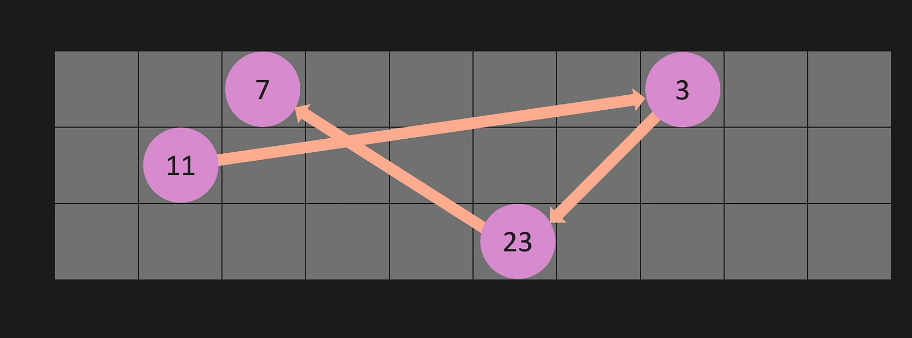
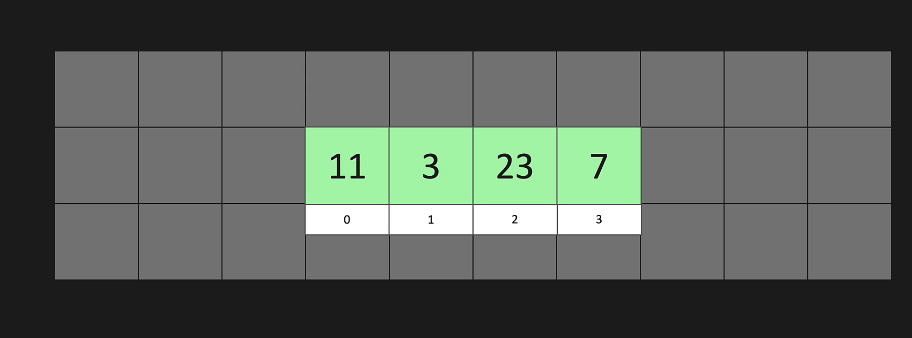
**About LinkedList**

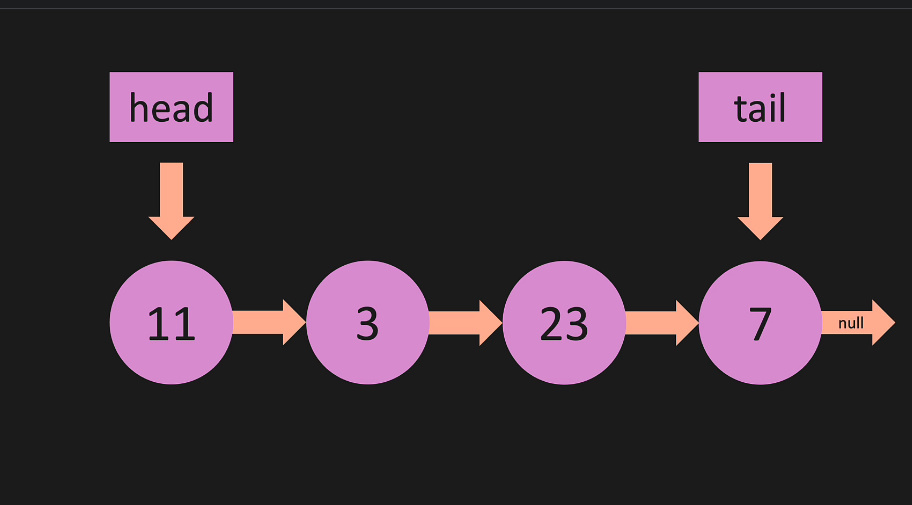
* LinkedList nu are indexi, spre deosebire de ArrayList
* LinkedList nu pastreaza elementele in memoria continua, ca ArrayList

