

■ Introduction to Linked List

Nodes connected by pointers — the foundation of dynamic data

- Linked List

■ EASY

■ What You Will Learn

- What a Node is and how it stores data + pointer
- How to create and traverse a linked list
- Difference between arrays and linked lists
- How to insert and print nodes

■ Intuition First

Imagine a treasure hunt where each clue tells you where the next clue is. Node 1 has value 10 and says 'next clue is at Node 2'. Node 2 has value 20 and says 'next is Node 3'. The last node says 'no more clues' (null). That's a linked list — a chain of self-referential boxes.

Linked List: 1 → 2 → 3 → 4 → null

Each Node:

Age Group	Percentage
18-24	15%
25-34	20%
35-44	25%
45-54	20%
55-64	15%
65-74	10%
75-84	5%
85+	5%

■ data ■ next ■■■→ (points to next node)

Page 10 of 10

In memory (NOT contiguous like arrays):

```
Node@100: [1 | @200]
```

```
Node@200: [2 | @450]
```

```
Node@450: [3 | @712]
```

```
Node@712: [4 | null ]
```

Array vs LinkedList:

Array: [1][2][3][4] ← contiguous, index access 0(1)

LinkedList: scattered, must traverse $O(n)$

■ Java Code — Node + Traversal

```
1 // Node definition
2 class ListNode {
3     int val;
4     ListNode next;
5     ListNode(int val) { this.val = val; }
6 }
7
8 // Create linked list 1→2→3→4
```

```

9   ListNode head = new ListNode(1);
10  head.next = new ListNode(2);
11  head.next.next = new ListNode(3);
12  head.next.next.next = new ListNode(4);
13
14  // Traverse and print
15  ListNode curr = head;
16  while (curr != null) {
17      System.out.print(curr.val + ' ');
18      curr = curr.next; // Move to next node
19  }

```

■ Complexity

Type	Value	Meaning
Time	$O(n)$	Traversal visits every node once
Space	$O(n)$	n nodes stored in memory

■ Common Mistakes

■ Mistake

Moving head directly during traversal (`head=head.next`). Always use a separate 'curr' pointer so you don't lose the reference to the start of the list!

■ Quick Revision

✓	ListNode has two fields: val (data) and next (pointer)
✓	Last node's next = null (marks end of list)
✓	Always traverse with a temp pointer, never move head
✓	Insertion at head: <code>newNode.next = head; head = newNode</code>
✓	Access by index needs $O(n)$ traversal — no random access