

DATA STRUCTURES ASSIGNMENT

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.

```
LinkedListMiddle.java ×
1 package assignment;
2 class ListNode {
3     int val;
4     ListNode next;
5
6     ListNode(int x) {
7         val = x;
8     }
9 }
10 public class LinkedListMiddle {
11     public ListNode findMiddle(ListNode head) {
12         if (head == null)
13             return null;
14         ListNode slow = head, fast = head;
15         while (fast != null && fast.next != null) {
16             slow = slow.next;
17             fast = fast.next.next;
18         }
19         return slow;
20     }
21     public static void main(String[] args) {
22         // Creating a linked list: 1 -> 2 -> 3 -> 4 -> 5
23         ListNode head = new ListNode(1);
24         head.next = new ListNode(2);
25         head.next.next = new ListNode(3);
26         head.next.next.next = new ListNode(4);
27         head.next.next.next.next = new ListNode(5);
28         LinkedListMiddle solution = new LinkedListMiddle();
29         ListNode middle = solution.findMiddle(head);
30         if (middle != null) {
31             System.out.println("The middle element is: " + middle.val); // Output: The middle element is: 3
32         } else {
33             System.out.println("The list is empty.");
34         }
35     }
36 }
37
```

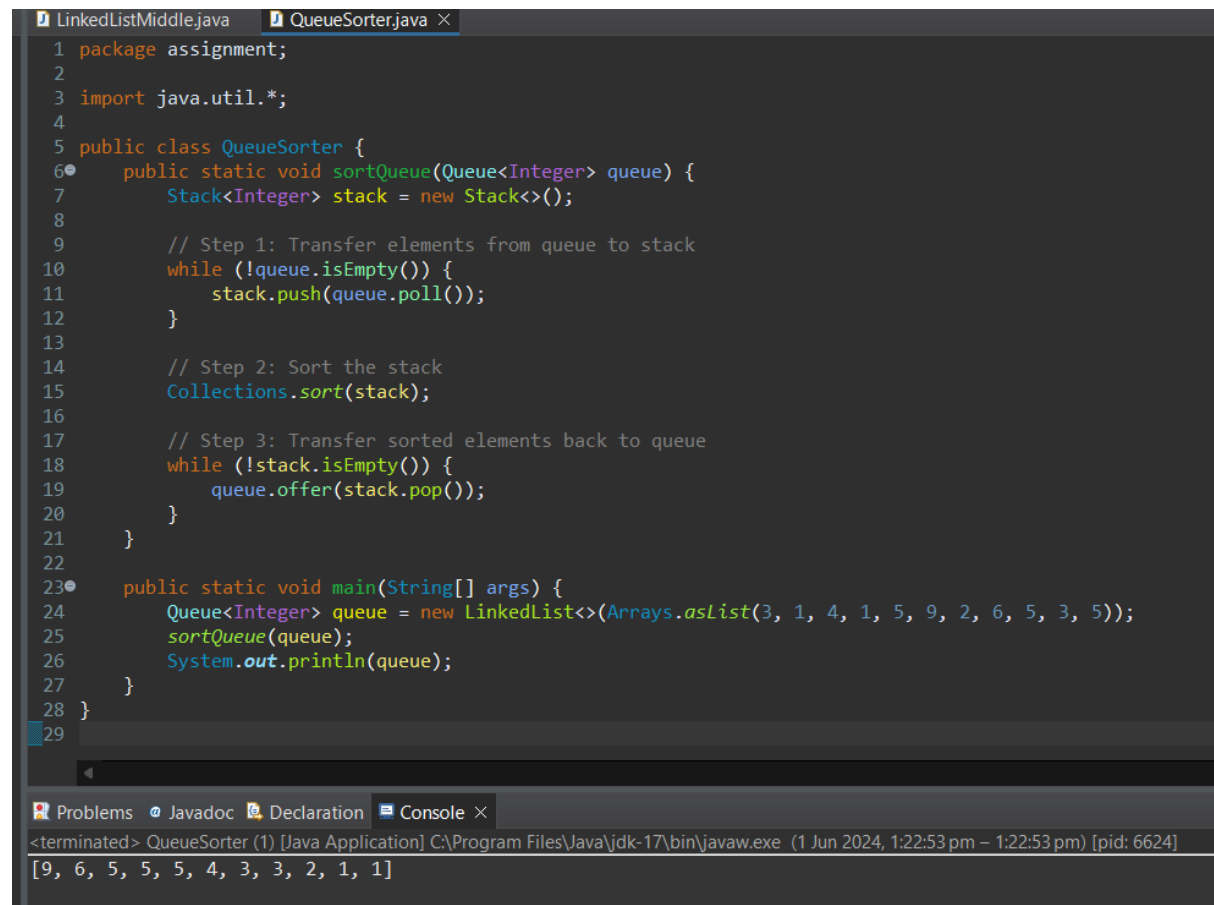
Problems Javadoc Declaration Console ×

<terminated> LinkedListMiddle (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (1 Jun 2024, 1:19:22 pm – 1:19:22 pm) [pid: 10716]

The middle element is: 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.



```
1 package assignment;
2
3 import java.util.*;
4
5 public class QueueSorter {
6     public static void sortQueue(Queue<Integer> queue) {
7         Stack<Integer> stack = new Stack<>();
8
9         // Step 1: Transfer elements from queue to stack
10        while (!queue.isEmpty()) {
11            stack.push(queue.poll());
12        }
13
14        // Step 2: Sort the stack
15        Collections.sort(stack);
16
17        // Step 3: Transfer sorted elements back to queue
18        while (!stack.isEmpty()) {
19            queue.offer(stack.pop());
20        }
21    }
22
23    public static void main(String[] args) {
24        Queue<Integer> queue = new LinkedList<>(Arrays.asList(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5));
25        sortQueue(queue);
26        System.out.println(queue);
27    }
28 }
29
```

