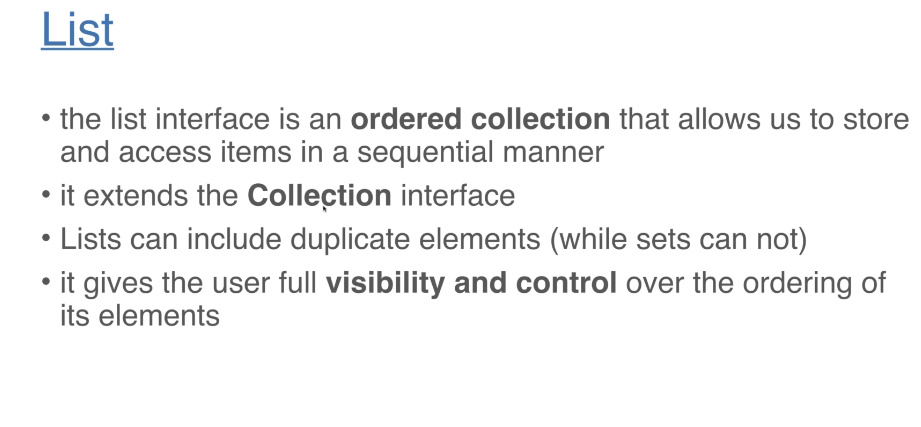
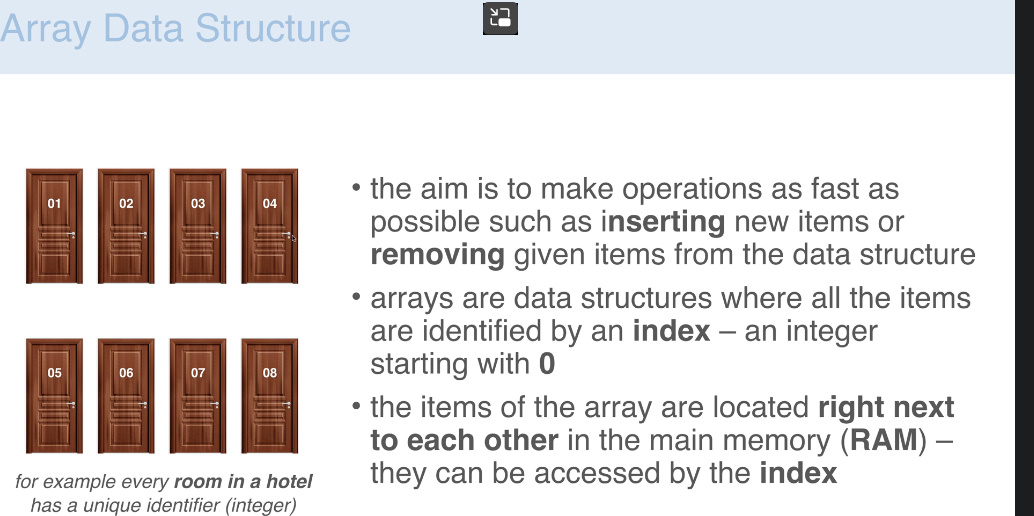
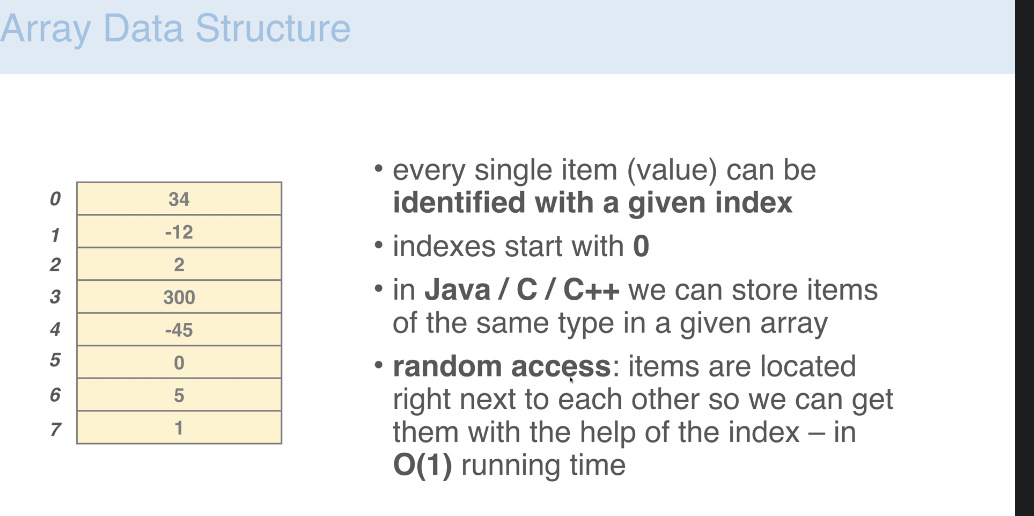
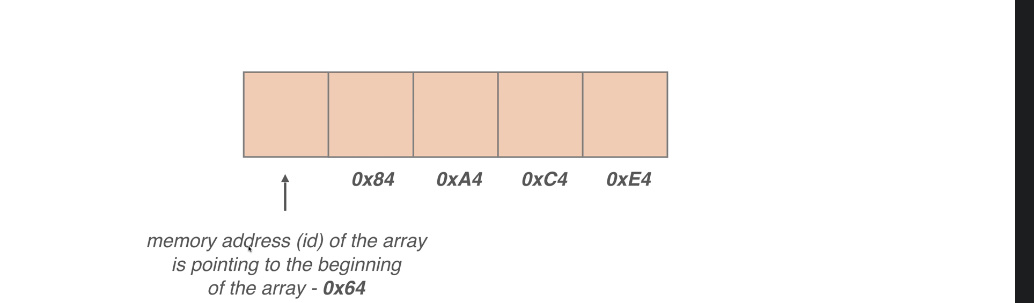
**List Interface**

