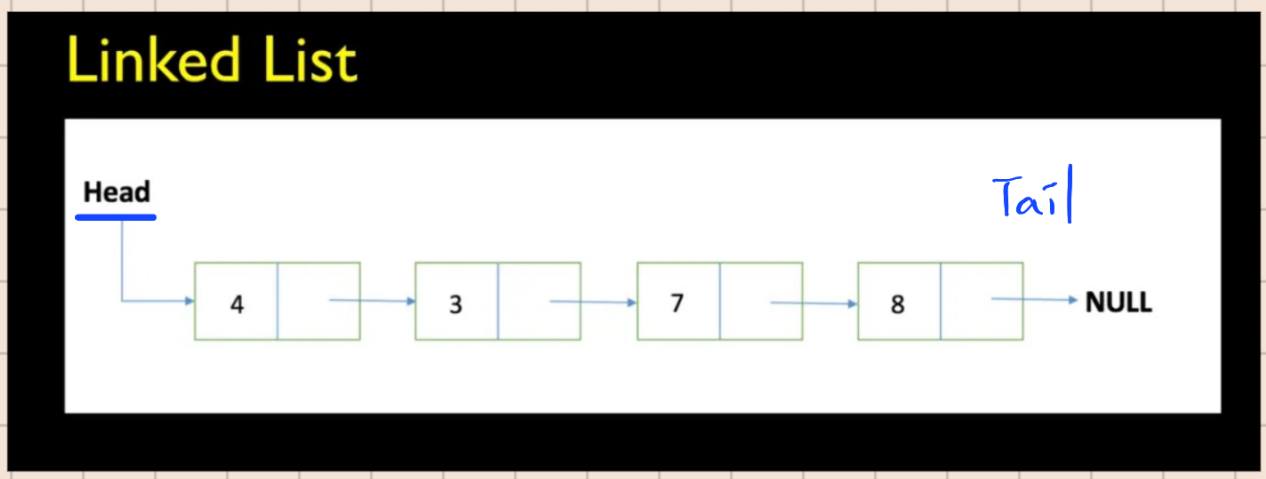
Linked List

Array drawback : Fixed size

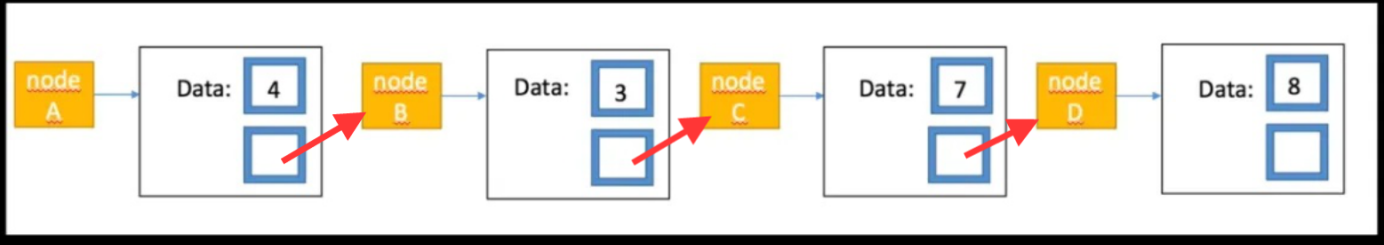
Node nodeA = new Node()

↑ reference variable, represent address!!




```
Node nodeA = new Node();
nodeA.data = 4;

Node nodeB = new Node();
nodeB.data = 3;

Node nodeC = new Node();
nodeC.data = 7;

Node nodeD = new Node();
nodeD.data = 8;
```

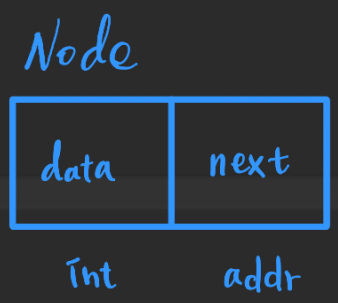
```
nodeA.next = nodeB;
nodeB.next = nodeC;
nodeC.next = nodeD;
```



