RPS DAY 1- 6 Assignments

Assignment 1

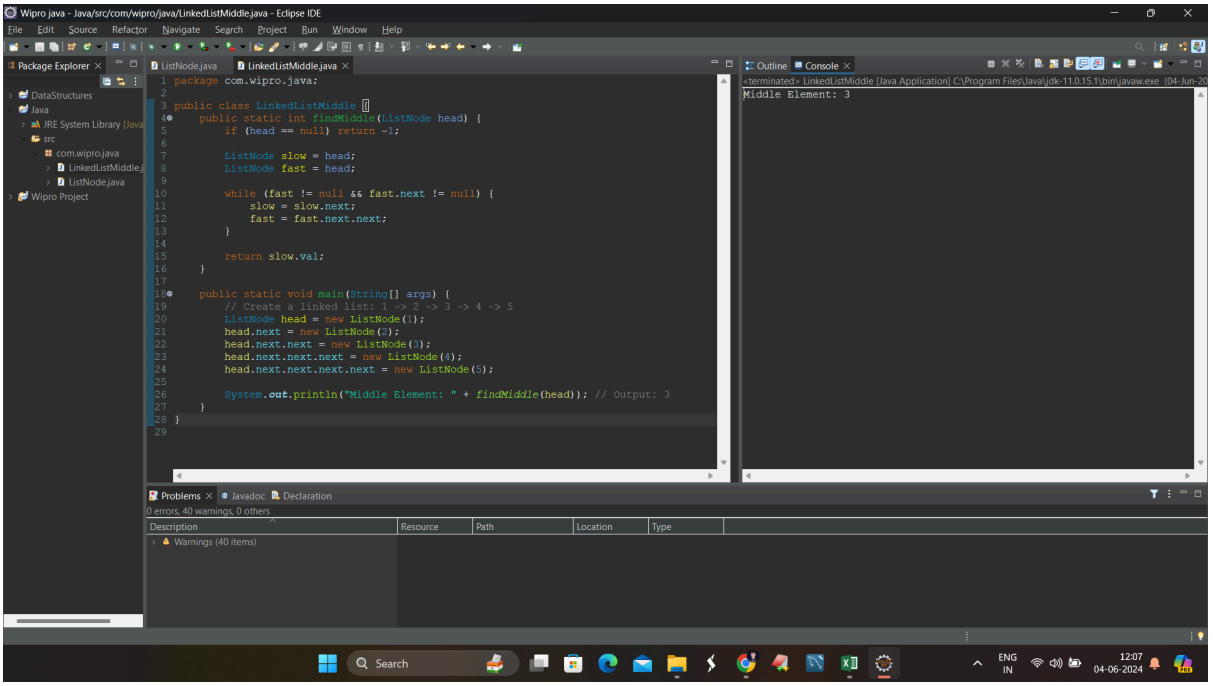
Name: Akshada Baad
Batch - CPPE

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.

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class LinkedListMiddle {
    public static int findMiddle(ListNode head) {
        if (head == null) return -1;

        ListNode slow = head;
        ListNode fast = head;

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

        return slow.val;
    }

    public static void main(String[] args) {
        // Create a linked list: 1 -> 2 -> 3 -> 4 -> 5
        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);

