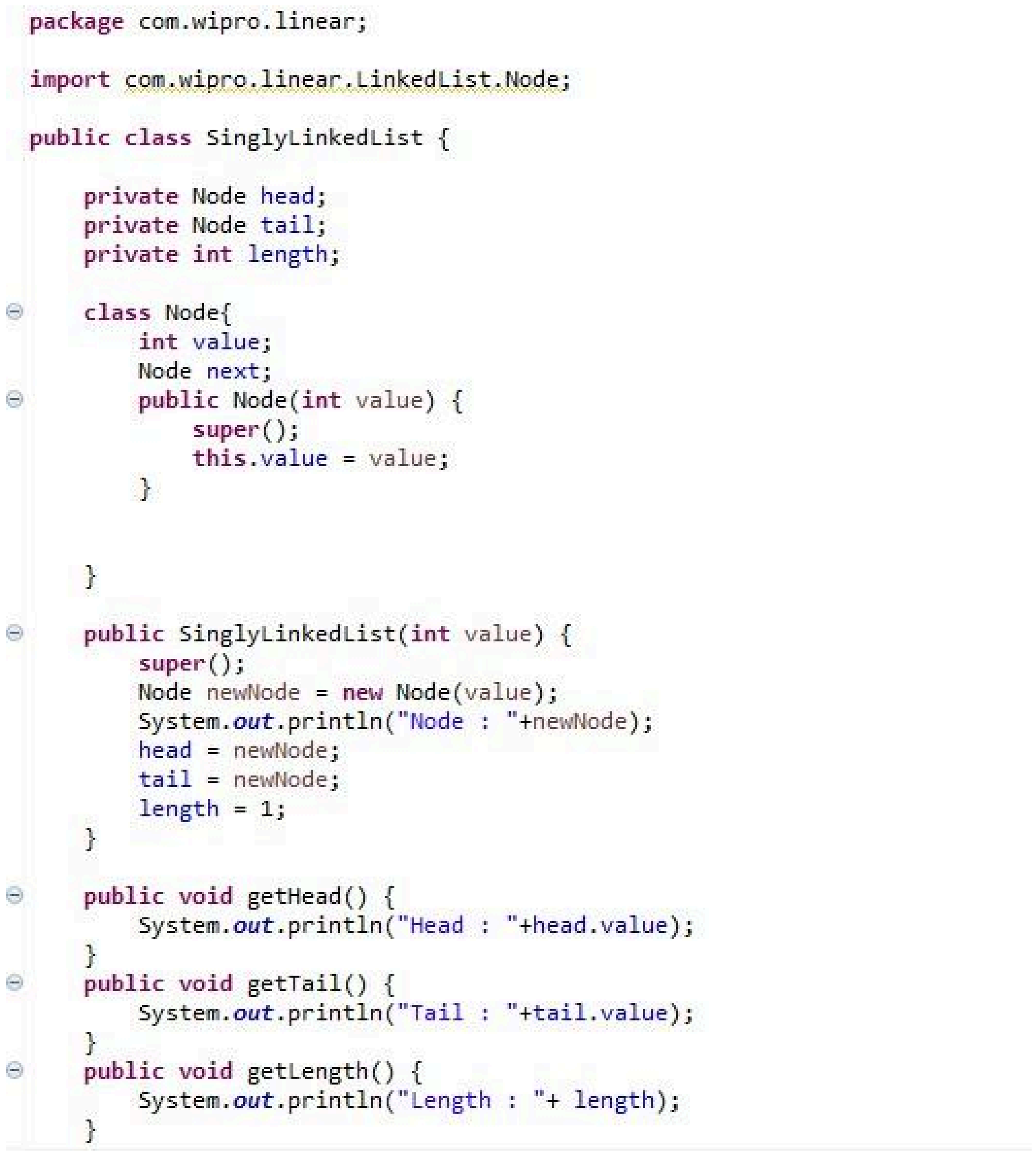
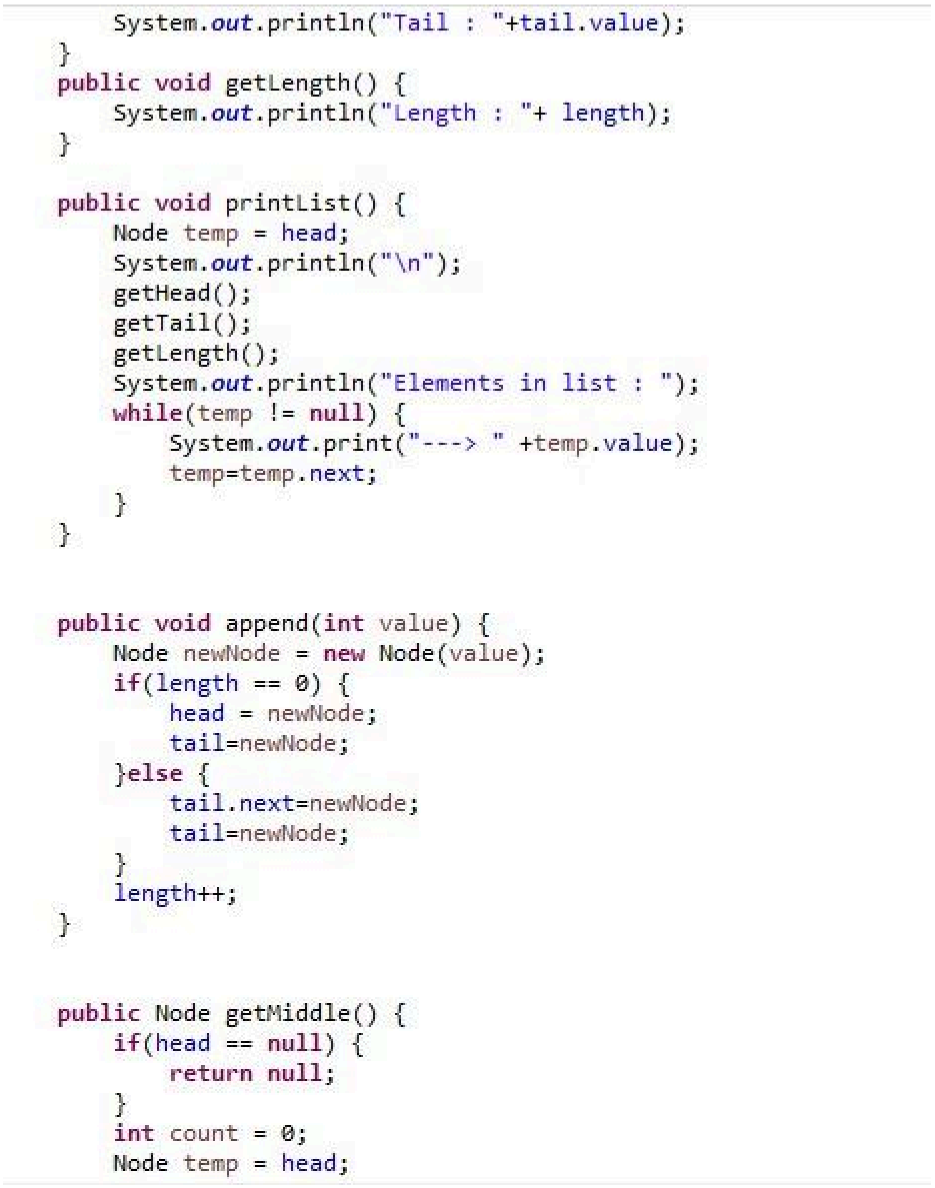
Day 1 to 6

1] 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.

