**LinkedList**

**Objective(s):**

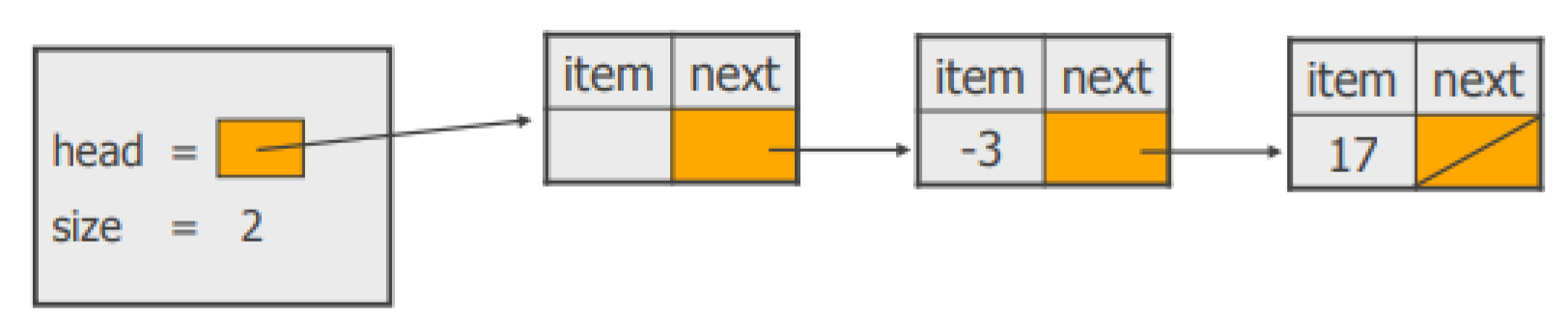
1. Understand LinkedList

# Lab Task(s):

**Exercise(s)**

1. In this task you will write a program that implements a variant of a linked list. This variant has a dummy node pointed to by the head link as shown in the following figure:

Linked list with a dummy first node:



This trick will allow your code to be a little simpler, not requiring a special case for add or remove operations. Your constructor method will be:

public LinkedList(){

head = new Node(null); size=0;

}

Methods you have to implement are:

**public boolean isEmpty();** // returns true if the list is empty, false otherwise

**public int size();** // returns the number of items in the list

**public void add(Object item);** // adds an item to the end of the list

**public void add(int index, Object item);** // adds an item to the list at the given index

// item is added at the given index; the indices start from 1.

# public void remove(int index);

// removes the item from the list that has the given index

**public void remove(Object item);** // removes an item from the list

// removes the first item in the list whose equal method matches

// that of the given item

**public List duplicate();** // creates a duplicate of the list

// returns a copy of the linked list

**public List duplicateReversed();** // creates a duplicate of the list with the nodes in reverse order

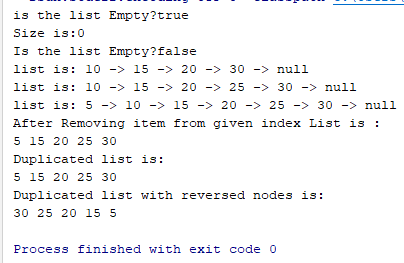
// returns a copy of the linked list with the nodes in reverse order

**}**

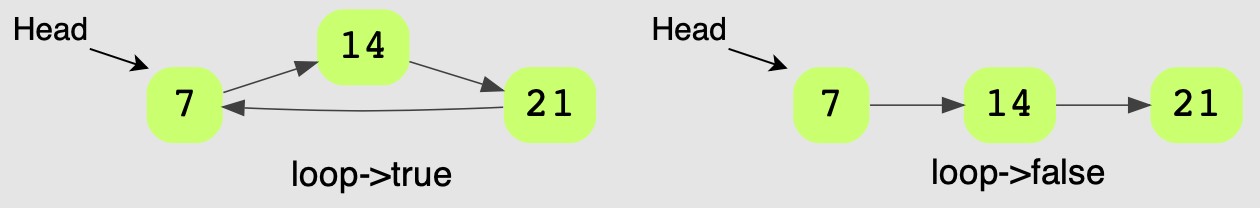
**CODE**

package SecondLabTasks;  
  
public class CustomLinkedList {  
  
 private Node head;  
  
  
  
 public static class Node {  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 this.next = null;  
 }  
 }  
  
  
 public boolean isEmpty() {  
  
 if (head == null) {  
  
 return true;  
  
 } else {  
 return false;  
 }  
  
  
 }  
  
 public int size(){  
  
 Node current=head;  
 int count=0;  
 while (current!=null){  
 count++;  
 current=current.next;  
  
 }  
  
 return count;  
  
 }  
  
 public void add(Node item){  
  
 if(head==null){  
  
 head=item;  
 }  
  
 else {  
 Node current=head;  
 while (current.next!=null){  
  
 current=current.next;  
 }  
 current.next=item;  
  
 }  
  
 }  
  
  
  
  
 public void add(int index, Node item) {  
 if (index < 1) {  
 System.*out*.println("Invalid index. Index must be greater than or equal to 1.");  
 return;  
 }  
 if (index == 1) {  
 item.next = head;  
 head = item;  
 }  
 else {  
 int currentIndex = 1;  
 Node current = head;  
 while (current != null && currentIndex < index - 1) {  
 current = current.next;  
 currentIndex++;  
  
  
 }  
 if (current == null) {  
 System.*out*.println("Index out of bounds. Cannot insert at index " + index + ".");  
 }  
 else {  
 item.next = current.next;  
 current.next = item;  
 }  
 }  
  
 *// Display the updated list* System.*out*.print("list is: ");  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.println("null");  
 }  
  
  
  
 public void Remove(int index){  
  
  
 if(index==1){  
 head= head.next;  
  
  
 }  
  
 else{  
  
 Node previous=head;  
 int count=1;  
  
 while(count<index-1){  
 previous=previous.next;  
 count++;  
 }  
  
 Node current=previous.next;  
 previous.next=current.next;  
  
  
  
 }  
  
 }  
  
  
 public void Display() {  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " ");  
 current = current.next;  
 }  
 System.*out*.println();  
 }  
  
  
  
  
  
  
  
 public CustomLinkedList duplicate(){  
 if(head == null){  
 System.*out*.println("No copy is formed.");  
 return null;  
 }  
 Node current = head;  
  
 CustomLinkedList duplicateList = new CustomLinkedList();  
 while(current!= null){  
 Node newNode = new Node(current.data);  
 duplicateList.add(newNode);  
 current=current.next;  
 }  
  
 return duplicateList;  
  
 }  
  
  
 public void print() {  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " ");  
 current = current.next;  
 }  
 System.*out*.println();  
 }  
  
  
  
 public CustomLinkedList duplicateReversed() {  
 CustomLinkedList duplicateList = duplicate(); *// Create a duplicate of the original list* duplicateList.reverse();  
 return duplicateList;  
 }  
  
  
  
 public void reverse() {  
 Node prev = null;  
 Node current = head;  
 Node next = null;  
  
 while (current != null) {  
 next = current.next;  
 current.next = prev;  
 prev = current;  
 current = next;  
 }  
  
 head = prev;  
  
  
  
  
 }  
  
  
  
 public static void main(String[] args) {  
 CustomLinkedList list1 = new CustomLinkedList();  
  
  
   
 System.*out*.println("is the list Empty?"+list1.isEmpty());  
  
 int length=list1.size();  
 System.*out*.println("Size is:"+length);  
  
 *//for adding items in add method* Node node1=new Node(10);  
 Node node2=new Node(20);  
 Node node3=new Node(30);  
  
 list1.add(node1);  
 list1.add(node2);  
 list1.add(node3);  
  
 System.*out*.println("Is the list Empty?"+list1.isEmpty());  
 *//it will return false because we have added node1,node2 node 3 in list  
  
  
 // Inserting a new node at index 2* Node newNode = new Node(15);  
 list1.add(2, newNode);  
  
 *// Inserting a new node at index 4* Node newNode2 = new Node(25);  
 list1.add(4, newNode2);  
  
 *// Inserting a new node at index 1* Node newNode3 = new Node(5);  
 list1.add(1, newNode3);  
  
  
 *//Removing item from given index* list1.Remove(2);  
 System.*out*.println("After Removing item from given index List is :");  
 list1.Display();  
  
  
  
 CustomLinkedList duplicateList=list1.duplicate();  
 System.*out*.println("Duplicated list is:");  
 duplicateList.print();  
  
  
 *// reversed nodes* CustomLinkedList reversedDuplicateList = list1.duplicateReversed();  
 System.*out*.println("Duplicated list with reversed nodes is:");  
 reversedDuplicateList.print();  
 }  
  
  
  
 }

**OUTPUT**

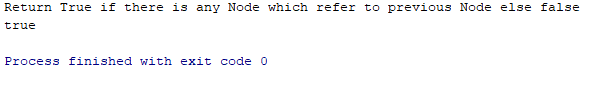


2: In this problem, you have to implement the **public Node detectLoop()** method, which will take a Singly linked list as input and find if there is a loop present in the list. A loop in a linked list occurs if any node contains a reference to any previous node, then a loop will be formed. An illustration is also provided for your understanding.

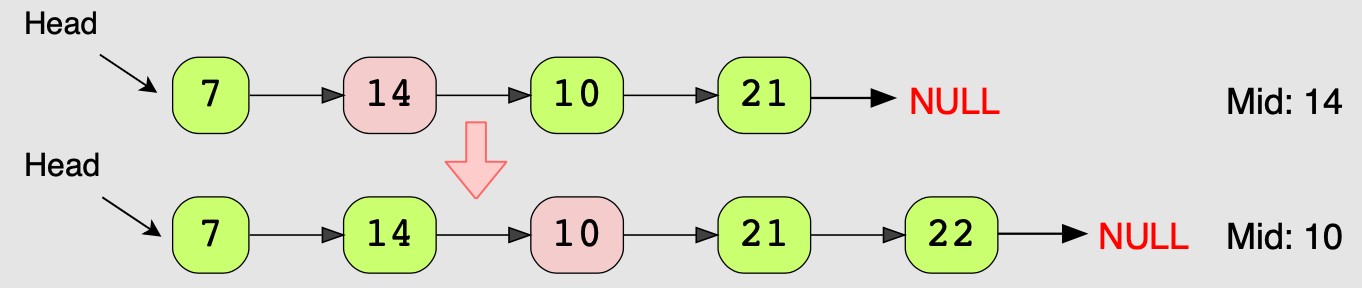


package SecondLabTasks;  
  
public class LoopDetectionTask {  
  
 Node head;  
  
 public class Node{  
  
 int data;  
 Node next;  
  
 public Node(int data){  
 this.data=data;  
 this.next=null;  
  
  
 }  
 }  
  
  
 public boolean detectLoop() {  
  
  
 Node fastPtr = head;  
 Node slowPtr = head;  
  
 while (fastPtr != null && fastPtr.next != null) {  
  
 fastPtr = fastPtr.next.next;  
 slowPtr = slowPtr.next;  
  
  
 if (slowPtr == fastPtr) {  
  
  
 return true;  
  
 }  
 }  
  
 return false;  
  
 }  
  
  
 public void createLoopInLinkedList(){  
  
 Node first =new Node(7);  
 Node second=new Node(14);  
 Node third=new Node(21);  
  
 head=first;  
 first.next=second;  
 second.next=third;  
 third.next=first;  
  
  
  
  
 }  
  
  
 public static void main(String[] args) {  
  
 LoopDetectionTask loop1=new LoopDetectionTask();  
 System.*out*.println("Return True if there is any Node which refer to previous Node else false");  
 loop1.createLoopInLinkedList();  
 System.*out*.println(loop1.detectLoop());  
  
 }  
  
  
}

**OUTPUT**

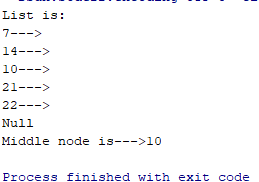


3: In this problem, you have to implement the **public Node findMiddle()** method, which will take a linked list as an input and return the value at the middle node of the list. An illustration is also provided for your understanding.