        System.out.println("Middle Element: " + findMiddle(head)); // Output: 3
    }
}
```

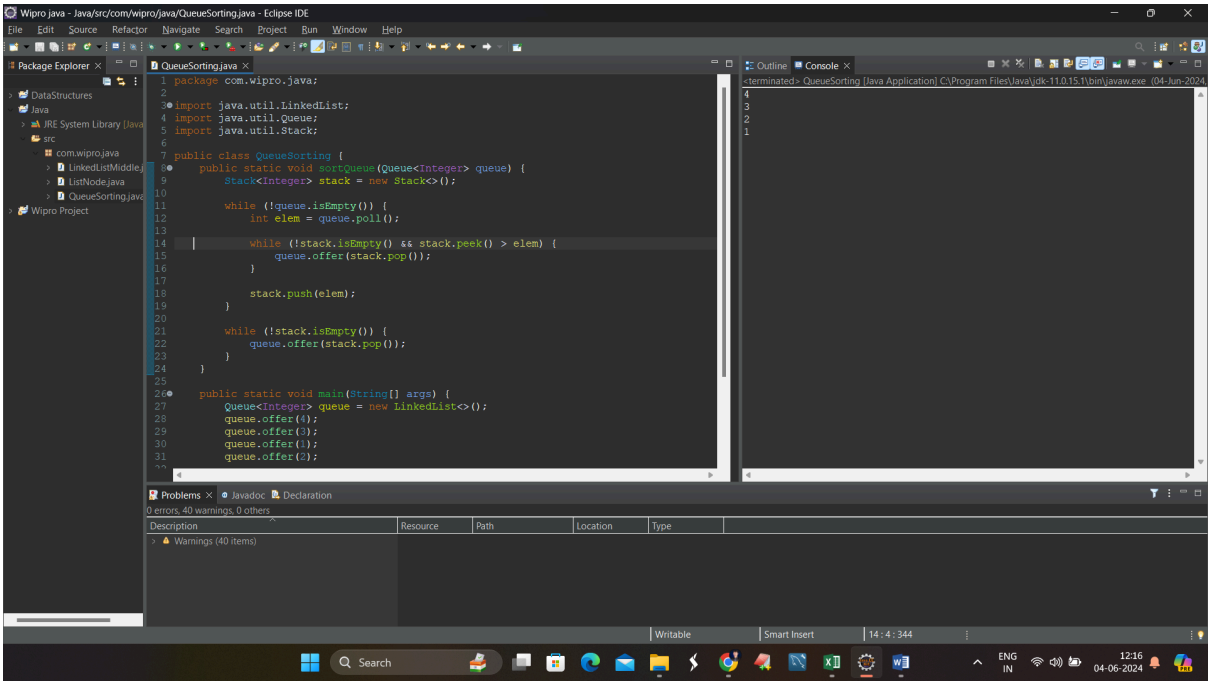


Task 3: Queue Sorting with Limited Space

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.

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

public class QueueSorting {
    public static void sortQueue(Queue<Integer> queue) {
```

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

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

            while (!stack.isEmpty() && stack.peek() > elem) {
                queue.offer(stack.pop());
            }

            stack.push(elem);
        }

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

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

        sortQueue(queue);

        while (!queue.isEmpty()) {
            System.out.println(queue.poll());
        }
    }
}
```

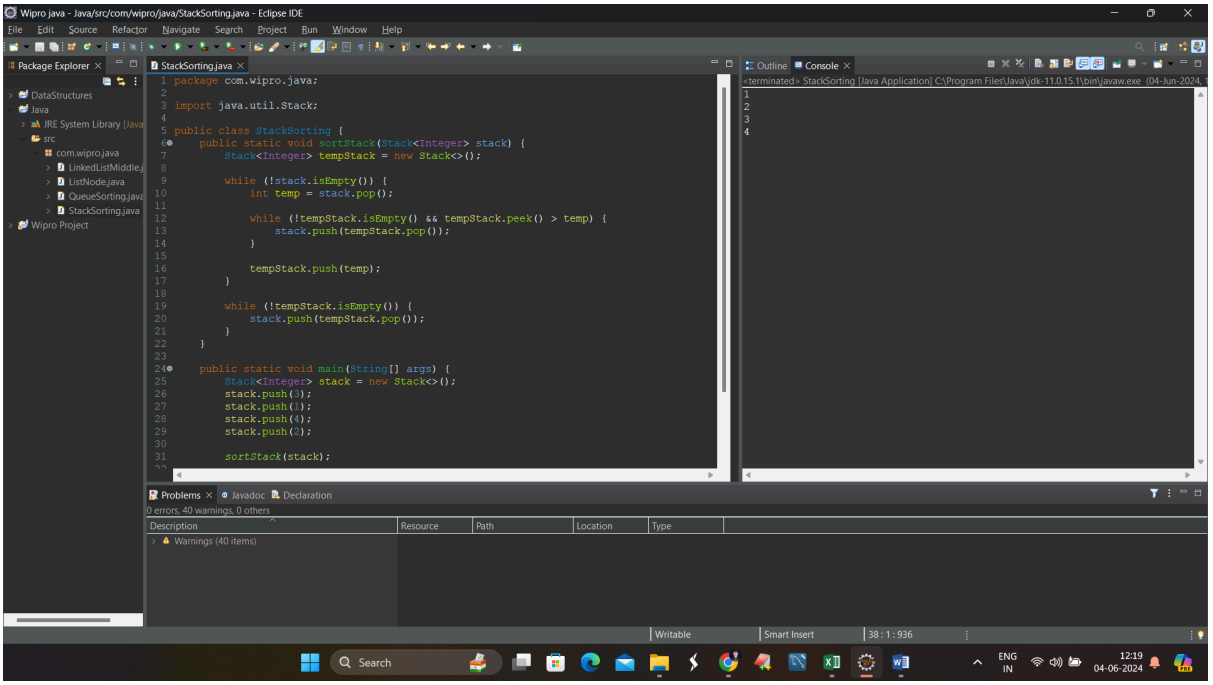


Task 4: Stack Sorting In-Place

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.