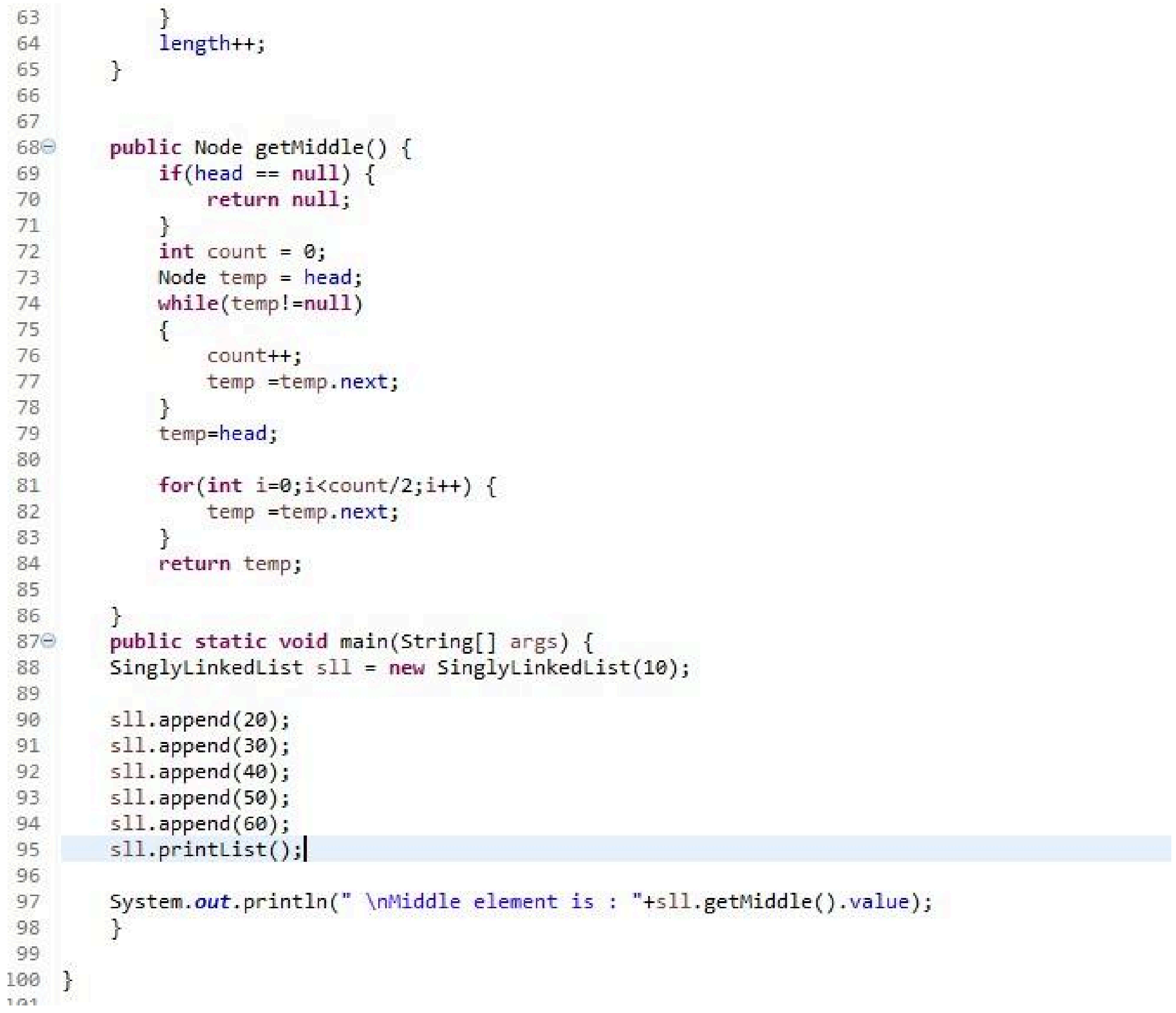
Solution:-



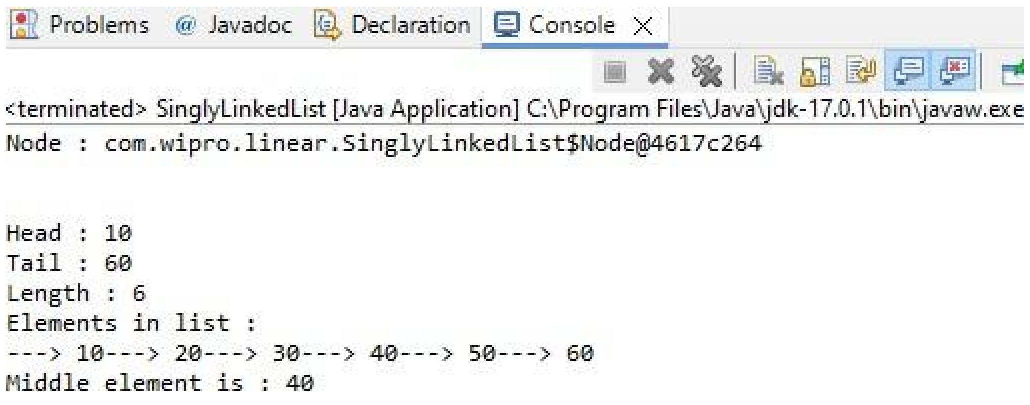
```java
package com.wipro.linear;

import com.wipro.linear.LinkedList.Node;

public class SinglyLinkedList {

    private Node head;
    private Node tail;
    private int length;

    class Node{
        int value;
        Node next;
        public Node(int value) {
            super();
            this.value = value;
        }


    }

    public SinglyLinkedList(int value) {
        super();
        Node newNode = new Node(value);
        System.out.println("Node : "+newNode);
        head = newNode;
        tail = newNode;
        length = 1;
    }

    public void getHead() {
        System.out.println("Head : "+head.value);
    }
    public void getTail() {
        System.out.println("Tail : "+tail.value);
    }
    public void getLength() {
        System.out.println("Length : "+ length);
    }
```



```java
        System.out.println("Tail : "+tail.value);
    }
    public void getLength() {
        System.out.println("Length : "+ length);
    }

    public void printList() {
        Node temp = head;
        System.out.println("\n");
        getHead();
        getTail();
        getLength();
        System.out.println("Elements in list : ");
        while(temp != null) {
            System.out.print("---> " +temp.value);
            temp=temp.next;
        }
    }


    public void append(int value) {
        Node newNode = new Node(value);
        if(length == 0) {
            head = newNode;
            tail=newNode;
        }else {
            tail.next=newNode;
            tail=newNode;
        }
        length++;
    }


    public Node getMiddle() {
        if(head == null) {
            return null;
        }
        int count = 0;
        Node temp = head;
```


```
        }
        length++;
    }


    public Node getMiddle() {
        if(head == null) {
            return null;
        }
        int count = 0;
        Node temp = head;
        while(temp!=null)
        {
            count++;
            temp =temp.next;
        }
        temp=head;

        for(int i=0;i<count/2;i++) {
            temp =temp.next;
        }
        return temp;

    }
    public static void main(String[] args) {
    SinglyLinkedList sll = new SinglyLinkedList(10);

    sll.append(20);
    sll.append(30);
    sll.append(40);
    sll.append(50);
    sll.append(60);
    sll.printList();

    System.out.println(" \nMiddle element is : "+sll.getMiddle().value);
    }

}
```

OUTPUT:-



```
<terminated> SinglyLinkedList [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe
Node : com.wipro.linear.SinglyLinkedList$Node@4617c264


Head : 10
Tail : 60
Length : 6
Elements in list :
---> 10---> 20---> 30---> 40---> 50---> 60
Middle element is : 40
```

2] 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.

Solution:-

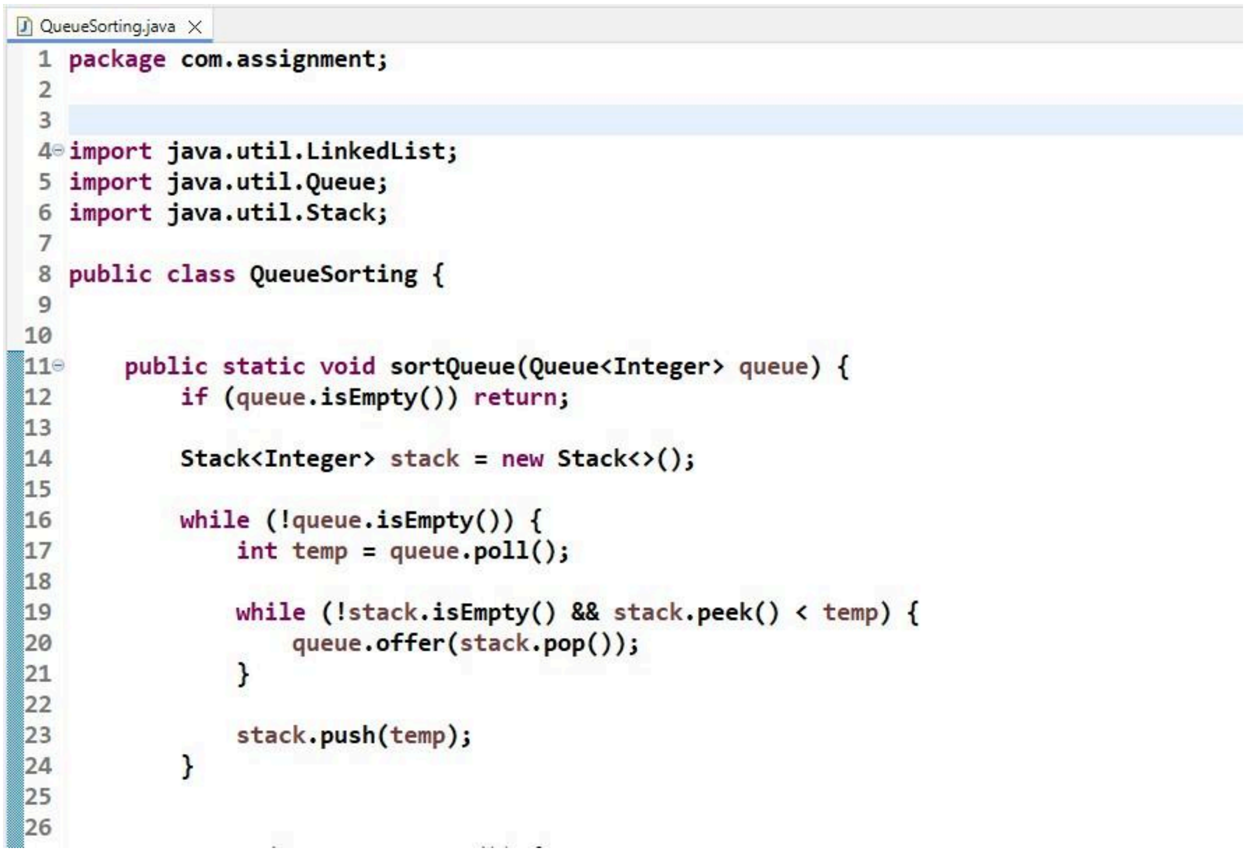
To sort a queue of integers using only one additional stack.Here is a step-by-step description of the process:

Steps:-

1. **Initialize the Queue and Stack**: Start with the queue containing the elements to be sorted and an empty stack.
2. **Find and Remove Maximum**:
   ○ Dequeue elements from the queue one by one and push them onto the stack.
   ○ Keep track of the maximum element encountered.
3. **Reinsert Elements**:
   ○ Pop elements from the stack and reinsert them into the queue.

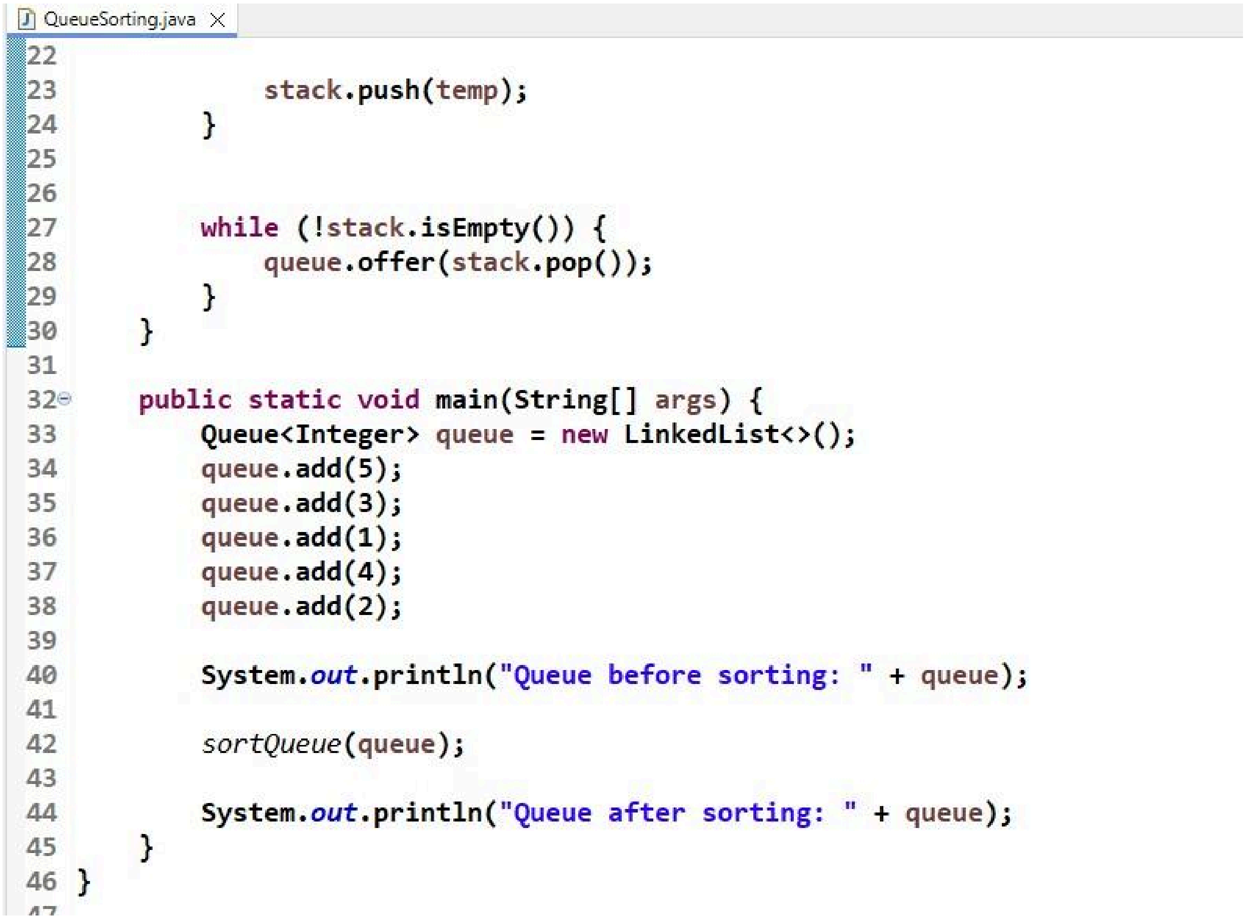
    - Skip reinserting the maximum element (since it’s the largest, it should stay out).
4. **Place Maximum in Sorted Position**:
    - After reinserting all other elements, the maximum element will be placed in its correct position.
5. **Repeat**: Repeat the process until the queue is sorted.

Code -



QueueSorting.java

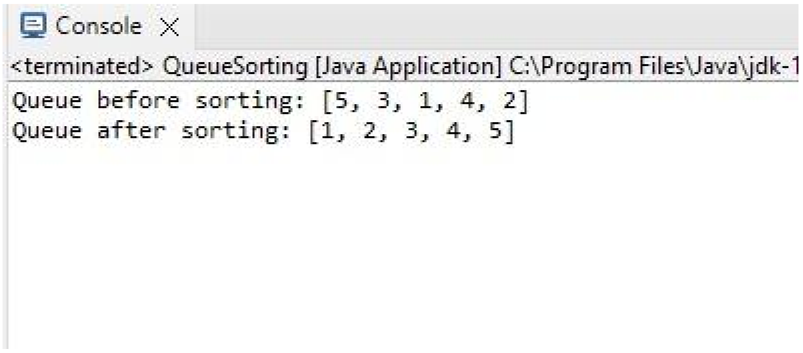
```java
package com.assignment;


import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class QueueSorting {


    public static void sortQueue(Queue<Integer> queue) {
        if (queue.isEmpty()) return;

        Stack<Integer> stack = new Stack<>();

        while (!queue.isEmpty()) {
            int temp = queue.poll();

            while (!stack.isEmpty() && stack.peek() < temp) {
                queue.offer(stack.pop());
            }

            stack.push(temp);
        }


```



QueueSorting.java

```java
            stack.push(temp);
        }


        while (!stack.isEmpty()) {
            queue.offer(stack.pop());
        }
    }

    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(5);
        queue.add(3);
        queue.add(1);
        queue.add(4);
        queue.add(2);

        System.out.println("Queue before sorting: " + queue);

        sortQueue(queue);

        System.out.println("Queue after sorting: " + queue);
    }
}
```

Output -



Console

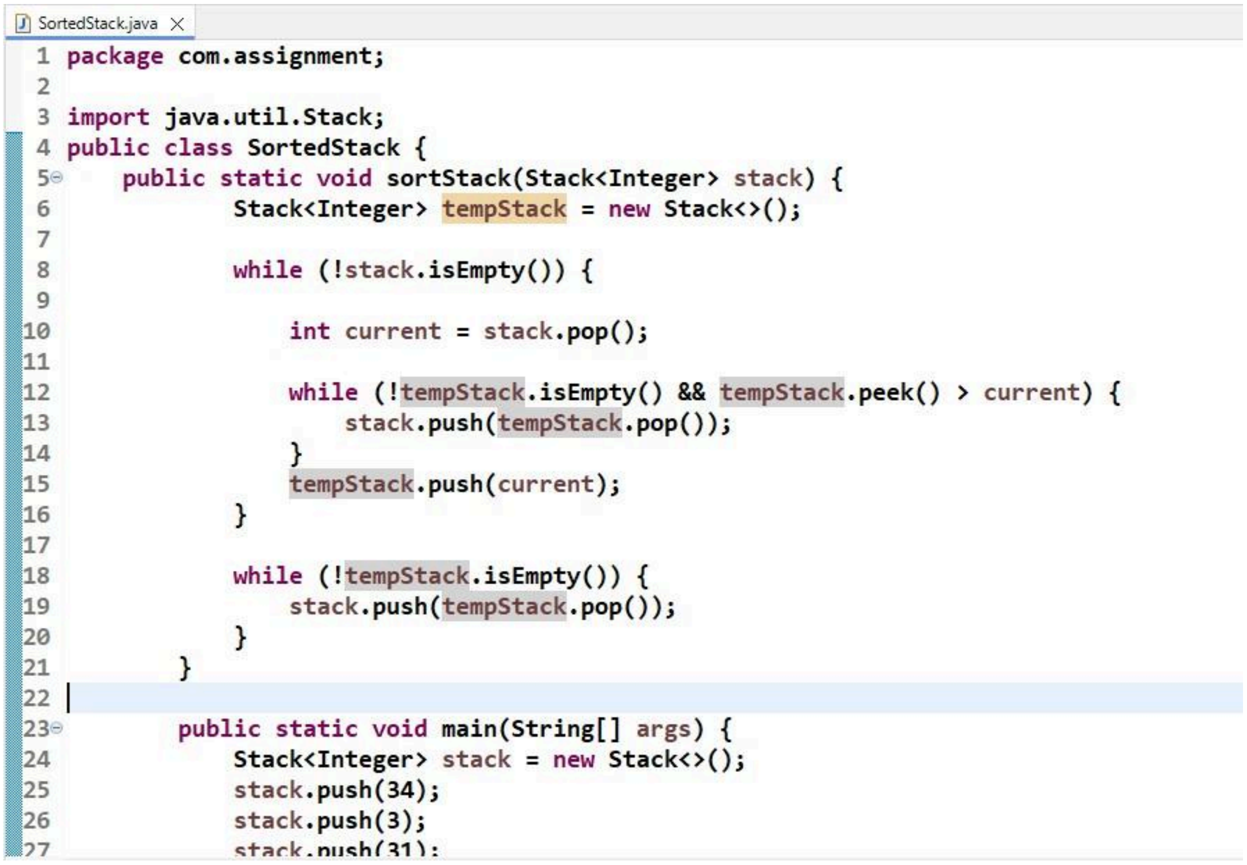
```
<terminated> QueueSorting [Java Application] C:\Program Files\Java\jdk-1
Queue before sorting: [5, 3, 1, 4, 2]
Queue after sorting: [1, 2, 3, 4, 5]
```

3] Task 4: Stack Sorting In-Place

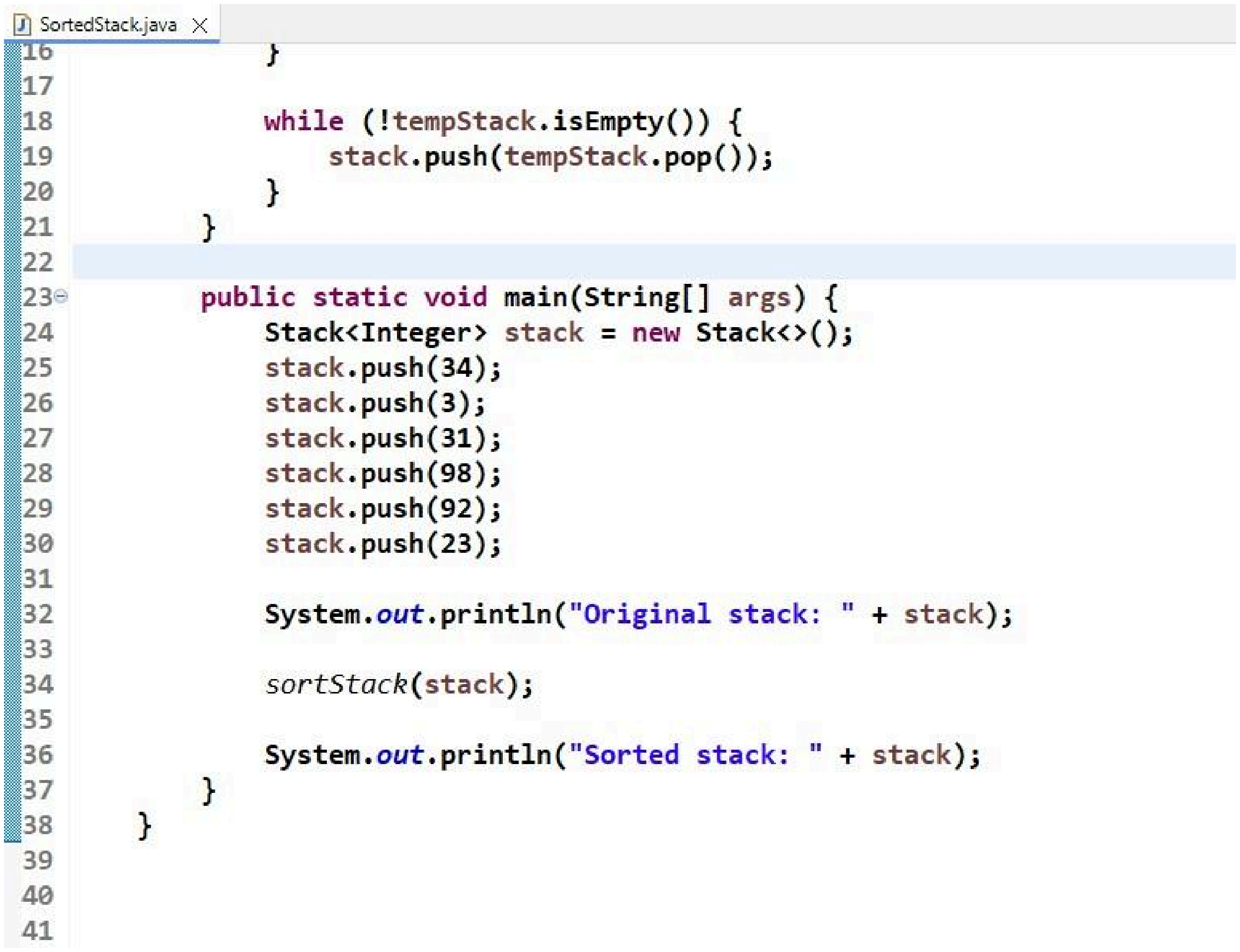
You must write a function to sort a stack such that the smallest items are on the top. You can use an additional temporary stack, but you may not copy the elements into any other data structure such as an array. The stack supports the following operations: push, pop, peek, and isEmpty.

Solution:-

Code -

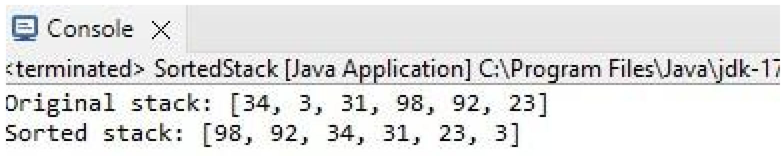

SortedStack.java

```
package com.assignment;

import java.util.Stack;
public class SortedStack {
    public static void sortStack(Stack<Integer> stack) {
            Stack<Integer> tempStack = new Stack<>();

            while (!stack.isEmpty()) {

                int current = stack.pop();

                while (!tempStack.isEmpty() && tempStack.peek() > current) {
                    stack.push(tempStack.pop());
                }
                tempStack.push(current);
            }

            while (!tempStack.isEmpty()) {
                stack.push(tempStack.pop());
            }
        }

        public static void main(String[] args) {
            Stack<Integer> stack = new Stack<>();
            stack.push(34);
            stack.push(3);
            stack.push(31);
```


SortedStack.java

```
            }

            while (!tempStack.isEmpty()) {
                stack.push(tempStack.pop());
            }
        }

        public static void main(String[] args) {
            Stack<Integer> stack = new Stack<>();
            stack.push(34);
            stack.push(3);
            stack.push(31);
            stack.push(98);
            stack.push(92);
            stack.push(23);

            System.out.println("Original stack: " + stack);

            sortStack(stack);

            System.out.println("Sorted stack: " + stack);
        }
    }
```

Output -


Console

```
<terminated> SortedStack [Java Application] C:\Program Files\Java\jdk-17
Original stack: [34, 3, 31, 98, 92, 23]
Sorted stack: [98, 92, 34, 31, 23, 3]
```

4] Task 5: Removing Duplicates from a Sorted Linked List

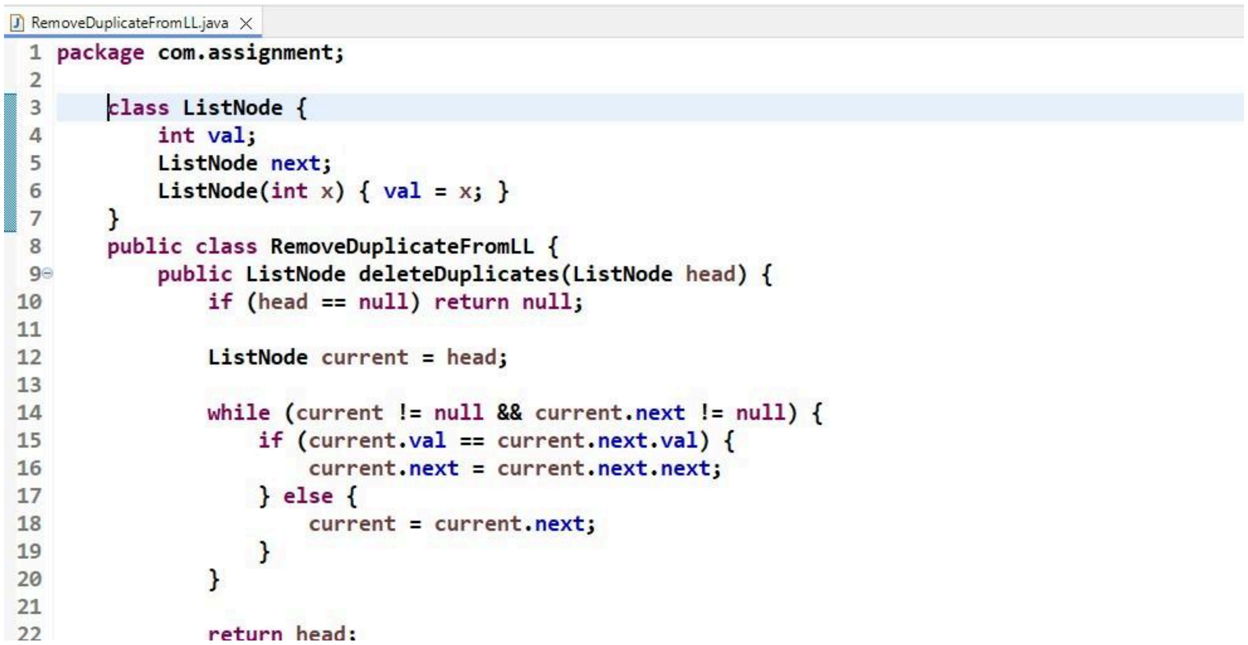
A sorted linked list has been constructed with repeated elements. Describe an algorithm to remove all duplicates from the linked list efficiently.

Solution:-

Algorithm:

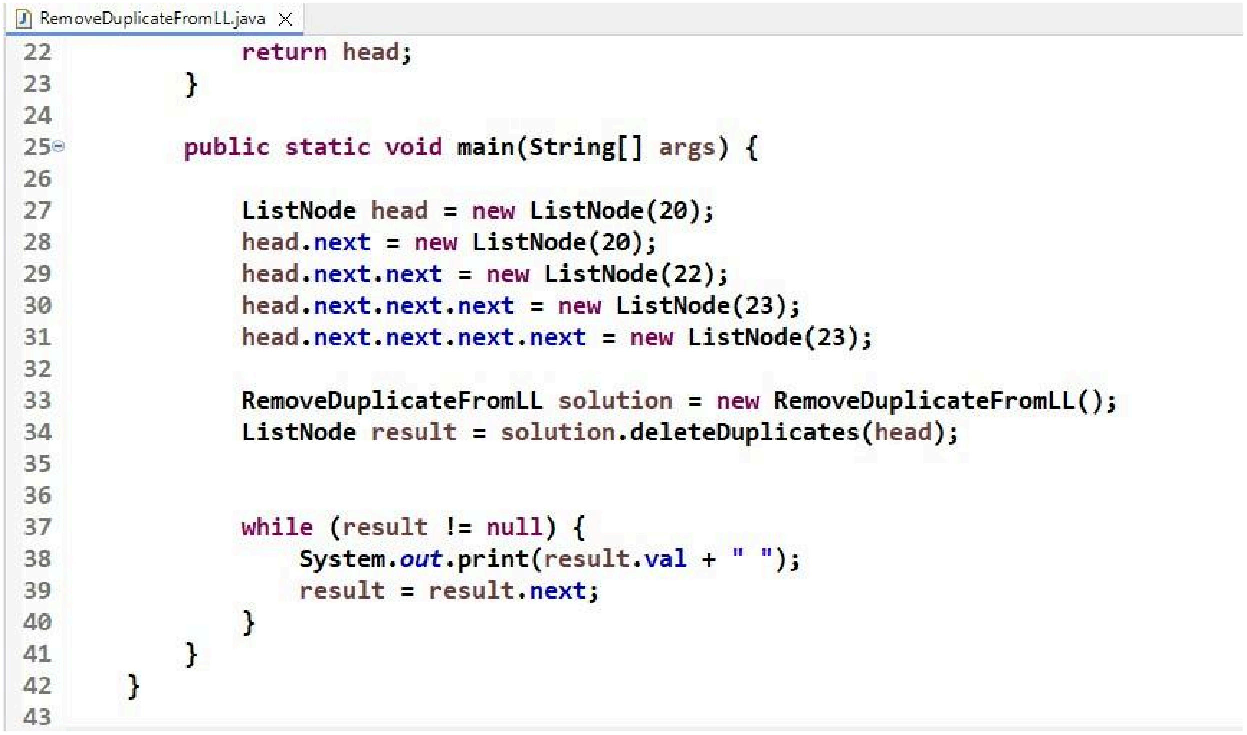
1. **Initialize a Current Pointer**: Start with the head of the linked list.
2. **Traverse the List**: Loop through the list until the end.
3. **Compare Current Node with Next Node**:
    - If the current node's value is equal to the next node's value, skip the next node by linking the current node's next pointer to the node after the next.
    - If the values are not equal, move the current pointer to the next node.
4. **Repeat**: Continue this process until the end of the list is reached.
5. **Return the Modified List**: The head of the list will point to the modified linked list without duplicates.

Code -



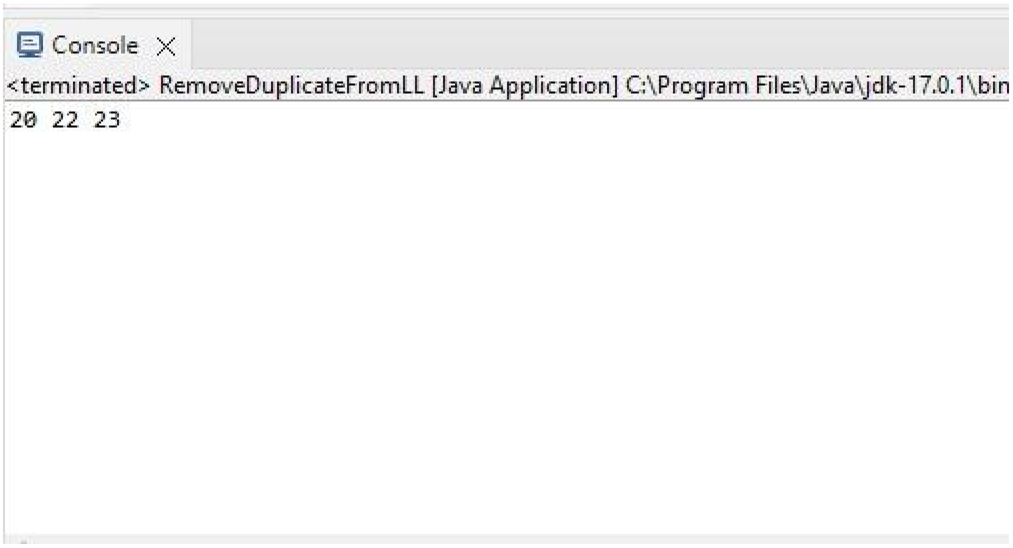
RemoveDuplicateFromLL.java

```java
package com.assignment;

    class ListNode {
        int val;
        ListNode next;
        ListNode(int x) { val = x; }
    }
    public class RemoveDuplicateFromLL {
        public ListNode deleteDuplicates(ListNode head) {
            if (head == null) return null;

            ListNode current = head;

            while (current != null && current.next != null) {
                if (current.val == current.next.val) {
                    current.next = current.next.next;
                } else {
                    current = current.next;
                }
            }

            return head;
```



```java
            return head;
        }

        public static void main(String[] args) {

            ListNode head = new ListNode(20);
            head.next = new ListNode(20);
            head.next.next = new ListNode(22);
            head.next.next.next = new ListNode(23);
            head.next.next.next.next = new ListNode(23);

            RemoveDuplicateFromLL solution = new RemoveDuplicateFromLL();
            ListNode result = solution.deleteDuplicates(head);


            while (result != null) {
                System.out.print(result.val + " ");
                result = result.next;
            }
        }
    }

```

Output -



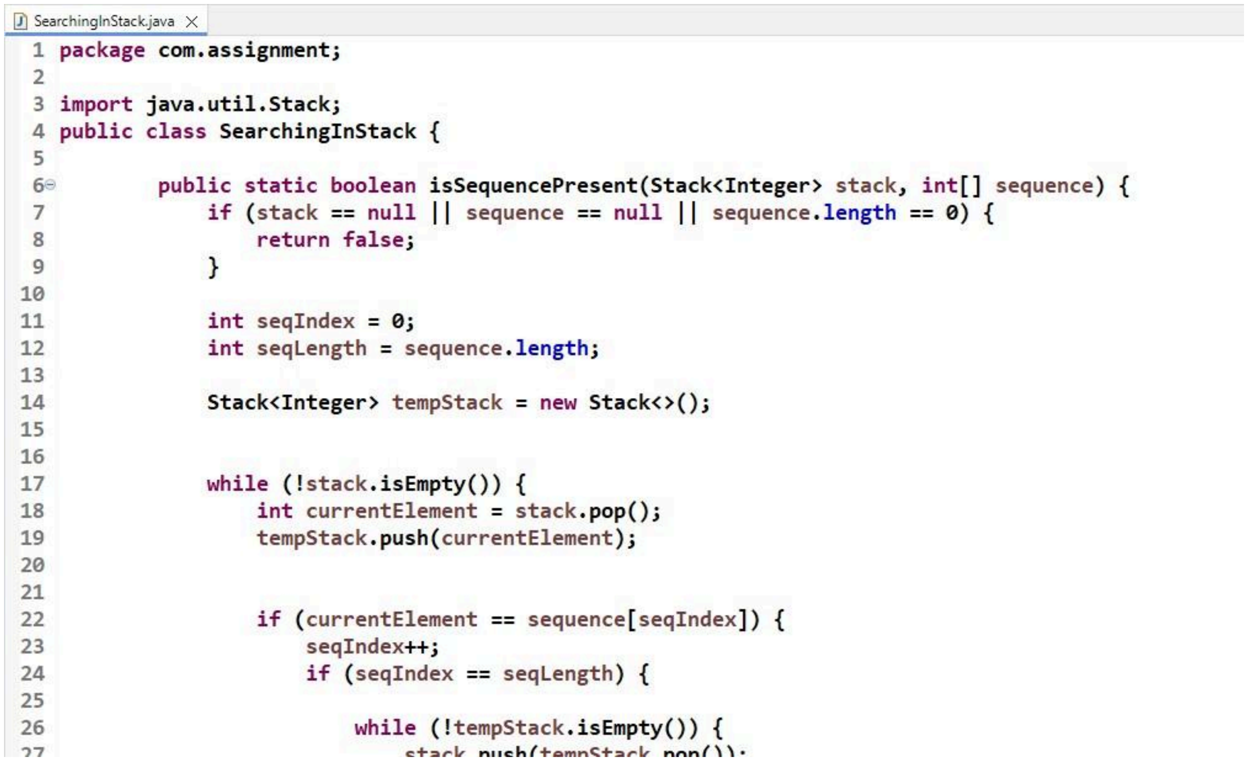
```
<terminated> RemoveDuplicateFromLL [Java Application] C:\Program Files\Java\jdk-17.0.1\bin
20 22 23
```

5] 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.

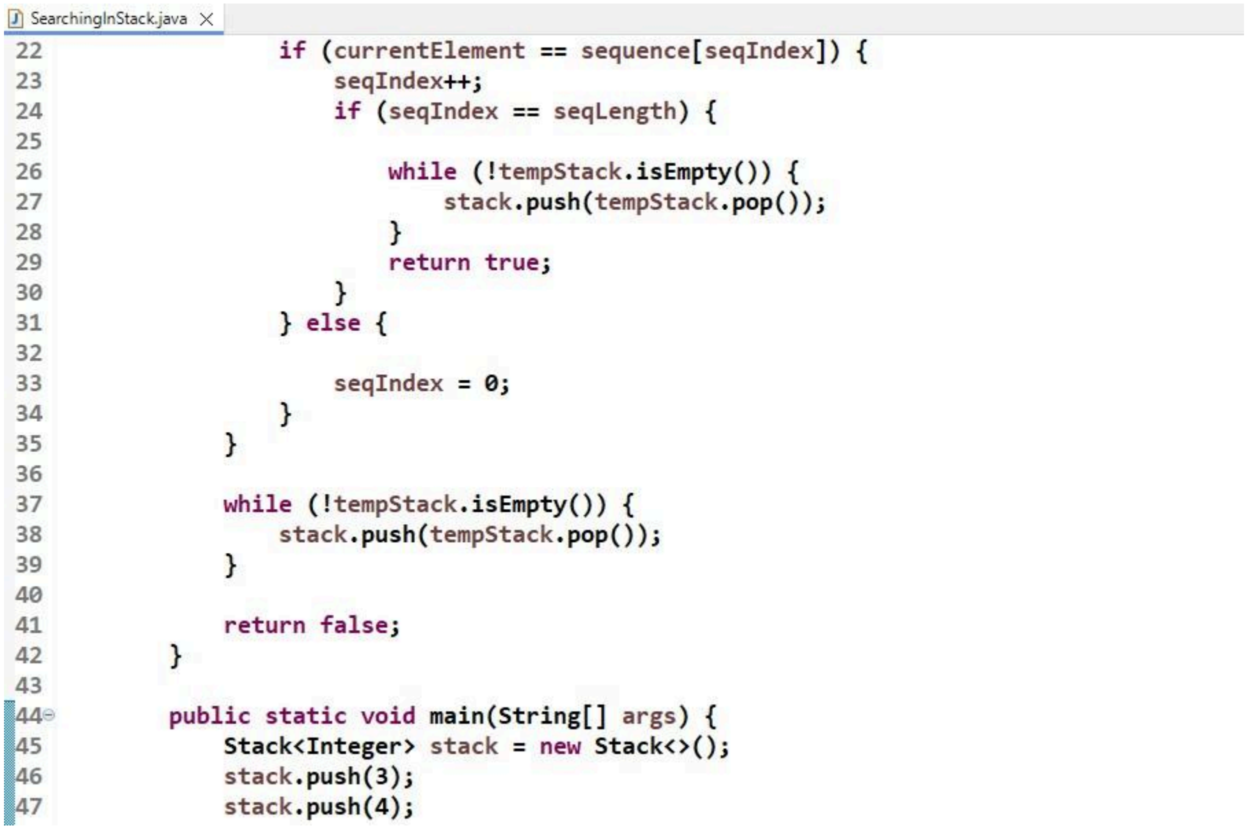
Solution:-

Code -

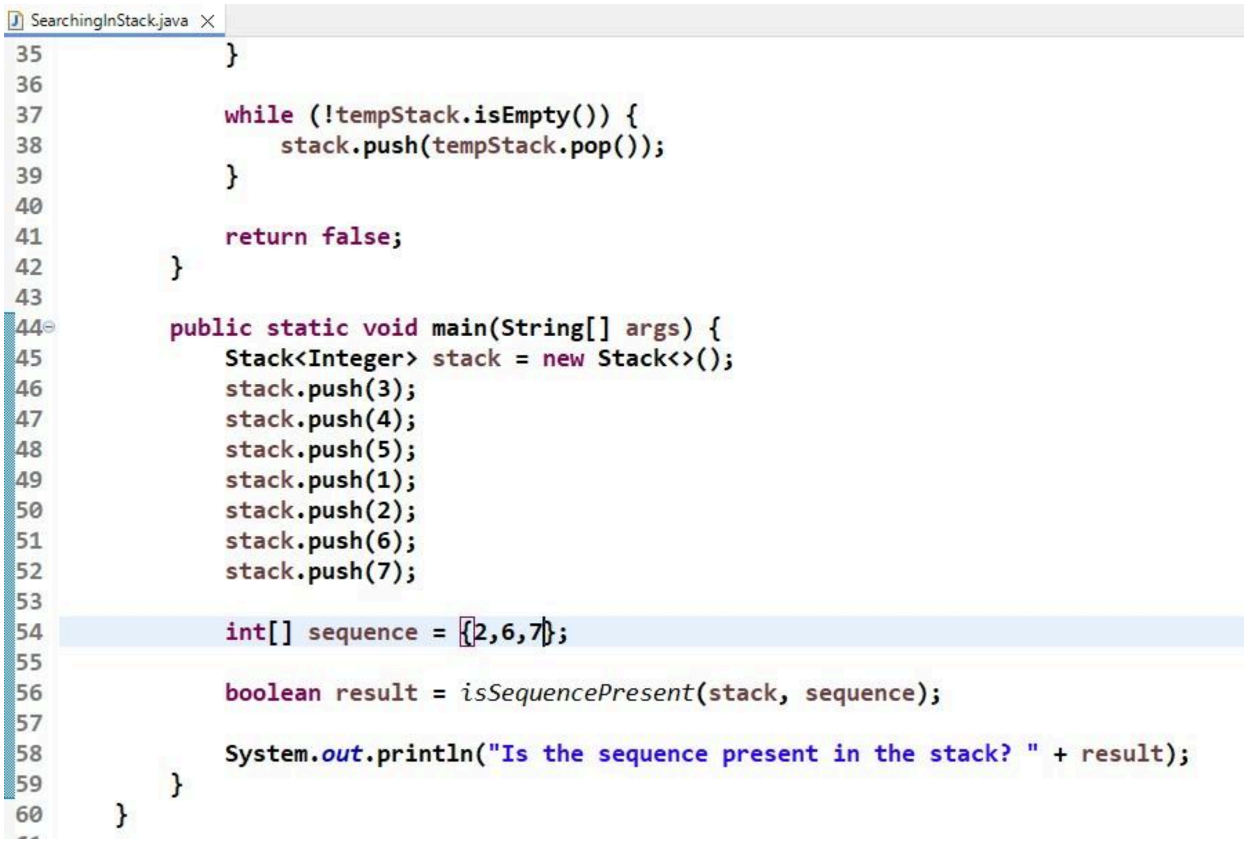


SearchingInStack.java

```java
package com.assignment;

import java.util.Stack;
public class SearchingInStack {

        public static boolean isSequencePresent(Stack<Integer> stack, int[] sequence) {
            if (stack == null || sequence == null || sequence.length == 0) {
                return false;
            }

            int seqIndex = 0;
            int seqLength = sequence.length;

            Stack<Integer> tempStack = new Stack<>();


            while (!stack.isEmpty()) {
                int currentElement = stack.pop();
                tempStack.push(currentElement);


                if (currentElement == sequence[seqIndex]) {
                    seqIndex++;
                    if (seqIndex == seqLength) {

                        while (!tempStack.isEmpty()) {
                            stack.push(tempStack.pop());
```

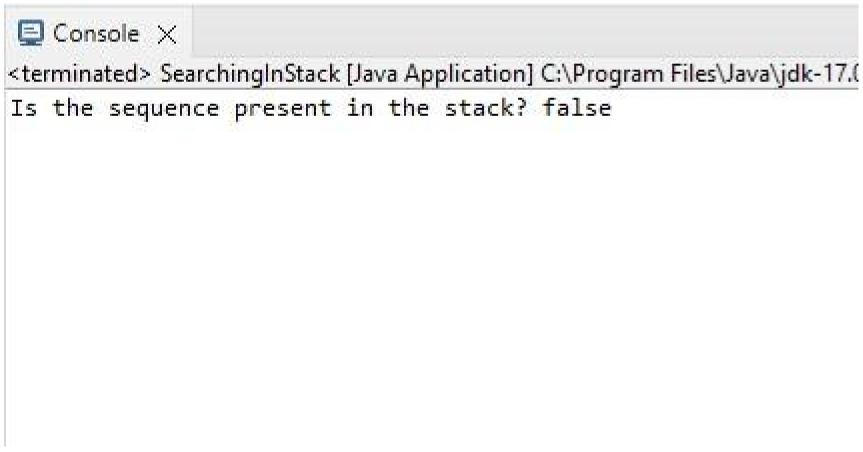

SearchingInStack.java

```java
                if (currentElement == sequence[seqIndex]) {
                    seqIndex++;
                    if (seqIndex == seqLength) {

                        while (!tempStack.isEmpty()) {
                            stack.push(tempStack.pop());
                        }
                        return true;
                    }
                } else {

                    seqIndex = 0;
                }
            }

            while (!tempStack.isEmpty()) {
                stack.push(tempStack.pop());
            }

            return false;
        }

        public static void main(String[] args) {
            Stack<Integer> stack = new Stack<>();
            stack.push(3);
            stack.push(4);
```


SearchingInStack.java

```java
            }

            while (!tempStack.isEmpty()) {
                stack.push(tempStack.pop());
            }

            return false;
        }

        public static void main(String[] args) {
            Stack<Integer> stack = new Stack<>();
            stack.push(3);
            stack.push(4);
            stack.push(5);
            stack.push(1);
            stack.push(2);
            stack.push(6);
            stack.push(7);

            int[] sequence = {2,6,7};

            boolean result = isSequencePresent(stack, sequence);

            System.out.println("Is the sequence present in the stack? " + result);
        }
    }
```

Output -


Console

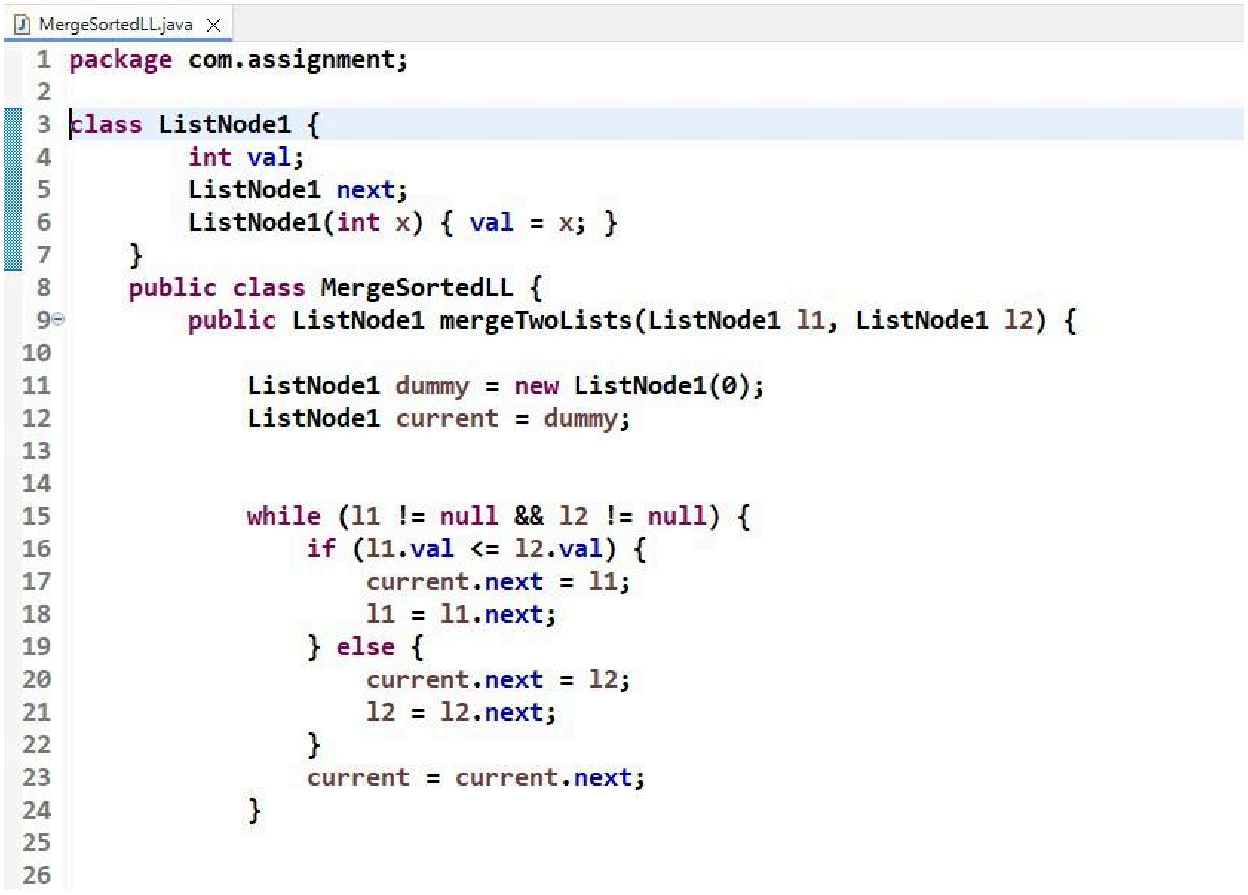
```
<terminated> SearchingInStack [Java Application] C:\Program Files\Java\jdk-17.
Is the sequence present in the stack? false
```

6] Task 7: Merging Two Sorted Linked Lists

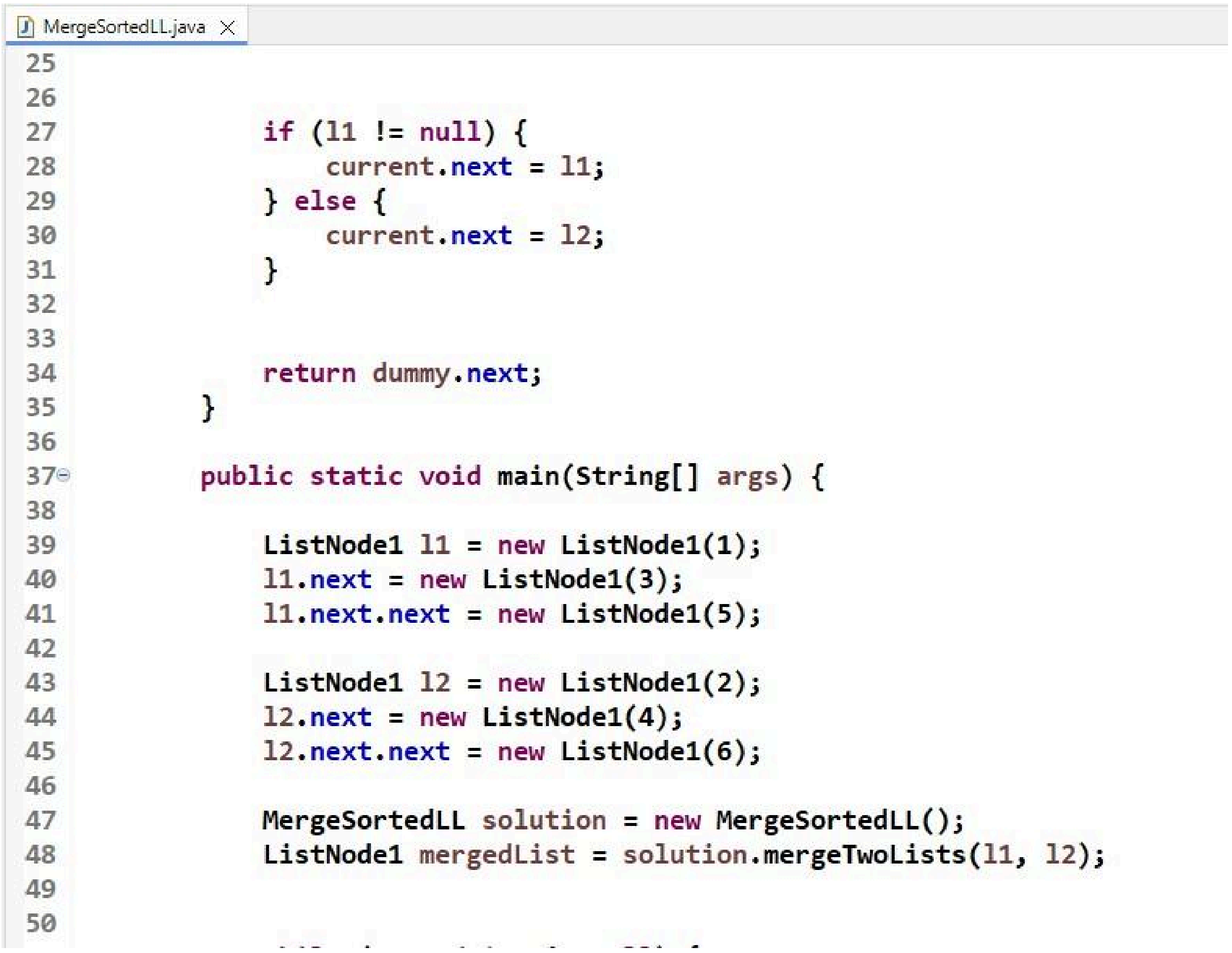
You are provided with the heads of two sorted linked lists. The lists are sorted in ascending order. Create a merged linked list in ascending order from the two input lists without using any extra space (i.e., do not create any new nodes).

