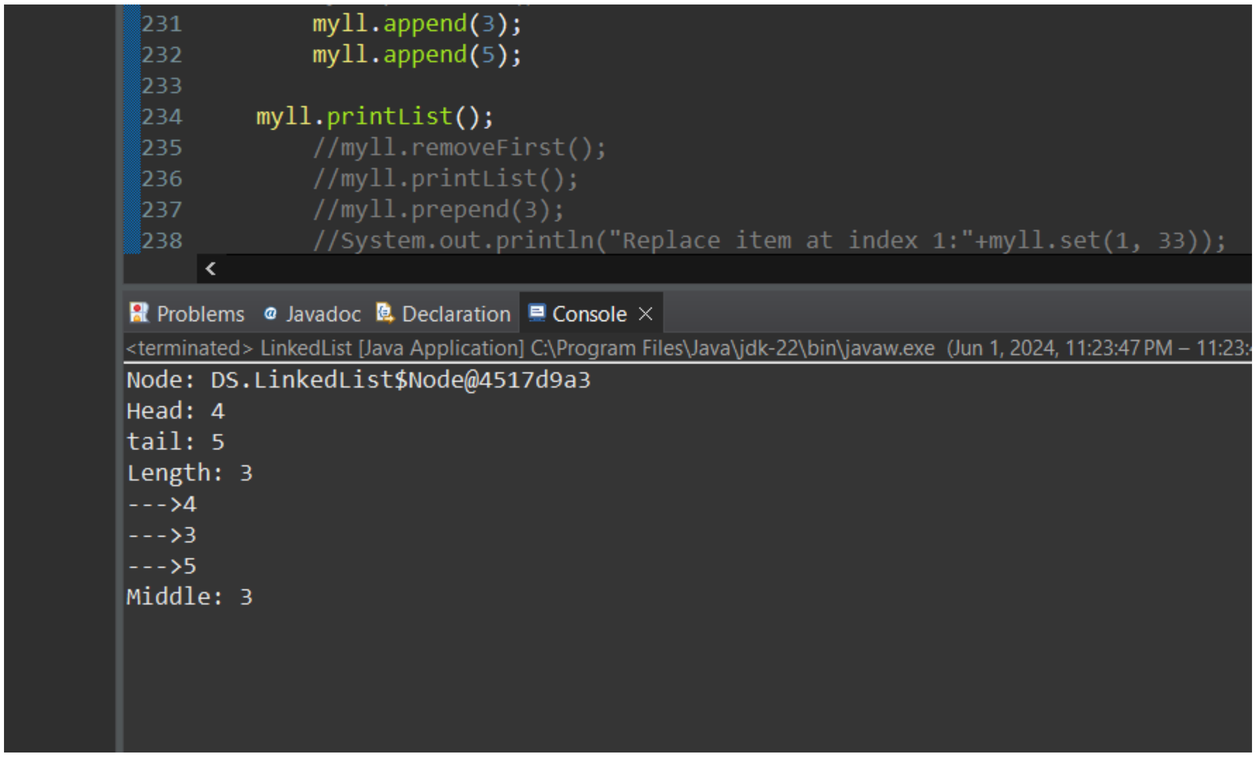
Task 2: Linked List Middle Element Search
You are given a singly linked list. Write a function to find the middle element without using any extra space and only one traversal through the linked list.

Ans.

```
public Node findMiddle()

        {

                if(head==null)

                        return null;

                Node current=head;

                Node current1=head;


                while(current1!=null && current1.next!=null)

                {

                        current=current.next;

                        current1=current1.next.next;

                }

                return current;

        }
```