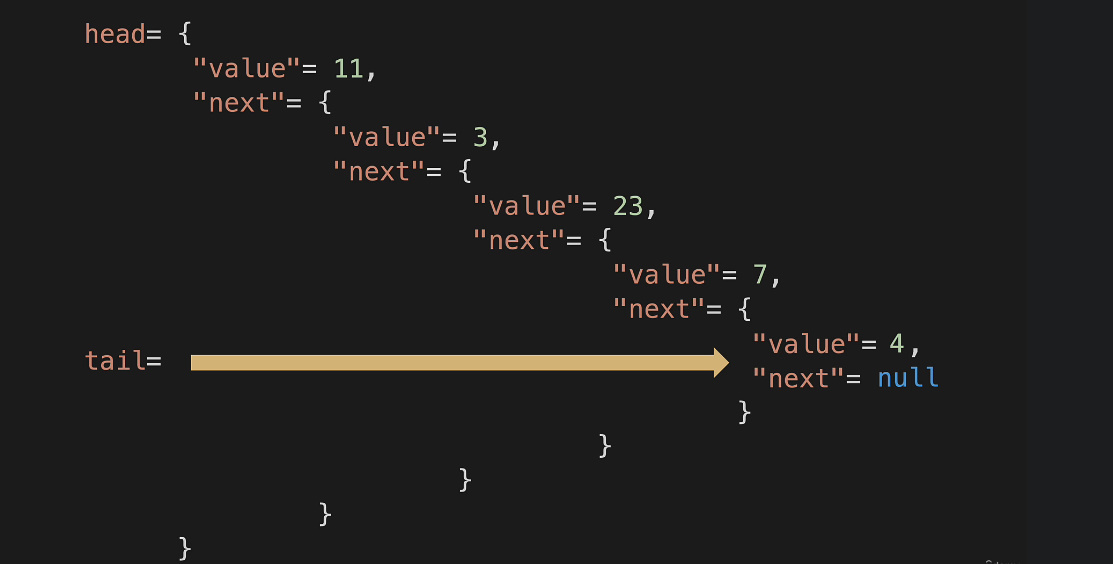


Dar ArrayList:



* Are head(cap) si tail(coada):





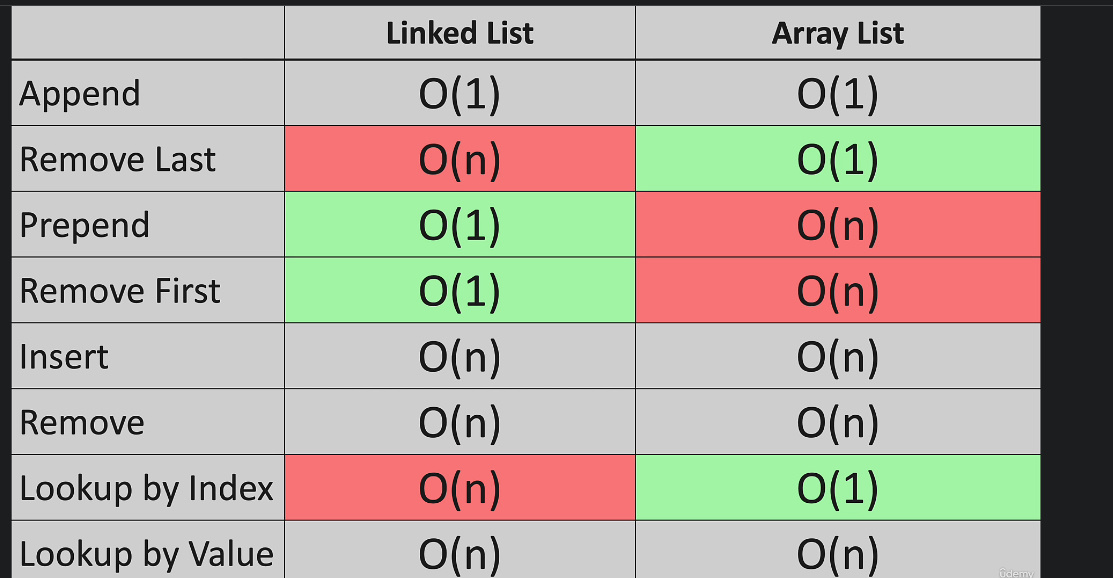
* Un LinkedList e format dintr-o valoare si un node next:



**LinkedList Big O**

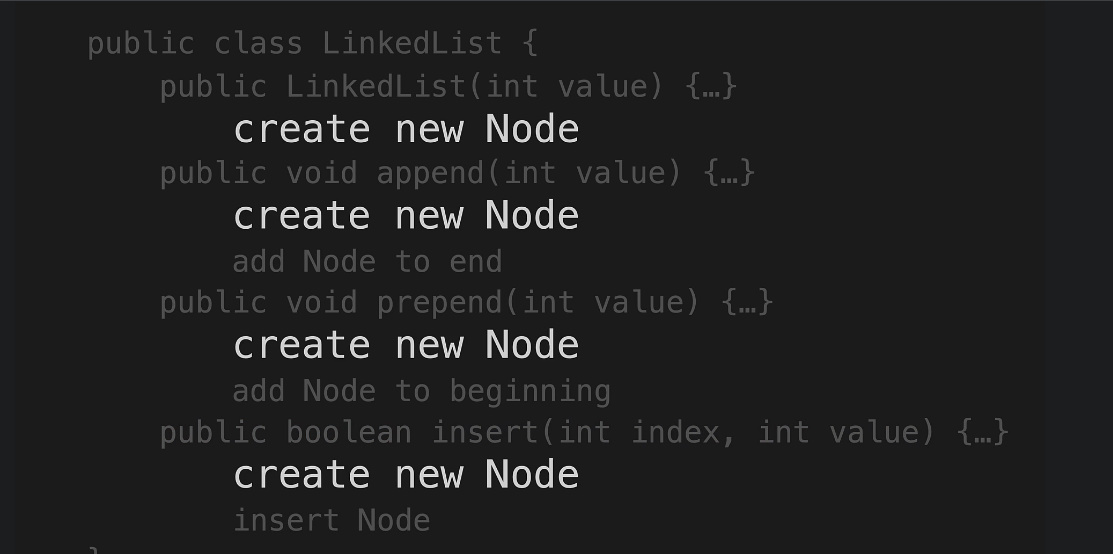
* **addLast**() – O(1) deoarece pur si simplu adaugam un element in tail si modificam tail si gata, nu parcurgem toata lista
* **addFirst() –** O(1) doar adaugam in noul node referinta la curent head si apoi mutam head la el
* **removeLast**() – O(n) , deoarece tail va fi null asa cum se sterge ultimul element si trebuie sa parcurgem toata lista pana la ultimul element ramas care va fi deja tail
* **removeFirst()** – O(1)
* **insert(index)** – O(n)
* **removeNode(index) –** O(n)
* **getValue(value)** – O(n)
* **getByIndex(index)** – O(n)

Anume la getByIndex() ArrayList e mai bun, deoarece in LinkedList trebuie sa parcurgem lista, dar la ArrayList e suficient sa accesamul indexul din zona de memorie continua si atat.

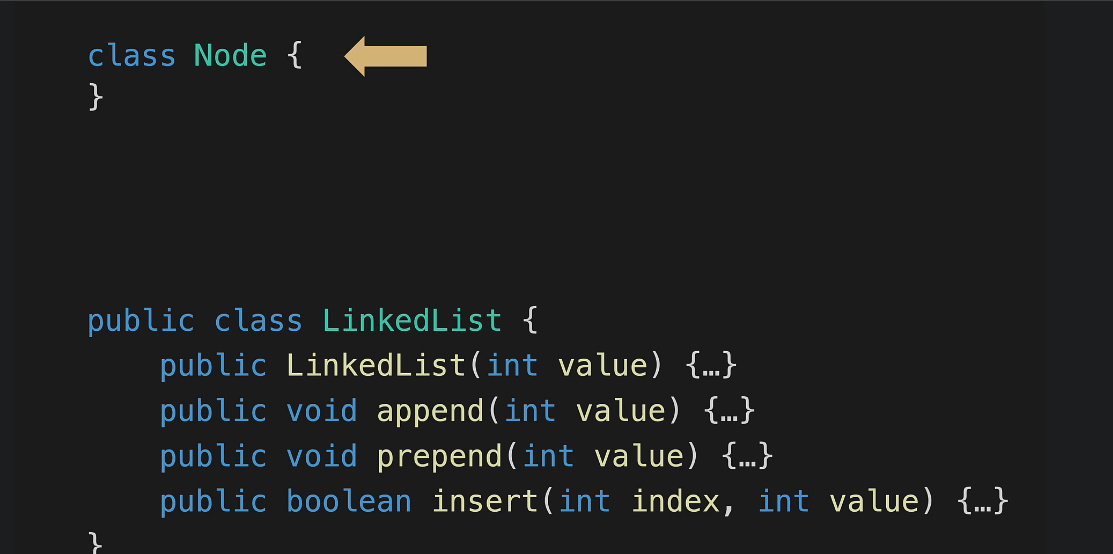


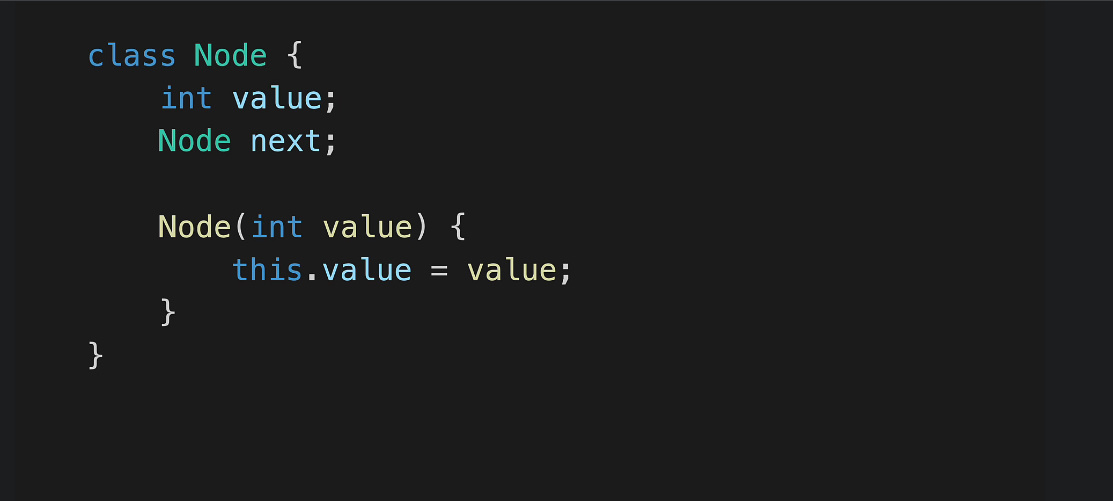
Dar ArrayList are problema ca trebuie mereu sa modificam indexii la alte elemente daca de ex stergem un element