Solution:-

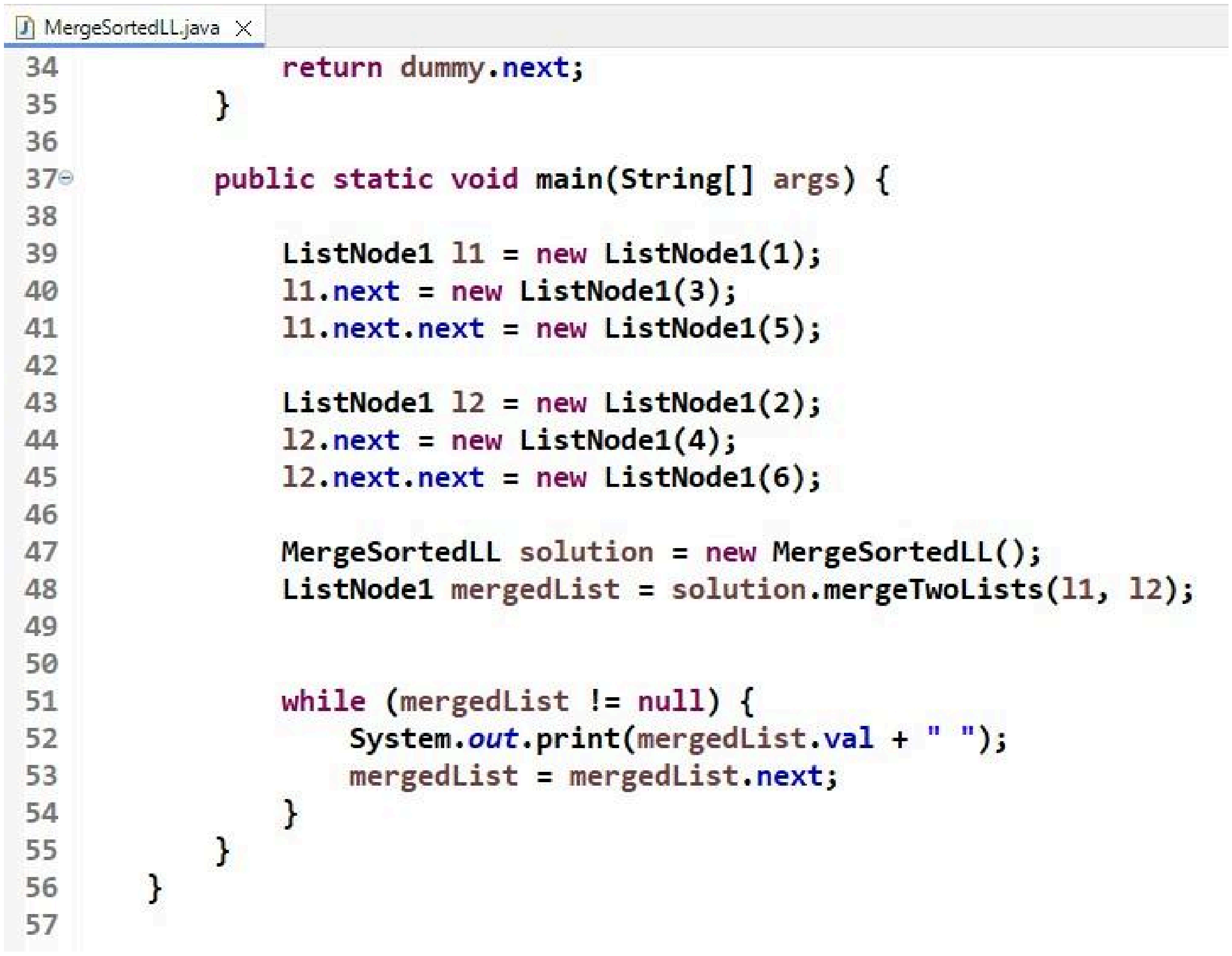
Code -



```java
package com.assignment;

class ListNode1 {
        int val;
        ListNode1 next;
        ListNode1(int x) { val = x; }
    }
    public class MergeSortedLL {
        public ListNode1 mergeTwoLists(ListNode1 l1, ListNode1 l2) {

            ListNode1 dummy = new ListNode1(0);
            ListNode1 current = dummy;


            while (l1 != null && l2 != null) {
                if (l1.val <= l2.val) {
                    current.next = l1;
                    l1 = l1.next;
                } else {
                    current.next = l2;
                    l2 = l2.next;
                }
                current = current.next;
            }


```


MergeSortedLL.java

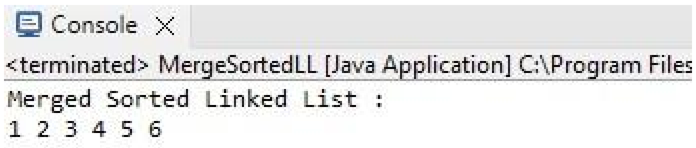
```java


            if (l1 != null) {
                current.next = l1;
            } else {
                current.next = l2;
            }


            return dummy.next;
        }

        public static void main(String[] args) {

            ListNode1 l1 = new ListNode1(1);
            l1.next = new ListNode1(3);
            l1.next.next = new ListNode1(5);

            ListNode1 l2 = new ListNode1(2);
            l2.next = new ListNode1(4);
            l2.next.next = new ListNode1(6);

            MergeSortedLL solution = new MergeSortedLL();
            ListNode1 mergedList = solution.mergeTwoLists(l1, l2);


```



MergeSortedLL.java

```java
            return dummy.next;
        }

        public static void main(String[] args) {

            ListNode1 l1 = new ListNode1(1);
            l1.next = new ListNode1(3);
            l1.next.next = new ListNode1(5);

            ListNode1 l2 = new ListNode1(2);
            l2.next = new ListNode1(4);
            l2.next.next = new ListNode1(6);

            MergeSortedLL solution = new MergeSortedLL();
            ListNode1 mergedList = solution.mergeTwoLists(l1, l2);


            while (mergedList != null) {
                System.out.print(mergedList.val + " ");
                mergedList = mergedList.next;
            }
        }
    }

```

Output -



Console

```
<terminated> MergeSortedLL [Java Application] C:\Program Files
Merged Sorted Linked List :
1 2 3 4 5 6
```

7] Task 8: Circular Queue Binary Search

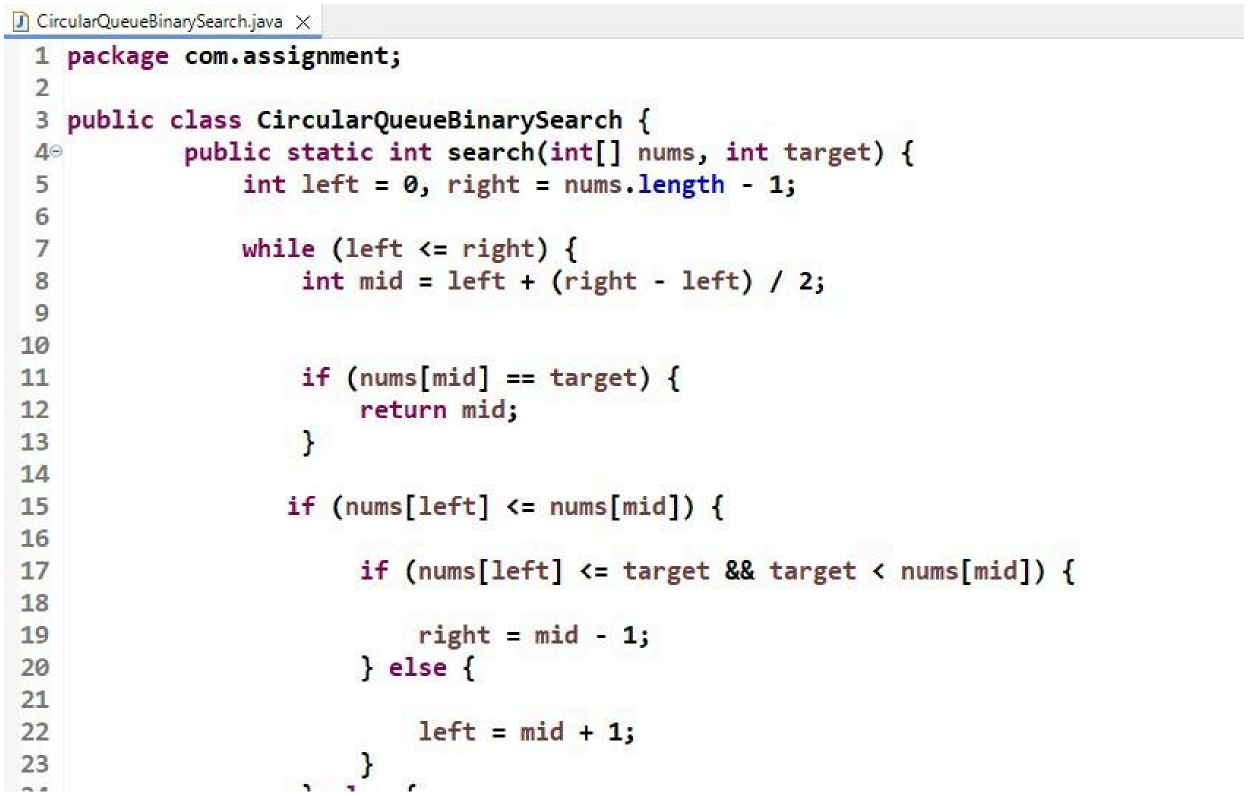
Consider a circular queue (implemented using a fixed-size array) where the elements are sorted but have been rotated at an unknown index. Describe an approach to perform a binary search for a given element within this circular queue.

