Java linked list traversal

☑️ Case 1:

```
while (temp != null) {
    temp = temp.next;
}
```

📌 **Purpose:** Traverse entire list
📍 **Loop ends:** when `temp == null`
🎯 **Final `temp`:** `null`
✅ **Correct** — Fully traverses the list, including the last node.

☑️ Case 2:

```
while (temp.next != null) {
    temp = temp.next;
}
```

📌 **Purpose:** Reach the **last node**
📍 **Loop ends:** when `temp.next == null`
🎯 **Final `temp`:** points to **last node**
✅ **Correct** — Works safely when list is non-empty.

☑️ Case 3:

```
while (temp.next.next != null) {
    temp = temp.next;
}
```

📌 **Purpose:** Reach the **second last node**
📍 **Loop ends:** when `temp.next.next == null`
🎯 **Final `temp`:** points to **second last node**
✅ **Correct for size ≥ 3**
⚠️ **Unsafe for size < 2** (will throw `NullPointerException`)

🛡️ **Safer version:**

```
if (temp != null && temp.next != null) {
    while (temp.next.next != null) {
```

```
            temp = temp.next;
        }
    }
```

✅ Summary Table:

Case	Condition	Final `temp`	Validity
1	`while (temp != null)`	`null` (after last node)	✅ Always safe
2	`while (temp.next != null)`	Last node	✅ Always safe for non-empty
3	`while (temp.next.next != null)`	Second last node	✅ For size ≥ 3 ⚠️ Unsafe for size < 2

Agar list me 3+ nodes hain, to tino loops sahi se kaam karenge. bas Case 3 ke liye input check zaroori hai.