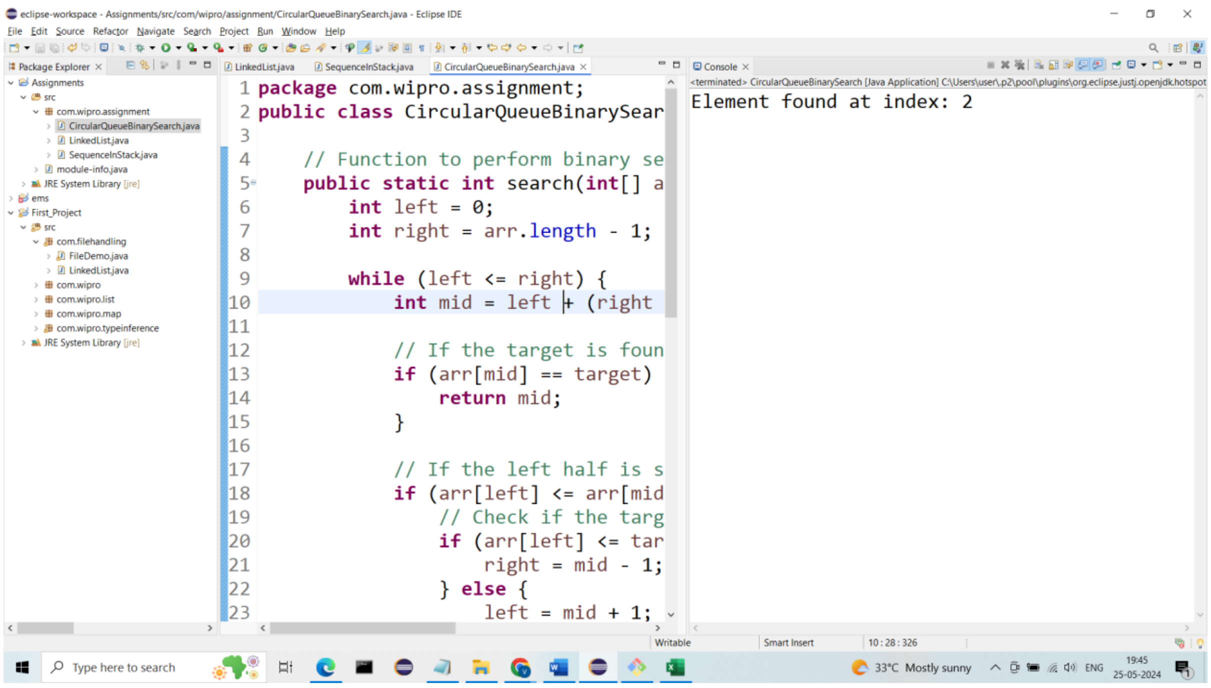
Task 8: Circular Queue Binary Search

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.

```
package com.wipro.assignment;
public class CircularQueueBinarySearch {

    // Function to perform binary search
on a circular queue
    public static int search(int[] arr,
int target) {
        int left = 0;
        int right = arr.length - 1;

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

            // If the target is found at
the middle
            if (arr[mid] == target) {
                return mid;
            }
            // If the left half is sorted
            if (arr[left] <= arr[mid]) {
                // Check if the target
lies in the left half
                if (arr[left] <= target
&& target < arr[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
            // If the right half is
sorted
            else {
                // Check if the target
lies in the right half
                if (arr[mid] < target &&
target <= arr[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }

        // Target not found
        return -1;
    }
```

```java
    public static void main(String[] args) {
        int[] arr = {4, 5, 6, 7, 8, 9, 1, 2, 3}; // Example circular queue
        int target = 6; // Element to search for
        int index = search(arr, target);
        if (index != -1) {
            System.out.println("Element found at index: " + index);
        } else {
            System.out.println("Element not found");
        }
    }
}
```

Output-:



eclipse-workspace - Assignments/src/com/wipro/assignment/CircularQueueBinarySearch.java - Eclipse IDE

```java
package com.wipro.assignment;
public class CircularQueueBinarySear

    // Function to perform binary se
    public static int search(int[] a
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right

            // If the target is foun
            if (arr[mid] == target)
                return mid;
            }

            // If the left half is s
            if (arr[left] <= arr[mid
                // Check if the targ
                if (arr[left] <= tar
                    right = mid - 1;
                } else {
                    left = mid + 1;
```

<terminated> CircularQueueBinarySearch [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot

```
Element found at index: 2
```

Explanation:

1. **CircularQueue Class:**
    - Represents a circular queue using an array items and keeps track of front, rear, and size.
    - Binary search and pivot finding methods are included.
2. **binarySearch Function:**
    - Takes the target element to search for.
    - Handles the empty queue case.
    - Calls findPivot to locate the potential rotation point in the queue.
    - Based on the target's value compared to the last element:
        - If the target is greater, it searches in the left part (including the front) using binarySearchUtil.
        - Otherwise, it searches in the right part (including the rear and checks the front element separately).
3. **findPivot Function:**

- Takes front and rear indices as input.
- Handles empty queue or single-element queue cases.
- Iterates through the queue using a while loop until front and rear meet.
- If the current element is less than or equal to the next element, it means we haven't passed the rotation point yet.
- If the current element is greater than the next element, the current index (front) is likely the pivot point and is returned.
- If the loop completes without finding a clear pivot (e.g., the queue might be sorted without rotation), it returns front as a potential starting point.

4. **binarySearchUtil Function:**
   - Standard binary search implementation for a sorted array within the specified low and high indices.