```
import java.util.Stack;


public class StackSorting {
    public static void sortStack(Stack<Integer> stack) {
        Stack<Integer> tempStack = new Stack<>();


        while (!stack.isEmpty()) {
            int temp = stack.pop();


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

```
            tempStack.push(temp);
        }

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

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

        sortStack(stack);

        while (!stack.isEmpty()) {
            System.out.println(stack.pop());
        }
    }
}
```

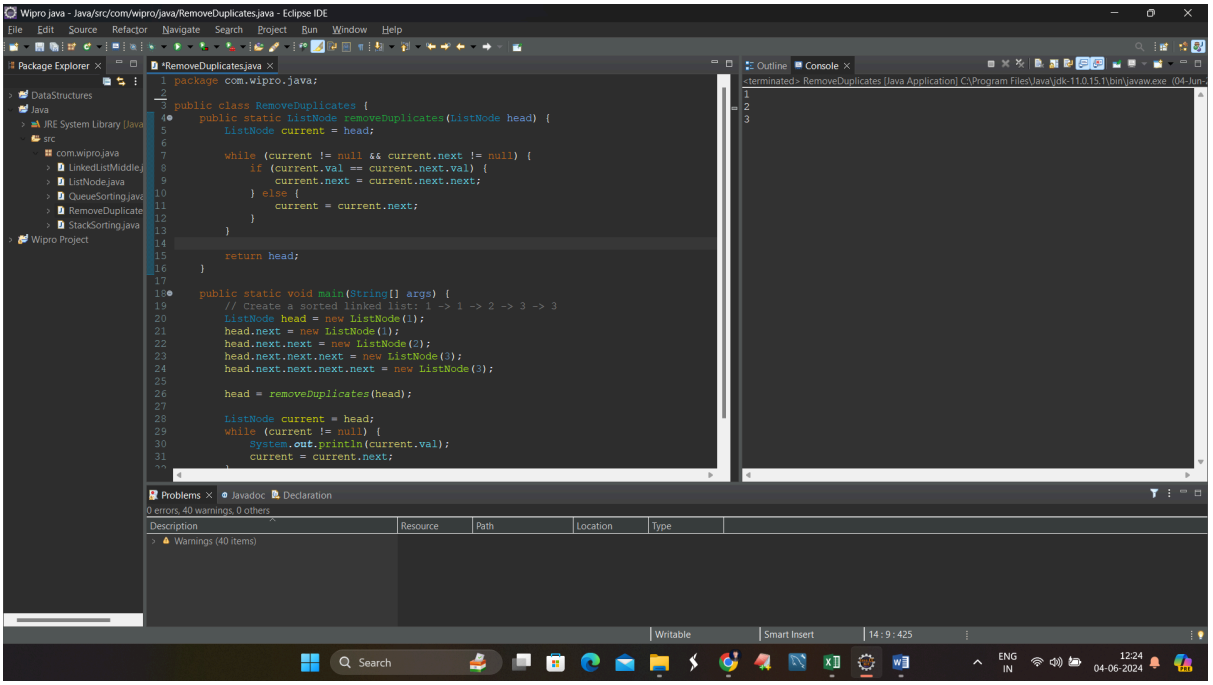


Task 5: Removing Duplicates from a Sorted Linked List

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

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class RemoveDuplicates {
    public static ListNode removeDuplicates(ListNode head) {
        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;
    }

