

Assignment-2

Linked List Middle Element Search

Q. 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 :

```
package com.wipro.day7;

public class LinkedList {
    private Node tail;
    private Node head;
    private int length;

    class Node {
        int value;
        Node next;

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

    public LinkedList(int value) {
        super();
        Node newNode = new Node(value);
        head = newNode;
        tail = newNode;
        length = 1;
    }

    public Node findMiddle() {
```

```

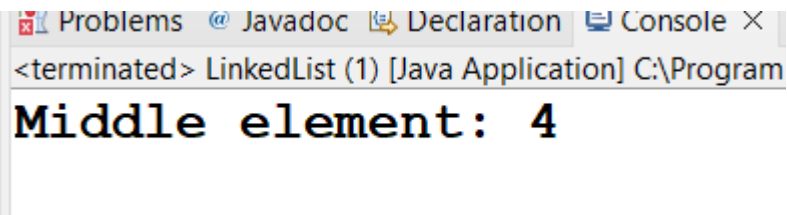
        if (length == 0) {
            return null;
        }
        Node slow = head;
        Node fast = head;
        while (fast != null && fast.next !=
null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

    public static void main(String[] args) {
        LinkedList ll = new LinkedList(1);
        ll.append(2);
        ll.append(3);
        ll.append(4);
        ll.append(5);
        ll.append(7);
        System.out.println("Middle element: " +
ll.findMiddle().value);
    }

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

```

Output:



```
<terminated> LinkedList (1) [Java Application] C:\Program
Middle element: 4
```

Assignment-3

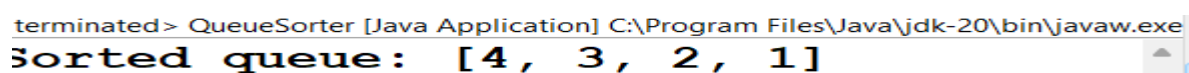
Queue Sorting with Limited Space

Q. 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:

```
1 import java.util.LinkedList;
2 import java.util.Queue;
3 import java.util.Stack;
4 public class QueueSorter {
5     public static void sortQueue(Queue<Integer> queue) {
6         Stack<Integer> stack = new Stack<>();
7
8         while (!queue.isEmpty()) {
9             int temp = queue.poll();
10
11             while (!stack.isEmpty() && stack.peek() > temp) {
12                 queue.add(stack.pop());
13             }
14
15             stack.push(temp);
16         }
17
18         while (!stack.isEmpty()) {
19             queue.add(stack.pop());
20         }
21     }
22     public static void main(String[] args) {
23         Queue<Integer> queue = new LinkedList<>();
24         queue.add(3);
25         queue.add(1);
26         queue.add(4);
27         queue.add(2);
28
29         sortQueue(queue);
30     }
31 }
```

OUTPUT:



```
terminated> QueueSorter [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe
Sorted queue: [4, 3, 2, 1]
```

Assignment-4

Stack Sorting In-Place

Q. 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:

```
1 package com.wipro.day7;
2 import java.util.Stack;
3
4 public class StackSorter {
5     public static void sortStack(Stack<Integer> stack) {
6         Stack<Integer> tempStack = new Stack<>();
7
8         while (!stack.isEmpty()) {
9             int temp = stack.pop();
10
11             while (!tempStack.isEmpty() && tempStack.peek() > temp) {
12                 stack.push(tempStack.pop());
13             }
14
15             tempStack.push(temp);
16         }
17
18         while (!tempStack.isEmpty()) {
19             stack.push(tempStack.pop());
20         }
21     }
22
23     public static void main(String[] args) {
24         Stack<Integer> stack = new Stack<>();
25         stack.push(3);
26         stack.push(1);
27         stack.push(4);
28         stack.push(2);
29
30         sortStack(stack);
31
32         System.out.println("Sorted stack: " + stack);
33     }
34 }
```

OUTPUT :

```
<terminated> StackSorter [Java Application] C:\Program Fi
Sorted stack: [4, 3, 2, 1]
```