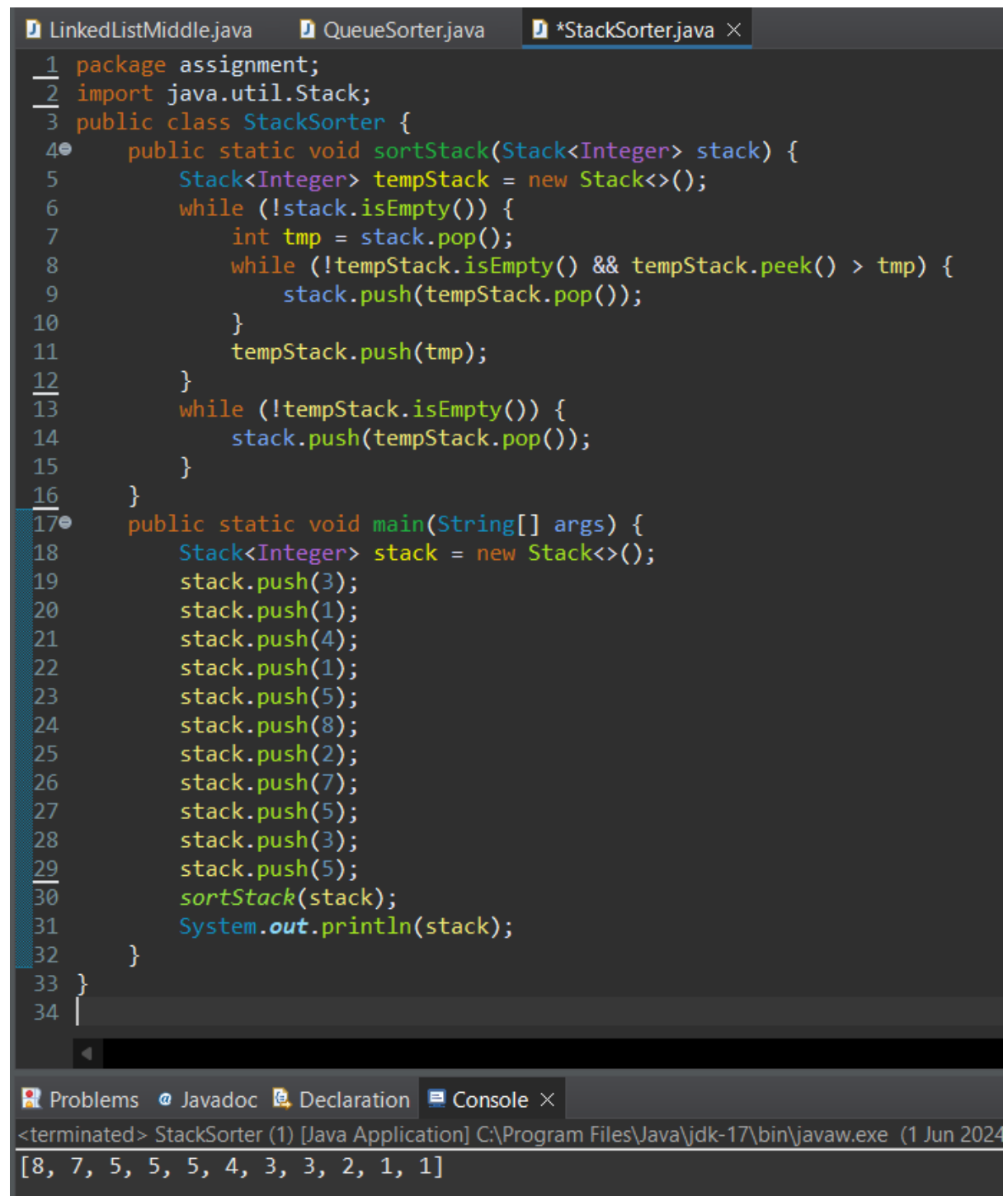
Problems Javadoc Declaration Console ×

<terminated> QueueSorter (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (1 Jun 2024, 1:22:53 pm – 1:22:53 pm) [pid: 6624]

[9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]

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.



```
1 package assignment;
2 import java.util.Stack;
3 public class StackSorter {
4     public static void sortStack(Stack<Integer> stack) {
5         Stack<Integer> tempStack = new Stack<>();
6         while (!stack.isEmpty()) {
7             int tmp = stack.pop();
8             while (!tempStack.isEmpty() && tempStack.peek() > tmp) {
9                 stack.push(tempStack.pop());
10            }
11            tempStack.push(tmp);
12        }
13        while (!tempStack.isEmpty()) {
14            stack.push(tempStack.pop());
15        }
16    }
17    public static void main(String[] args) {
18        Stack<Integer> stack = new Stack<>();
19        stack.push(3);
20        stack.push(1);
21        stack.push(4);
22        stack.push(1);
23        stack.push(5);
24        stack.push(8);
25        stack.push(2);
26        stack.push(7);
27        stack.push(5);
28        stack.push(3);
29        stack.push(5);
30        sortStack(stack);
31        System.out.println(stack);
32    }
33 }
34
```

Problems Javadoc Declaration Console

<terminated> StackSorter (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (1 Jun 2024)

[8, 7, 5, 5, 5, 4, 3, 3, 2, 1, 1]

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.

```
RemoveDuplicates.java ×
1 package assignment;
2 class List {
3     int val;
4     List next;
5     List(int x) {
6         val = x;
7     }
8 }
9 public class RemoveDuplicates {
10     public List deleteDuplicates(List head) {
11         List current = head;
12         while (current != null && current.next != null) {
13             if (current.val == current.next.val) {
14                 current.next = current.next.next;
15             } else {
16                 current = current.next;
17             }
18         }
19         return head;
20     }
21     public static void main(String[] args) {
22         List head = new List(1);
23         head.next = new List(1);
24         head.next.next = new List(2);
25         head.next.next.next = new List(3);
26         head.next.next.next.next = new List(3);
27         head.next.next.next.next.next = new List(4);
28         head.next.next.next.next.next.next = new List(4);
29         head.next.next.next.next.next.next.next = new List(4);
30         head.next.next.next.next.next.next.next.next = new List(5);
31         RemoveDuplicates solution = new RemoveDuplicates();
32         List newHead = solution.deleteDuplicates(head);
33         List current = newHead;
34         while (current != null) {
35             System.out.print(current.val + " ");
36             current = current.next;
37         }
38     }
39 }
```