Solution:-

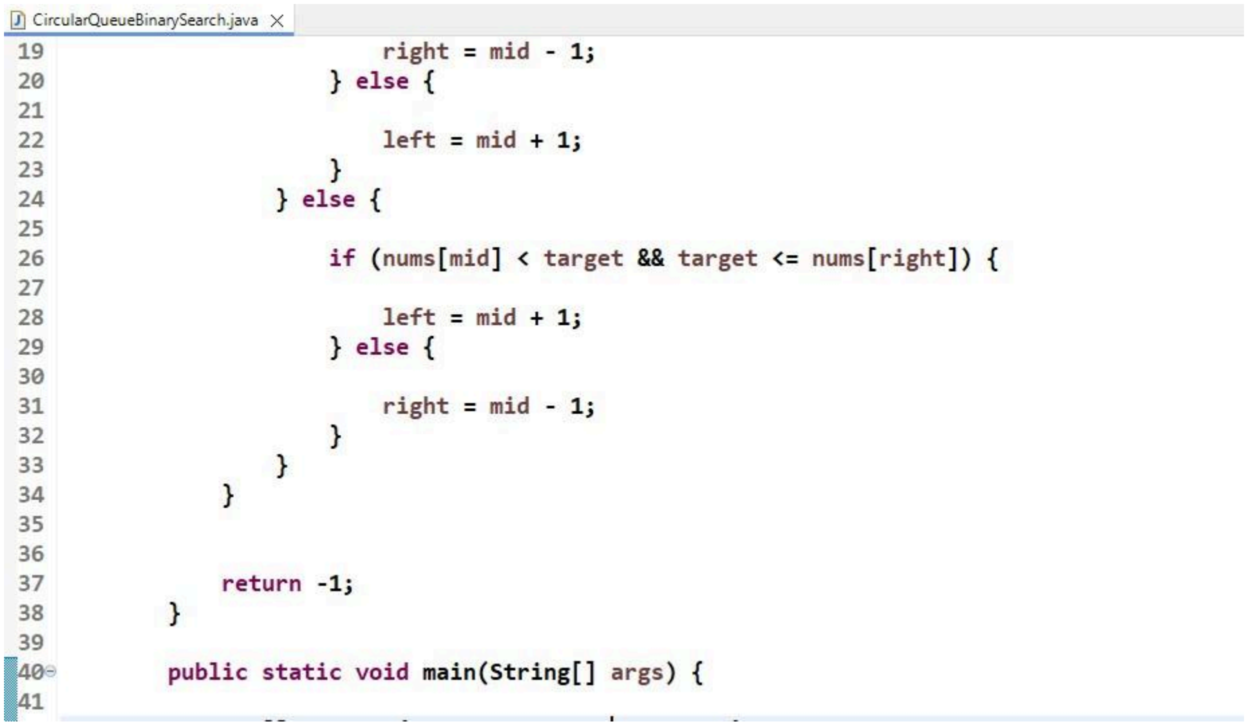
Steps for Binary Search on a Rotated Sorted Array:

1. **Initialize Pointers**: Start with two pointers, left and right, pointing to the beginning and end of the array.
2. **Find the Middle**: Calculate the midpoint of the current subarray.
3. **Check if the Middle Element is the Target**: If the middle element is the target, return its index.
4. **Determine the Sorted Half**: Check whether the left half or the right half is properly sorted.
5. **Decide Which Half to Search**:
    - If the left half is sorted:
        - Check if the target is within the range of the left half.
        - If so, continue the search in the left half.
        - Otherwise, search in the right half.
    - If the right half is sorted:
        - Check if the target is within the range of the right half.
        - If so, continue the search in the right half.
        - Otherwise, search in the left half.
6. **Repeat**: Adjust the left and right pointers based on the decision and repeat the process until the target is found or the search space is exhausted.

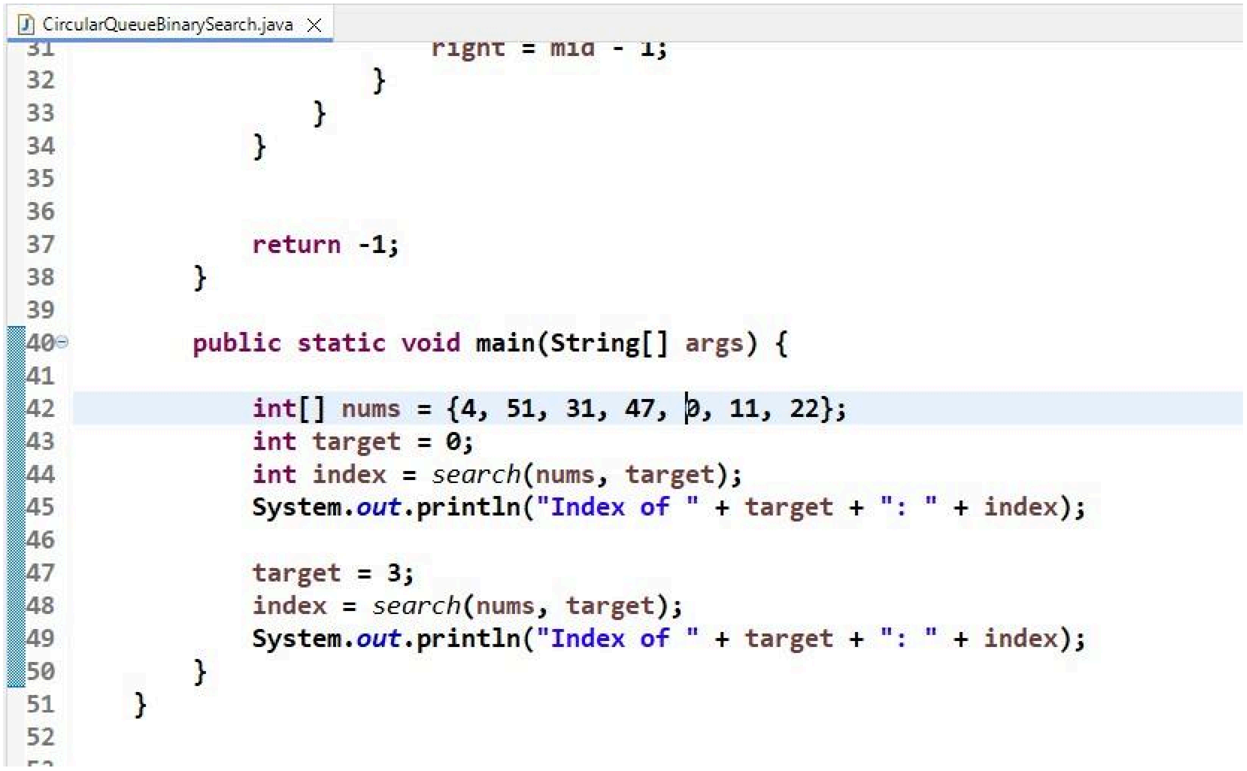
Code -


CircularQueueBinarySearch.java

```java
package com.assignment;

public class CircularQueueBinarySearch {
        public static int search(int[] nums, int target) {
            int left = 0, right = nums.length - 1;

            while (left <= right) {
                int mid = left + (right - left) / 2;


                if (nums[mid] == target) {
                    return mid;
                }

               if (nums[left] <= nums[mid]) {

                    if (nums[left] <= target && target < nums[mid]) {

                        right = mid - 1;
                    } else {

                        left = mid + 1;
                    }
```


CircularQueueBinarySearch.java

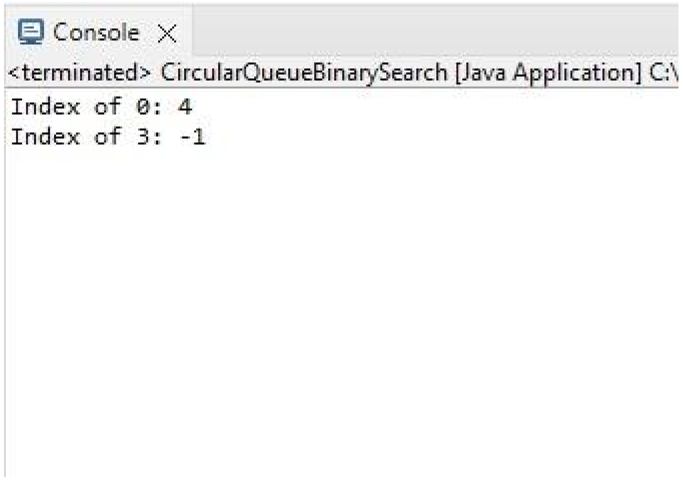
```java
                        right = mid - 1;
                    } else {

                        left = mid + 1;
                    }
                } else {

                    if (nums[mid] < target && target <= nums[right]) {

                        left = mid + 1;
                    } else {

                        right = mid - 1;
                    }
                }
            }


            return -1;
        }

        public static void main(String[] args) {

```



CircularQueueBinarySearch.java

```java
                        right = mid - 1;
                    }
                }
            }


            return -1;
        }

        public static void main(String[] args) {

            int[] nums = {4, 51, 31, 47, 0, 11, 22};
            int target = 0;
            int index = search(nums, target);
            System.out.println("Index of " + target + ": " + index);

            target = 3;
            index = search(nums, target);
            System.out.println("Index of " + target + ": " + index);
        }
    }
```

Output -



Console

```
<terminated> CircularQueueBinarySearch [Java Application] C:\
Index of 0: 4
Index of 3: -1
```