Arrays versus Listas ligadas

		Array (não ordenado)	Lista simplesmente ligada
Inserção	Princípio	$O(N)$ ①	$O(1)$ ②
	Meio	$O(N/2) = O(N)$	$O(1)$ ②
	Fim	$O(1)$	$O(1)$ ②
Remoção	Princípio	$O(N)$	$O(1)$ ②
	Meio	$O(N/2)$	$O(1)$ ②
	Fim	$O(1)$	$O(1)$ ②
Indexação	Princípio	$O(1)$	$O(1)$
	Meio	$O(1)$	$O(N/2)$
	Fim	$O(1)$	$O(N)$

① Porque implica deslocar os elementos para a direita para arranjar espaço para o novo elemento.

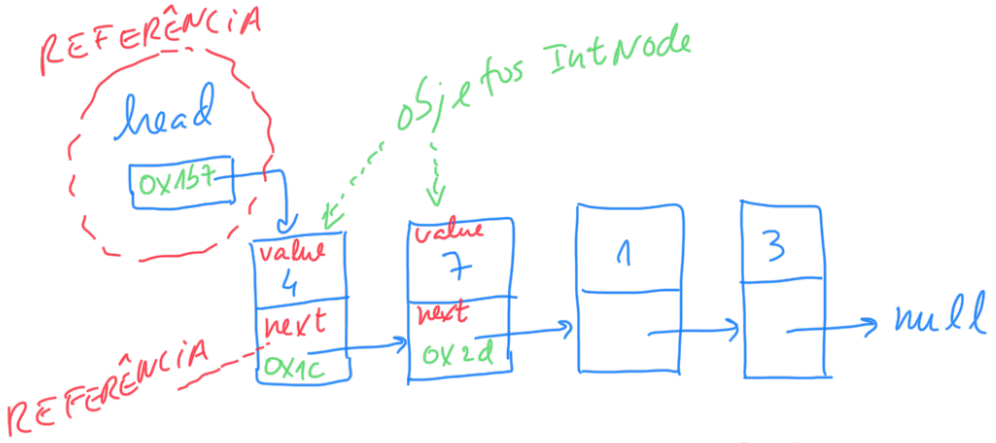
② Admitindo que é passado um ponteiro "prev" que aponta para o nó anterior.

Lista simplesmente ligada

- listas podem ter ou não nós auxiliares, designados de nós sentinela
  ↓
  usados para simplificar os algoritmos
- listas também podem ser circulares
  ↓
  último nó aponta para o primeiro nó
- listas simplesmente ligadas também podem ter um ponteiro para o último nó ("tail" pointer)
  ↓
  simplifica inserção no fim