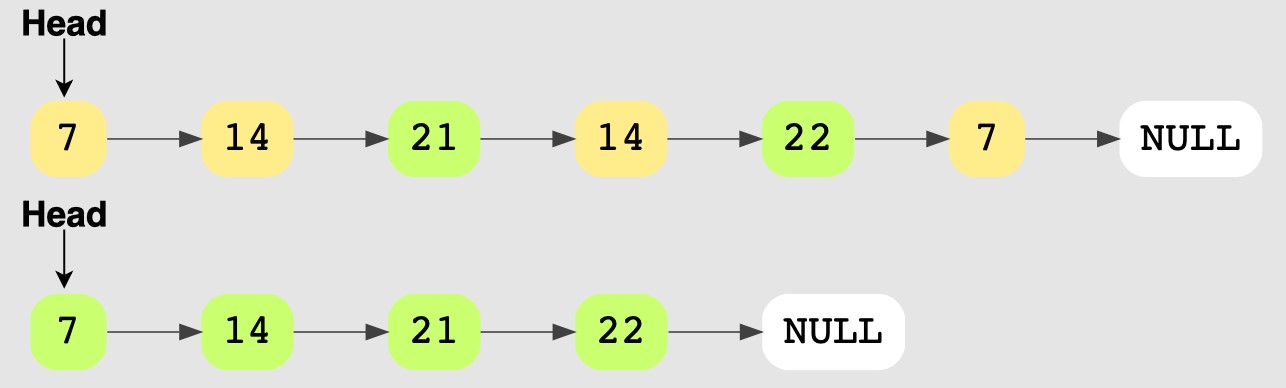


package SecondLabTasks;  
  
  
  
public class FindMiddleNode {  
  
 Node head;  
  
 public class Node {  
  
 int data;  
 Node next;  
  
 public Node(int data) {  
  
 this.data = data;  
 this.next = null;  
 }  
  
 }  
  
  
  
  
  
  
 public void printList( ){  
  
 if(head==null){  
  
 System.*out*.println("Null");  
 }  
  
  
 Node current=head;  
 while (null!=current){  
  
 System.*out*.println(current.data+"--->");  
 current=current.next;  
 }  
  
 System.*out*.println("Null");  
 }  
  
  
  
 public Node MiddleNode() {  
  
  
 if (head == null) {  
  
 return null;  
 }  
  
 Node fastPtr = head;  
 Node slowPtr = head;  
  
 while (fastPtr != null && fastPtr.next != null) {  
  
 slowPtr = slowPtr.next;  
 fastPtr = fastPtr.next.next;  
  
 }  
 return slowPtr;  
  
 }  
  
  
 public static void main(String[] args) {  
  
 FindMiddleNode object = new FindMiddleNode();  
  
  
 Node node1 = object.new Node(7);  
 FindMiddleNode.Node node2 = object.new Node(14);  
 FindMiddleNode.Node node3 = object.new Node(10);  
 FindMiddleNode.Node node4 = object.new Node(21);  
 FindMiddleNode.Node node5 = object.new Node(22);  
  
  
  
 object.head = node1;  
 node1.next = node2;  
 node2.next = node3;  
 node3.next = node4;  
 node4.next = node5;  
  
 System.*out*.println ("List is: ");  
 object.printList();  
  
 Node middleNode=object.MiddleNode();  
 System.*out*.println("Middle node is--->"+middleNode.data);  
  
  
  