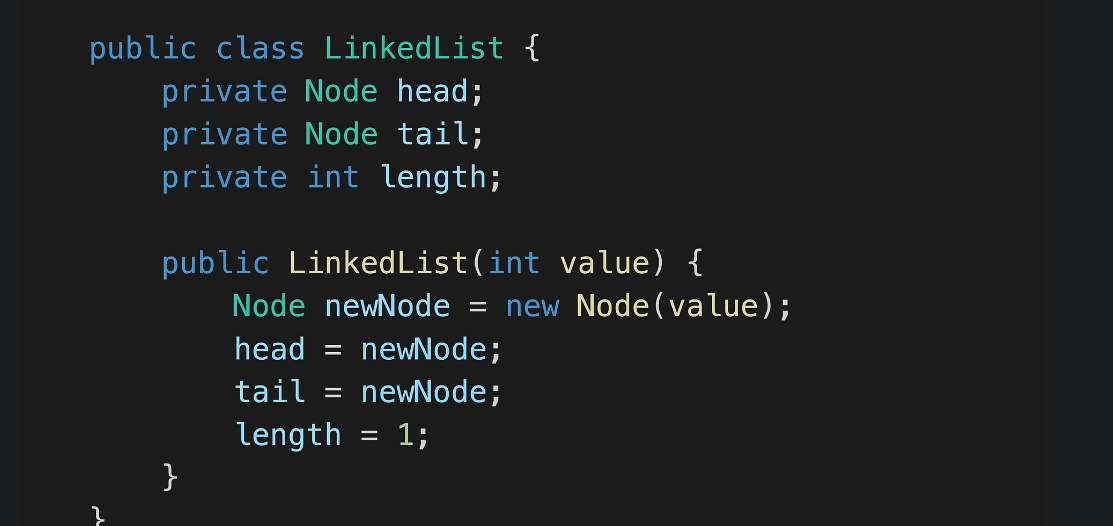
**Constructor**

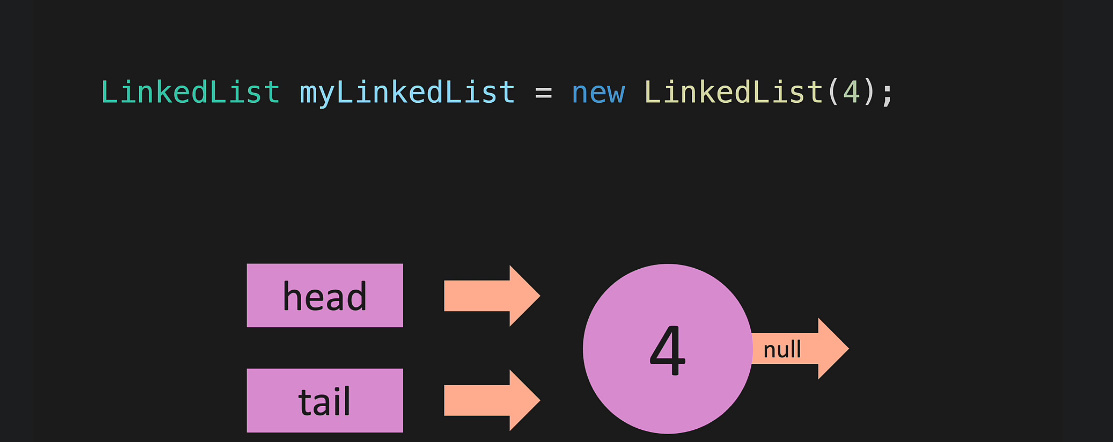


Constructorul LinkedList(value) – doar creaza o LinkedList cu un nod cu valoarea trimisa