    public static void main(String[] args) {
        // Create a sorted linked list: 1 -> 1 -> 2 -> 3 -> 3
        ListNode head = new ListNode(1);
        head.next = new ListNode(1);
        head.next.next = new ListNode(2);
        head.next.next.next = new ListNode(3);
        head.next.next.next.next = new ListNode(3);

        head = removeDuplicates(head);

        ListNode current = head;
        while (current != null) {
            System.out.println(current.val);
            current = current.next;
        }
    }
}
```

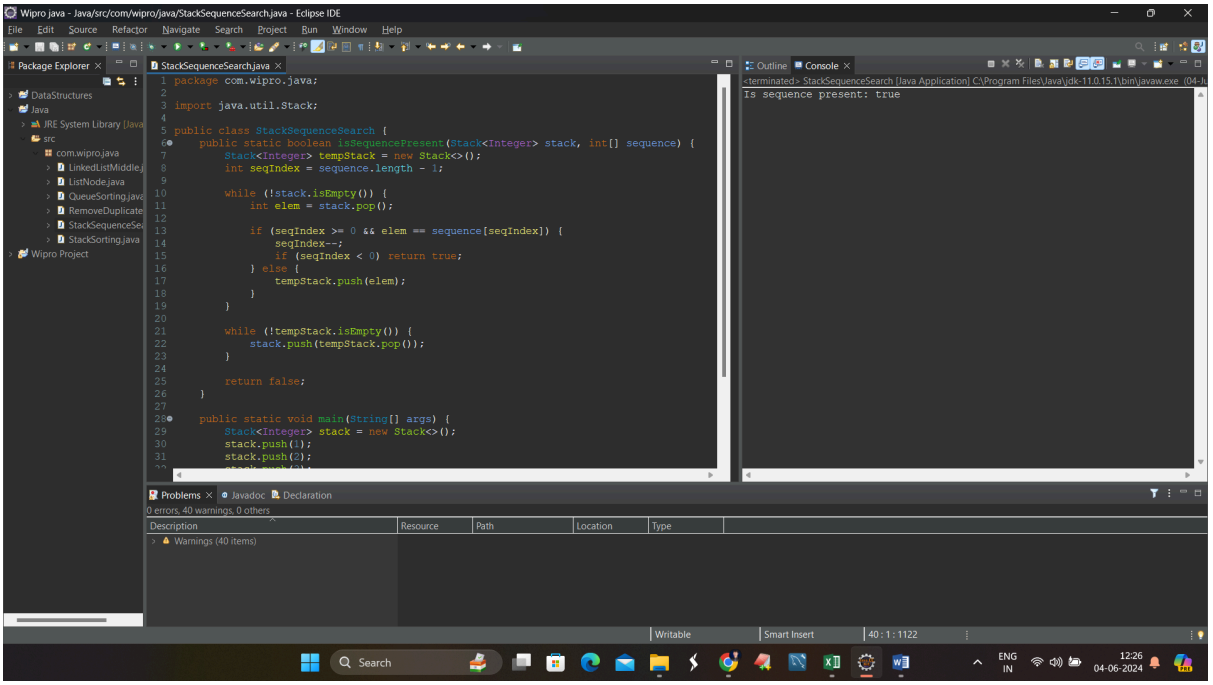


Task 6: Searching for a Sequence in a Stack

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

```
import java.util.Stack;


public class StackSequenceSearch {
    public static boolean isSequencePresent(Stack<Integer> stack, int[] sequence) {
        Stack<Integer> tempStack = new Stack<>();
        int seqIndex = sequence.length - 1;


        while (!stack.isEmpty()) {
            int elem = stack.pop();


            if (seqIndex >= 0 && elem == sequence[seqIndex]) {
                seqIndex--;
                if (seqIndex < 0) return true;
            } else {
```