Problems Javadoc Declaration Console ×

<terminated> RemoveDuplicates [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (1 Jun 2024, 1:10:10 PM)

1 2 3 4 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.

```
RemoveDuplicates.java  *SequenceInStack.java ×
1 package assignment;
2
3 import java.util.Stack;
4
5 public class SequenceInStack {
6     public static boolean isSequenceInStack(Stack<Integer> stack, int[] sequence) {
7         Stack<Integer> tempStack = new Stack<>();
8         int seqIndex = sequence.length - 1;
9
10        while (!stack.isEmpty()) {
11            int element = stack.pop();
12            tempStack.push(element);
13            if (element == sequence[seqIndex]) {
14                seqIndex--;
15                if (seqIndex < 0) {
16                    break;
17                }
18            }
19        }
20        while (!tempStack.isEmpty()) {
21            stack.push(tempStack.pop());
22        }
23        return seqIndex < 0;
24    }
25    public static void main(String[] args) {
26        Stack<Integer> stack = new Stack<>();
27        stack.push(1);
28        stack.push(2);
29        stack.push(3);
30        stack.push(4);
31        stack.push(5);
32
33        int[] sequence = { 3, 4, 5 };
34        System.out.println(isSequenceInStack(stack, sequence)); // Output: true
35    }
36 }
37
```

Problems Javadoc Declaration Console ×

<terminated> SequenceInStack [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (1 Jun 2024, 1:32:10 pm – 1:32:10 pm)
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).

```
MergeSortedLists.java ×
1 package assignment;
2 class Node {
3     int val;
4     Node next;
5     Node(int x) { val = x; }
6 }
7 public class MergeSortedLists {
8     public Node mergeTwoLists(Node l1, Node l2) {
9         Node dummy = new Node(0);
10        Node current = dummy;
11        while (l1 != null && l2 != null) {
12            if (l1.val < l2.val) {
13                current.next = l1;
14                l1 = l1.next;
15            } else {
16                current.next = l2;
17                l2 = l2.next;
18            }
19            current = current.next;
20        }
21        current.next = (l1 != null) ? l1 : l2;
22        return dummy.next;
23    }
24    public static void main(String[] args) {
25        // Creating first sorted linked list: 1 -> 3 -> 5
26        Node l1 = new Node(1);
27        l1.next = new Node(3);
28        l1.next.next = new Node(5);
29        // Creating second sorted linked list: 2 -> 4 -> 6
30        Node l2 = new Node(2);
31        l2.next = new Node(4);
32        l2.next.next = new Node(6);
33        MergeSortedLists solution = new MergeSortedLists();
34        Node mergedHead = solution.mergeTwoLists(l1, l2);
35        Node current = mergedHead;
36        while (current != null) {
37            System.out.print(current.val + " ");
38            current = current.next;
39        }
40    }
41 }
```

Console ×
<terminated> M
1 2 3 4 5 6

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.

```
CircularQueueBinarySearch.java ×
1 package assignment;
2
3 public class CircularQueueBinarySearch {
4     public static int search(int[] nums, int target) {
5         int left = 0, right = nums.length - 1;
6
7         while (left <= right) {
8             int mid = (left + right) / 2;
9             if (nums[mid] == target) {
10                 return mid;
11             }
12
13             if (nums[left] <= nums[mid]) {
14                 if (nums[left] <= target && target < nums[mid]) {
15                     right = mid - 1;
16                 } else {
17                     left = mid + 1;
18                 }
19             } else {
20                 if (nums[mid] < target && target <= nums[right]) {
21                     left = mid + 1;
22                 } else {
23                     right = mid - 1;
24                 }
25             }
26         }
27
28         return -1;
29     }
30
31     public static void main(String[] args) {
32         int[] nums = {4, 5, 6, 7, 0, 1, 2};
33         int target = 0;
34         System.out.println(search(nums, target));
35     }
36 }
37
38
```

Console ×
<terminated> CircularQueue
4

