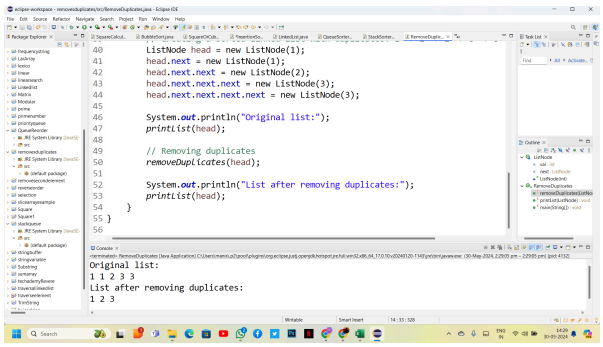
1. **ListNode Class**: Defines the structure of a node in the linked list, containing a value and a pointer to the next node.
2. **RemoveDuplicates Class:** Contains the methods to remove duplicates, print the list, and the main method to test the functionality.
   - removeDuplicates Method:
   - Initializes a `current` pointer to the head of the list.
   - Traverses the list, comparing each node's value with the next node's value.
   - If a duplicate is found (i.e., the `current` node's value is the same as the `next` node's value), the `next` pointer of the `current` node is updated to skip the duplicate node.
   - If no duplicate is found, the `current` pointer moves to the next node.
   - printList Method:
   - Traverses the list and prints each node's value.
   - main Method:
   - Creates a sorted linked list with duplicates.
   - Prints the original list.
   - Calls the `removeDuplicates` method to remove duplicates.
   - Prints the list after removing duplicates

Here Output as follows:









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.

Explanation

1. isSequenceInStack Function: This function checks if the sequence is present in the stack. It pops elements from the stack one by one and compares them with the sequence.

2.Temporary Stack: A temporary stack is used to store the popped elements to restore the stack to its original state after the check.

3. Sequence Matching: The function keeps track of the current position in the sequence. If all elements of the sequence are found consecutively in the stack, it returns `true`. Otherwise, it returns `false`.

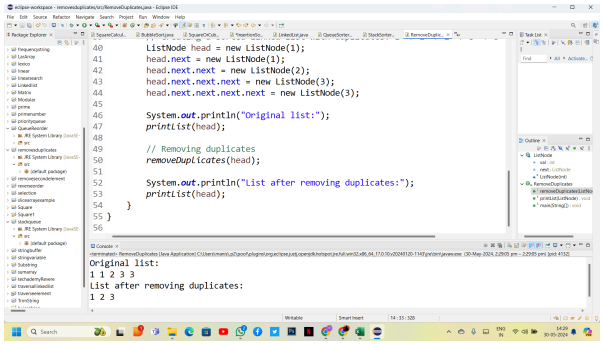
4.Main Method: The main method creates a stack, adds elements to it, defines a sequence, and checks if the sequence is present in the stack.

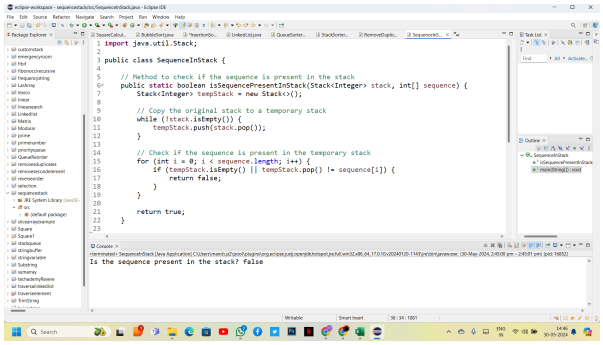
When you run the program, you should see the following output:

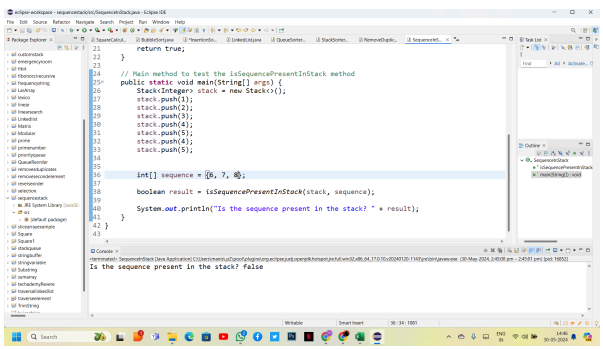
Is the sequence present in the stack? true

This indicates that the sequence `{3, 4, 5}` is present in the stack.