```
                tempStack.push(elem);
            }
        }

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

        return false;
    }

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

        int[] sequence = {3, 4, 5};
        System.out.println("Is sequence present: " + isSequencePresent(stack, sequence)); //
Output: true
    }
}
```

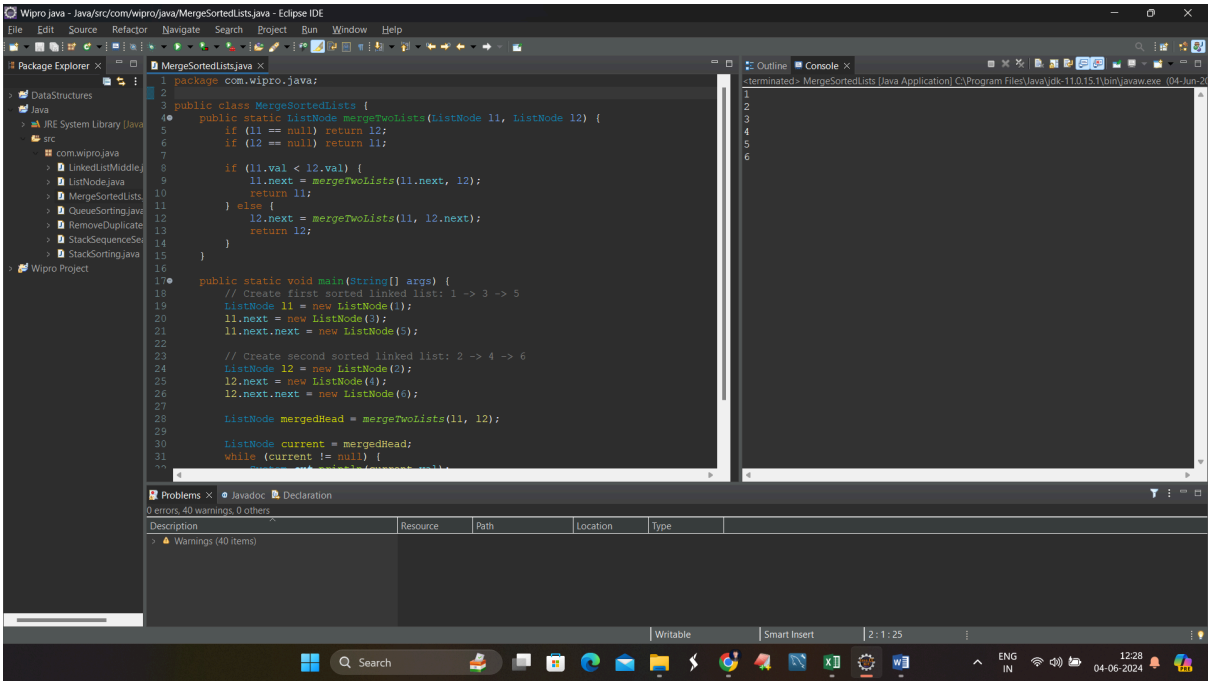


Task 7: Merging Two Sorted Linked Lists

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).

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class MergeSortedLists {
    public static ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;

        if (l1.val < l2.val) {
            l1.next = mergeTwoLists(l1.next, l2);
            return l1;
```

```
        } else {
            l2.next = mergeTwoLists(l1, l2.next);
            return l2;
        }
    }

    public static void main(String[] args) {
        // Create first sorted linked list: 1 -> 3 -> 5
        ListNode l1 = new ListNode(1);
        l1.next = new ListNode(3);
        l1.next.next = new ListNode(5);

        // Create second sorted linked list: 2 -> 4 -> 6
        ListNode l2 = new ListNode(2);
        l2.next = new ListNode(4);
        l2.next.next = new ListNode(6);

        ListNode mergedHead = mergeTwoLists(l1, l2);

        ListNode current = mergedHead;
        while (current != null) {
            System.out.println(current.val);
            current = current.next;
        }
    }
}
```

