* Stack se bazeaza pe LIFO sau FILO, care sunt tot aceeasi
* **Teoretic**, stack se bazeaza pe aceea ca itemele sunt puse unul peste altul si tot asa eliminate. De ex, pe masa punem o foaie, apoi fiecare nou item va sta peste acea foaie, adica nu va fi pus sub foaie, dar peste, ca si cum in fata la primul element de deasupra, si e evident ca eliminarea foilor incepe din varf, de la ultimul item pus, nu de la primul cum se face in vector, linkedlist etc.
* Totusi, in code, stack se implementeaza ca si LinkedList sau Vector,si modul in care cele 2 o implementeaza difera:

**ArrayList:**

* Adauarea se face la ultimul item,dar eliminarea tot incepe de la ultimul index,adica de la capat, nu de la 0, si tot se primeste ca cu exemplul cu foile, sau cel putin asa pare, desi implementarea nu ciar asa ca in realitate
* Stack cu ArrayList se bazeaza pe principiul ca eliminarea elementelor nu incepe de la indexul 0, ci de la urma, dar adauarea lor se face la urma.
* Elementele noi tot se pun la final de lista. De ex, daca avem:

1 2 3 4 5

si dam push(9) , tot vom avea o lista cu 9 la urma, ca la toate structurile de date cu liste:

1 2 3 4 5 9



* **Dar anume ceea ce face stack diferit la implementarea cu ArrayList, e ca eliminarea element cu element se face de la urma , de la dreapta la stana,daca nu oferim index:**

1 5 1 2 3 1

remove()

1 5 1 2 3 1



* Daca oferim index, tot se incepe de la stanga la dreapta
* O stack cu ArrayList are indexi exact ca si un ArrayList, adica 0 incepe de la stana si creste spre dreapta. **Daca indexii nu ar fi asa, am avea o complexitate rea, asa cum ar trebui sa modificam indexii la fiecare element.**
* Un stack poate fi implementat si cu un ArrayList si cu un LinkedList. Unica cerinta e ca sa incepem stererea elementelor tot de acolo unde incepem adauarea lor(fara a oferi indexi)
* **In Java, stack extinde Vector, si deci si ArrayList!Asa dar, indexii incep de la stana la dreapta, elementele noi se pun la urma, si eliminarea incepe tot de la urma, daca nu oferim indexi**
* **Top e ultimul element, adica din ultimul index**

**Stack cu LinkedList**

* Un stack poate fi implementat si cu un LinkedList
* Aici, am putea fie face ca ordinea de sterere si inserare sa fie ca la ArrayList, dar am putea face si altfel
* Daca am face adauarile la urma si eliminarile tot de la urma,ca si cu ArrayList, complexitatea va fi rea(vezi la complexitate)
* Iata de ce, cand un Stack e implementat cu LinkedList, adauarea se face in indexul 0, si stererea tot din indexul 0 incepe:
* 1 2 3 4 5

add(9)



9 1 2 3 4 5



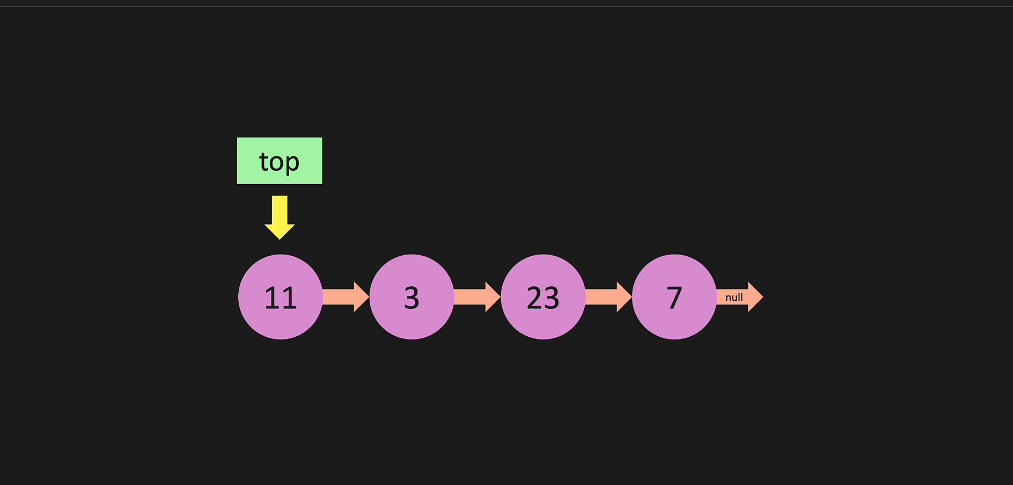
* 1 2 3 4 1

pop(1)

1 2 3 4 1



* **Deci stack cu LinkedList e exact ca stack din realitate**
* **Top e primul element!**



**Complexitate**

**Stack mai mult e facut pentru a adaua un element si a-l stere pe cel din varf, si deci anume complexitatea lor conteaza cel mai mult**

**Stack Implementant cu ArrayList:**

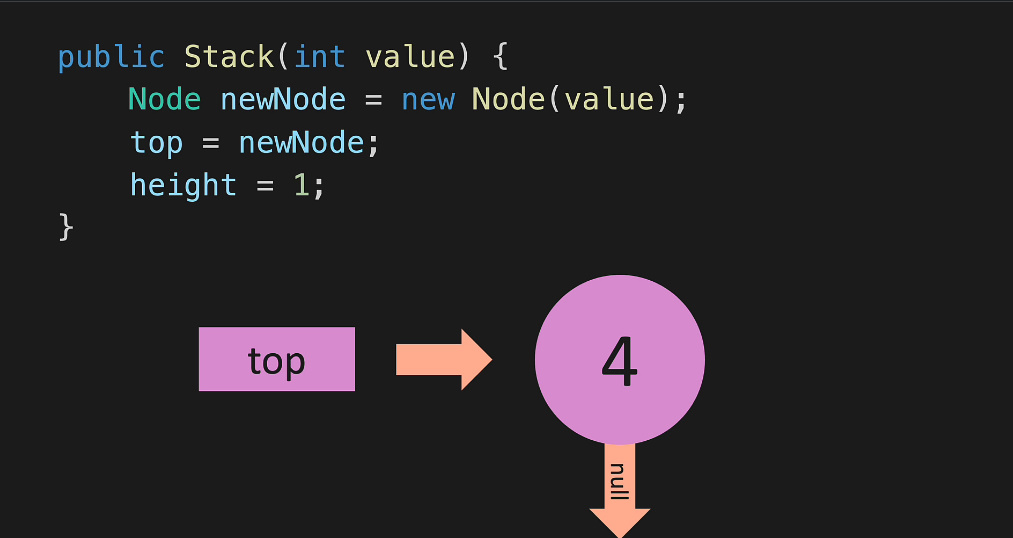
* append() – O(1). In viata reala, stack prevede ca punem elementele unul peste altul,si asa si le luam, dar daca am face asa in code, folosind un ArrayList, daca vom pune fiecare element nou in indexul 0, va trebui sa modificam toti indexii la toate elemntele, si asa vom avea O(n), nu O(1), de aia si punem fiecare nou element la urma, si de acolo si incepem deja eliminarea, ca tot se primeste aceeasi ca in viata reala
* remove() – O(1) – se refera la a stere ultimul element
* remove(index) – O(n)

**Stack implementant cu LinkedList:**

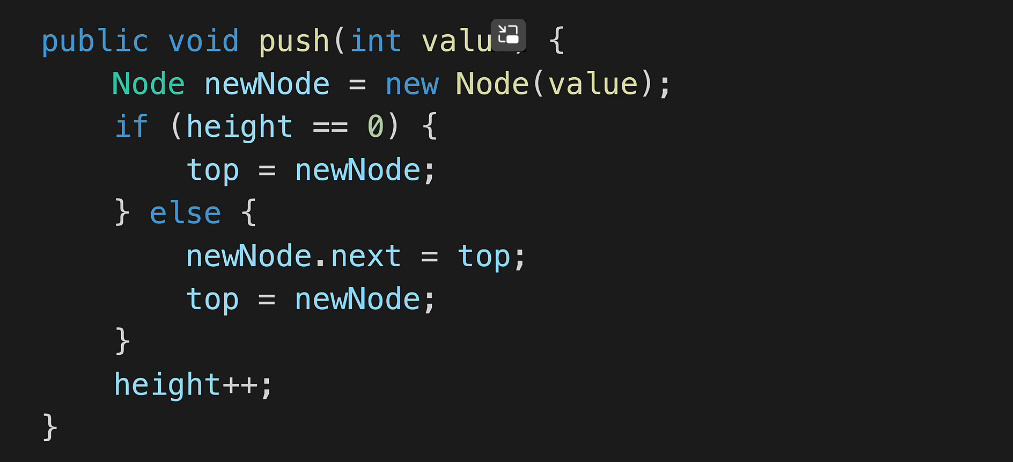
Totusi, cu LinkedList e mai comod sa cream o stack ca in viata reala. Adica, fiecare nou nod va fi pus la inceput, nu la urma, si eliminarea deci tot va incepe la inceput

* pop() – O(1) – se refera la a stere ultimul element, dar daca am pune elementele noi la urma si am incepe a stere de la urma, ar fi O(n)
* append() – O(1)

**Metode**



* push(value)



* pop()

