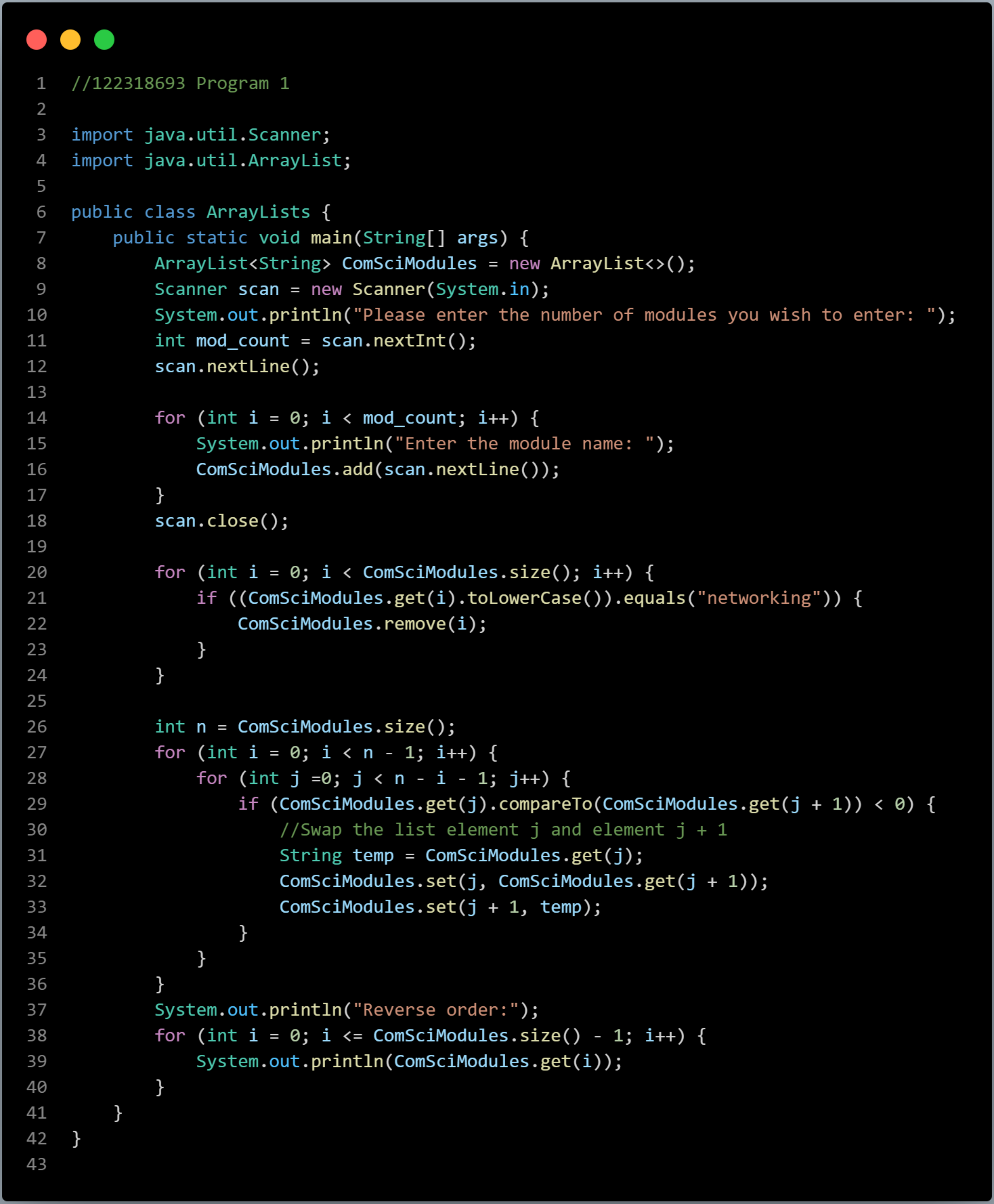
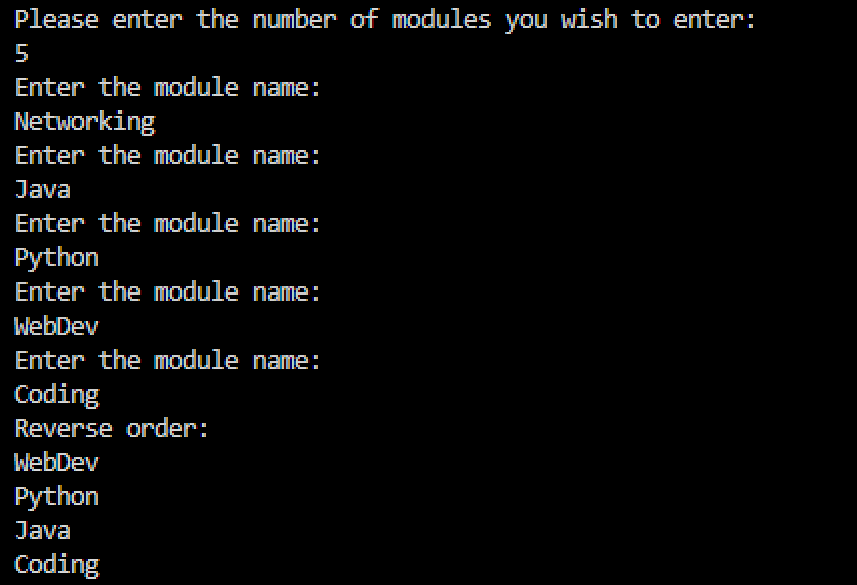
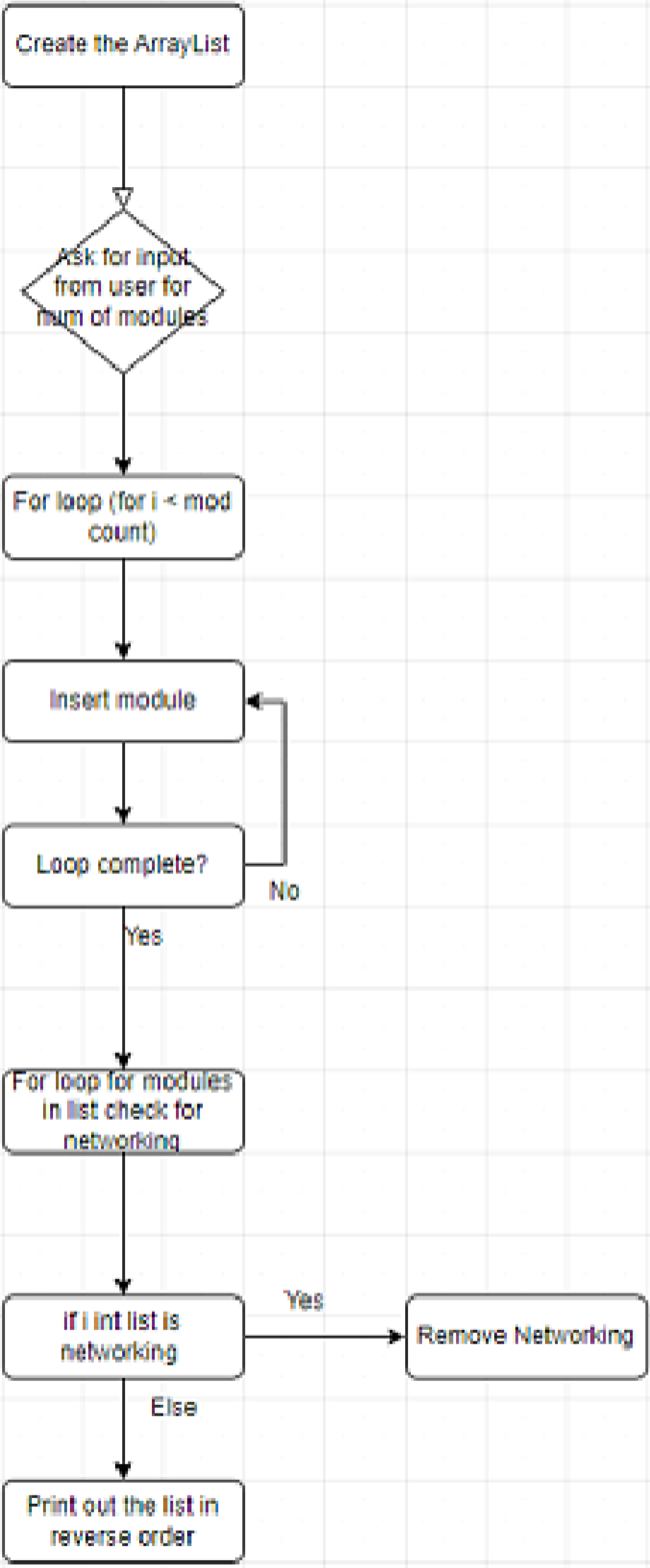


```java
//122318693 Program 1

import java.util.Scanner;
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<String> ComSciModules = new ArrayList<>();
        Scanner scan = new Scanner(System.in);
        System.out.println("Please enter the number of modules you wish to enter: ");
        int mod_count = scan.nextInt();
        scan.nextLine();

        for (int i = 0; i < mod_count; i++) {
            System.out.println("Enter the module name: ");
            ComSciModules.add(scan.nextLine());
        }
        scan.close();

        for (int i = 0; i < ComSciModules.size(); i++) {
            if ((ComSciModules.get(i).toLowerCase()).equals("networking")) {
                ComSciModules.remove(i);
            }
        }

        int n = ComSciModules.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j =0; j < n - i - 1; j++) {
                if (ComSciModules.get(j).compareTo(ComSciModules.get(j + 1)) < 0) {
                    //Swap the list element j and element j + 1
                    String temp = ComSciModules.get(j);
                    ComSciModules.set(j, ComSciModules.get(j + 1));
                    ComSciModules.set(j + 1, temp);
                }
            }
        }
        System.out.println("Reverse order:");
        for (int i = 0; i <= ComSciModules.size() - 1; i++) {
            System.out.println(ComSciModules.get(i));
        }
    }
}

```



```
Please enter the number of modules you wish to enter:
5
Enter the module name:
Networking
Enter the module name:
Java
Enter the module name:
Python
Enter the module name:
WebDev
Enter the module name:
Coding
Reverse order:
WebDev
Python
Java
Coding
```


Create the ArrayList
Ask for input from user for num of modules
For loop (for i < mod count)
Insert module
Loop complete?
No
Yes
For loop for modules in list check for networking
If i int list is networking
Yes
Remove Networking
Else
Print out the list in reverse order

Program 1

ArrayList Initialization:

An ArrayList named ComSciModules is initialized to store strings (module names).

User Input:

The program prompts the user to enter the number of modules they wish to input.

It reads the input count using a Scanner.

Module Input Loop:

A loop runs mod_count times, where the user is prompted to input each module name

one by one.

The module names are added to the ComSciModules ArrayList using the add() method.

Removing "Networking" Module:

After all modules are input, another loop iterates through the list.

If a module name (converted to lowercase) equals "networking", it's removed from the

list using the remove() method.

Sorting in Reverse Order:

The program sorts the modules in reverse alphabetical order.

It uses a bubble sort algorithm:

Two nested loops are used to compare adjacent elements and swap them if necessary.

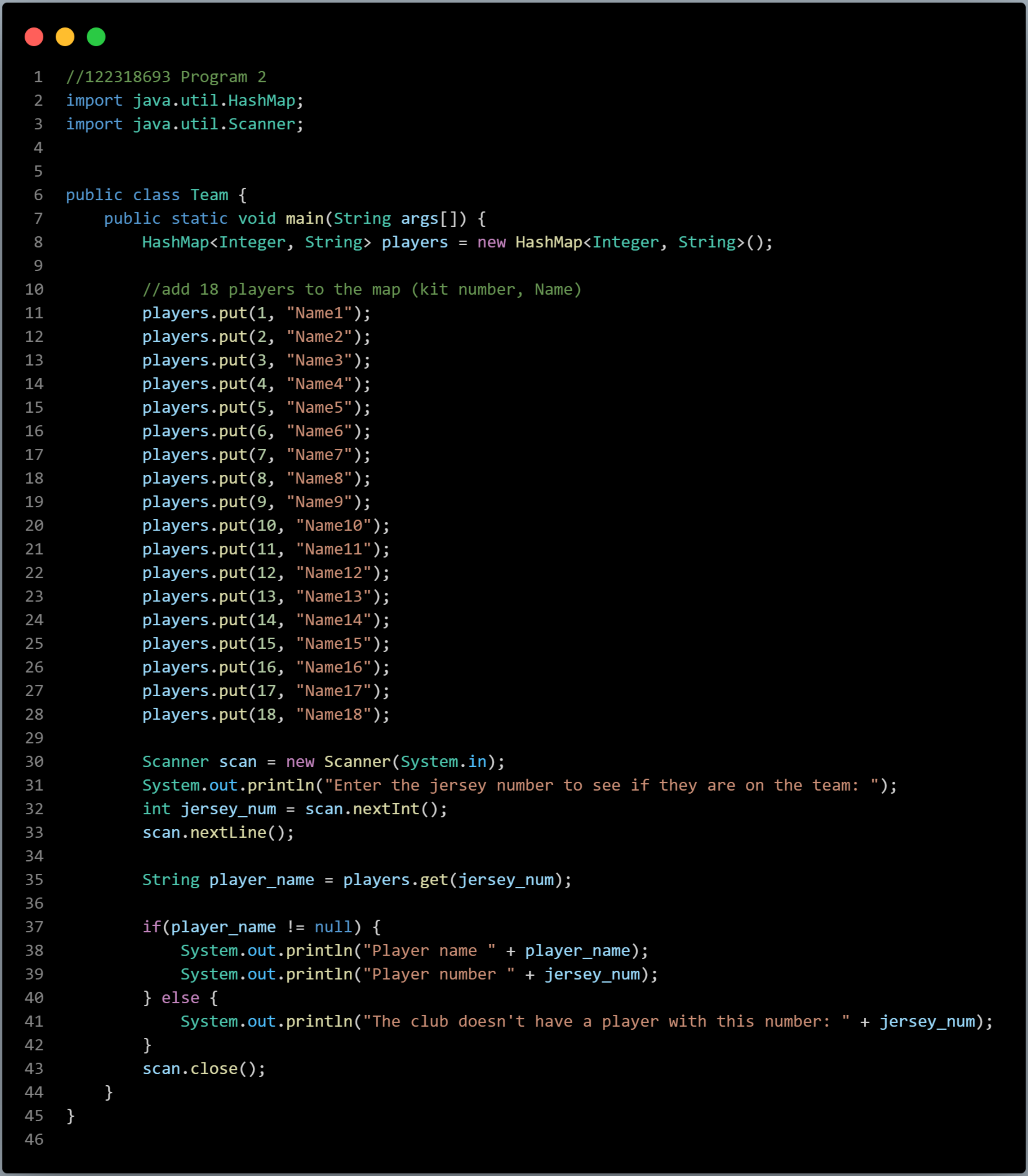
Sorting is performed based on lexicographical comparison (compareTo() method).

The loop stops when no more swaps are needed.

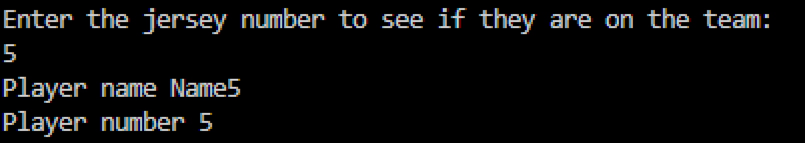
Output:

Finally, the sorted (in reverse order) module names are printed out one by one using a

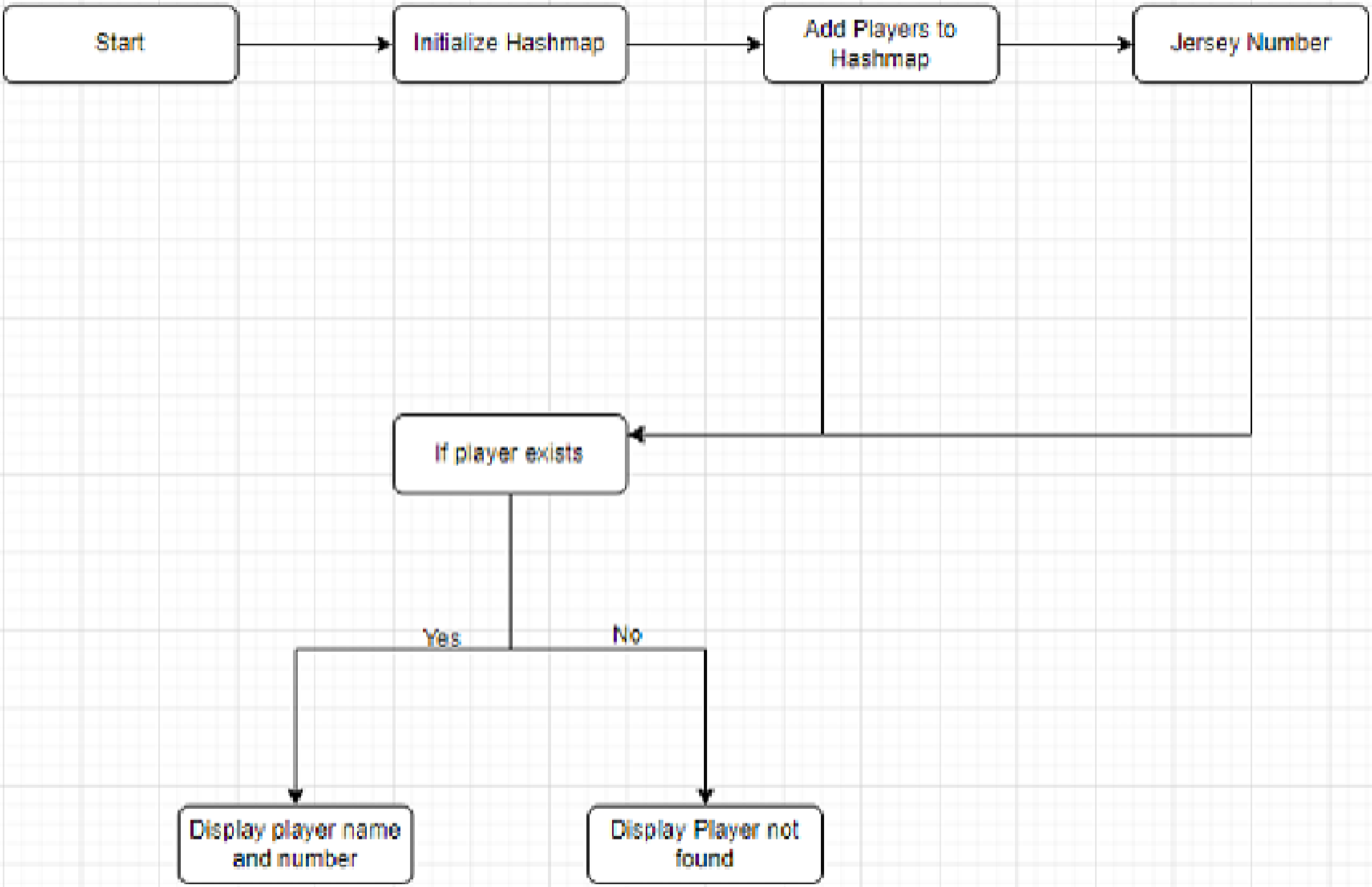
loop.



```java
//122318693 Program 2
import java.util.HashMap;
import java.util.Scanner;


public class Team {
    public static void main(String args[]) {
        HashMap<Integer, String> players = new HashMap<Integer, String>();

        //add 18 players to the map (kit number, Name)
        players.put(1, "Name1");
        players.put(2, "Name2");
        players.put(3, "Name3");
        players.put(4, "Name4");
        players.put(5, "Name5");
        players.put(6, "Name6");
        players.put(7, "Name7");
        players.put(8, "Name8");
        players.put(9, "Name9");
        players.put(10, "Name10");
        players.put(11, "Name11");
        players.put(12, "Name12");
        players.put(13, "Name13");
        players.put(14, "Name14");
        players.put(15, "Name15");
        players.put(16, "Name16");
        players.put(17, "Name17");
        players.put(18, "Name18");

        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the jersey number to see if they are on the team: ");
        int jersey_num = scan.nextInt();
        scan.nextLine();

        String player_name = players.get(jersey_num);

        if(player_name != null) {
            System.out.println("Player name " + player_name);
            System.out.println("Player number " + jersey_num);
        } else {
            System.out.println("The club doesn't have a player with this number: " + jersey_num);
        }
        scan.close();
    }
}

```



```
Enter the jersey number to see if they are on the team:
5
Player name Name5
Player number 5
```


Start
Initialize Hashmap
Add Players to Hashmap
Jersey Number
If player exists
Yes
No
Display player name and number
Display Player not found

Program 2

HashMap Initialization:

A HashMap<Integer, String> named players is initialized to store player information, where the key is the jersey number (an integer) and the value is the player's name (a string).

Adding Players:

Players' information (jersey number and name) is added to the HashMap using the put() method.

The program adds 18 players to the HashMap, each with a unique jersey number and name.

User Input:

The program prompts the user to enter a jersey number to search for.

It reads the input jersey number using a Scanner object.

Player Lookup:

The program retrieves the player's name associated with the input jersey number from the HashMap using the get() method.

If a player with the given jersey number exists in the HashMap, the associated player name is stored in the variable player_name.

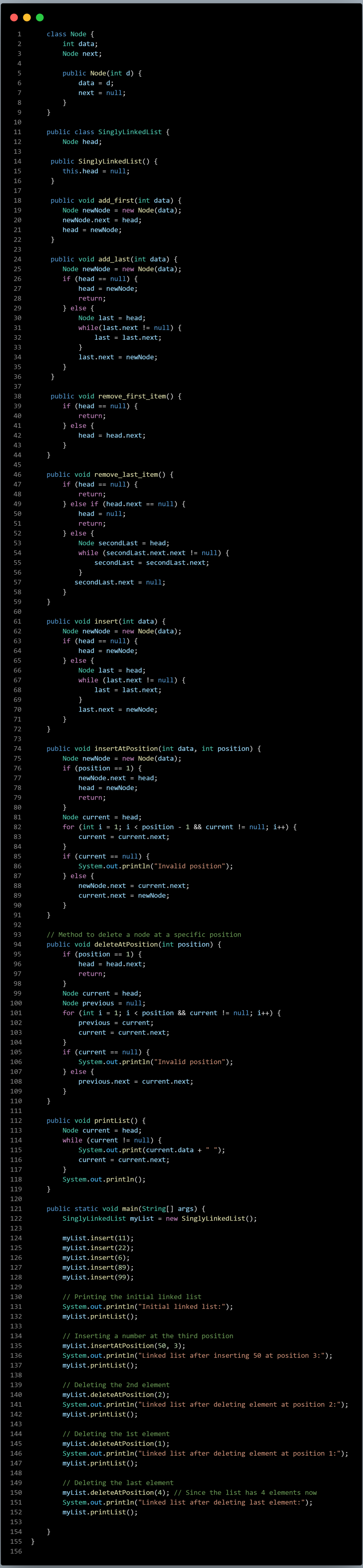
Output:

If player_name is not null, indicating that a player with the given jersey number exists in the HashMap:

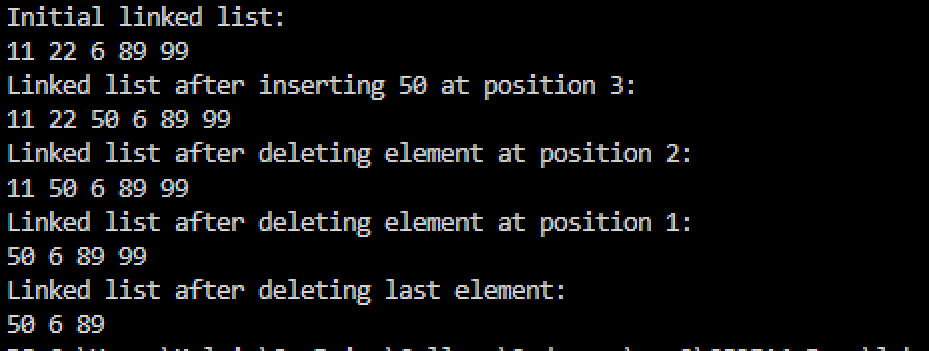
The program prints the player's name and jersey number.

If player_name is null, indicating that no player with the given jersey number exists in the HashMap:

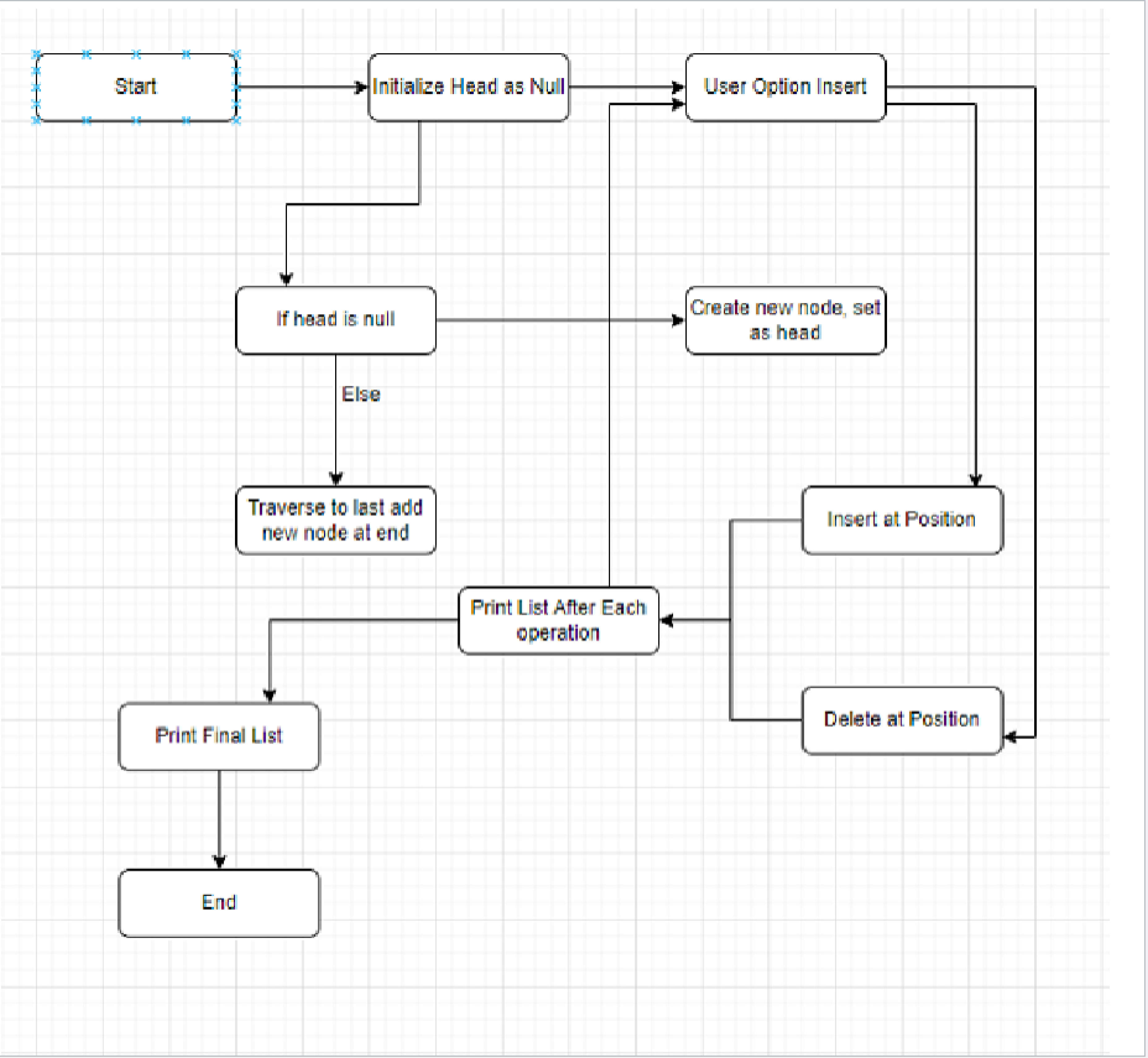
The program prints a message indicating that the player is not found.



```java
    class Node {
        int data;
        Node next;

        public Node(int d) {
            data = d;
            next = null;
        }
    }

    public class SinglyLinkedList {
        Node head;

     public SinglyLinkedList() {
        this.head = null;
     }

     public void add_first(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
     }

     public void add_last(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        } else {
            Node last = head;
            while(last.next != null) {
                last = last.next;
            }
            last.next = newNode;
        }
     }

     public void remove_first_item() {
        if (head == null) {
            return;
        } else {
            head = head.next;
        }
    }

    public void remove_last_item() {
        if (head == null) {
            return;
        } else if (head.next == null) {
            head = null;
            return;
        } else {
            Node secondLast = head;
            while (secondLast.next.next != null) {
                secondLast = secondLast.next;
            }
           secondLast.next = null;
        }
    }

    public void insert(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node last = head;
            while (last.next != null) {
                last = last.next;
            }
            last.next = newNode;
        }
    }

    public void insertAtPosition(int data, int position) {
        Node newNode = new Node(data);
        if (position == 1) {
            newNode.next = head;
            head = newNode;
            return;
        }
        Node current = head;
        for (int i = 1; i < position - 1 && current != null; i++) {
            current = current.next;
        }
        if (current == null) {
            System.out.println("Invalid position");
        } else {
            newNode.next = current.next;
            current.next = newNode;
        }
    }

    // Method to delete a node at a specific position
    public void deleteAtPosition(int position) {
        if (position == 1) {
            head = head.next;
            return;
        }
        Node current = head;
        Node previous = null;
        for (int i = 1; i < position && current != null; i++) {
            previous = current;
            current = current.next;
        }
        if (current == null) {
            System.out.println("Invalid position");
        } else {
            previous.next = current.next;
        }
    }

    public void printList() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        SinglyLinkedList myList = new SinglyLinkedList();

        myList.insert(11);
        myList.insert(22);
        myList.insert(6);
        myList.insert(89);
        myList.insert(99);

        // Printing the initial linked list
        System.out.println("Initial linked list:");
        myList.printList();

        // Inserting a number at the third position
        myList.insertAtPosition(50, 3);
        System.out.println("Linked list after inserting 50 at position 3:");
        myList.printList();

        // Deleting the 2nd element
        myList.deleteAtPosition(2);
        System.out.println("Linked list after deleting element at position 2:");
        myList.printList();

        // Deleting the 1st element
        myList.deleteAtPosition(1);
        System.out.println("Linked list after deleting element at position 1:");
        myList.printList();

        // Deleting the last element
        myList.deleteAtPosition(4); // Since the list has 4 elements now
        System.out.println("Linked list after deleting last element:");
        myList.printList();

    }
}
```



```
Initial linked list:
11 22 6 89 99
Linked list after inserting 50 at position 3:
11 22 50 6 89 99
Linked list after deleting element at position 2:
11 50 6 89 99
Linked list after deleting element at position 1:
50 6 89 99
Linked list after deleting last element:
50 6 89
```


Start
Initialize Head as Null
User Option Insert
If head is null
Create new node, set as head
Else
Traverse to last add new node at end
Insert at Position
Print List After Each operation
Delete at Position
Print Final List
End

Program 3

Node Class:

The Node class represents a single node of the linked list.

Each node contains an integer data field (data) and a reference to the next node (next).

SinglyLinkedList Class:

This class represents the singly linked list data structure.

It has a reference to the head node (head) which initially points to null, indicating an empty list.

Insertion Methods:

add_first(int data): Adds a new node with the given data at the beginning of the list. It adjusts the next reference of the new node to point to the current head, and then updates the head to the new node.

add_last(int data): Adds a new node with the given data at the end of the list. It traverses the list to find the last node and updates its next reference to point to the new node.

Deletion Methods:

remove_first_item(): Removes the first node from the list by updating the head to point to the next node.

remove_last_item(): Removes the last node from the list by traversing to the second last node and updating its next reference to null.

Other Methods:

insert(int data): Inserts a new node with the given data at the end of the list. It behaves similarly to add_last(int data).

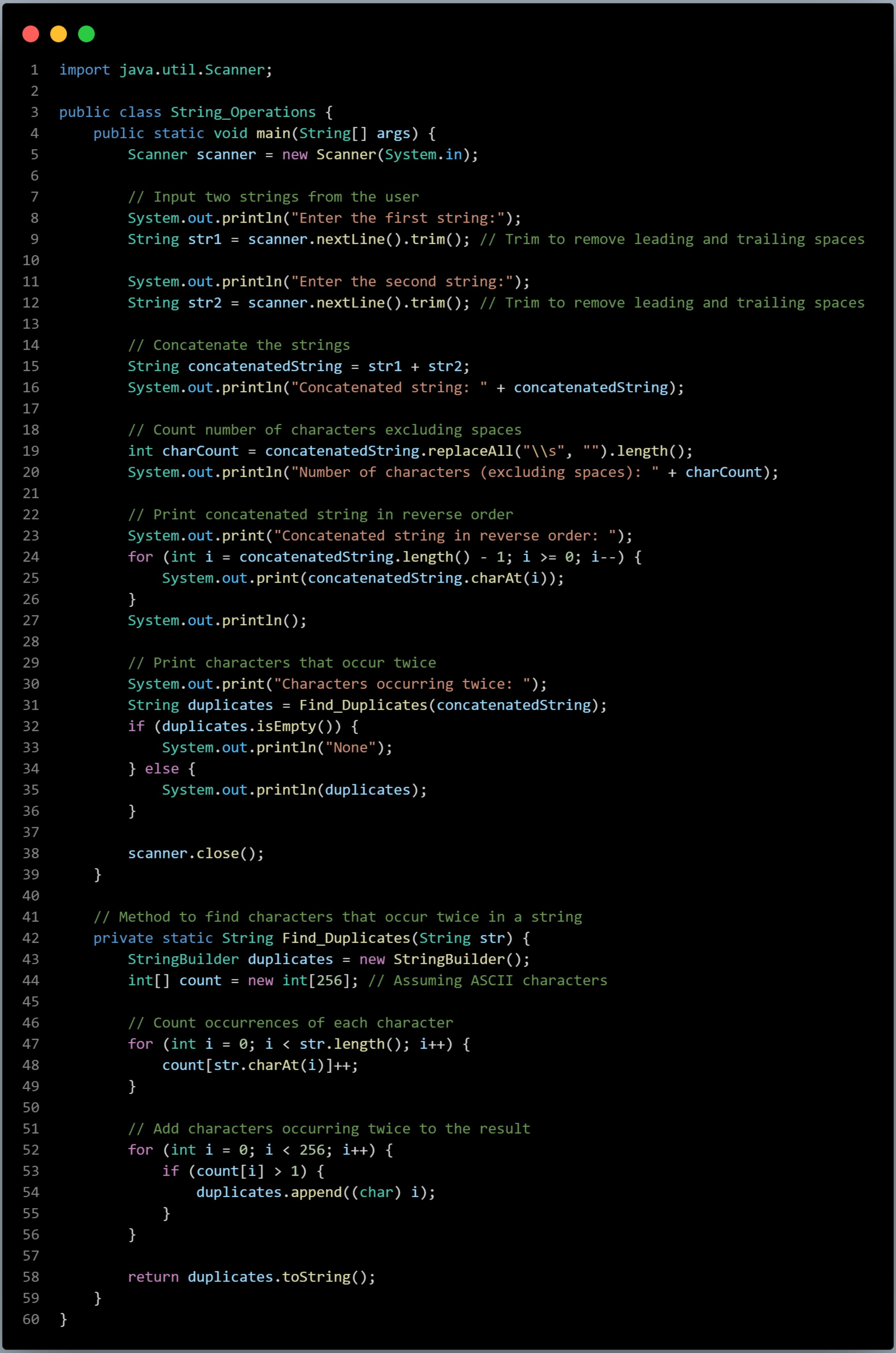
insertAtPosition(int data, int position): Inserts a new node with the given data at the specified position in the list. It traverses to the node at the specified position and adjusts the next references accordingly.

deleteAtPosition(int position): Deletes the node at the specified position in the list. It traverses to the node before the specified position and adjusts the next reference to skip the node to be deleted.

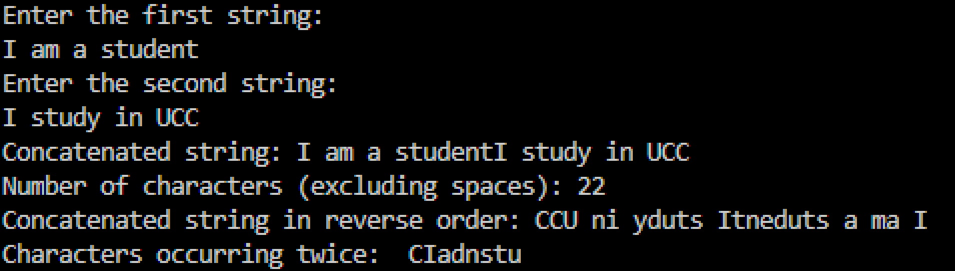
printList(): Traverses the list from the head and prints the data of each node.

Main Method:

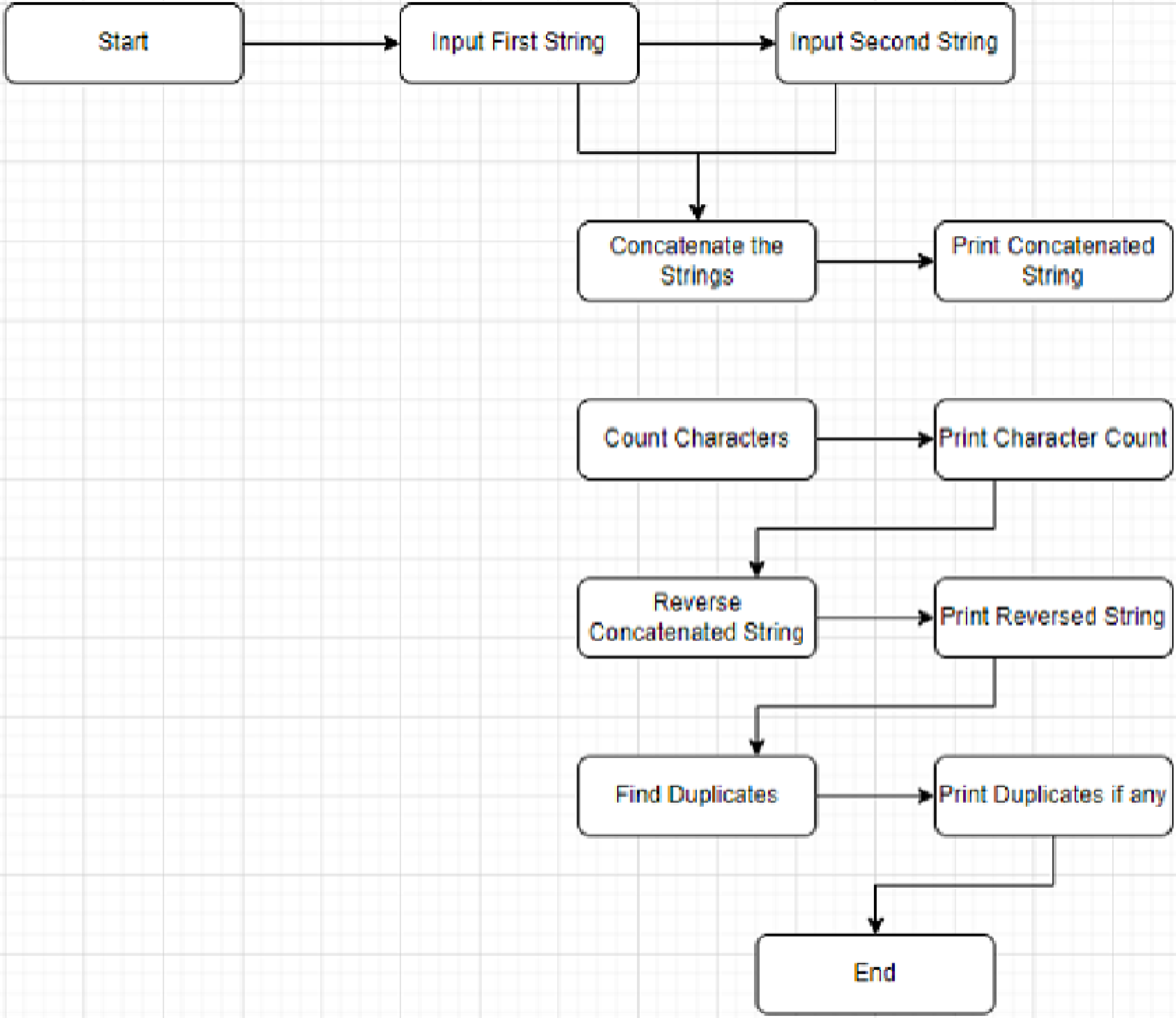
Creates an instance of SinglyLinkedList and inserts some initial data.Demonstrates various operations such as insertion at a specific position and deletion at a specific position. Prints the list after each operation to demonstrate the effect of the operations.



```java
import java.util.Scanner;

public class String_Operations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input two strings from the user
        System.out.println("Enter the first string:");
        String str1 = scanner.nextLine().trim(); // Trim to remove leading and trailing spaces

        System.out.println("Enter the second string:");
        String str2 = scanner.nextLine().trim(); // Trim to remove leading and trailing spaces

        // Concatenate the strings
        String concatenatedString = str1 + str2;
        System.out.println("Concatenated string: " + concatenatedString);

        // Count number of characters excluding spaces
        int charCount = concatenatedString.replaceAll("\\s", "").length();
        System.out.println("Number of characters (excluding spaces): " + charCount);

        // Print concatenated string in reverse order
        System.out.print("Concatenated string in reverse order: ");
        for (int i = concatenatedString.length() - 1; i >= 0; i--) {
            System.out.print(concatenatedString.charAt(i));
        }
        System.out.println();

        // Print characters that occur twice
        System.out.print("Characters occurring twice: ");
        String duplicates = Find_Duplicates(concatenatedString);
        if (duplicates.isEmpty()) {
            System.out.println("None");
        } else {
            System.out.println(duplicates);
        }

        scanner.close();
    }

    // Method to find characters that occur twice in a string
    private static String Find_Duplicates(String str) {
        StringBuilder duplicates = new StringBuilder();
        int[] count = new int[256]; // Assuming ASCII characters

        // Count occurrences of each character
        for (int i = 0; i < str.length(); i++) {
            count[str.charAt(i)]++;
        }

        // Add characters occurring twice to the result
        for (int i = 0; i < 256; i++) {
            if (count[i] > 1) {
                duplicates.append((char) i);
            }
        }

        return duplicates.toString();
    }
}
```



```
Enter the first string:
I am a student
Enter the second string:
I study in UCC
Concatenated string: I am a studentI study in UCC
Number of characters (excluding spaces): 22
Concatenated string in reverse order: CCU ni yduts Itneduts a ma I
Characters occurring twice:  CIadnstu
```


Start
Input First String
Input Second String
Concatenate the Strings
Print Concatenated String
Count Characters
Print Character Count
Reverse Concatenated String
Print Reversed String
Find Duplicates
Print Duplicates if any
End

Program 4

User Input:

The program prompts the user to input two strings: str1 and str2.

It uses the nextLine() method of the Scanner class to read the input strings.

The trim() method is called on each input string to remove any leading or trailing whitespace.

Concatenation:

The two input strings (str1 and str2) are concatenated together using the + operator and stored in the concatenatedString variable.

Character Count (excluding spaces):

The program counts the number of characters in the concatenated string excluding spaces.

It uses the replaceAll() method with a regular expression ("\\s") to remove all whitespace characters, and then calculates the length of the resulting string.

Printing Concatenated String:

The concatenated string (concatenatedString) is printed.

Printing Concatenated String in Reverse Order:

The program iterates through the characters of the concatenated string in reverse order and prints them one by one.

Finding Characters Occurring Twice:

The program calls the Find_Duplicates() method to find characters that occur twice in the concatenated string.

Inside the Find_Duplicates() method:

It initializes an array count of size 256 (assuming ASCII characters) to store the count of occurrences of each character.

It iterates through each character of the input string and increments the count of occurrences in the count array.

After counting, it iterates through the count array and appends characters occurring more than once to a StringBuilder named duplicates.

Finally, it returns the string representation of the duplicates StringBuilder.

Printing Characters Occurring Twice:

The program prints the characters that occur twice in the concatenated string, or "None" if no duplicates are found.