Assignment-5

Removing Duplicates from a Sorted Linked List

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

Solution:

```
package com.wipro.day7;
public class LinkedList1 {
    private Node tail;
    private Node head;
    private int length;

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

    public LinkedList1(int value) {
        super();
        Node newNode = new Node(value);
        head = newNode;
        tail = newNode;
        length = 1;
    }

    public void removeDuplicates() {
        Node current = head;
        while (current != null && current.next !=
null) {
            if (current.value == current.next.value)
            {
                current.next = current.next.next;
                length--;
            } else {
                current = current.next;
            }
        }
    }
}
```

```

    }

    public static void main(String[] args) {
        LinkedList1 ll = new LinkedList1(1);
        ll.append(1);
        ll.append(2);
        ll.append(3);
        ll.append(3);
        ll.printList();
        ll.removeDuplicates();
        ll.printList();
    }

    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 void printList() {
        Node temp = head;
        while (temp != null) {
            System.out.print("--->" + temp.value);
            temp = temp.next;
        }
        System.out.println();
    }
}

```

OUTPUT:

```

C:\Users\user> java Application1
--->1--->1--->2--->3--->3
--->1--->2--->3

```

Assignment-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 :

```
1 package com.wipro.day7;
2 import java.util.Stack;
3 public class StackSequenceChecker {
4     public static boolean isSequenceInStack(Stack<Integer> stack, int[] sequence) {
5         Stack<Integer> tempStack = new Stack<>();
6         int index = sequence.length - 1;
7         while (!stack.isEmpty()) {
8             int value = stack.pop();
9             tempStack.push(value);
10            if (index >= 0 && value == sequence[index]) {
11                index--;
12            }
13            if (index < 0) {
14                break;
15            }
16        }
17        while (!tempStack.isEmpty()) {
18            stack.push(tempStack.pop());
19        }
20        return index < 0;
21    }
22    public static void main(String[] args) {
23        Stack<Integer> stack = new Stack<>();
24        stack.push(1);
25        stack.push(2);
26        stack.push(3);
27        stack.push(4);
28        stack.push(5);
29
30        int[] sequence = {3, 4, 5};
31        System.out.println("Sequence present: " + isSequenceInStack(stack, sequence)); /
32    }
33 }
34
```

OUTPUT:

```
<terminated> StackSequenceChecker [Java Applic
Sequence present: true
```

Assignment-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).

```
package com.wipro.day7;

public class LinkedList2 {
    private Node tail;
    private Node head;
    private int length;
    class Node {
        int value;
        Node next;

        public Node(int value) {
            super();
            this.value = value;
        }
    }
    public LinkedList2(int value) {
        super();
        Node newNode = new Node(value);
        head = newNode;
        tail = newNode;
        length = 1;
    }
    public static Node mergeTwoLists(Node l1, Node
12) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;

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



```

    }
    public static void main(String[] args) {
        LinkedList2 list1 = new LinkedList2(1);
        list1.append(3);
        list1.append(5);

        LinkedList2 list2 = new LinkedList2(2);
        list2.append(4);
        list2.append(6);
        Node mergedHead = mergeTwoLists(list1.head,
list2.head);

        Node current = mergedHead;
        while (current != null) {
            System.out.print("--->" + current.value);
            current = current.next;
        }
        System.out.println();
    }

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

```

OUTPUT:

```

<terminated> LinkedList2 [Java Application] C:\Program Files\Java\jdk-
|--->1--->2--->3--->4--->5--->6

```

Assignment-8

Circular Queue Binary Search

Q.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.day7;

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) / 2;
            if (nums[mid] > nums[right]) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }

        int pivot = left;
        left = 0;
        right = nums.length - 1;

        if (target >= nums[pivot] && target <=
nums[right]) {
            left = pivot;
        } else {
            right = pivot - 1;
        }

        while (left <= right) {
            int mid = (left + right) / 2;
            if (nums[mid] == target) {
                return mid;
            }
        }
    }
}
```

```

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

    return -1;
}

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

```

OUTPUT:

```

<terminated> CircularQueueBinarySearch
Index of target: 4

```

----END----