lista simplesmente ligada, sem sentinela, não circular

conteúdo on elementos: 4, 7, 1, 3



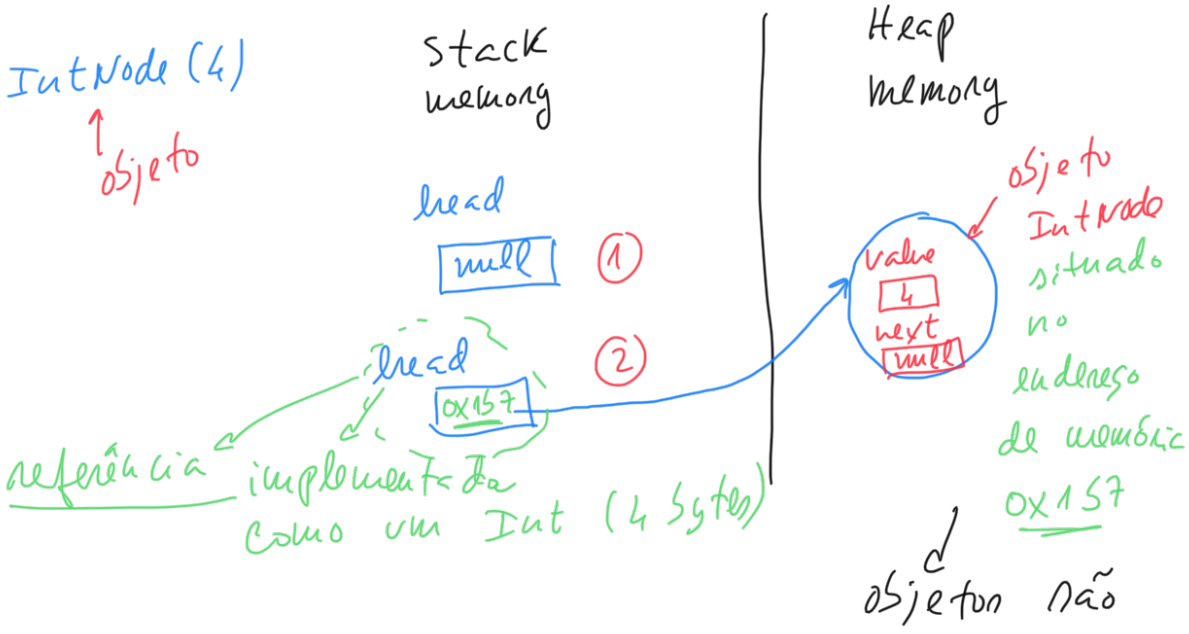
Uma referência é um valor/variável que aponta/referencia um objeto que reside num dado sítio da memória (heap memory)

main:

① var head : IntNode? = null

② head = IntNode(4)
↑ objeto

Memória



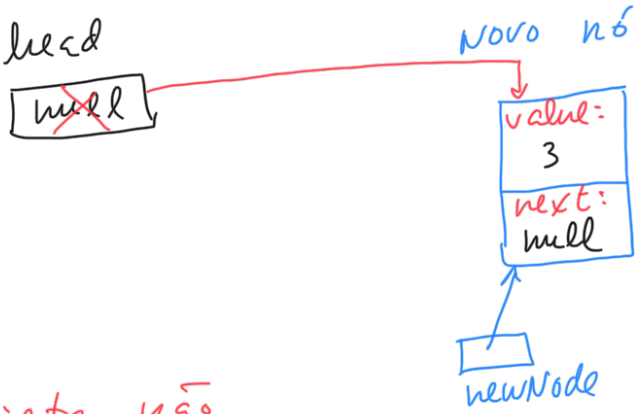
Pelo GC - Garbage Collector ← limpa da memória quando não são referenciados por ninguém

(Processo que corre em segundo plano na JVM)

addFront

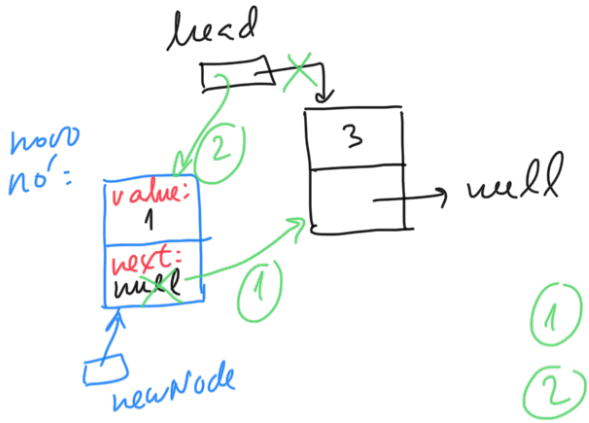
1º) caso: lista vazia

Inserir (3)



2º) caso: lista não vazia

Inserir (1)



(1) newNode.next = head

(2) head = newNode

6.

main:

```
val list : SIntList = SIntList()   // 4, 7, 1, 3

list.addFront(3)
 "    "      (1)
 "    "      (7)
 "    "      (4)

list.print()  // 4, 7, 1, 3
```

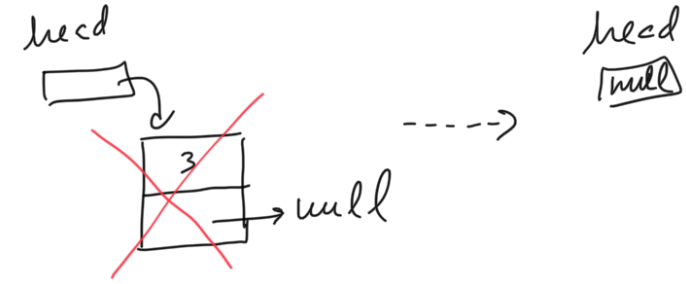
```
class IntNode {
    var value: Int
    var next : IntNode?
    constructor() { }
    constructor(v: Int) {
        value = v
        next = null
    }
}
```

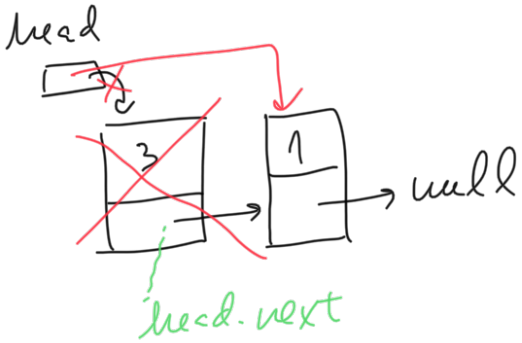
```
class SIntList {
    var head : IntNode?
    var size : Int
    constructor() { }
    fun addFront(v : Int) {
        val newNode = IntNode(v)
(1)     if (head == null
            head = newNode
        else {
(2)         newNode.next = head   (1)
            head = newNode        (2)
        }
        ++size
    }
```

removeFront

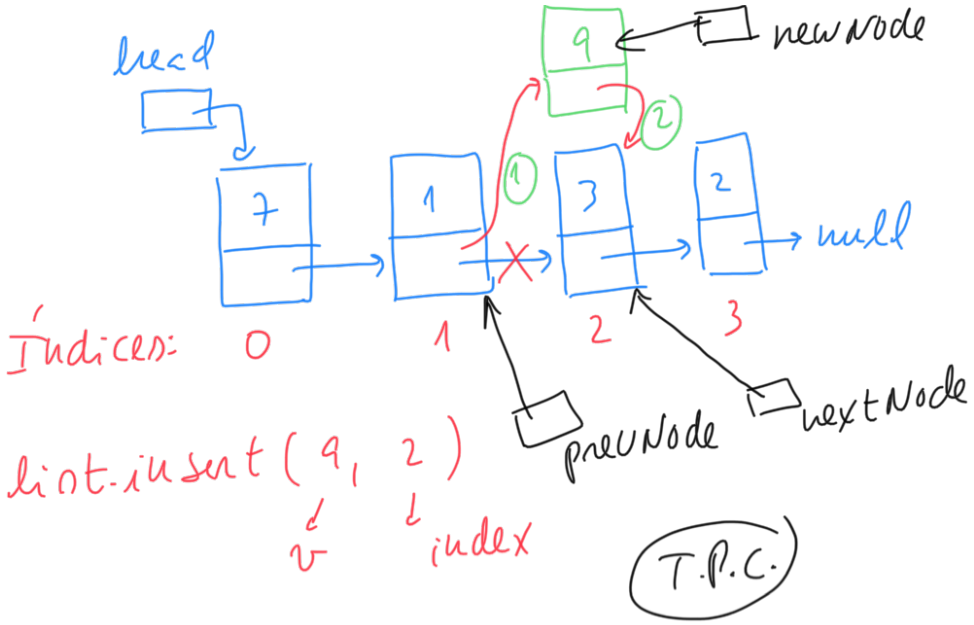
1°) caso: lista vazia → return null

2°) caso: lista não vazia





```
fun removeFront(): Int? {
    if (head == null) // vazia
        return null
    //não vazia
    val elem: Int? = head.value
    head = head.next
    -- size
    return elem
}
fun insert(v: Int, index: Int) {
```



list.insert(9, 2)

v index

T.P.C.