Here Output as Follows:





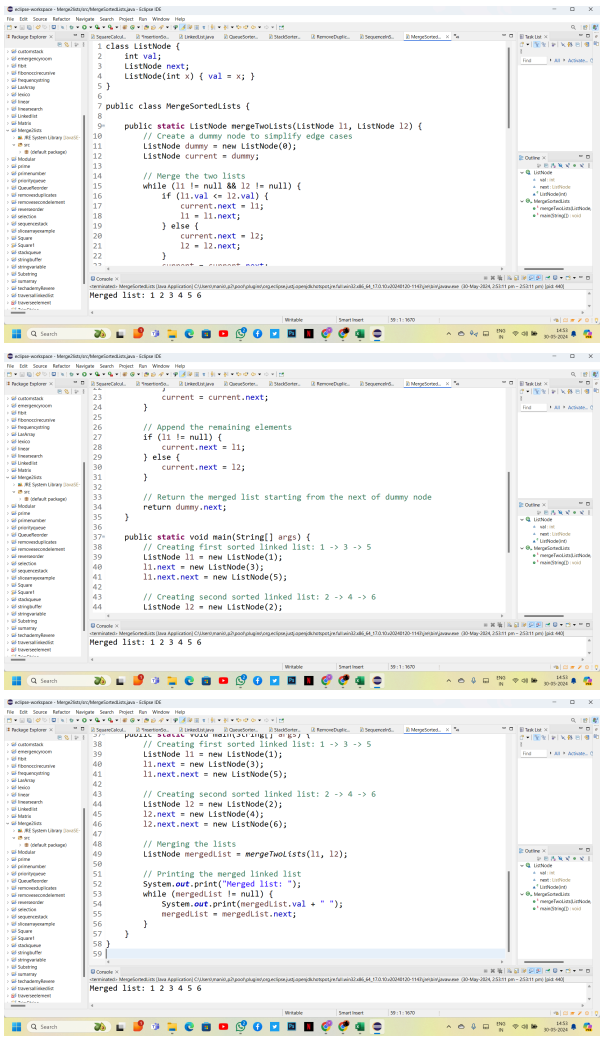


Task 7: Merging Two Sorted Linked Lists

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).

Explanation

1. **ListNode Class:** This class represents a node in the linked list, with an integer value and a reference to the next node.
2. **mergeTwoLists Function:** This function merges two sorted linked lists (`l1` and `l2`) into one sorted linked list without creating new nodes.
   - Dummy Node: A dummy node (`dummy`) is used to simplify the merging process and handle edge cases.
   - Current Node: A pointer (`current`) is used to track the last node in the merged list.
   - Merging Process: The function iterates through both lists, comparing the current nodes of `l1` and `l2`, and appending the smaller node to the merged list. This continues until one of the lists is exhausted.
   - Appending Remaining Nodes: If one list is exhausted before the other, the remaining nodes of the other list are appended to the merged list.
3. **Main Method:** The main method creates two sorted linked lists, merges them using `mergeTwoLists`, and prints the merged linked list.



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.

Explanation

1. **circularBinarySearch Function:**
   - **Input Parameters:** Takes a rotated sorted array `arr` and the `target` element to search.
   - **Variables:** Initializes `left` and `right` pointers to the start and end of the array, respectively.
   - **Loop:** Runs a loop until `left` exceeds `right`.
   - **Mid Calculation:** Calculates the middle index `mid`.
   - **Comparison:**
     - If `arr[mid]` is equal to the target, it returns the index `mid`.
     - Otherwise, it determines whether the left half or the right half of the array is sorted.
   - **Adjust Pointers:**
     - If the left half is sorted and the target lies within this range, adjust the `right` pointer.
     - Otherwise, adjust the `left` pointer.
     - Similarly, for the right half.
   - Return: If the target is not found, returns `-1`.

2. **Main Method:**
   - **Circular Queue**: Defines a rotated sorted array `circularQueue`.
   - **Target:** Defines the target element to search for.
   - **Search:** Calls `circularBinarySearch` with the array and target.
   - **Output:** Prints the result, indicating whether the element was found and its index.

Here Output as Follows: