|  |
| --- |
| **Ex.No: 6 DATE: 13-09-23**  **LINKED LIST IMPLEMENTATION** |

Here, Node Class is used as static in all programs for the sole purpose of **better readability of code** and to **avoid unnecessary coupling between the two classes**. In real-time use, it could just be created as a separate public class and can be accessed by all others in the directory.

Also, all of these Linked List functions can be **combined together as a single public class** and can be accessed by other programs as per required.

1) Aim: To implement the following insertion functions in Linked List

**Insertion:**

Insert at the beginning.

Insert at the end.

Insert at a specific position.

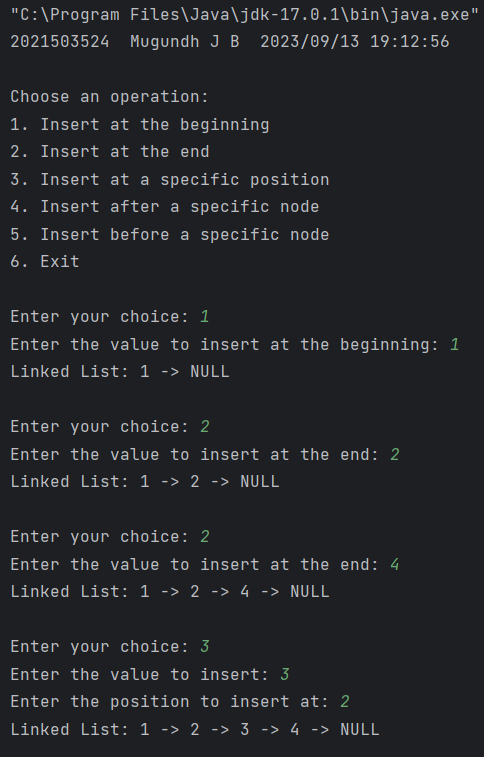
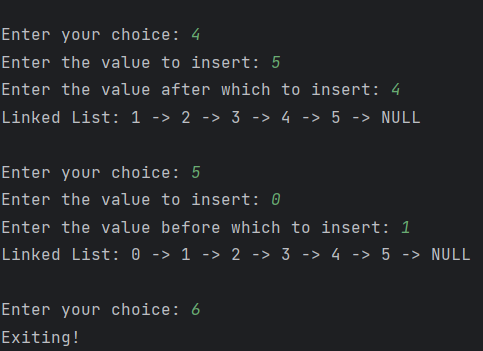
Insert after a specific node.

Insert before a specific node.

**Code:**

import java.util.\*;  
import java.time.format.DateTimeFormatter;  
import java.time.LocalDateTime;  
  
public class LinkedListInsertion3524 {  
 // Inner class representing a Node in the linked list  
 static class Node {  
 int data;  
 Node next;  
  
 Node(int d) {  
 data = d;  
 next = null;  
 }  
 }  
  
 Node head; // Head of the linked list  
  
 // Function to insert a node at the beginning of the linked list  
 public void insertAtBeginning(int data) {  
 Node newNode = new Node(data);  
 newNode.next = head; // New node points to the current head  
 head = newNode; // Update head to the new node  
 }  
  
 // Function to insert a node at the end of the linked list  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode; // If the list is empty, set the new node as the head  
 return;  
 }  
  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode; // Set the new node as the next of the last node  
 }  
  
 // Function to insert a node at a specific position in the linked list  
 public void insertAtPosition(int data, int pos) {  
 Node newNode = new Node(data);  
  
 if (pos <= 0) { // Insert at the beginning  
 newNode.next = head;  
 head = newNode;  
 return;  
 }  
  
 if (head == null) { // If the list is empty, insert at the beginning  
 head = newNode;  
 return;  
 }  
  
 Node current = head;  
 Node previous = null;  
 int count = 0;  
  
 while (count < pos && current != null) {  
 previous = current;  
 current = current.next;  
 count++;  
 }  
  
 if (current == null) { // Insert at the end  
 previous.next = newNode;  
 } else { // Insert at the specified position  
 newNode.next = current;  
 previous.next = newNode;  
 }  
 }  
  
 // Function to insert a node after a specific node with a target data value  
 public void insertAfterNode(int data, int targetData) {  
 Node newNode = new Node(data);  
 Node current = head;  
  
 while (current != null) {  
 if (current.data == targetData) {  
 newNode.next = current.next;  
 current.next = newNode;  
 return;  
 }  
 current = current.next;  
 }  
  
 // Target node not found, insert at the end  
 insertAtEnd(data);  
 }  
  
 // Function to insert a node before a specific node with a target data value  
 public void insertBeforeNode(int data, int targetData) {  
 Node newNode = new Node(data);  
  
 if (head != null && head.data == targetData) {  
 newNode.next = head;  
 head = newNode;  
 return;  
 }  
  
 Node current = head;  
 Node previous = null;  
  
 while (current != null) {  
 if (current.data == targetData) {  
 newNode.next = current;  
 if (previous != null) {  
 previous.next = newNode;  
 } else {  
 head = newNode; // If target is the head, update the head  
 }  
 return;  
 }  
 previous = current;  
 current = current.next;  
 }  
  
 // Target node not found, insert at the end  
 insertAtEnd(data);  
 }  
  
 // Function to display the elements of the linked list  
 public void display() {  
 System.*out*.print("Linked List: ");  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.print("NULL");  
 System.*out*.println();  
 }  
  
 public static void main(String[] args) {  
 // Code for getting the current date and time  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");  
 LocalDateTime now = LocalDateTime.*now*();  
 System.*out*.println("2021503524 " + "Mugundh J B " + dtf.format(now));  
  
 Scanner in = new Scanner(System.*in*);  
 LinkedListInsertion3524 list = new LinkedListInsertion3524();  
 int choice, data, position, targetData;  
  
 // Menu for user interaction  
 System.*out*.println("\nChoose an operation:");  
 System.*out*.println("1. Insert at the beginning");  
 System.*out*.println("2. Insert at the end");  
 System.*out*.println("3. Insert at a specific position");  
 System.*out*.println("4. Insert after a specific node");  
 System.*out*.println("5. Insert before a specific node");  
 System.*out*.println("6. Exit");  
  
 while (true) {  
 System.*out*.print("\nEnter your choice: ");  
 choice = in.nextInt();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter the value to insert at the beginning: ");  
 data = in.nextInt();  
 list.insertAtBeginning(data);  
 list.display();  
 break;  
  
 case 2:  
 System.*out*.print("Enter the value to insert at the end: ");  
 data = in.nextInt();  
 list.insertAtEnd(data);  
 list.display();  
 break;  
  
 case 3:  
 System.*out*.print("Enter the value to insert: ");  
 data = in.nextInt();  
 System.*out*.print("Enter the position to insert at: ");  
 position = in.nextInt();  
 list.insertAtPosition(data, position);  
 list.display();  
 break;  
  
 case 4:  
 System.*out*.print("Enter the value to insert: ");  
 data = in.nextInt();  
 System.*out*.print("Enter the value after which to insert: ");  
 targetData = in.nextInt();  
 list.insertAfterNode(data, targetData);  
 list.display();  
 break;  
  
 case 5:  
 System.*out*.print("Enter the value to insert: ");  
 data = in.nextInt();  
 System.*out*.print("Enter the value before which to insert: ");  
 targetData = in.nextInt();  
 list.insertBeforeNode(data, targetData);  
 list.display();  
 break;  
  
 case 6:  
 System.*out*.print("Exiting!");  
 System.*exit*(0);  
 break;  
  
 default:  
 System.*out*.print("Invalid choice. Please try again.");  
 break;  
 }  
 }  
 }  
}

Output:

2) Aim: To implement the following deletion functions in Linked List

**Deletion:**

Delete from the beginning.

Delete from the end.

Delete a specific element by value.

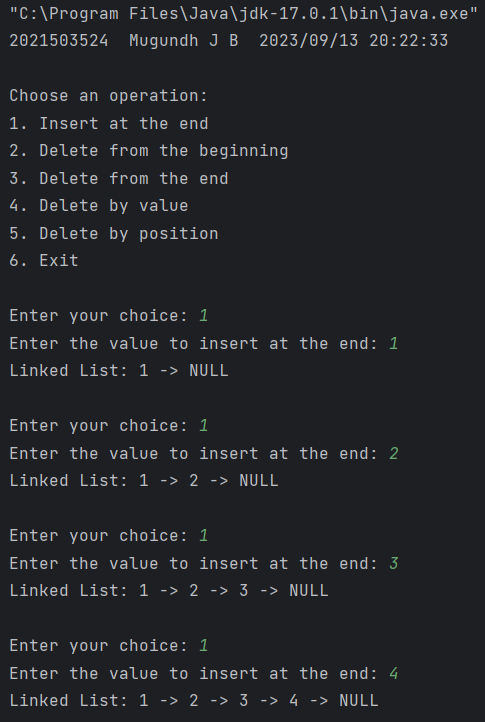
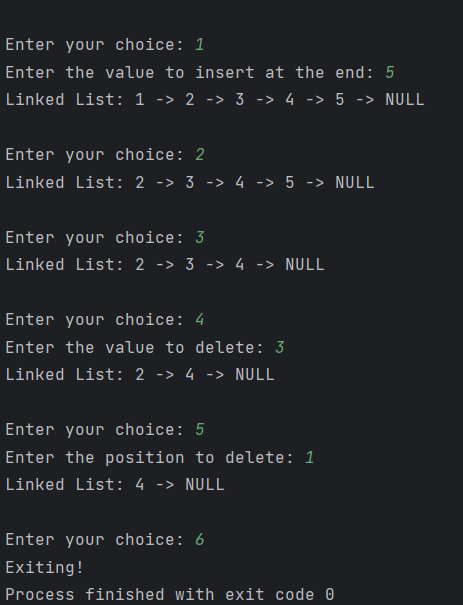
Delete a specific element by position.

Code:

import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Scanner;  
  
public class LinkedListDeletion3524 {  
 static class Node {  
 int data;  
 Node next;  
  
 Node(int d) {  
 data = d;  
 next = null;  
 }  
 }  
  
 Node head;  
  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 return;  
 }  
  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
  
 public void display() {  
 System.*out*.print("Linked List: ");  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.print("NULL");  
 System.*out*.println();  
 }  
  
 // Function to delete the first node (beginning) of the linked list  
 public void deleteFromBeginning() {  
 if (head == null) {  
 return;  
 }  
 head = head.next; // Update the head to the next node, effectively removing the first node.  
 }  
  
 // Function to delete the last node (end) of the linked list  
 public void deleteFromEnd() {  
 if (head == null) {  
 return;  
 }  
 if (head.next == null) {  
 head = null; // If there is only one node, set head to null.  
 return;  
 }  
 Node current = head;  
 Node previous = null;  
 while (current.next != null) {  
 previous = current;  
 current = current.next;  
 }  
 previous.next = null; // Set the next of the second-to-last node to null.  
 }  
  
 // Function to delete a node with a specific value from the linked list  
 public void deleteByValue(int targetData) {  
 if (head == null) {  
 return;  
 }  
 if (head.data == targetData) {  
 head = head.next; // If the target data is in the first node, update the head.  
 return;  
 }  
 Node current = head;  
 Node previous = null;  
 while (current != null) {  
 if (current.data == targetData) {  
 previous.next = current.next; // Skip the current node to remove it.  
 return;  
 }  
 previous = current;  
 current = current.next;  
 }  
 System.*out*.println("Element with value " + targetData + " not found.");  
 }  
  
 // Function to delete a node at a specific position from the linked list  
 public void deleteByPosition(int pos) {  
 if (head == null) {  
 return;  
 }  
 if (pos <= 0) {  
 System.*out*.println("Invalid position. Please provide a positive position.");  
 return;  
 }  
 if (pos == 1) {  
 head = head.next; // If deleting the first node, update the head.  
 return;  
 }  
 Node current = head;  
 Node previous = null;  
 int count = 1;  
 while (current != null) {  
 if (count == pos) {  
 previous.next = current.next; // Skip the current node to remove it.  
 return;  
 }  
 previous = current;  
 current = current.next;  
 count++;  
 }  
 System.*out*.println("Position " + pos + " is out of range.");  
 }  
  
 public static void main(String[] args) {  
 // Code for getting the current date and time  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");  
 LocalDateTime now = LocalDateTime.*now*();  
 System.*out*.println("2021503524 " + "Mugundh J B " + dtf.format(now));  
 Scanner in = new Scanner(System.*in*);  
 LinkedListDeletion3524 list = new LinkedListDeletion3524();  
 int choice, data, position;  
  
 System.*out*.println("\nChoose an operation:");  
 System.*out*.println("1. Insert at the end");  
 System.*out*.println("2. Delete from the beginning");  
 System.*out*.println("3. Delete from the end");  
 System.*out*.println("4. Delete by value");  
 System.*out*.println("5. Delete by position");  
 System.*out*.println("6. Exit");

while (true) {  
 System.*out*.print("\nEnter your choice: ");  
 choice = in.nextInt();  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter the value to insert at the end: ");  
 data = in.nextInt();  
 list.insertAtEnd(data);  
 list.display();  
 break;  
 case 2:  
 list.deleteFromBeginning();  
 list.display();  
 break;  
 case 3:  
 list.deleteFromEnd();  
 list.display();  
 break;  
 case 4:  
 System.*out*.print("Enter the value to delete: ");  
 data = in.nextInt();  
 list.deleteByValue(data);  
 list.display();  
 break;  
 case 5:  
 System.*out*.print("Enter the position to delete: ");  
 position = in.nextInt();  
 list.deleteByPosition(position);  
 list.display();  
 break;  
 case 6:  
 System.*out*.print("Exiting!");  
 System.*exit*(0);  
 break;  
 default:  
 System.*out*.print("Invalid choice. Please try again.");  
 break;  
 }  
 }  
 }  
}

Output:

3) Aim: To implement the following traversal and display functions in Linked List

**Traversal and Display:**

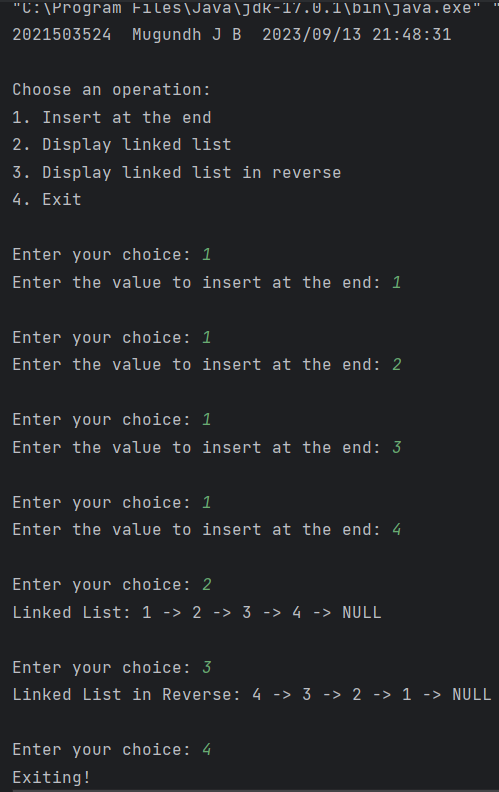
Traverse and print the elements in the linked list.

Reverse and print the elements in the linked list.

Code:

import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Scanner;  
  
public class LinkedListDisplay3524 {  
 static class Node {  
 int data;  
 Node next;  
  
 Node(int d) {  
 data = d;  
 next = null;  
 }  
 }  
  
 Node head;  
 // Function to insert a node at the end of the linked list  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode; // If the list is empty, set the new node as the head  
 return;  
 }  
  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode; // Traverse the list to find the last node and append the new node  
 }  
  
 // Function to display the linked list  
 public void display() {  
 System.*out*.print("Linked List: ");  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.print("NULL");  
 System.*out*.println();  
 }  
  
 // Function to display the linked list in reverse order using recursion  
 public void displayReverse(Node node) {  
 if (node == null) {  
 return; // Base case for recursion  
 }  
 displayReverse(node.next); // Recursively call the function for the next node  
 System.*out*.print(node.data + " -> "); // Print the data of the current node during backtracking  
 }  
  
 public static void main(String[] args) {  
 // Code for getting the current date and time  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");  
 LocalDateTime now = LocalDateTime.*now*();  
 System.*out*.println("2021503524 " + "Mugundh J B " + dtf.format(now));  
 Scanner in = new Scanner(System.*in*);  
 LinkedListDisplay3524 list = new LinkedListDisplay3524();  
 int choice, data;  
  
 System.*out*.println("\nChoose an operation:");  
 System.*out*.println("1. Insert at the end");  
 System.*out*.println("2. Display linked list");  
 System.*out*.println("3. Display linked list in reverse");  
 System.*out*.println("4. Exit");  
  
 while (true) {  
 System.*out*.print("\nEnter your choice: ");  
 choice = in.nextInt();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter the value to insert at the end: ");  
 data = in.nextInt();  
 list.insertAtEnd(data);  
 break;  
  
 case 2:  
 list.display();  
 break;  
  
 case 3:  
 System.*out*.print("Linked List in Reverse: ");  
 list.displayReverse(list.head);  
 System.*out*.print("NULL\n");  
 break;  
  
 case 4:  
 System.*out*.print("Exiting!");  
 System.*exit*(0);  
 break;  
  
 default:  
 System.*out*.print("Invalid choice. Please try again.");  
 break;  
 }  
 }  
 }  
}

Output:



4)Aim: To implement the following search and access functions in Linked List

**Search and Access:**

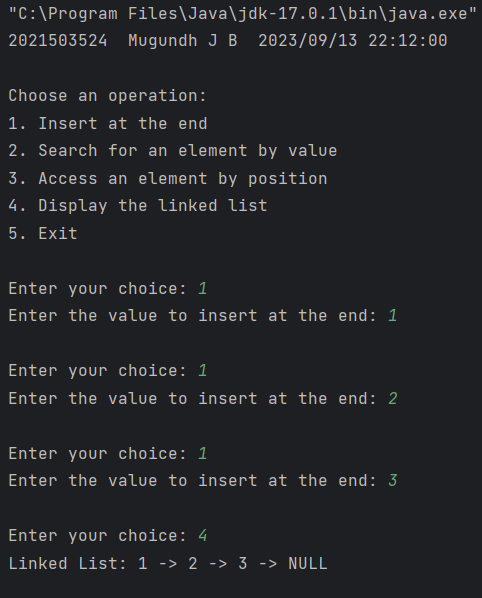
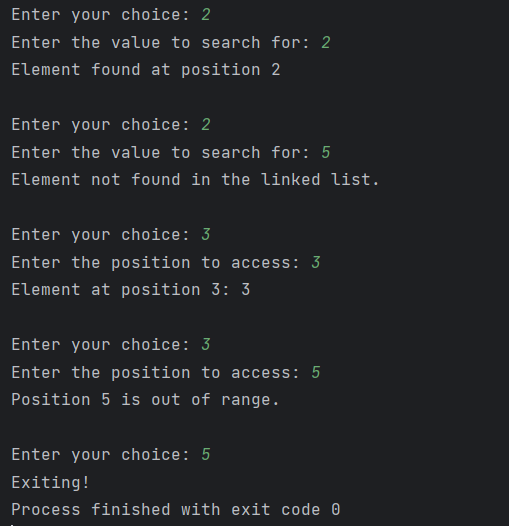
Search for an element by value.

Access an element by position.

Code:

import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Scanner;  
  
public class LinkedListSearchAccess3524 {  
 static class Node {  
 int data;  
 Node next;  
  
 Node(int d) {  
 data = d;  
 next = null;  
 }  
 }  
  
 Node head;  
  
 // Function to insert a node at the end of the linked list  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 return;  
 }  
  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
  
 // Function to display the linked list  
 public void display() {  
 System.*out*.print("Linked List: ");  
 LinkedListSearchAccess3524.Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.print("NULL");  
 System.*out*.println();  
 }  
  
 // Function to search for an element by value  
 public int searchByValue(int targetData) {  
 Node current = head;  
 int position = 1;  
 while (current != null) {  
 if (current.data == targetData) {  
 return position; // Element found, return its position  
 }  
 current = current.next;  
 position++;  
 }  
 return -1; // Element not found  
 }  
  
 // Function to access an element by position  
 public int accessByPosition(int pos) {  
 Node current = head;  
 int count = 1;  
 while (current != null) {  
 if (count == pos) {  
 return current.data; // Return the data of the element at the specified position  
 }  
 current = current.next;  
 count++;  
 }  
 // If the position is out of range, return a sentinel value (e.g., -1)  
 return -1;  
 }  
  
 public static void main(String[] args) {  
 // Code for getting the current date and time  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");  
 LocalDateTime now = LocalDateTime.*now*();  
 System.*out*.println("2021503524 " + "Mugundh J B " + dtf.format(now));  
 Scanner in = new Scanner(System.*in*);  
 LinkedListSearchAccess3524 list = new LinkedListSearchAccess3524();  
 int choice, data, position;  
  
 System.*out*.println("\nChoose an operation:");  
 System.*out*.println("1. Insert at the end");  
 System.*out*.println("2. Search for an element by value");  
 System.*out*.println("3. Access an element by position");  
 System.*out*.println("4. Display the linked list");  
 System.*out*.println("5. Exit");  
  
 while (true) {  
 System.*out*.print("\nEnter your choice: ");  
 choice = in.nextInt();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter the value to insert at the end: ");  
 data = in.nextInt();  
 list.insertAtEnd(data);  
 break;  
  
 case 2:  
 System.*out*.print("Enter the value to search for: ");  
 data = in.nextInt();  
 int foundPosition = list.searchByValue(data);  
 if (foundPosition != -1) {  
 System.*out*.println("Element found at position " + foundPosition);  
 } else {  
 System.*out*.println("Element not found in the linked list.");  
 }  
 break;  
  
 case 3:  
 System.*out*.print("Enter the position to access: ");  
 position = in.nextInt();  
 int accessedData = list.accessByPosition(position);  
 if (accessedData != -1) {  
 System.*out*.println("Element at position " + position + ": " + accessedData);  
 } else {  
 System.*out*.println("Position " + position + " is out of range.");  
 }  
 break;  
  
 case 4:  
 list.display();  
 break;  
  
 case 5:  
 System.*out*.print("Exiting!");  
 System.*exit*(0);  
 break;  
  
 default:  
 System.*out*.print("Invalid choice. Please try again.");  
 break;  
 }  
 }  
 }  
}

Output:

5) Aim: To implement the following length and counting functions in Linked List

**Length and Counting:**

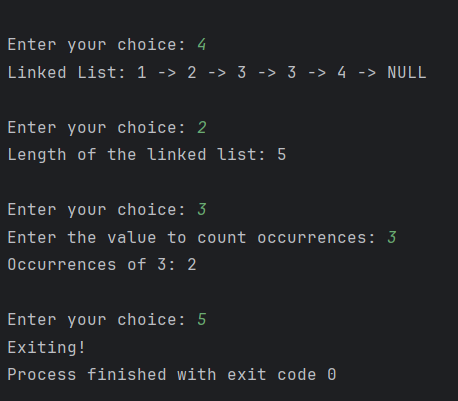
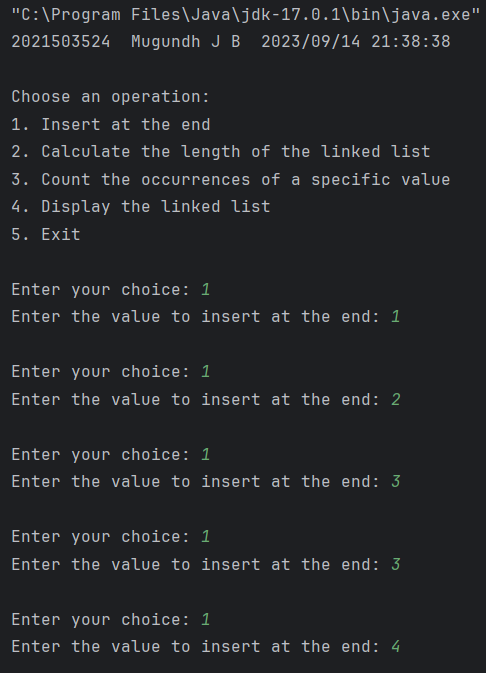
Find the length (number of nodes) of the linked list.

Count the occurrences of a specific value in the list.

Code:

import java.util.Scanner;  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
  
public class LinkedListLenOccurrence3524 {  
 static class Node {  
 int data;  
 Node next;  
  
 Node(int d) {  
 data = d;  
 next = null;  
 }  
 }  
  
 Node head;  
  
 // Function to insert a node at the end of the linked list  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 return;  
 }  
 Node cur = head;  
 while (cur.next != null) {  
 cur = cur.next;  
 }  
 cur.next = newNode;  
 }  
  
 // Function to display the linked list  
 public void display() {  
 System.*out*.print("Linked List: ");  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.print("NULL");  
 System.*out*.println();  
 }  
  
 // Function to calculate the length of the linked list  
 public int len() {  
 if (head == null) // If head is null, then length is 0  
 return 0;  
 int count = 1;  
 Node cur = head;  
 while (cur.next != null) {  
 cur = cur.next;  
 count += 1; // Counting the no of nodes  
 }  
 return count;  
 }  
  
 // Function to count the occurrences of a specific data value in the linked list  
 public int countOccurrence(int data) {  
 if (head == null) // If head is null, then occurrence is 0  
 return 0;  
 int count = 0;  
 Node cur = head;  
 while (cur != null) {  
 if (cur.data == data)   
 count += 1; // Counting the occurrence  
 cur = cur.next;  
 }  
 return count;  
 }  
  
 public static void main(String[] args) {  
 // Code for getting the current date and time  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");  
 LocalDateTime now = LocalDateTime.*now*();  
 System.*out*.println("2021503524 " + "Mugundh J B " + dtf.format(now));  
  
 Scanner in = new Scanner(System.*in*);  
 LinkedListLenOccurrence3524 list = new LinkedListLenOccurrence3524();  
 int choice, data;  
  
 System.*out*.println("\nChoose an operation:");  
 System.*out*.println("1. Insert at the end");  
 System.*out*.println("2. Calculate the length of the linked list");  
 System.*out*.println("3. Count the occurrences of a specific value");  
 System.*out*.println("4. Display the linked list");  
 System.*out*.println("5. Exit");  
  
 while (true) {  
 System.*out*.print("\nEnter your choice: ");  
 choice = in.nextInt();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter the value to insert at the end: ");  
 data = in.nextInt();  
 list.insertAtEnd(data);  
 break;  
  
 case 2:  
 int length = list.len();  
 System.*out*.println("Length of the linked list: " + length);  
 break;  
  
 case 3:  
 System.*out*.print("Enter the value to count occurrences: ");  
 data = in.nextInt();  
 int occurrence = list.countOccurrence(data);  
 System.*out*.println("Occurrences of " + data + ": " + occurrence);  
 break;  
  
 case 4:  
 list.display();  
 break;  
  
 case 5:  
 System.*out*.print("Exiting!");  
 System.*exit*(0);  
 break;  
  
 default:  
 System.*out*.print("Invalid choice. Please try again.");  
 break;  
 }  
 }  
 }  
}

Output:



**6)** Aim: To implement the following sorting and merging functions in Linked List

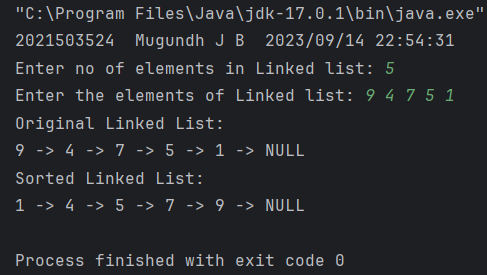
**Sorting and Merging:**

Sort the linked list (best sort).

Code:

import java.util.Scanner;  
  
public class MergeSortLinkedList3524 {  
 static class Node {  
 int data;  
 Node next;  
  
 Node(int d) {  
 data = d;  
 next = null;  
 }  
 }  
  
 Node head;  
  
 // Function to insert a node at the end of the linked list  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 return;  
 }  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
  
 // Function to display the linked list  
 public void display() {  
 System.*out*.print("Linked List: ");  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.print("NULL");  
 System.*out*.println();  
 }  
  
 // Function to perform Merge Sort on the linked list  
 public Node mergeSort(Node head) {  
 if (head == null || head.next == null) {  
 return head;  
 }  
  
 // Find the middle of the linked list  
 Node middle = findMiddle(head);  
 Node middleNext = middle.next;  
 middle.next = null;  
  
 // Recursively sort the left and right halves  
 Node left = mergeSort(head);  
 Node right = mergeSort(middleNext);  
  
 // Merge the sorted halves  
 return merge(left, right);  
 }  
  
 // Function to find the middle of the linked list  
 private Node findMiddle(Node head) {  
 if (head == null) {  
 return null;  
 }  
  
 Node slow = head;  
 Node fast = head;  
  
 while (fast.next != null && fast.next.next != null) {  
 slow = slow.next;  
 fast = fast.next.next;  
 }  
  
 return slow;  
 }  
  
 // Function to merge two sorted linked lists  
 private Node merge(Node left, Node right) {  
 if (left == null) {  
 return right;  
 }  
 if (right == null) {  
 return left;  
 }  
  
 if (left.data < right.data) {  
 left.next = merge(left.next, right);  
 return left;  
 } else {  
 right.next = merge(left, right.next);  
 return right;  
 }  
 }  
  
 public static void main(String[] args) {  
 MergeSortLinkedList3524 list = new MergeSortLinkedList3524();  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Enter no of elements in Linked list: ");  
 int n = scanner.nextInt();  
  
 System.*out*.print("Enter the elements of Linked list: ");  
  
 for (int i = 0; i < n; i++) {  
 int element = scanner.nextInt();  
 list.insertAtEnd(element);  
 }  
  
 System.*out*.print("Original ");  
 list.display();  
  
 // Perform Merge Sort on the linked list  
 list.head = list.mergeSort(list.head);  
  
 System.*out*.print("Sorted ");  
 list.display();  
  
 }  
}

Output:



7) Aim: To implement concatenation function in Linked List

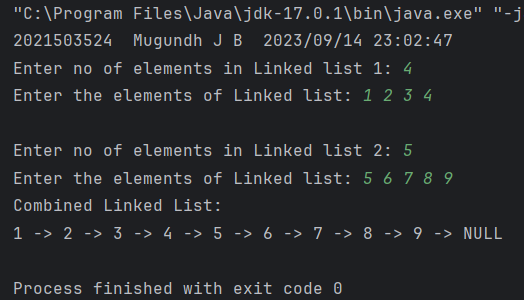
**Concatenation:**

Concatenate (combine) two linked lists together.

Code:

import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Scanner;  
  
public class LinkedListCombine3524 {  
 static class Node {  
 int data;  
 Node next;  
  
 Node(int d) {  
 data = d;  
 next = null;  
 }  
 }  
  
 Node head;  
  
 // Function to insert a node at the end of the linked list  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 return;  
 }  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
  
 // Function to display the linked list  
 public void display() {  
 System.*out*.print("Linked List:\n");  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.print("NULL");  
 System.*out*.println();  
 }  
  
 public Node combine(LinkedListCombine3524 l2){  
 if(head == null){ // If head is null, then no need to combine. Just, return 2nd LL  
 return l2.head;  
 }  
 Node cur = head;  
 while(cur.next != null){ // Traverse until end of the first LL  
 cur = cur.next;  
 }   
 cur.next = l2.head; // Point the first LL's next to 2nd LL  
  
 return head;  
 }  
  
 public static void main(String[] args) {  
 // Code for getting the current date and time  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");  
 LocalDateTime now = LocalDateTime.*now*();  
 System.*out*.println("2021503524 " + "Mugundh J B " + dtf.format(now));  
  
 LinkedListCombine3524 list1 = new LinkedListCombine3524();  
 LinkedListCombine3524 list2 = new LinkedListCombine3524();  
 LinkedListCombine3524 combined = new LinkedListCombine3524();  
 Scanner in = new Scanner(System.*in*);  
  
 System.*out*.print("Enter no of elements in Linked list 1: ");  
 int n = in.nextInt();  
  
 System.*out*.print("Enter the elements of Linked list: ");  
  
 for (int i = 0; i < n; i++) {  
 int element = in.nextInt();  
 list1.insertAtEnd(element);  
 }  
 System.*out*.print("\nEnter no of elements in Linked list 2: ");  
 n = in.nextInt();  
 System.*out*.print("Enter the elements of Linked list: ");  
 for (int i = 0; i < n; i++) {  
 int element = in.nextInt();  
 list2.insertAtEnd(element);  
 }  
   
 combined.head = list1.combine(list2);  
 System.*out*.print("Combined ");  
 combined.display();  
 }  
}

Output:



8) Aim: To implement duplication removal function in Linked List

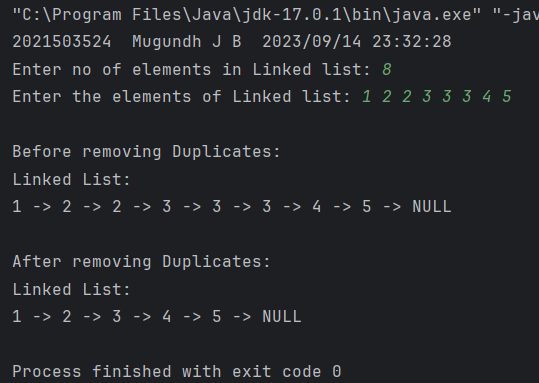
**Duplicate Removal:**

Remove duplicate elements from a linked list.

Code:

import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Scanner;  
  
public class LinkedListRemDuplicates3524 {  
 static class Node {  
 int data;  
 Node next;  
  
 Node(int d) {  
 data = d;  
 next = null;  
 }  
 }  
  
 Node head;  
  
 // Function to insert a node at the end of the linked list  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 return;  
 }  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
  
 // Function to display the linked list  
 public void display() {  
 System.*out*.print("Linked List:\n");  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.print("NULL");  
 System.*out*.println();  
 }  
  
 public void removeDuplicates(){  
 if(head == null || head.next == null) // If no element or only one element, no possibility for duplicates  
 return;  
 Node curNode = head;  
 while(curNode.next != null){ // Outer traversal  
 Node nextNode = curNode;  
 while(nextNode.next != null){ // Inner traversal for next right elements of current node  
 if(nextNode.next.data == curNode.data) {  
 nextNode.next = nextNode.next.next;  
 }  
 else  
 nextNode = nextNode.next;  
 }  
 curNode = curNode.next; // Moving to next node  
 }  
 }  
  
 public static void main(String[] args) {  
 // Code for getting the current date and time  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");  
 LocalDateTime now = LocalDateTime.*now*();  
 System.*out*.println("2021503524 " + "Mugundh J B " + dtf.format(now));  
  
 LinkedListRemDuplicates3524 list = new LinkedListRemDuplicates3524();  
 Scanner in = new Scanner(System.*in*);  
  
 System.*out*.print("Enter no of elements in Linked list: ");  
 int n = in.nextInt();  
  
 System.*out*.print("Enter the elements of Linked list: ");  
  
 for (int i = 0; i < n; i++) {  
 int element = in.nextInt();  
 list.insertAtEnd(element);  
 }  
  
 System.*out*.println("\nBefore removing Duplicates: ");  
 list.display();  
  
 list.removeDuplicates();  
  
 System.*out*.println("\nAfter removing Duplicates: ");  
 list.display();  
 }  
  
}

Output:



9) Aim: To implement polynomial addition and multiplication in Linked List

**Polynomial Representation:**

Implement polynomial addition and multiplication using linked lists.

Code:

import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Scanner;  
  
public class PolynomialOperations3524 {  
 static class Node {  
 int coefficient;  
 int exponent;  
 Node next;  
  
 Node(int coef, int exp) {  
 coefficient = coef;  
 exponent = exp;  
 next = null;  
 }  
 }  
  
 Node head;  
  
 // Function to insert a term into the polynomial linked list  
 public void insertTerm(int coef, int exp) {  
 Node newNode = new Node(coef, exp);  
 if (head == null) {  
 head = newNode;  
 return;  
 }  
  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
  
 // Function to display the polynomial  
 public void display() {  
 System.*out*.print("Polynomial: ");  
 Node current = head;  
 while (current != null) {  
 if (current.coefficient != 0) {  
 if(current.exponent != 0) {  
 System.*out*.print(current.coefficient + "x^" + current.exponent);  
 if (current.next != null && current.next.coefficient != 0) {  
 System.*out*.print(" + ");  
 }  
 }else{  
 System.*out*.print(current.coefficient);  
 }  
 }  
 current = current.next;  
 }  
 System.*out*.println();  
 }  
  
 // Function to add two polynomials  
 public static PolynomialOperations3524 addPolynomials(PolynomialOperations3524 poly1, PolynomialOperations3524 poly2) {  
 PolynomialOperations3524 result = new PolynomialOperations3524();  
 Node current1 = poly1.head;  
 Node current2 = poly2.head;  
  
 while (current1 != null && current2 != null) {  
 if (current1.exponent > current2.exponent) {  
 result.insertTerm(current1.coefficient, current1.exponent);  
 current1 = current1.next;  
 } else if (current2.exponent > current1.exponent) {  
 result.insertTerm(current2.coefficient, current2.exponent);  
 current2 = current2.next;  
 } else {  
 int sumCoeff = current1.coefficient + current2.coefficient;  
 if (sumCoeff != 0) {  
 result.insertTerm(sumCoeff, current1.exponent);  
 }  
 current1 = current1.next;  
 current2 = current2.next;  
 }  
 }  
  
 // Append any remaining terms from poly1 or poly2  
 while (current1 != null) {  
 result.insertTerm(current1.coefficient, current1.exponent);  
 current1 = current1.next;  
 }  
 while (current2 != null) {  
 result.insertTerm(current2.coefficient, current2.exponent);  
 current2 = current2.next;  
 }  
  
 return result;  
 }  
  
 // Function to multiply two polynomials  
 public static PolynomialOperations3524 multiplyPolynomials(PolynomialOperations3524 poly1, PolynomialOperations3524 poly2) {  
 PolynomialOperations3524 result = new PolynomialOperations3524();  
 Node current1 = poly1.head;  
  
 while (current1 != null) {  
 Node current2 = poly2.head;  
 while (current2 != null) {  
 int productCoeff = current1.coefficient \* current2.coefficient;  
 int productExp = current1.exponent + current2.exponent;  
 result.insertTerm(productCoeff, productExp);  
 current2 = current2.next;  
 }  
 current1 = current1.next;  
 }  
  
 return result;  
 }  
  
 public static void main(String[] args) {  
 // Code for getting the current date and time  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd HH:mm:ss");  
 LocalDateTime now = LocalDateTime.*now*();  
 System.*out*.println("2021503524 " + "Mugundh J B " + dtf.format(now));  
   
 Scanner in = new Scanner(System.*in*);  
 PolynomialOperations3524 poly1 = new PolynomialOperations3524();  
 PolynomialOperations3524 poly2 = new PolynomialOperations3524();  
  
 // Input for the first polynomial  
 System.*out*.print("Enter the number of terms in the first polynomial: ");  
 int n1 = in.nextInt();  
 System.*out*.println("Enter the terms for the first polynomial (like coeff, exp => ex : 5 2 , which means 5x^2) :");  
 for (int i = 0; i < n1; i++) {  
 int coef = in.nextInt();  
 int exp = in.nextInt();  
 poly1.insertTerm(coef, exp);  
 }  
  
 // Input for the second polynomial  
 System.*out*.print("Enter the number of terms in the second polynomial: ");  
 int n2 = in.nextInt();  
 System.*out*.println("Enter the terms for the second polynomial (like coeff, exp => ex : 5 2 , which means 5x^2) :");  
 for (int i = 0; i < n2; i++) {  
 int coef = in.nextInt();  
 int exp = in.nextInt();  
 poly2.insertTerm(coef, exp);  
 }  
  
 // Perform polynomial addition  
 PolynomialOperations3524 sum = *addPolynomials*(poly1, poly2);  
 System.*out*.print("Sum of polynomials: ");  
 sum.display();  
  
 // Perform polynomial multiplication  
 PolynomialOperations3524 product = *multiplyPolynomials*(poly1, poly2);  
 System.*out*.print("Product of polynomials: ");  
 product.display();  
  
 }  
}

Output:

