CM1602 : Data Structures and Algorithms for AI

3. Array and Linked List

Lecture 3| R. Sivaraman

MODULE CONTENT

Lecture	Topic
Lecture 01	Introduction to Fundamentals of Algorithms
Lecture 02	Analysis of Algorithms
Lecture 03	Array and Linked Lists
Lecture 04	Stack
Lecture 05	Queue
Lecture 06	Searching algorithms and Sorting algorithms
Lecture 07	Trees
Lecture 08	Maps, Sets, and Lists
Lecture 09	Graph algorithms

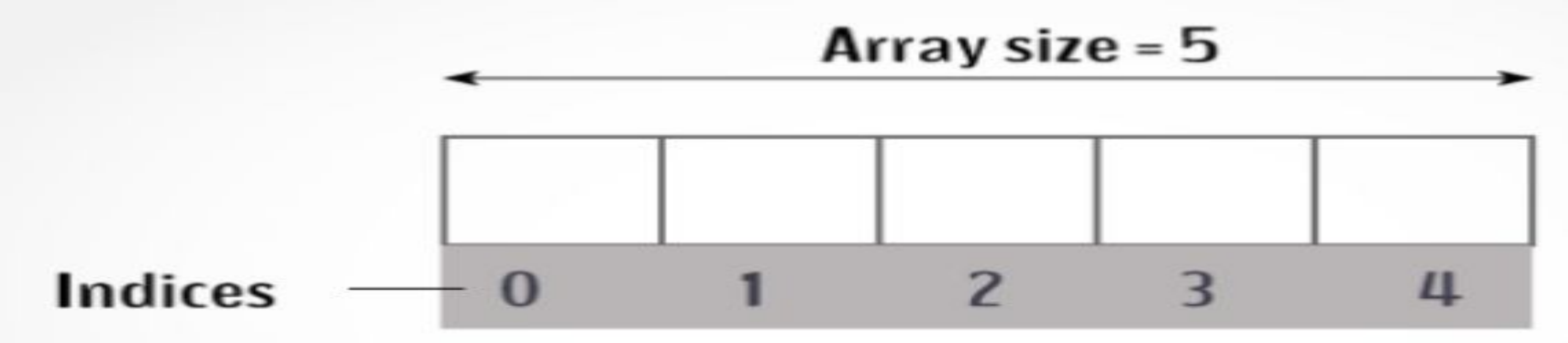
Learning Outcomes

- Covers LO1 : Describe the fundamental concepts of algorithms and data structures.
- Covers LO3 : Apply appropriate data structures given a real-world problem to meet requirements of programming language APIs.
- On completion of this lecture, students are expected to be able to:
  - Describe Array and Linked List
  - Implement an Array
  - Implement a Linked List

Array

Array

- Arrays are used to store multiple values in a single variable

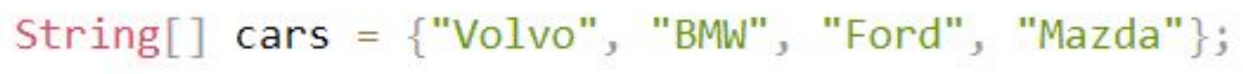


- To declare an array, define the variable type with **square brackets**

```
String[] cars;
```

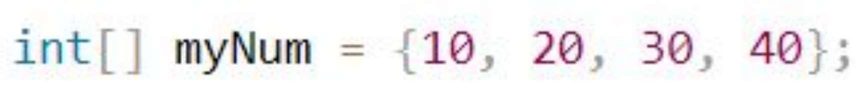
Array

- To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces.



```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

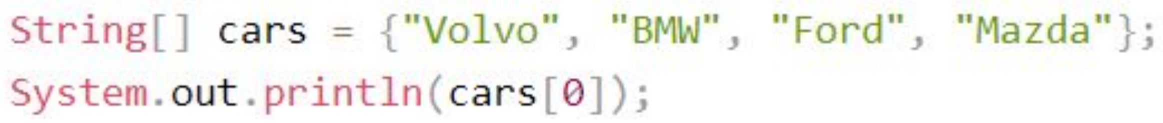
- To create an array of integers, you could write:



```
int[] myNum = {10, 20, 30, 40};
```

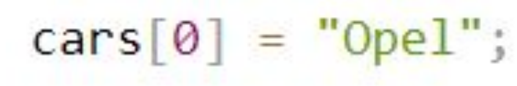
Array

- You access an array element by referring to the index number.



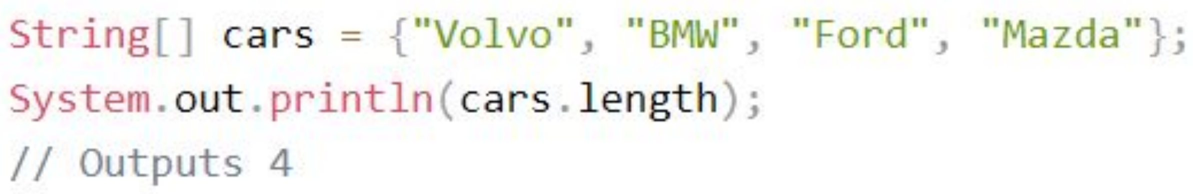
```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
```

- To change the value of a specific element, refer to the index number:



```
cars[0] = "Opel";
```

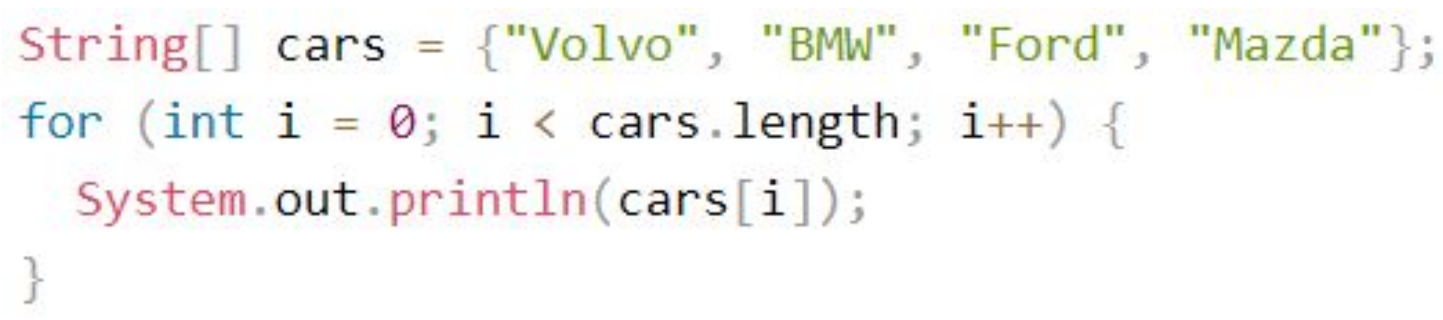
- To find out how many elements an array has, use the ‘length’ property:



```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
// Outputs 4
```

Array

- You can loop through the array elements.

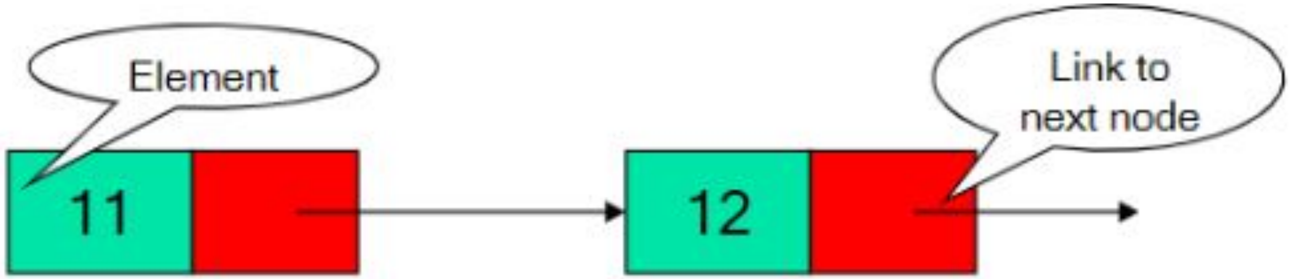


```java
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
  System.out.println(cars[i]);
}
```

Linked List

Linked List - Introduction

- Linked List is a Linear Data Structure represented by nodes.
- It is made of collection of connected, dynamically allocated Nodes
- Each node will have at least two elements
  - Element (The data)
  - The next node



Why Linked List

- Inserting or Deleting elements into or from list is easy, where it requires extensive data movement if array is used

- Linked List can grow or shrink dynamically based on the size of the list, but in array size is fixed once it is created

Linked List - Types

- Single Linked List
  - Can access the next node only
  - Last Node is set to null
- Doubly Linked List
  - Can access the next and Previous node
  - Last Node is set to null
- Circular Linked List
  - Similar to Doubly linked List, but Last node points to the first node

Header & Trailer Nodes

- Header Node: A placeholder node at the beginning of list, used to simplify list processing. It doesn't hold any data but satisfies that every node has a previous node.

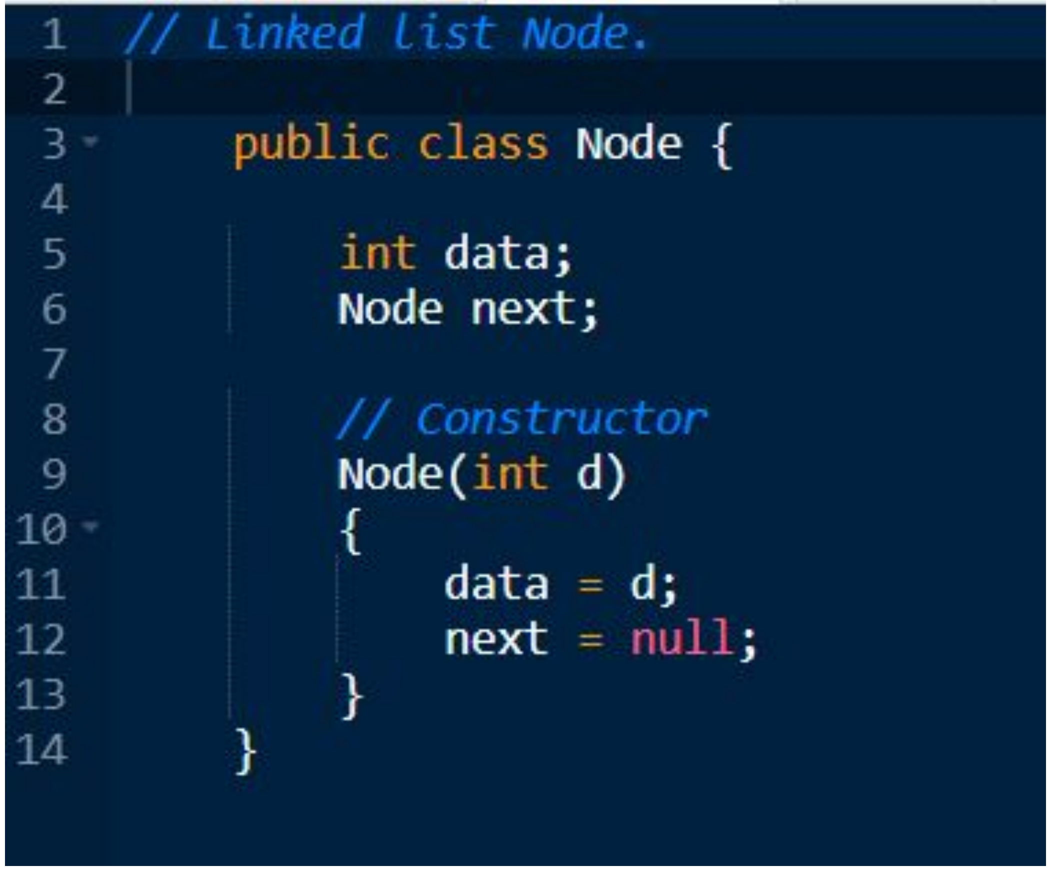
- Trailer Node: A Placeholder node at the end of list, used to simplify list processing.

Linked List Implementation

Linked List - Implementation

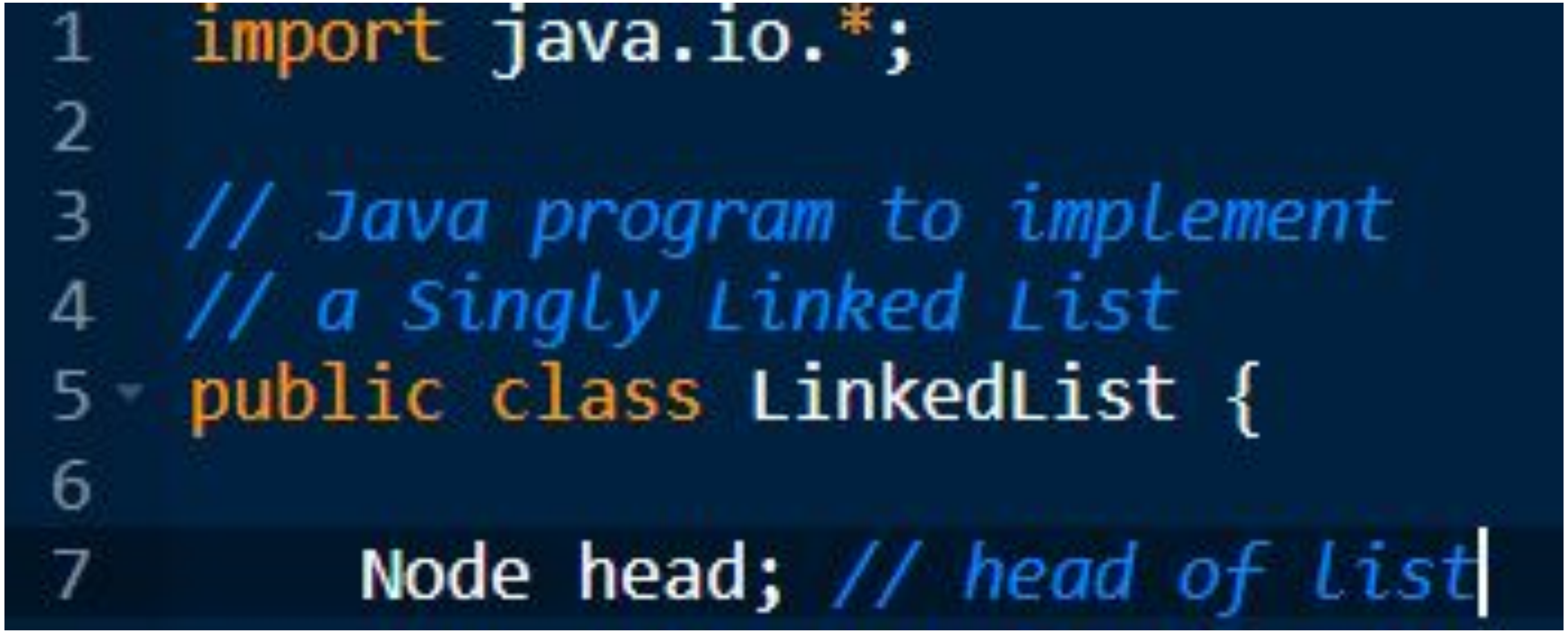
- It requires two classes
  - A class for one Node (Node class)
  - A class for Linked List (LinkedList class)

Node Class



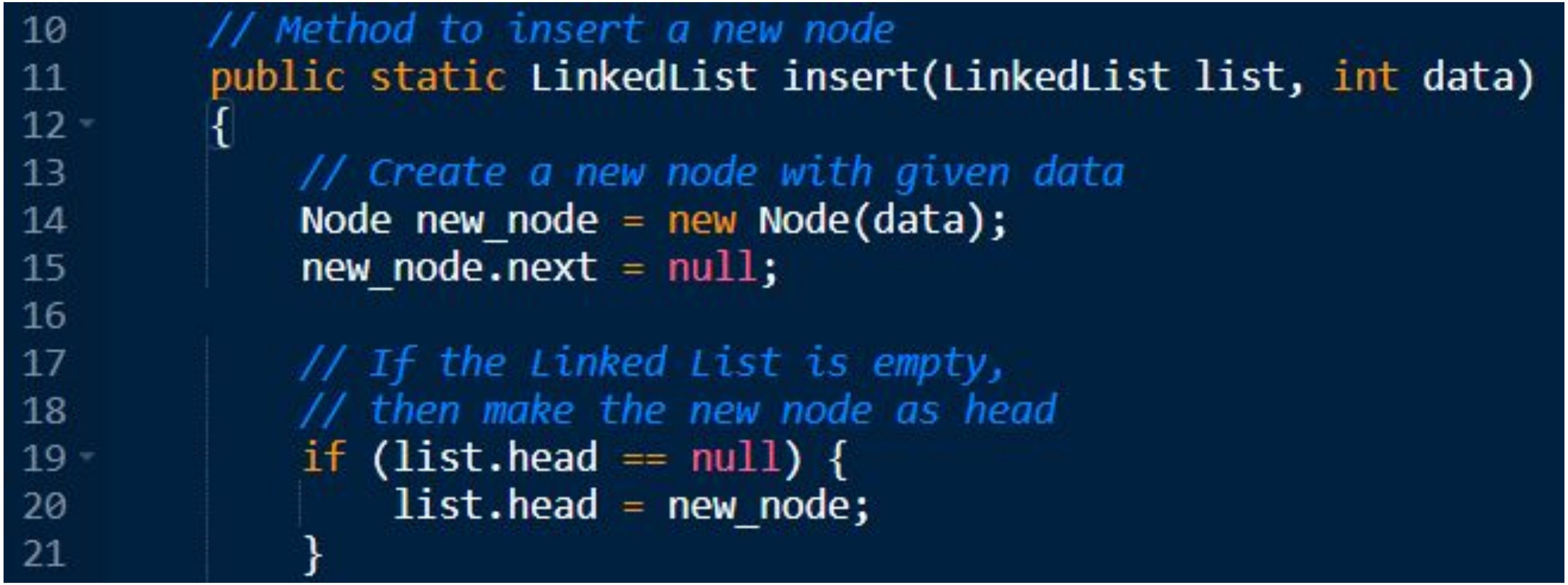
```java
// Linked list Node.

    public class Node {

        int data;
        Node next;

        // Constructor
        Node(int d)
        {
            data = d;
            next = null;
        }
    }
```

Linked List Class



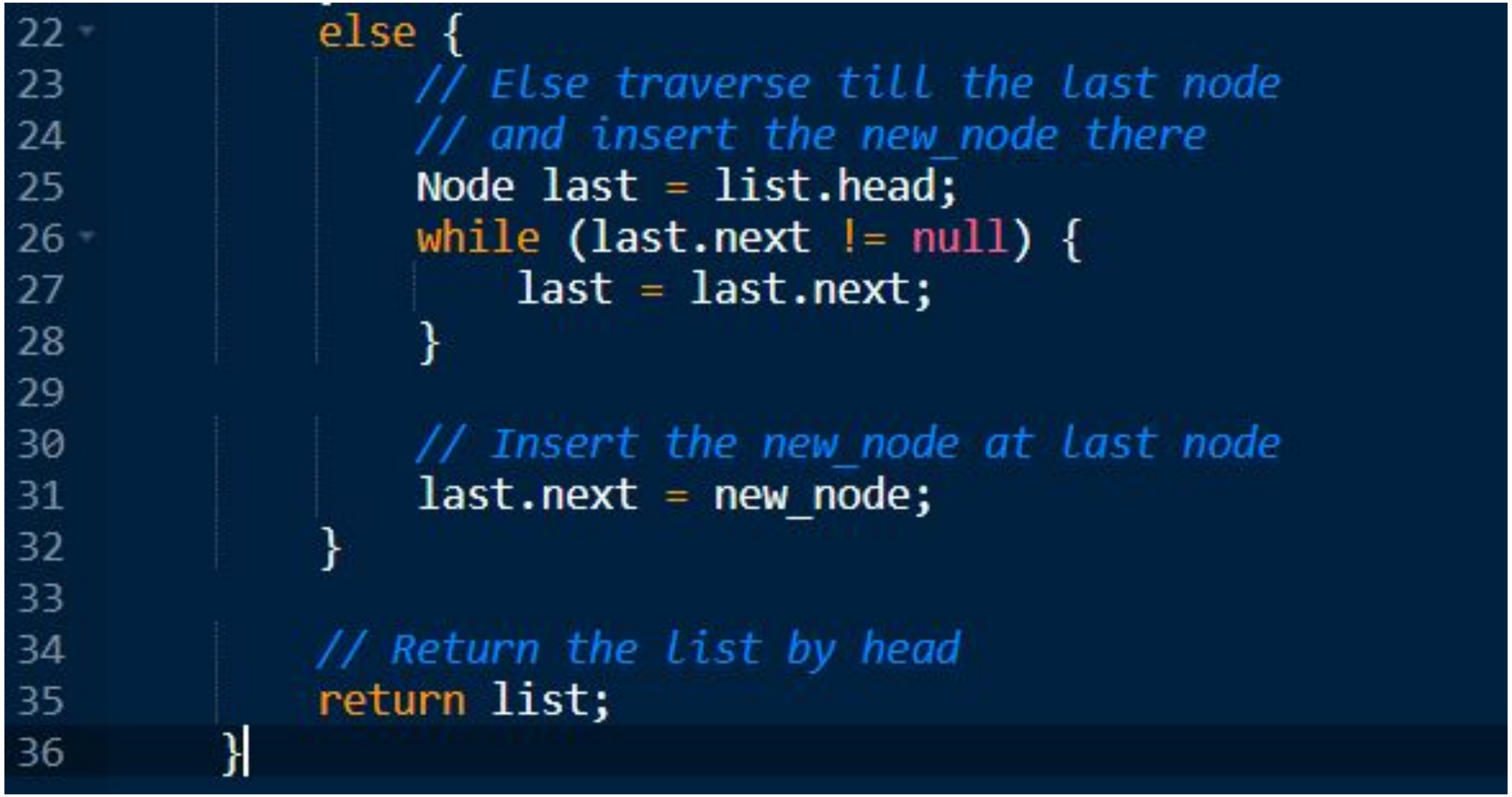
```java
import java.io.*;

// Java program to implement
// a Singly Linked List
public class LinkedList {

    Node head; // head of list
```

Linked List Class – Insert



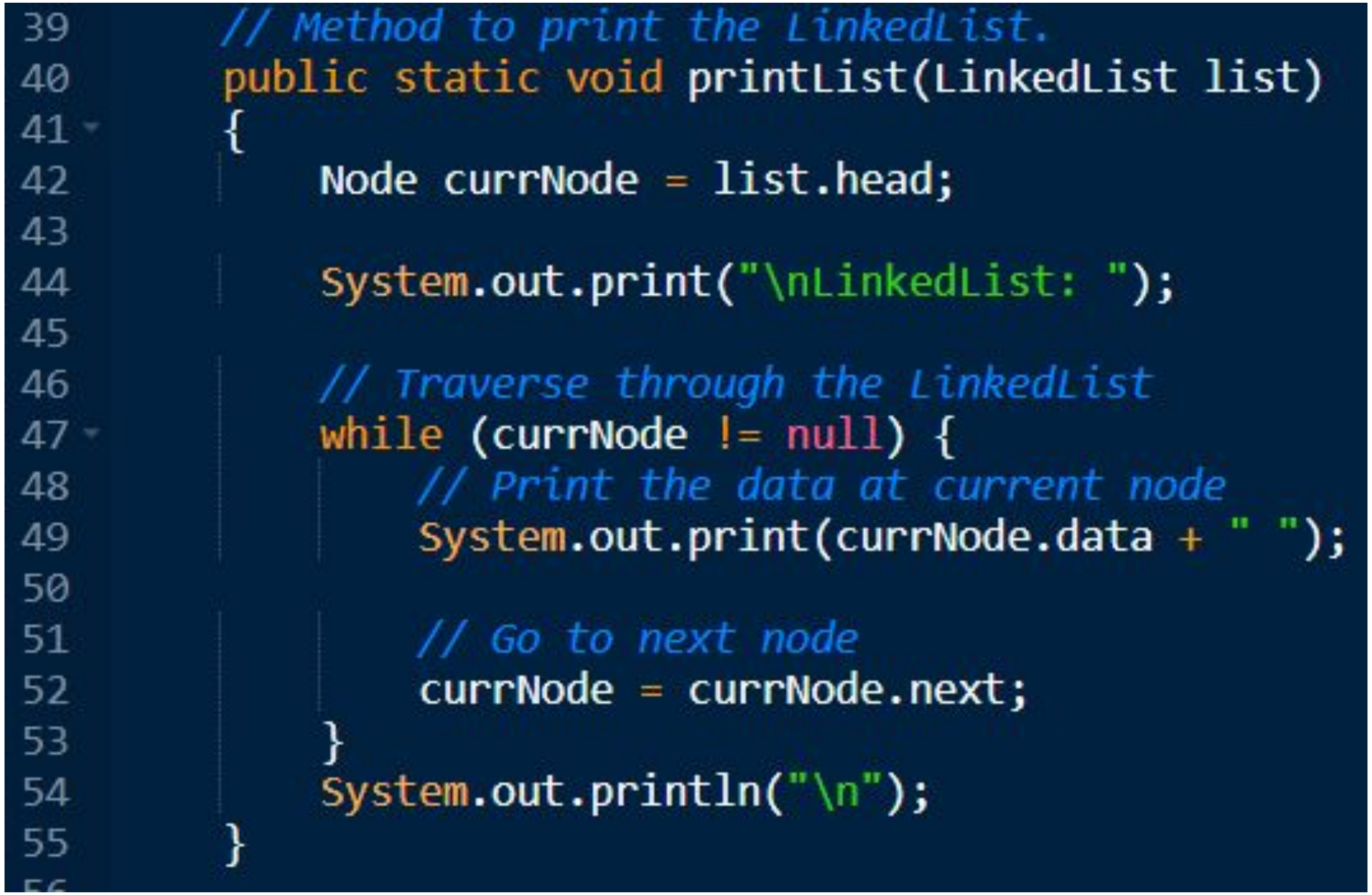
```
    // Method to insert a new node
    public static LinkedList insert(LinkedList list, int data)
    {
        // Create a new node with given data
        Node new_node = new Node(data);
        new_node.next = null;

        // If the Linked List is empty,
        // then make the new node as head
        if (list.head == null) {
            list.head = new_node;
        }
```

Linked List Class - Insert



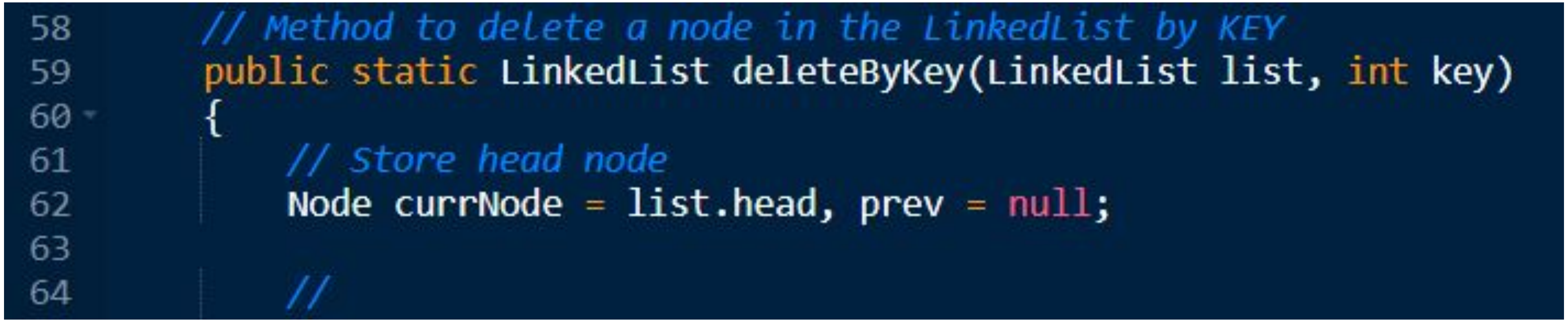
```
        else {
            // Else traverse till the last node
            // and insert the new_node there
            Node last = list.head;
            while (last.next != null) {
                last = last.next;
            }

            // Insert the new_node at last node
            last.next = new_node;
        }

        // Return the list by head
        return list;
    }
```

Linked List Class – Print



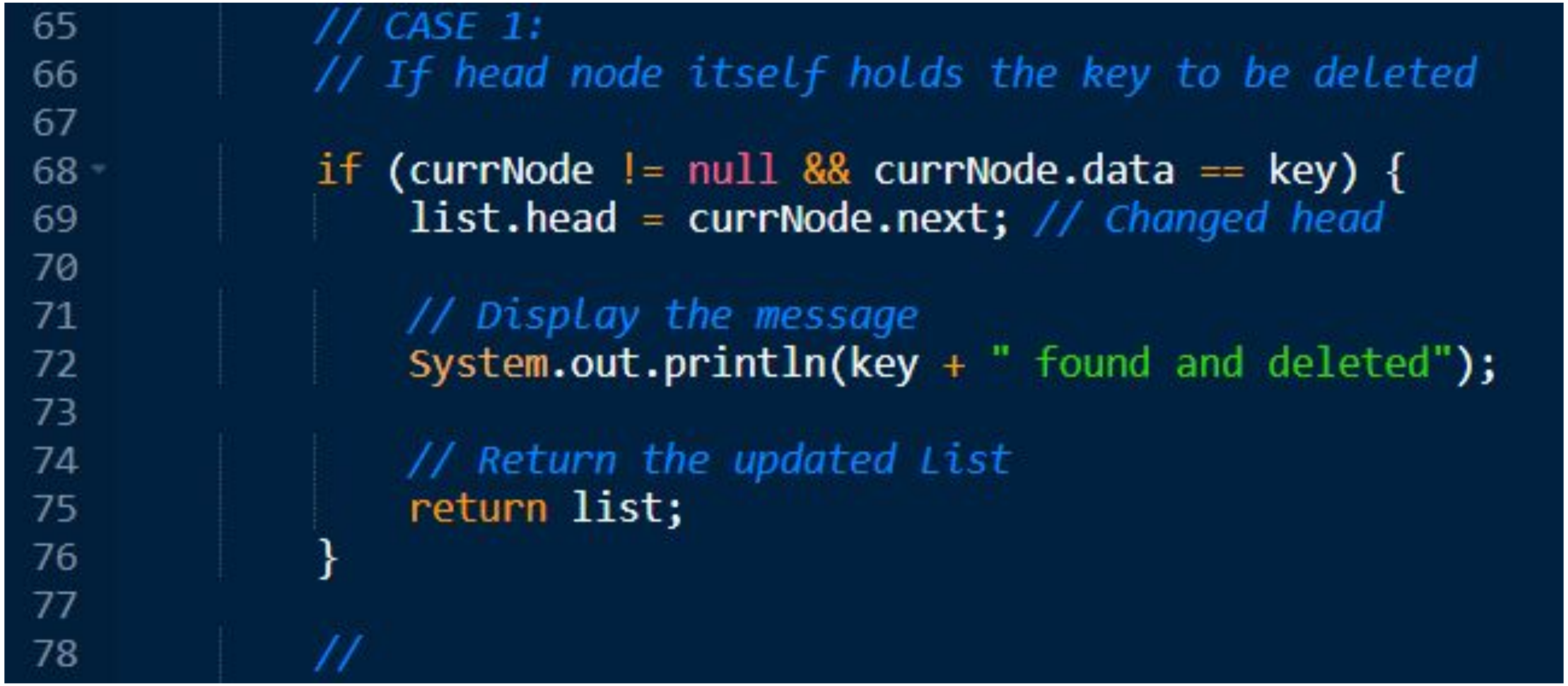
```java
    // Method to print the LinkedList.
    public static void printList(LinkedList list)
    {
        Node currNode = list.head;

        System.out.print("\nLinkedList: ");

        // Traverse through the LinkedList
        while (currNode != null) {
            // Print the data at current node
            System.out.print(currNode.data + " ");

            // Go to next node
            currNode = currNode.next;
        }
        System.out.println("\n");
    }
```

Linked List Class - Delete



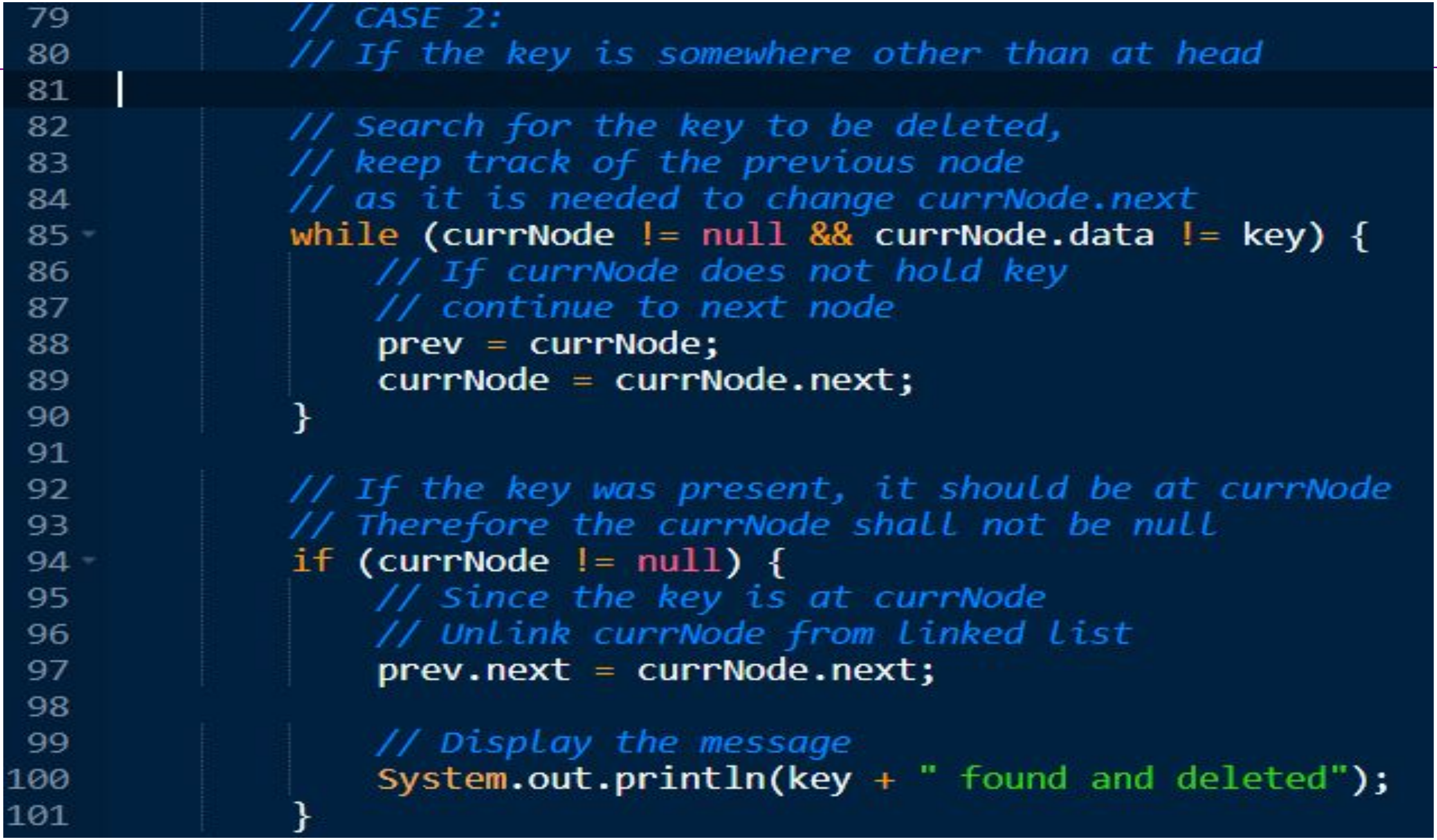
```
    // Method to delete a node in the LinkedList by KEY
    public static LinkedList deleteByKey(LinkedList list, int key)
    {
        // Store head node
        Node currNode = list.head, prev = null;

        //
```

Linked List Class - Delete



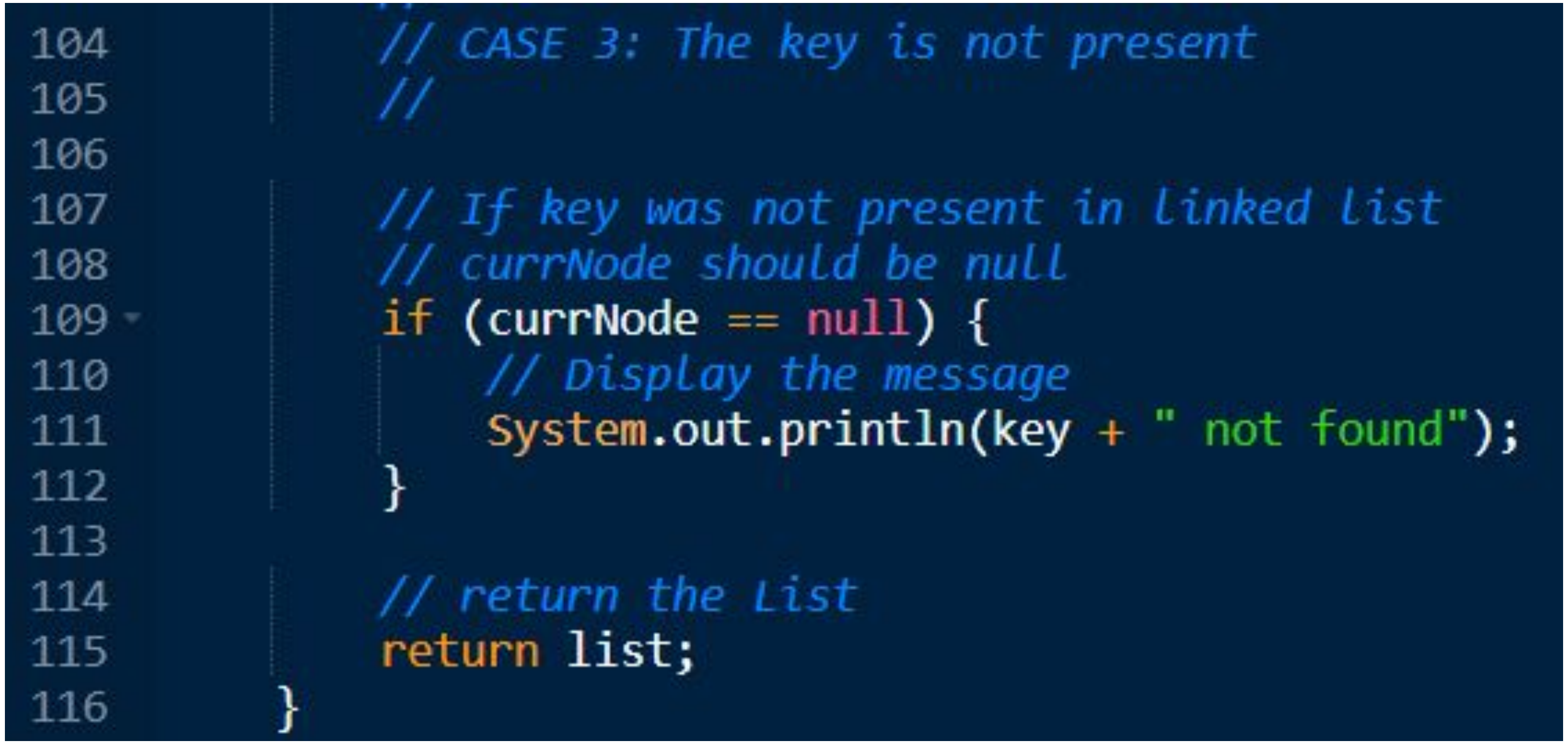
```
        // CASE 1:
        // If head node itself holds the key to be deleted

        if (currNode != null && currNode.data == key) {
            list.head = currNode.next; // Changed head

            // Display the message
            System.out.println(key + " found and deleted");

            // Return the updated List
            return list;
        }

        //
```

Linked List Class - Delete



```
        // CASE 2:
        // If the key is somewhere other than at head

        // Search for the key to be deleted,
        // keep track of the previous node
        // as it is needed to change currNode.next
        while (currNode != null && currNode.data != key) {
            // If currNode does not hold key
            // continue to next node
            prev = currNode;
            currNode = currNode.next;
        }

        // If the key was present, it should be at currNode
        // Therefore the currNode shall not be null
        if (currNode != null) {
            // Since the key is at currNode
            // Unlink currNode from linked list
            prev.next = currNode.next;

            // Display the message
            System.out.println(key + " found and deleted");
        }
```

Linked List Class - Delete



```
    // CASE 3: The key is not present
    //

    // If key was not present in linked list
    // currNode should be null
    if (currNode == null) {
        // Display the message
        System.out.println(key + " not found");
    }

    // return the List
    return list;
}
```