Q1class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class MergeLinkedLists {

public static ListNode mergeLinkedLists(ListNode head1, ListNode head2) {

ListNode dummy = new ListNode(0);

ListNode current = dummy;

while (head1 != null && head2 != null) {

if (head1.val >= head2.val) {

current.next = new ListNode(head1.val);

head1 = head1.next;

} else {

current.next = new ListNode(head2.val);

head2 = head2.next;

}

current = current.next;

}

// Add the remaining nodes from the first linked list

while (head1 != null) {

current.next = new ListNode(head1.val);

head1 = head1.next;

current = current.next;

}

// Add the remaining nodes from the second linked list

while (head2 != null) {

current.next = new ListNode(head2.val);

head2 = head2.next;

current = current.next;

}

return dummy.next;

}

public static void main(String[] args) {

// Create the first linked list

ListNode list1 = new ListNode(5);

list1.next = new ListNode(2);

list1.next.next = new ListNode(3);

list1.next.next.next = new ListNode(8);

// Create the second linked list

ListNode list2 = new ListNode(1);

list2.next = new ListNode(7);

list2.next.next = new ListNode(4);

list2.next.next.next = new ListNode(5);

// Merge the linked lists

ListNode newList = mergeLinkedLists(list1, list2);

// Print the new linked list

while (newList != null) {

System.out.print(newList.val + " ");

newList = newList.next;

}

}

}

Q2class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class RemoveDuplicates {

public static ListNode removeDuplicates(ListNode head) {

if (head == null || head.next == null) {

return head;

}

ListNode current = head;

while (current != null && current.next != null) {

if (current.val == current.next.val) {

current.next = current.next.next;

} else {

current = current.next;

}

}

return head;

}

public static void main(String[] args) {

// Create the linked list

ListNode head = new ListNode(11);

head.next = new ListNode(11);

head.next.next = new ListNode(11);

head.next.next.next = new ListNode(21);

head.next.next.next.next = new ListNode(43);

head.next.next.next.next.next = new ListNode(43);

head.next.next.next.next.next.next = new ListNode(60);

// Remove duplicates

ListNode newList = removeDuplicates(head);

// Print the modified linked list

while (newList != null) {

System.out.print(newList.val + " ");

newList = newList.next;

}

}

}

Q3.class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class ReverseKNodes {

public static ListNode reverseKNodes(ListNode head, int k) {

if (head == null || k == 1) {

return head;

}

ListNode dummy = new ListNode(0);

dummy.next = head;

ListNode prev = dummy;

ListNode current = head;

int count = 0;

while (current != null) {

count++;

if (count % k == 0) {

prev = reverseNodes(prev, current.next);

current = prev.next;

} else {

current = current.next;

}

}

return dummy.next;

}

private static ListNode reverseNodes(ListNode prev, ListNode next) {

ListNode last = prev.next;

ListNode current = last.next;

while (current != next) {

last.next = current.next;

current.next = prev.next;

prev.next = current;

current = last.next;

}

return last;

}

public static void printList(ListNode head) {

while (head != null) {

System.out.print(head.val + " ");

head = head.next;

}

}

public static void main(String[] args) {

// Create the linked list

ListNode head = new ListNode(1);

head.next = new ListNode(2);

head.next.next = new ListNode(2);

head.next.next.next = new ListNode(4);

head.next.next.next.next = new ListNode(5);

head.next.next.next.next.next = new ListNode(6);

head.next.next.next.next.next.next = new ListNode(7);

head.next.next.next.next.next.next.next = new ListNode(8);

int k = 4;

// Reverse every k nodes

ListNode newHead = reverseKNodes(head, k);

// Print the modified linked list

printList(newHead);

}

}

Q4.class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class ReverseAlternateKNodes {

public static ListNode reverseAlternateKNodes(ListNode head, int k) {

if (head == null || k == 1) {

return head;

}

ListNode dummy = new ListNode(0);

dummy.next = head;

ListNode prev = dummy;

ListNode current = head;

boolean reverse = true;

int count = 0;

while (current != null) {

count++;

if (count % k == 0) {

if (reverse) {

prev = reverseNodes(prev, current.next);

current = prev.next;

} else {

prev = current;

current = current.next;

}

reverse = !reverse;

} else {

current = current.next;

}

}

return dummy.next;

}

private static ListNode reverseNodes(ListNode prev, ListNode next) {

ListNode last = prev.next;

ListNode current = last.next;

while (current != next) {

last.next = current.next;

current.next = prev.next;

prev.next = current;

current = last.next;

}

return last;

}

public static void printList(ListNode head) {

while (head != null) {

System.out.print(head.val + " ");

head = head.next;

}

}

public static void main(String[] args) {

// Create the linked list

ListNode head = new ListNode(1);

head.next = new ListNode(2);

head.next.next = new ListNode(3);

head.next.next.next = new ListNode(4);

head.next.next.next.next = new ListNode(5);

head.next.next.next.next.next = new ListNode(6);

head.next.next.next.next.next.next = new ListNode(7);

head.next.next.next.next.next.next.next = new ListNode(8);

head.next.next.next.next.next.next.next.next = new ListNode(9);

int k = 3;

// Reverse every alternate k nodes

ListNode newHead = reverseAlternateKNodes(head, k);

// Print the modified linked list

printList(newHead);

}

}

Q5class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class DeleteLastOccurrence {

public static ListNode deleteLastOccurrence(ListNode head, int key) {

if (head == null) {

return null;

}

ListNode prev = null;

ListNode last = null;

ListNode current = head;

while (current != null) {

if (current.val == key) {

last = prev;

}

prev = current;

current = current.next;

}

if (last == null) {

return head; // Key not found

}

if (last == head) {

head = head.next; // Remove head node

} else {

last.next = last.next.next; // Remove last occurrence

}

return head;

}

public static void printList(ListNode head) {

while (head != null) {

System.out.print(head.val + " ");

head = head.next;

}

}

public static void main(String[] args) {

// Create the linked list

ListNode head = new ListNode(1);

head.next = new ListNode(2);

head.next.next = new ListNode(3);

head.next.next.next = new ListNode(5);

head.next.next.next.next = new ListNode(2);

head.next.next.next.next.next = new ListNode(10);

int key = 2;

// Delete last occurrence of key

ListNode newHead = deleteLastOccurrence(head, key);

// Print the modified linked list

printList(newHead);

}

}

Q6.class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class MergeSortedLists {

public static ListNode mergeLists(ListNode a, ListNode b) {

// Create a dummy node as the head of the merged list

ListNode dummy = new ListNode(0);

ListNode tail = dummy;

// Merge the lists until either one becomes empty

while (a != null && b != null) {

if (a.val <= b.val) {

tail.next = a;

a = a.next;

} else {

tail.next = b;

b = b.next;

}

tail = tail.next;

}

// Append the remaining nodes of the non-empty list

if (a != null) {

tail.next = a;

}

if (b != null) {

tail.next = b;

}

return dummy.next; // Return the head of the merged list

}

public static void printList(ListNode head) {

while (head != null) {

System.out.print(head.val + " ");

head = head.next;

}

}

public static void main(String[] args) {

// Create the first sorted linked list

ListNode a = new ListNode(5);

a.next = new ListNode(10);

a.next.next = new ListNode(15);

// Create the second sorted linked list

ListNode b = new ListNode(2);

b.next = new ListNode(3);

b.next.next = new ListNode(20);

// Merge the lists

ListNode mergedHead = mergeLists(a, b);

// Print the merged list

printList(mergedHead);

}

}

Q7.class Node {

int data;

Node prev;

Node next;

Node(int data) {

this.data = data;

this.prev = null;

this.next = null;

}

}

public class ReverseDoublyLinkedList {

public static Node reverseList(Node head) {

if (head == null || head.next == null) {

return head; // Empty list or single node, no need to reverse

}

Node current = head;

Node newHead = null;

// Swap the prev and next pointers of each node

while (current != null) {

Node next = current.next;

current.next = current.prev;

current.prev = next;

newHead = current; // Update the new head

current = next;

}

return newHead; // Return the new head of the reversed list

}

public static void printList(Node head) {

while (head != null) {

System.out.print(head.data + " ");

head = head.next;

}

}

public static void main(String[] args) {

// Create the original doubly linked list

Node head = new Node(10);

Node second = new Node(8);

Node third = new Node(4);

Node fourth = new Node(2);

head.next = second;

second.prev = head;

second.next = third;

third.prev = second;

third.next = fourth;

fourth.prev = third;

// Print the original list

System.out.print("Original Linked list: ");

printList(head);

System.out.println();

// Reverse the list

Node newHead = reverseList(head);

// Print the reversed list

System.out.print("Reversed Linked list: ");

printList(newHead);

System.out.println();

}

}

Q8.class Node {

int data;

Node prev;

Node next;

Node(int data) {

this.data = data;

this.prev = null;

this.next = null;

}

}

public class DeleteNodeDoublyLinkedList {

public static Node deleteNode(Node head, int position) {

if (head == null) {

return null; // Empty list

}

if (position == 1) {

head = head.next;

if (head != null) {

head.prev = null;

}

return head;

}

Node current = head;

int currentPosition = 1;

// Traverse to the node at the given position

while (currentPosition < position && current != null) {

current = current.next;

currentPosition++;

}

if (current == null) {

return head; // Position exceeds the length of the list

}

// Adjust the pointers of the previous and next nodes

if (current.prev != null) {

current.prev.next = current.next;

}

if (current.next != null) {

current.next.prev = current.prev;

}

return head;

}

public static void printList(Node head) {

while (head != null) {

System.out.print(head.data + " ");

head = head.next;

}

}

public static void main(String[] args) {

// Create the doubly linked list

Node head = new Node(1);

Node second = new Node(3);

Node third = new Node(4);

head.next = second;

second.prev = head;

second.next = third;

third.prev = second;

// Print the original list

System.out.print("Original Linked list: ");

printList(head);

System.out.println();

// Delete the node at position 3

int position = 3;

head = deleteNode(head, position);

// Print the updated list

System.out.print("Updated Linked list: ");

printList(head);

System.out.println();

}

}