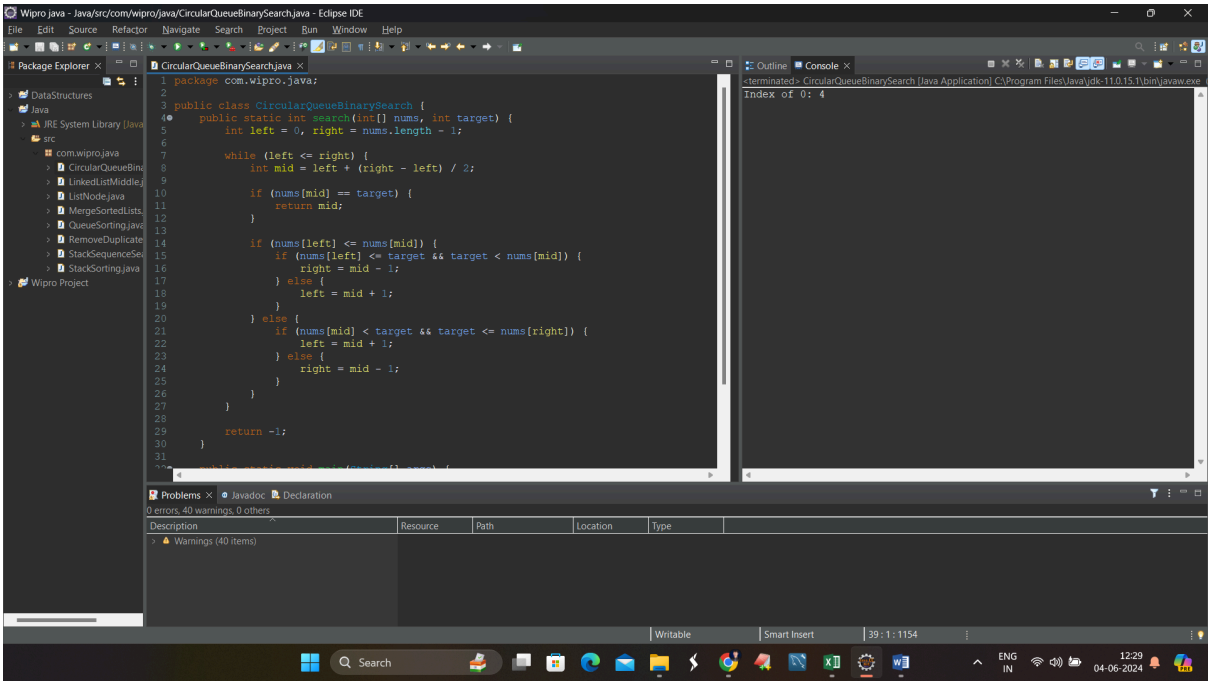


Task 8: Circular Queue Binary Search

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.

```
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;
            }
        } else {
            if (nums[mid] < target && target <= nums[right]) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
    }

    return -1;
}

public static void main(String[] args) {
    int[] circularQueue = {4, 5, 6, 7, 0, 1, 2};
    int target = 0;
    int index = search(circularQueue, target);
    System.out.println("Index of " + target + ": " + index); // Output: 4
}
}
```


Wipro java - Java/src/com/wipro/java/CircularQueueBinarySearch.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
DataStructures
Java
JRE System Library [Java
src
com.wipro.java
CircularQueueBina
LinkedListMiddle.j
ListNode.java
MergeSortedLists.
QueueSorting.java
RemoveDuplicate
StackSequenceSe
StackSorting.java
Wipro Project
CircularQueueBinarySearch.java
package com.wipro.java;
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;
}
} else {
if (nums[mid] < target && target <= nums[right]) {
left = mid + 1;
} else {
right = mid - 1;
}
}
}
return -1;
}
Outline
Console
<terminated> CircularQueueBinarySearch [Java Application] C:\Program Files\Java\jdk-11.0.15.1\bin\javaw.exe
Index of 0: 4
Problems
Javadoc
Declaration
0 errors, 40 warnings, 0 others
Description
Resource
Path
Location
Type
Warnings (40 items)
Writable
Smart Insert
39 : 1 : 1154
Search
ENG
IN
12:29
04-06-2024