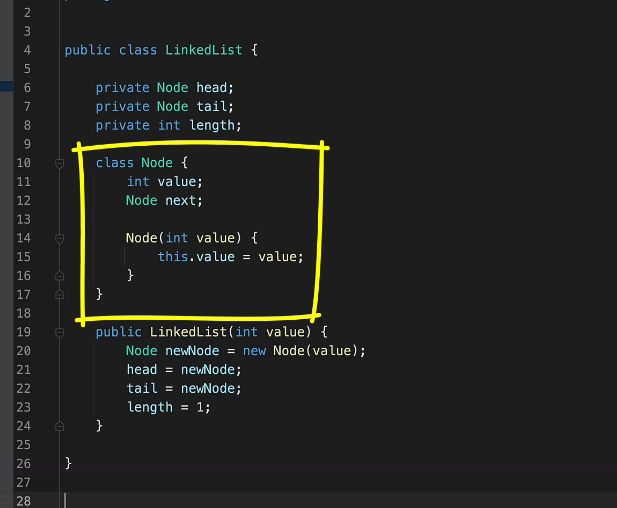






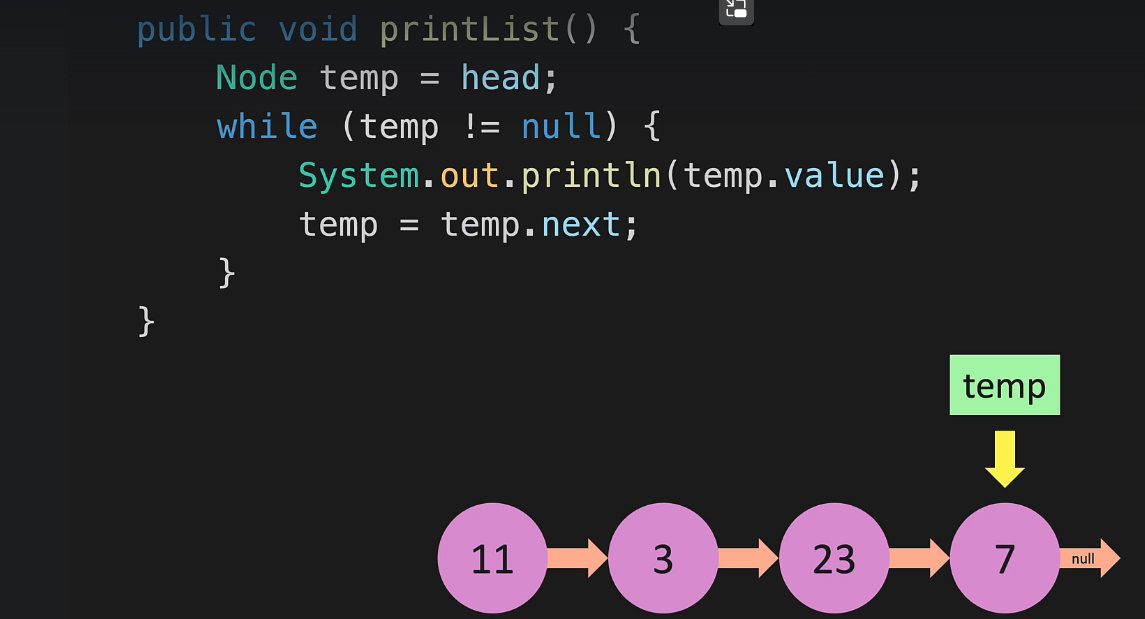


Node class va fi inner class!

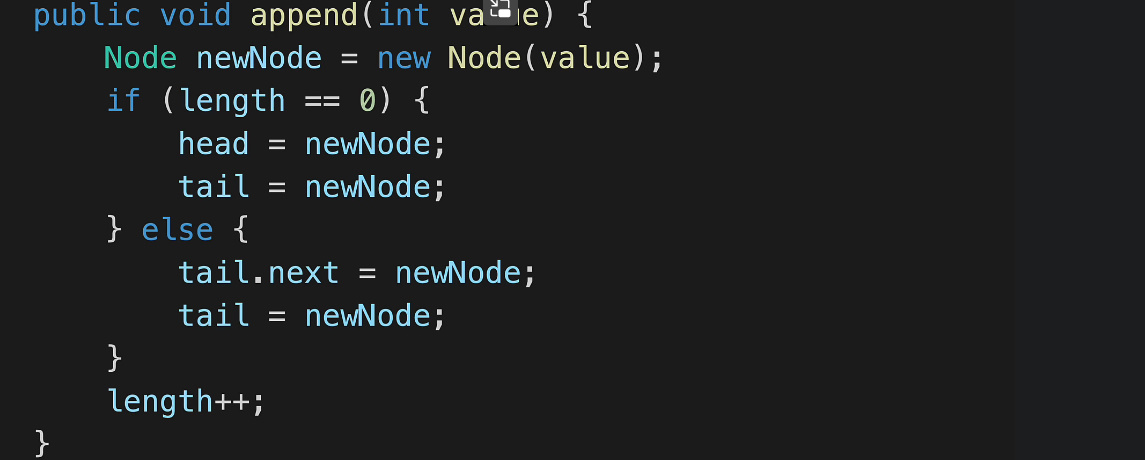


Nu punem nimic din inner class ca private, deoarece mai apoi vom vrea sa cream un fisier separat pentru ea in package, ca si alte data structures sa o foloseasca

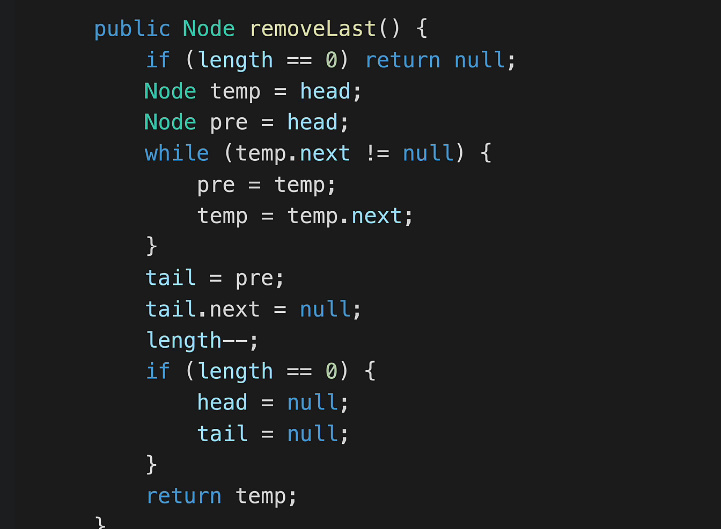
* printList()