  
}  
  
}

**OUTPUT**

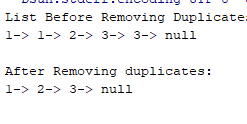


4: In this problem, you have to implement **public int removeDuplicates( )** method. This method will take a Singly linked list as input and remove all the elements that appear more than once in the list. An illustration is also provided for your understanding.



package SecondLabTasks;  
  
public class RemoveDuplicate {  
 ListNode head;  
 public class ListNode{  
 int data;  
 ListNode next;  
  
 ListNode(int data){  
 this.data=data;  
 this.next=null;  
 }  
 }  
  
 public void insertAtBeginning(int data){  
 ListNode newNode=new ListNode(data);  
 newNode.next=head;  
 head=newNode;  
  
 }  
 public void removeDuplicate(){  
 ListNode current=head;  
 while(current!=null && current.next!=null){  
 if(current.data == current.next.data){  
 current.next=current.next.next;  
 }  
 else{  
 current=current.next;  
 }  
 }  
 }  
 public void display() {  
 ListNode current = head;  
 while (current != null) {  
 System.*out*.print(current.data + "-> ");  
 current = current.next;  
 }  
 System.*out*.println("null");  
 System.*out*.println();  
 }  
  
  
 public static void main(String[] args) {  
 RemoveDuplicate list = new RemoveDuplicate();  
 list.insertAtBeginning(3);  
 list.insertAtBeginning(3);  
 list.insertAtBeginning(2);  
 list.insertAtBeginning(1);  
 list.insertAtBeginning(1);  
  
 System.*out*.println("List Before Removing Duplicates:");  
 list.display();  
  
 list.removeDuplicate();  
  
 System.*out*.println("After Removing duplicates:");  
 list.display();  
  
 }  
}

**OUTPUT**



6: You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

# Method Prototype:

ListNode **mergeTwoLists**(ListNode list1, ListNode list2)

# Example 1:

**Input:** list1 = [1,2,4], list2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

# CODE

package SecondLabTasks;  
  
public class SortingList {  
 public class ListNode {  
 int data;  
 ListNode next;  
  
 ListNode(int data) {  
 this.data = data;  
 this.next = null;  
 }  
 }  
 private ListNode head;  
  
 public void insertLast(int data) {  
 ListNode newNode = new ListNode(data);  
 if (head == null) {  
 head = newNode;  
 } else {  
 ListNode current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
 }  
  
 public void display() {  
 ListNode current = head;  
 while (current != null) {  
 System.*out*.print(+current.data +" ");  
 current = current.next;  
 }  
 System.*out*.println("null");  
  
 }  
  
 public ListNode mergeTwoLists(ListNode list1, ListNode list2) {  
 ListNode dummy = new ListNode(0);  
 ListNode tail=dummy;  
 while(list1!= null && list2!=null){  
 if(list1.data <= list2.data){  
 tail.next=list1;  
 list1=list1.next;  
 }  
 else{  
 tail.next=list2;  
 list2=list2.next;  
 }  
 tail=tail.next;  
  
 }  
 if(list1==null){  
 tail.next=list2;  
  
 }  
 else{  
 tail.next=list1;  
 }  
 return dummy.next;  
  
 }  
  
 public static void main(String[] args){  
 SortingList object1=new SortingList();  
 object1.insertLast(1);  
 object1.insertLast(2);  
 object1.insertLast(4);  
  
 SortingList object2=new SortingList();  
 object2.insertLast(1);  
 object2.insertLast(3);  
 object2.insertLast(4);  
   
 System.*out*.println("List 1:");  
 object1.display();

System.*out*.println();

System.*out*.println("List 2 :");  
 object2.display();  
  
 SortingList result = new SortingList();  
  
 result.head = result.mergeTwoLists(object1.head,object2.head);  
 System.*out*.println();  
 System.*out*.println("Result after Sorting two lists are:");  
 result.display();  
  
  
 }  
  
  
}

# OUTPUT

# 