OutPUT:



```java
        myll.append(3);
        myll.append(5);

    myll.printList();
        //myll.removeFirst();
        //myll.printList();
        //myll.prepend(3);
        //System.out.println("Replace item at index 1:"+myll.set(1, 33));
```

```
<terminated> LinkedList [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (Jun 1, 2024, 11:23:47 PM – 11:23:
Node: DS.LinkedList$Node@4517d9a3
Head: 4
tail: 5
Length: 3
--->4
--->3
--->5
Middle: 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.

Ans.

1. **Iterate:** Multiple times (n iterations, n being queue size).
2. **Find Minimum:** Dequeue and enqueue all elements, tracking the smallest found (store on stack - optional).
3. **Enqueue Minimum:** Dequeue all again, enqueue the minimum (from stack or comparison), then remaining elements.

Repeat steps 2-3 progressively places the smallest elements at the front, achieving a sorted queue.

Trade-off: Using the stack simplifies minimum tracking but might overflow. Not using the stack saves space but requires comparing with the previous minimum.

Complexity:

- Time Complexity: O(n^2). Each iteration requires dequeuing and enqueueing all elements, leading to a quadratic runtime.
- Space Complexity: O(1) or O(log n) depending on the approach. Without using the stack for minimum tracking, it's constant. Using the stack introduces a space complexity of O(log n) in the worst case (when the stack holds all minimum elements during initial iterations for a large queue).

Task 4: Stack Sorting In-Place

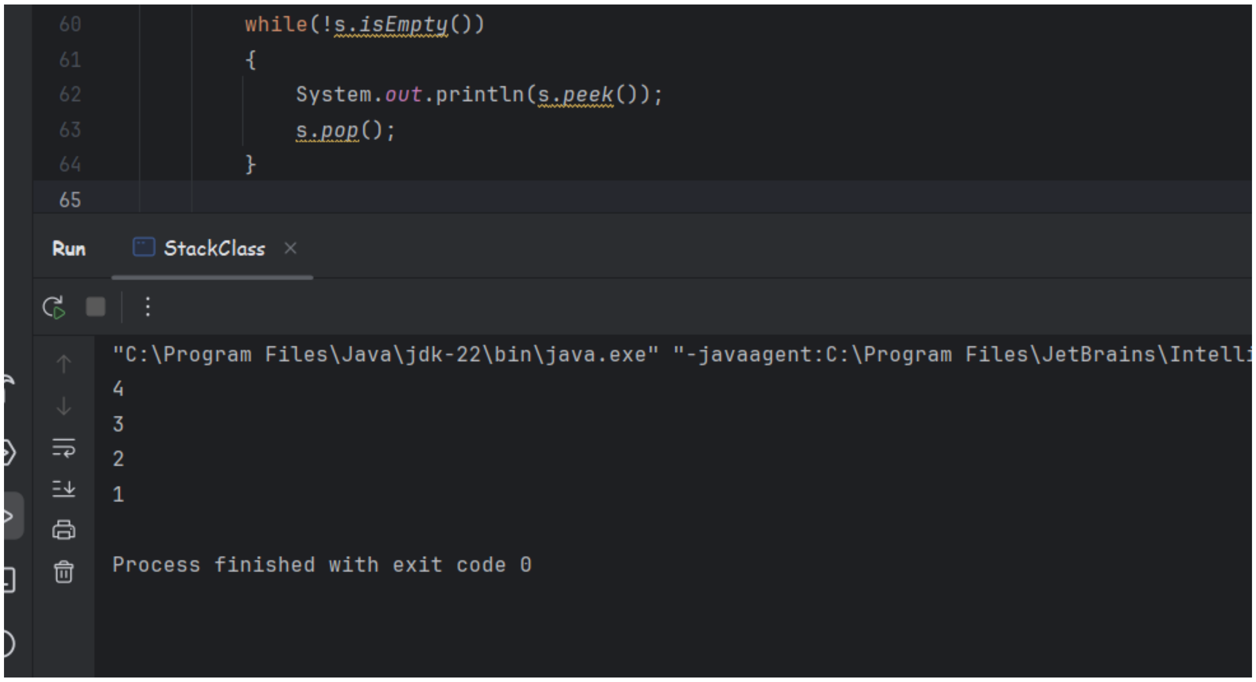
You must write a function to sort a stack such that the smallest items are on the top. You can use an additional temporary stack, but you may not copy the elements into any other data structure such as an array. The stack supports the following operations: push, pop, peek, and isEmpty.

Ans:

```
package Stack;



public class StackClass {
  static  class Node{
    int data;
    Node next;

    public Node(int data)
    {
      this.data=data;
      next=null;
    }
  }
  static  class Stack{
    public static Node head;
    public static boolean isEmpty()
    {
      return head==null;
```

```
        }
        public static void push(int data)
        {
            Node newNode=new Node(data);
            if(isEmpty())
            {
                head=newNode;
                return;
            }
            newNode.next=head;
            head=newNode;
        }
        public static int pop()
        {
            if(isEmpty())
            {
                return -1;
            }
            int top=head.data;
            head=head.next;
            return top;
        }
        public static int peek()
        {
            if(isEmpty())
            {
                return -1;
            }
            int top= head.data;    //return head.data
            return top;
        }
    }
    public static void main(String[] args) {
        Stack s=new Stack();
        s.push(1);
        s.push(2);
        s.push(3);
        s.push(4);

        while(!s.isEmpty())
        {
            System.out.println(s.peek());
            s.pop();
        }

    }
}
```

OUTPUT:



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

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Intelli
4
3
2
1

Process finished with exit code 0
```