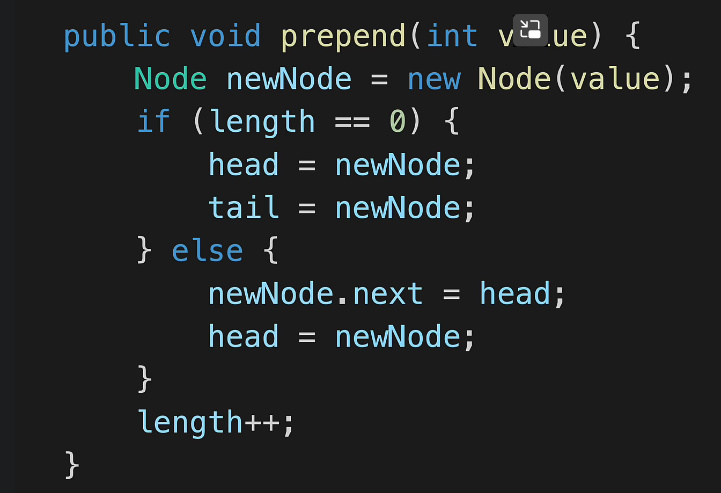
* append()



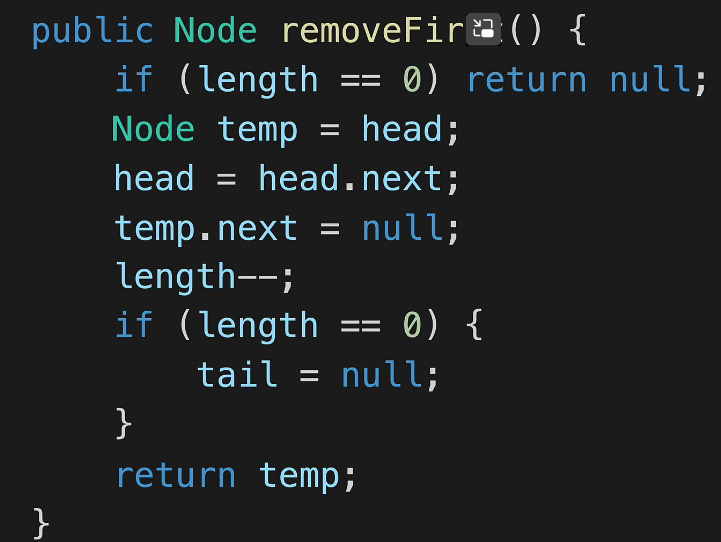
* removeLast()



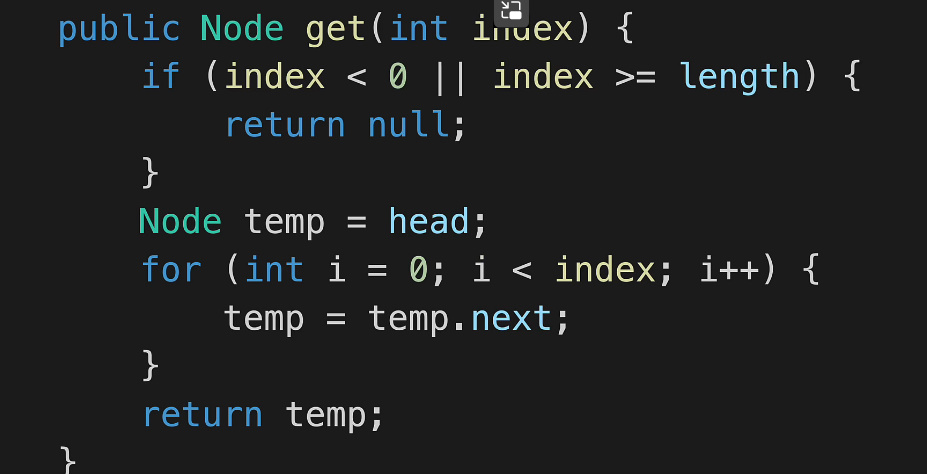
* prepend() – adauga un element pe prima pozitie:



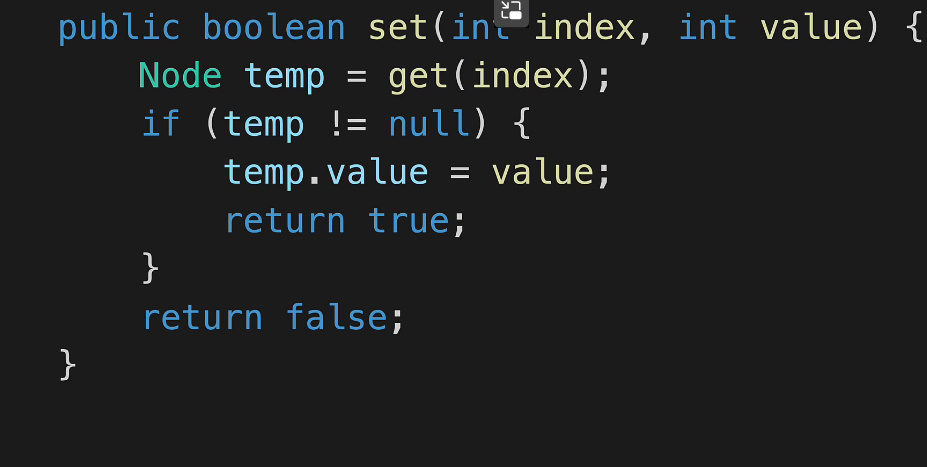
* removeFirst()



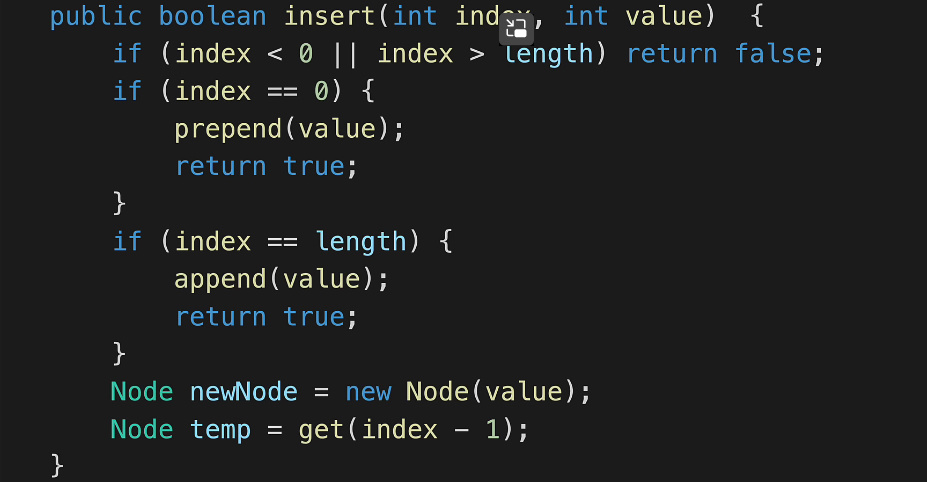
* get(index)



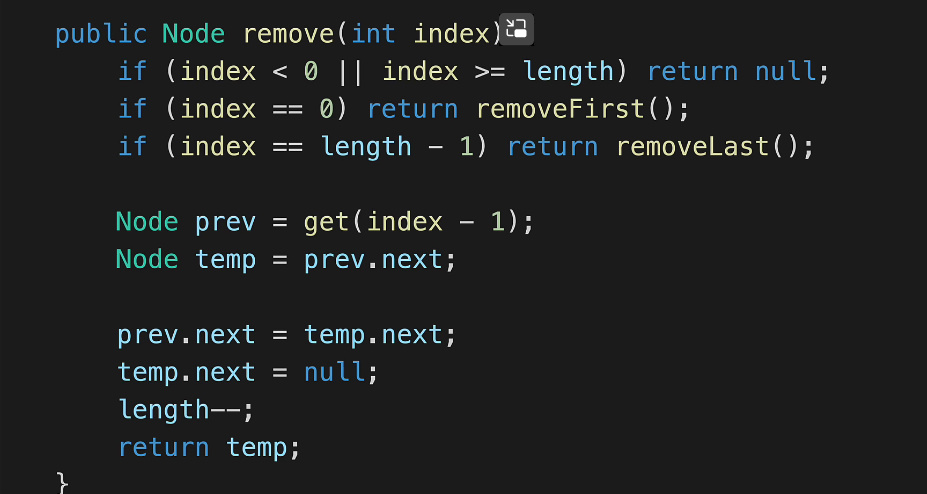
* set(element)



* insert(index, value)



* remove(index)



* reverse()