*. Node.java

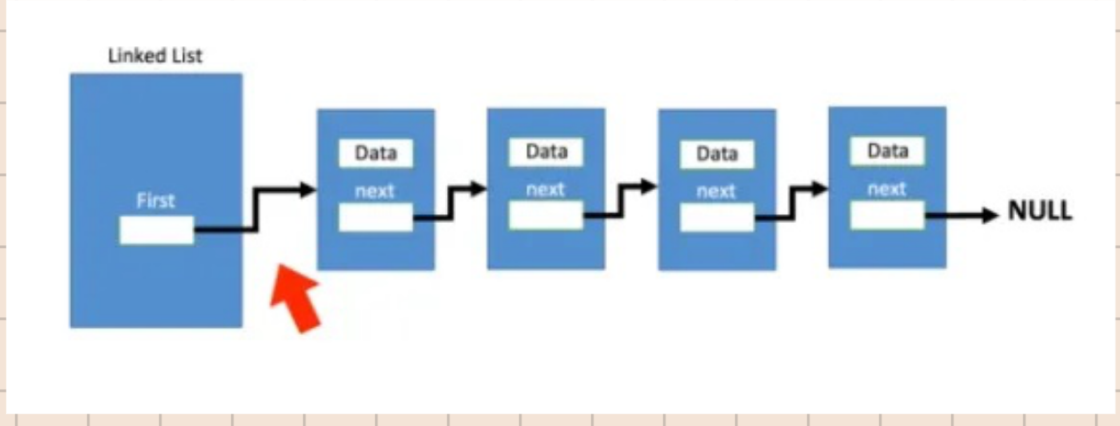
SinglyLinkedList.java | CircularLinkedList.java | Node.java | LinkedListApp.java

```java
package LinkedList;

public class Node {
    public int data;
    public Node next = null;

    public Node() {
    }

    public void displayNode(){
        System.out.println(this.data);
    }
}
```

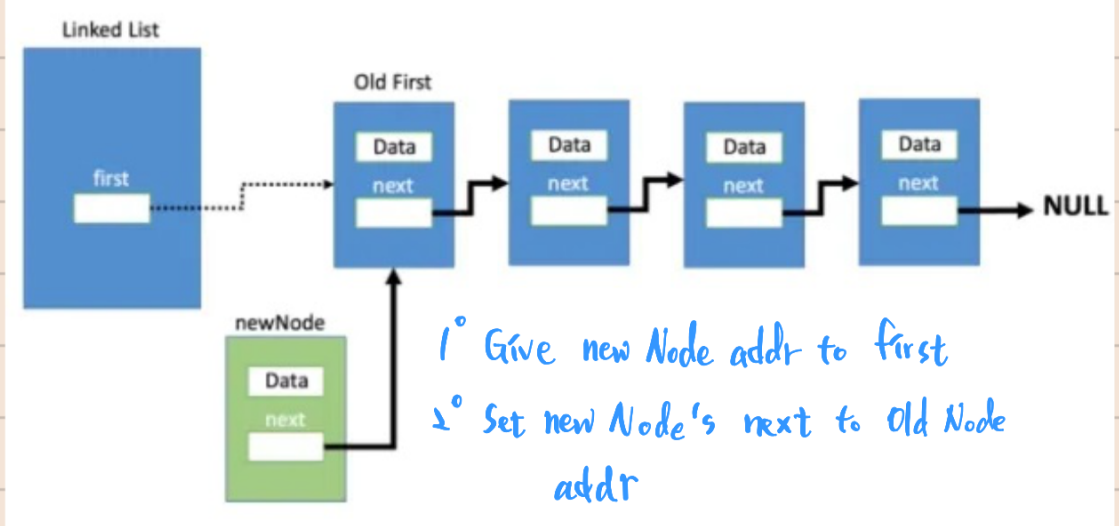


*. SinglyLinkedList.java

Singly Linked List Structure



Insert in Singly Linked List



SinglyLinkedList.java | CircularLinkedList.java | Node.java | LinkedListApp.java

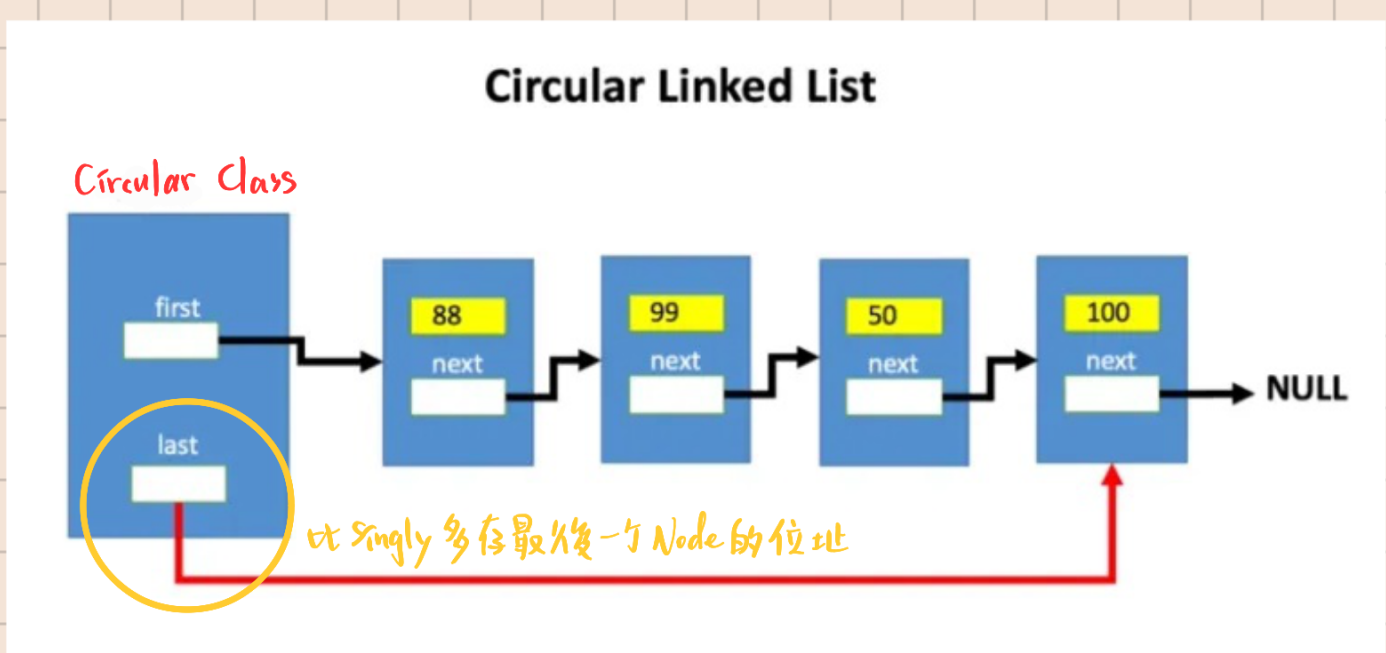
```java
public class SinglyLinkedList {
    private Node firstNode; // First Node Address

    public SinglyLinkedList(){}

    public boolean isEmpty() { return (firstNode == null); }
```

```java
    // Insert Node to the beginning
    public void insertFirst(int data){
        Node newNode = new Node();
        newNode.data = data;
        newNode.next = firstNode; // Give old first node address to New First node's next
        firstNode = newNode;      // give new node address to firstNode
    }

    public Node deleteFirst(){...}

    public void displayList(){
        System.out.println("List (first ---> last)");
        Node currentNode = firstNode;
        while (currentNode != null){
            currentNode.displayNode();
            currentNode = currentNode.next;
        }
        System.out.println();
    }

    public void insertLast(int data){
        Node tmp = firstNode;
        while (tmp != null){...}
        Node lastNode = new Node();
        lastNode.data = data;
        tmp.next = lastNode;
    }
}
```

firstNode = firstNode.next;

tmp = tmp.next;

Circular Linked List



SinglyLinkedList.java | CircularLinkedList.java | Node.java | LinkedListApp.java

```java
public class CircularLinkedList {
    private Node firstNode;
    private Node lastNode;  // Add this
```

```java
    public CircularLinkedList(){

    }

    public boolean isEmpty() { return (firstNode == null); }

    // Insert Node to the beginning
    public void insertFirst(int data){
        Node newNode = new Node();
        newNode.data = data;

        if (isEmpty()){           // Difference between Singly Linked List
            lastNode = newNode;
        }
        newNode.next = firstNode; // Give old first node address to New First node's next
        firstNode = newNode;      // give new node address to firstNode
    }

    public void insertLast(int data){
        Node tmp = new Node();
        tmp.data = data;

        if (isEmpty()){
            firstNode = tmp;
        }else {
            lastNode.next = tmp; // Remove null in old last node's next
            lastNode = tmp;      // Update last node address in CircularLinkedList
        }
    }

    public Node deleteFirst(){
        Node temp = firstNode;
        if (firstNode.next == null){  // If firstNode is the only node
            lastNode = null;          // update lastNode to null
        }
        firstNode = firstNode.next;
        return temp;
    }

    public void displayList(){
        System.out.println("List (first ---> last)");
        Node currentNode = firstNode;
        while (currentNode != null){
            currentNode.displayNode();
            currentNode = currentNode.next;
        }
        System.out.println();
    } // Same in Singly
}
```

當 circular insert 第一個 Node 時，指標 first 跟 last 都會指第1个 node

這一行可理解為將 newNode 的 Next 設為 null

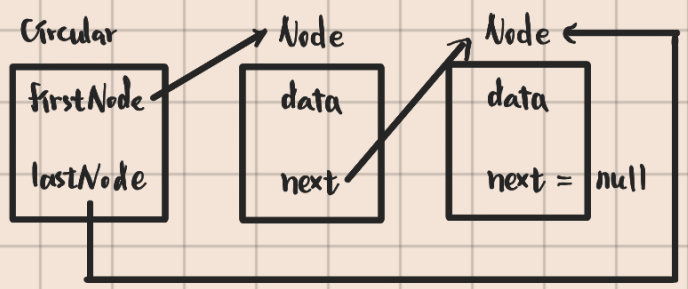
∵ circular 為空時 firstNode = null

← 加上 lastNode = tmp ;

- Insert first

*. List not empty

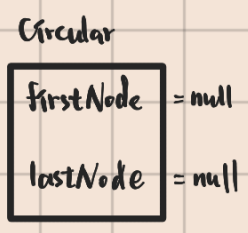
1°



2°

*. List empty

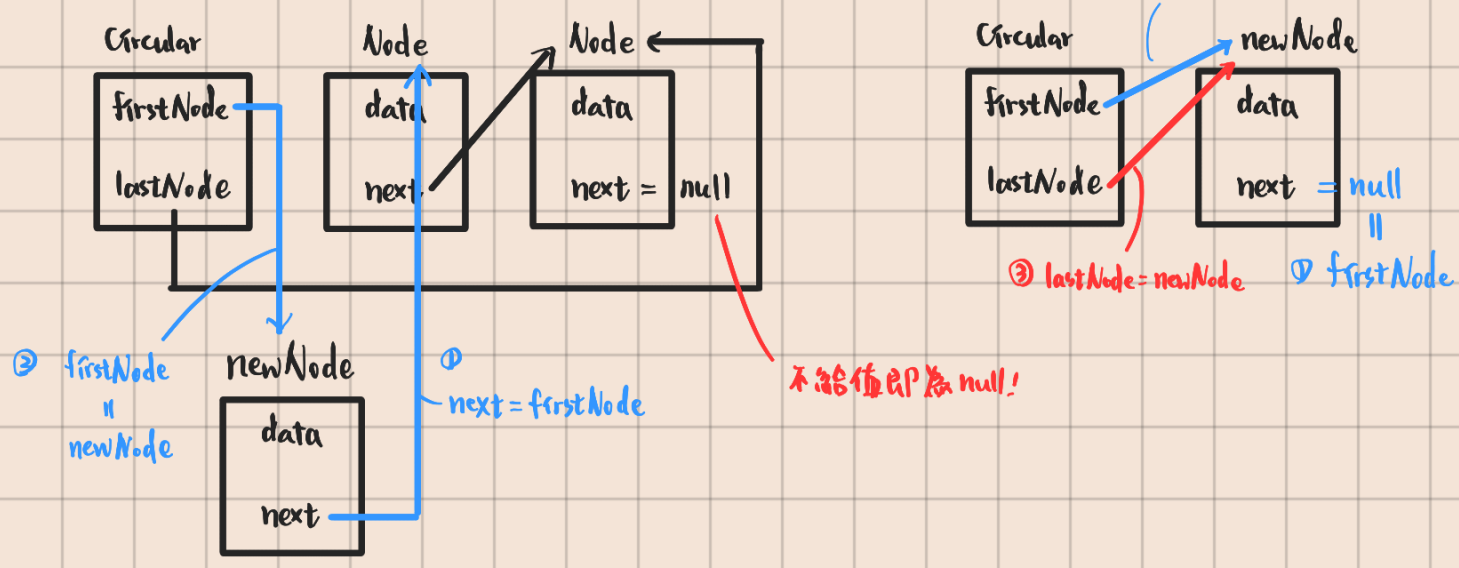
1°



*. 初始化都是"null"!

2°

② firstNode = newNode

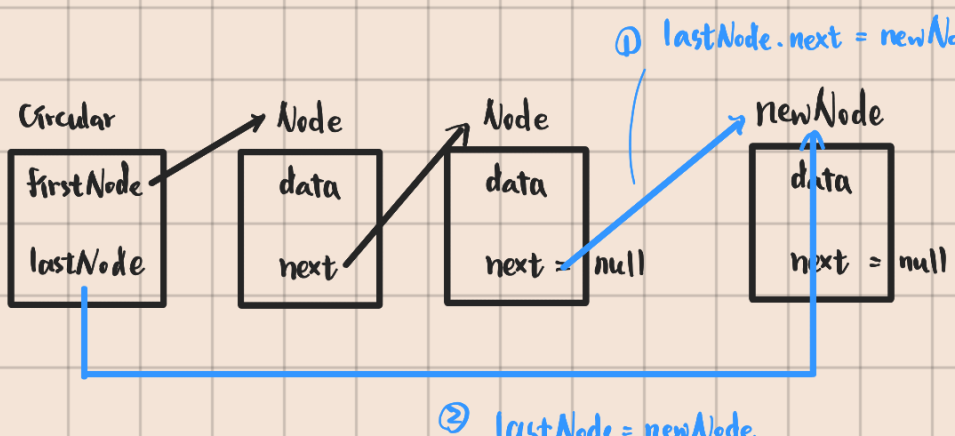


結論：

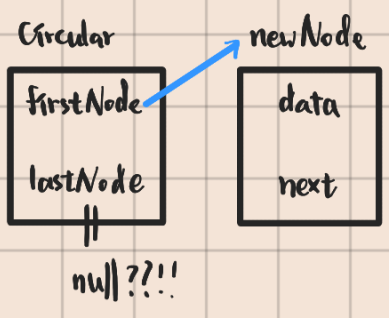
- Insert 不論 List 是不是 empty，firstNode 步驟皆相同
- List 為 empty，記得給 lastNode 新 Node 的 addr，否則為 null
- Class 內的 Data (ex: firstNode.next...) 沒有給值，則 default 為 null

◦ Insert last

*. List not empty



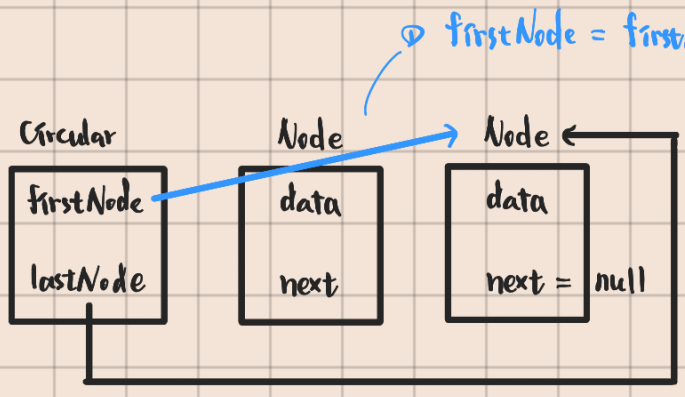
*. List empty

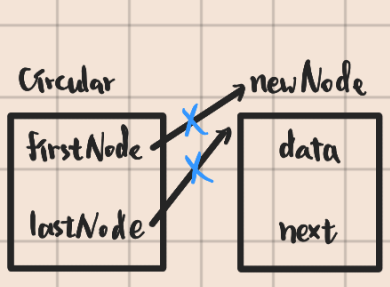


***. 問題

- 教學版本問題，当 circular list 為 empty 時
  insertFirst 有將 first, last 都設為 1st node 的 addr
  但 insertLast 的 lastNode 還是 null
  ∵ deleteFirst 只有1 node 時，first 跟 last 都有值
  ∴ 這裡 insertLast 的 lastNode 也要給值

- Delete first





結論：当 circularList 只有一个 Node 时
firstNode 和 lastNode 都是
指向這个 Node！

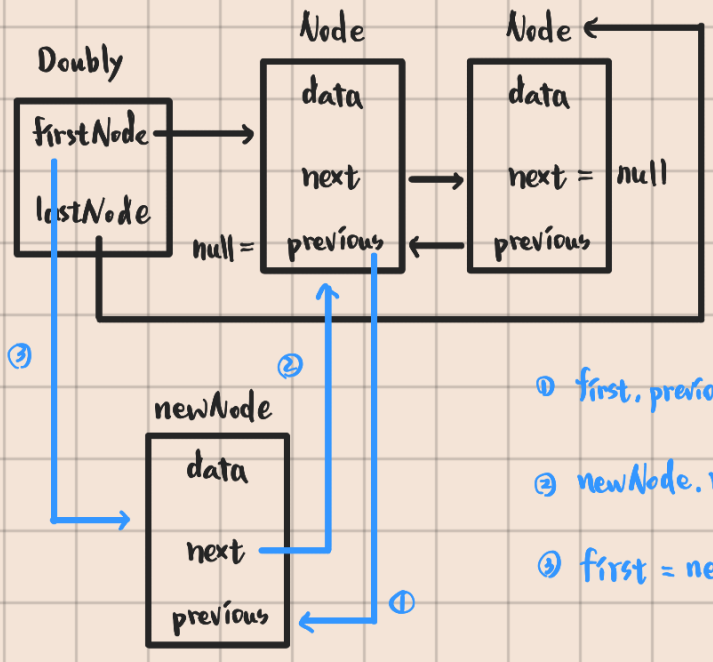
① firstNode = null = firstNode.next

② lastNode = null

Doubly Linked List

- Insert first

*. List not empty

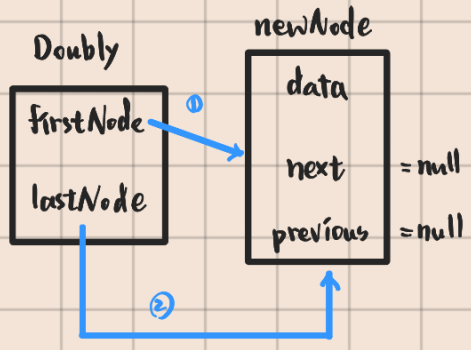


① first.previous = newNode

② newNode.next = first

③ first = newNode

*. List empty

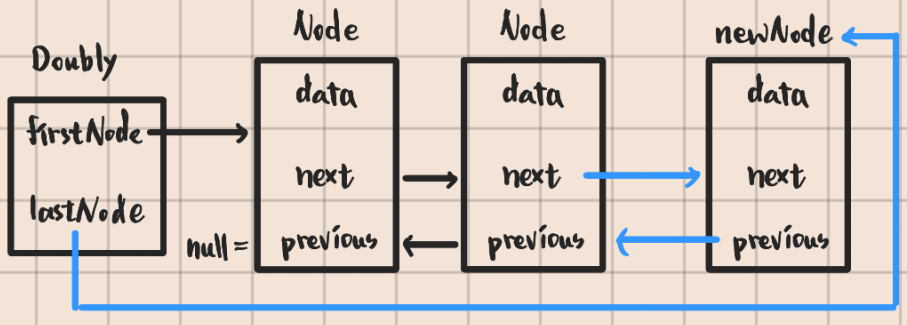


① first = newNode

② last = newNode

- Insert last

*. List not empty

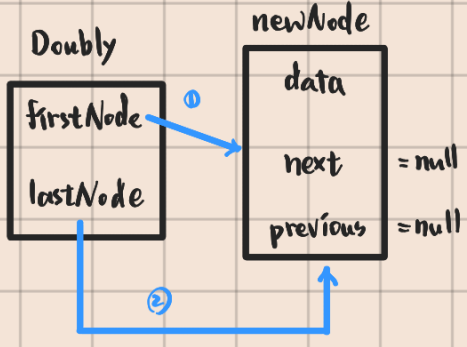


① last.next = newNode

② newNode.previous = last

③ last = newNode

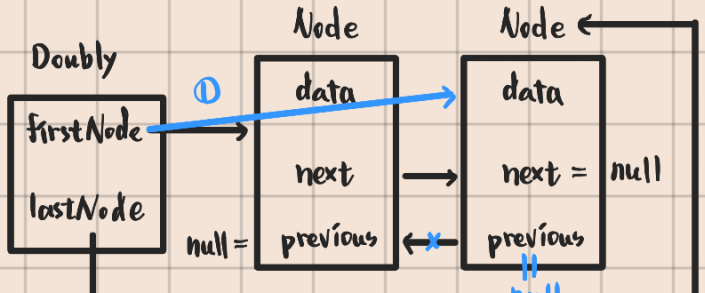
*. List empty



① first = newNode

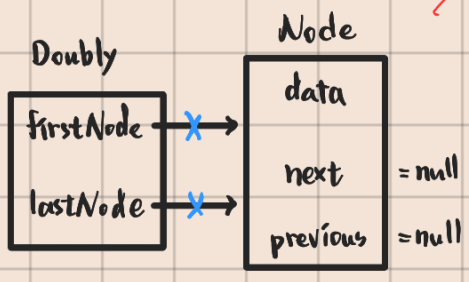
② last = newNode

- Delete first



① first = first.next

② first.previous = null

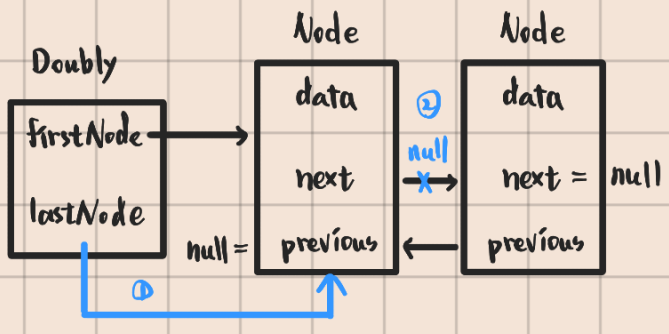


刪除要考慮 List 是否只有1个 node

① first = null

② last = null

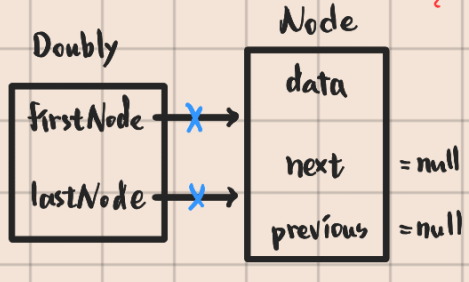
- Delete last



① last = last.previous

② last.next = null

刪除要考慮 List 是否只有1个 node

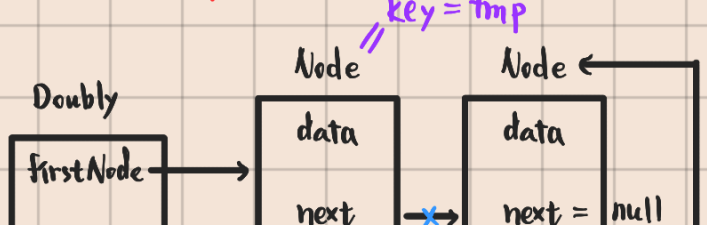


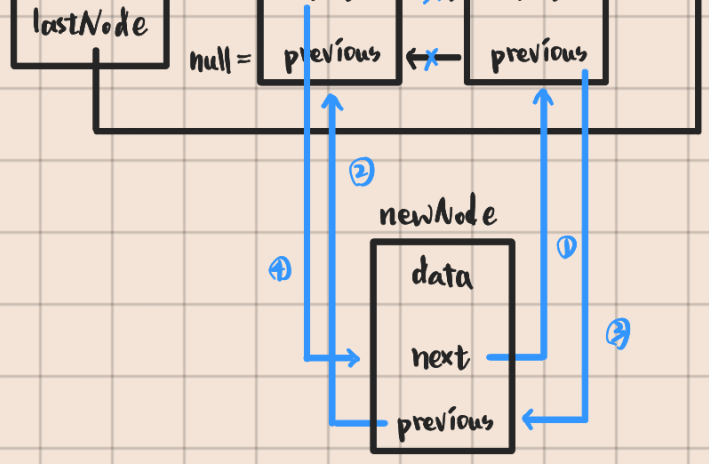
① first = null

② last = null

- Insert after

*. 插入在中間

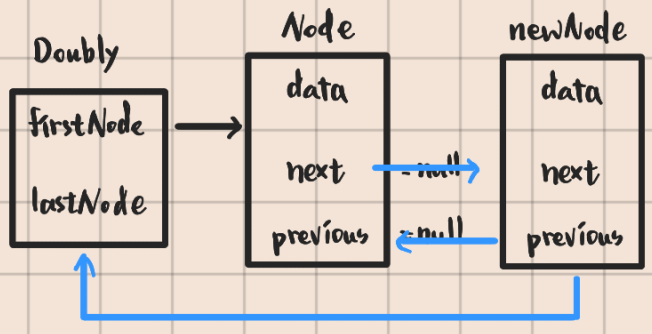




① newNode.next = tmp.next
② newNode.previous = tmp
} 先綁 newNode

③ tmp.next.previous = newNode
④ tmp.next = newNode
} 再改前後

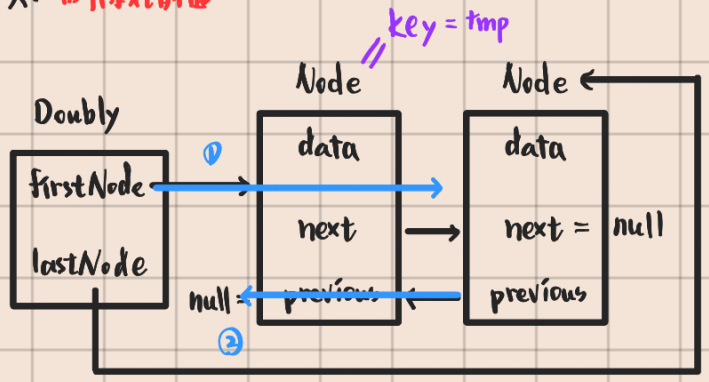
*. 插入在最後 if(tmp.next == null)



① newNode.previous = tmp
② tmp.next = newNode
③ last = newNode

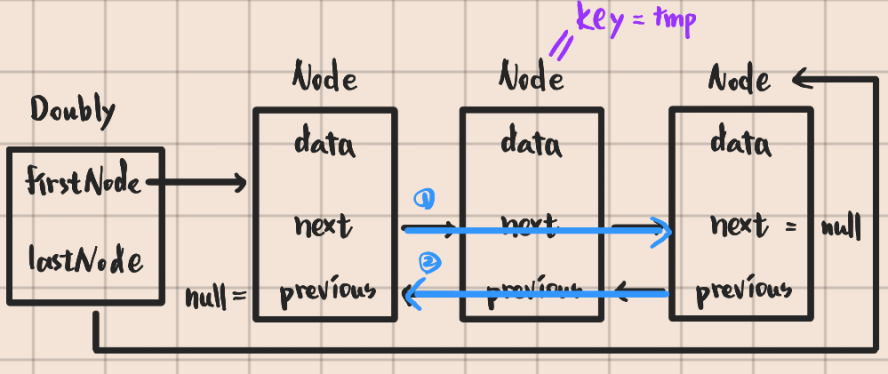
○ Delete key

*. 刪除在前面



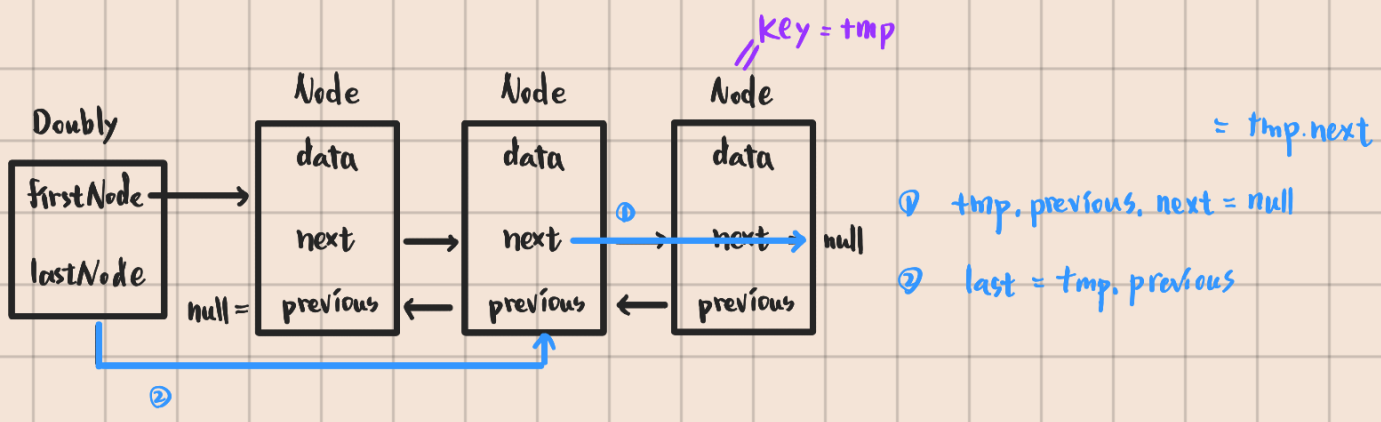
① first = tmp.next
② tmp.next.previous = null = tmp.previous

*. 刪除在中間



① tmp.previous.next = tmp.next
② tmp.next.previous = tmp.previous

X. 刪除在最後



X. 統整以上

改成 tmp = first 更好

if (~~tmp.previous == null~~) ← key是 1st Node

first = tmp.next

tmp.next.previous = null = tmp.previous

tmp = last

else if (~~tmp.next == null~~) ← key是 last Node

last = tmp.previous

tmp.previous.next = null = tmp.next

else ← key在中間

tmp.previous.next = tmp.next

tmp.next.previous = tmp.previous

再簡化

if (tmp = first)

first = tmp.next

else ← tmp"不是" fist Node

tmp.previous.next = null = tmp.next

if (tmp = last)

last = tmp.previous

else ← tmp"不是" last Node

tmp.next.previous = null = tmp.previous