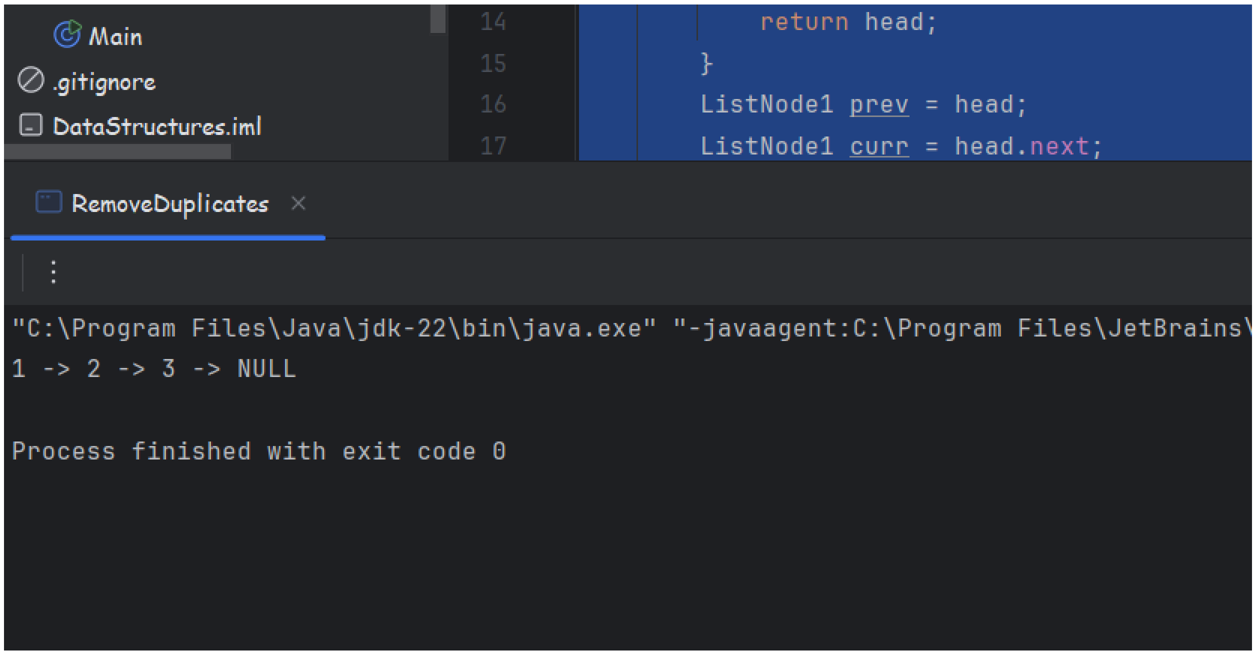
Task 5: Removing Duplicates from a Sorted Linked List

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

Ans:

```
package List;

 class ListNode1 {
    int val;
    ListNode1 next;
    ListNode1() {}
    ListNode1(int val) { this.val = val; }
    ListNode1(int val, ListNode1 next) { this.val = val; this.next = next; }
}

public class RemoveDuplicates {
    public ListNode1 removeDuplicates(ListNode1 head) {
        if (head == null || head.next == null) {
            return head;
        }
        ListNode1 prev = head;
        ListNode1 curr = head.next;

        while (curr != null) {
            if (prev.val == curr.val) {
                // Duplicate found, remove current node
                prev.next = curr.next;
            } else {
                // Move both pointers
                prev = curr;
            }
            curr = curr.next;
        }
        return head;
    }

    public static void main(String[] args) {
        RemoveDuplicates solution = new RemoveDuplicates();

        // Sample linked list with duplicates
        ListNode1 head = new ListNode1(1);
        head.next = new ListNode1(1);
        head.next.next = new ListNode1(2);
        head.next.next.next = new ListNode1(3);
        head.next.next.next.next = new ListNode1(3);

        // Remove duplicates
        head = solution.removeDuplicates(head);

        // Print the modified list
        ListNode1 temp = head;
        while (temp != null) {
            System.out.print(temp.val + " -> ");
            temp = temp.next;
        }
        System.out.println("NULL");
    }
}
```

OutPut:



```
            return head;
        }
        ListNode1 prev = head;
        ListNode1 curr = head.next;
```

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\
1 -> 2 -> 3 -> NULL

Process finished with exit code 0
```

Task 6: Searching for a Sequence in a Stack

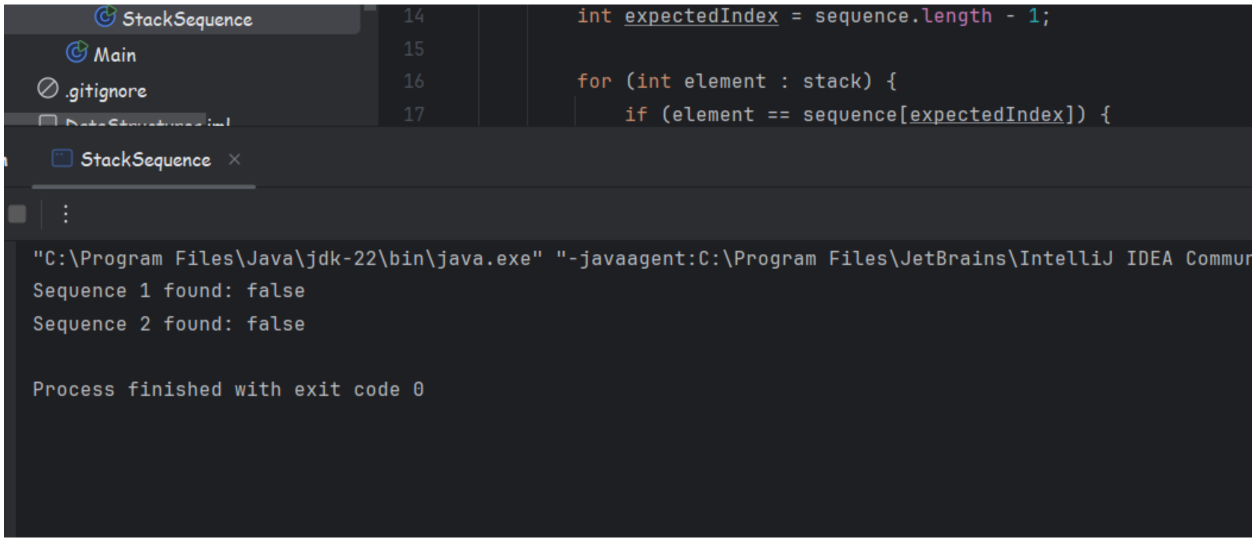
Given a stack and a smaller array representing a sequence, write a function that determines if the sequence is present in the stack. Consider the sequence present if, upon popping the elements, all elements of the array appear consecutively in the stack.

Ans:

```
package Stack;

import java.util.Stack;

public class StackSequence {
```

```java
public static boolean searchSequence(Stack<Integer> stack, int[] sequence) {
    if (stack.isEmpty() || sequence.length > stack.size()) {
        return false;
    }

    int expectedIndex = sequence.length - 1;

    for (int element : stack) {
        if (element == sequence[expectedIndex]) {
            expectedIndex--;
            if (expectedIndex < 0) {
                return true;
            }
        } else {
            expectedIndex = sequence.length - 1; // Reset the search when mismatch occurs
        }
    }

    return false; // Sequence not found if loop finishes
}

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

    int[] sequence1 = {1, 2, 3};
    int[] sequence2 = {4, 2};

    System.out.println("Sequence 1 found: " + searchSequence(stack, sequence1));
    System.out.println("Sequence 2 found: " + searchSequence(stack, sequence2));
}
}
```

OutPut:



```
        int expectedIndex = sequence.length - 1;

        for (int element : stack) {
            if (element == sequence[expectedIndex]) {
```

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Commur
Sequence 1 found: false
Sequence 2 found: false

Process finished with exit code 0
```

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).

Ans:

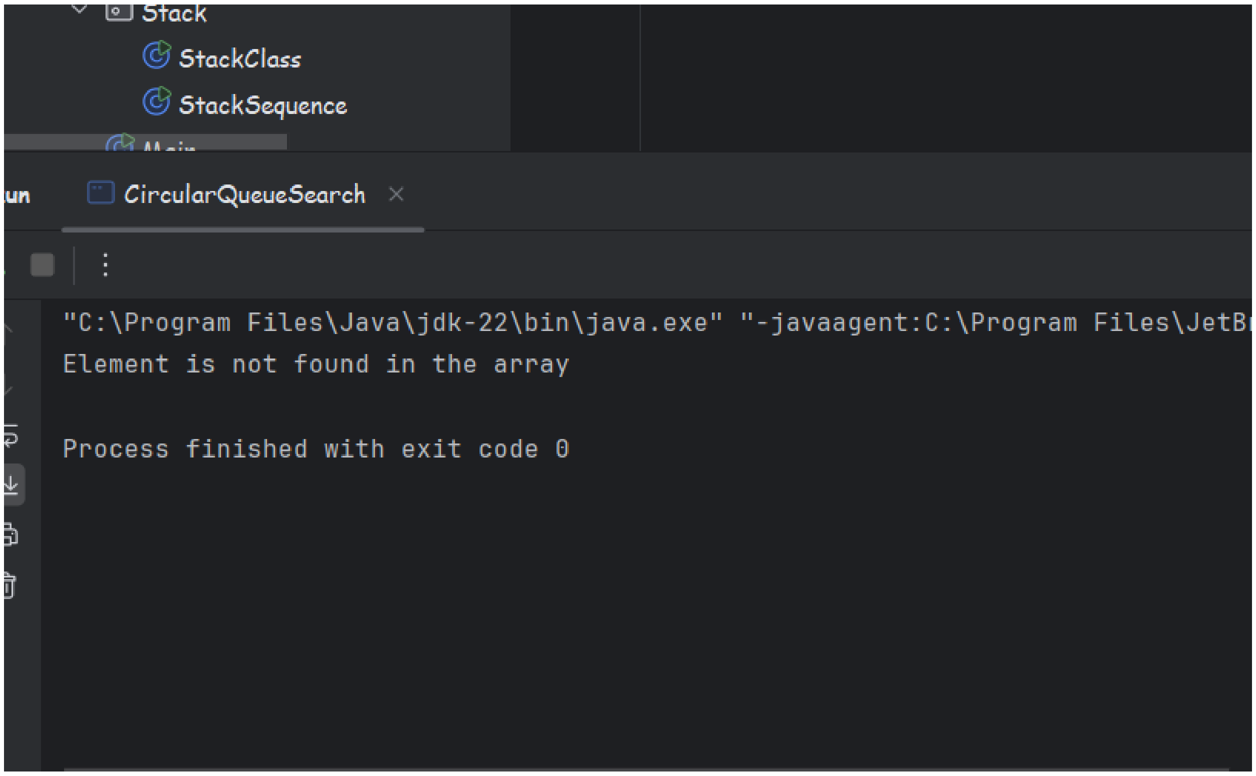
```
class ListNode {
  int val;
  ListNode next;

  ListNode(int val) {
    this.val = val;
    this.next = null;
  }
}
```

```
public class MergeSortedLinkedLists {

    public static ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        // Create a dummy node to act as the starting point of the merged list
        ListNode dummy = new ListNode(0);
        ListNode current = dummy;

        // Traverse both lists and link nodes in ascending order
        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;
        }

        // Link the remaining nodes, if any
        if (l1 != null) {
            current.next = l1;
        } else {
            current.next = l2;
        }
        // The merged list is next to the dummy node
        return dummy.next;
    }

    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);

        // Merge the two lists
        ListNode mergedList = mergeTwoLists(l1, l2);

        // Print the merged list
        while (mergedList != null) {
            System.out.print(mergedList.val + " ");
            mergedList = mergedList.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.

```java
package Queue;

public class CircularQueueSearch {


    static int findPivot(int arr[], int low, int high) {
        if (high < low) return -1;
        if (high == low) return low;

        int mid = (low + high) / 2;

        if (arr[mid] > arr[(mid + 1) % arr.length])
            return (mid + 1) % arr.length;


        if (arr[mid] < arr[(mid - 1 + arr.length) % arr.length])
            return mid;


        if (arr[low] >= arr[mid])
            return findPivot(arr, low, mid - 1);

        return findPivot(arr, mid + 1, high);
    }


    static int binarySearch(int arr[], int low, int high, int key) {
        if (high < low)
            return -1;

        int mid = (low + high) / 2;

        if (arr[mid] == key)
            return mid;

        if (key > arr[mid]) {
            if (arr[low] <= key)
                return binarySearch(arr, low, mid - 1, key);
            else
                return binarySearch(arr, (mid + 1) % arr.length, high, key);
        }

        return binarySearch(arr, low, mid - 1, key);
    }


    static int search(int arr[], int n, int key) {
        int pivot = findPivot(arr, 0, n - 1);

        if (pivot == -1) return -1;

        if (arr[pivot] <= key && key <= arr[n - 1])
            return binarySearch(arr, pivot, n - 1, key);

        return binarySearch(arr, 0, pivot - 1, key);
    }

    public static void main(String[] args) {
        int arr[] = {5, 6, 7, 8, 1, 2, 3, 4};
        int key = 3;

        int result = search(arr, arr.length, key);

        if (result == -1)
            System.out.println("Element is not found in the array");
        else
            System.out.println("Element is found at index " + result);
    }
}
```

OutPut:



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBr
Element is not found in the array

Process finished with exit code 0
```