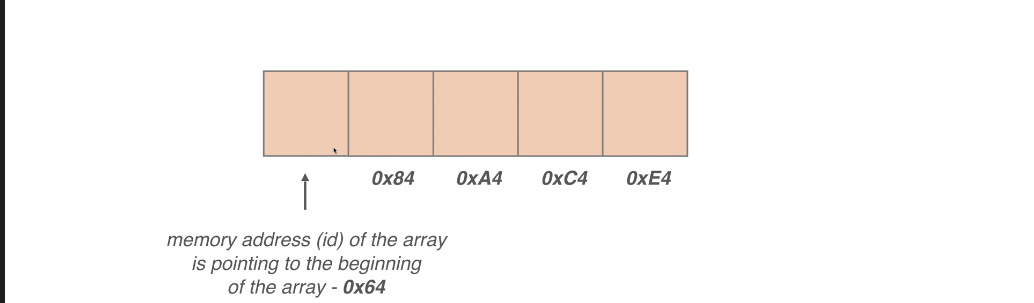


**Arrays**

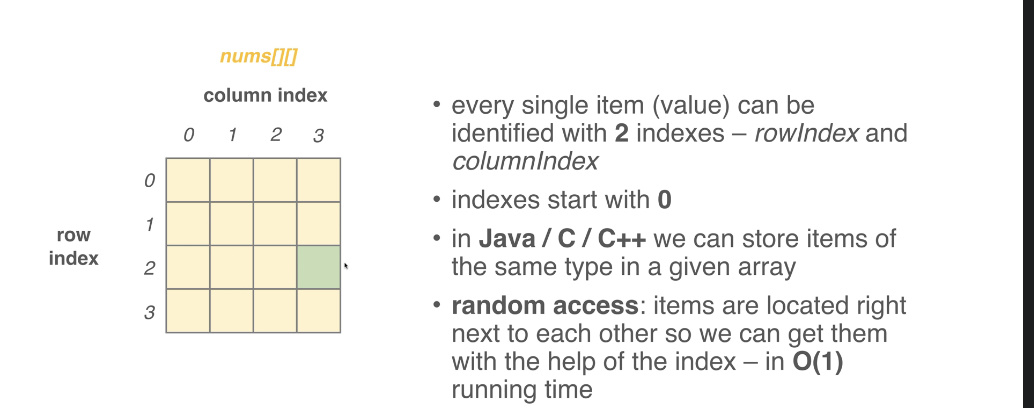


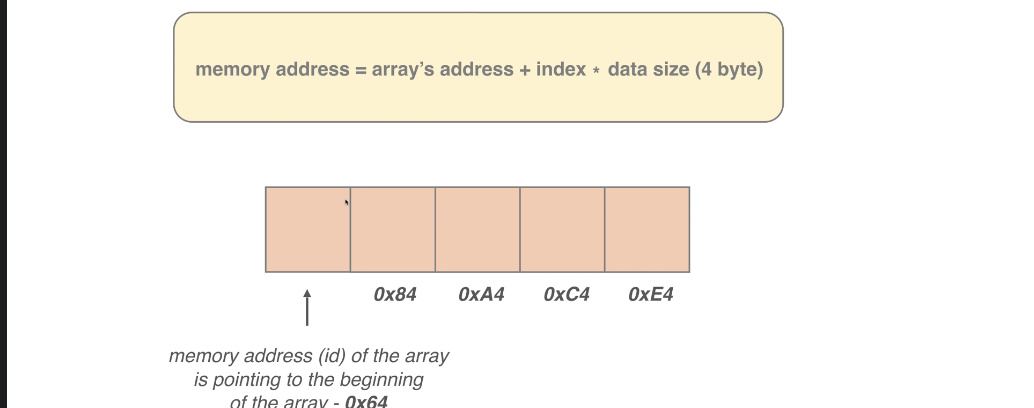
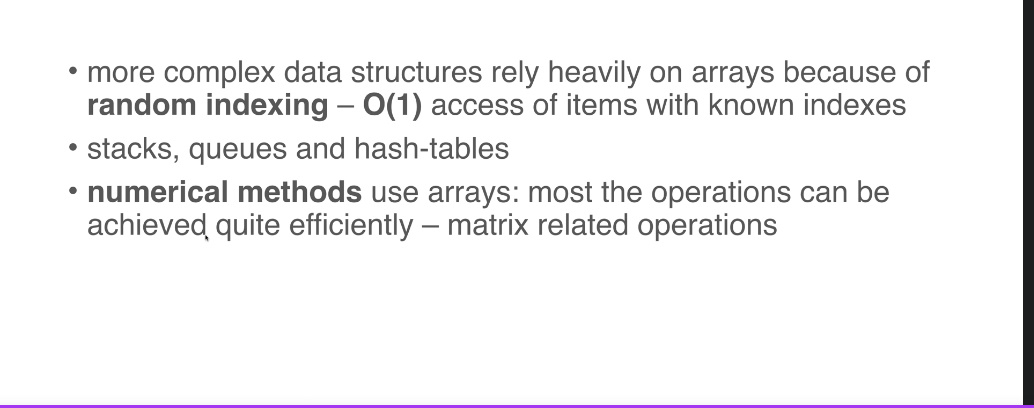
* Deci, avantajul la arrays e ca putem accesa iteme dupa index foarte repede si usor.
* 
* 

Deci, fiecare iteme din array sunt puse unul langa altul in RAM

* 

adresa de memorie a unui array este adresa de memorie a primului element din array



* 
* 
* 

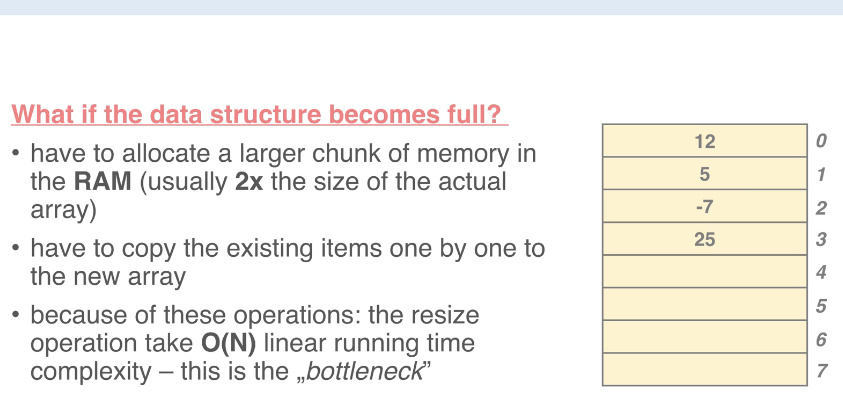
**List Methods**

* **Metode**:
* add(Item) – pune un item la final de array
* add(index, Item) – pune un item la indexul dat
* addAll(Collection) - adauga toate itemele din colectie la urma. Returneaza true daca reuseste, false daca nu
* addAll(index,Collection) – adauga itemele in lista curenta incepand cu indexul specificat, nu la urma
* set(index,Item) – actualizeaza un element de pe pozitia data
* indexOf(Item) – returneaza prima aparitie a lui Item
* lastIndexOf(Item) – returneaza ultima aparitie a lui Item

Ambele returneaza -1 daca elementul nu e gasit

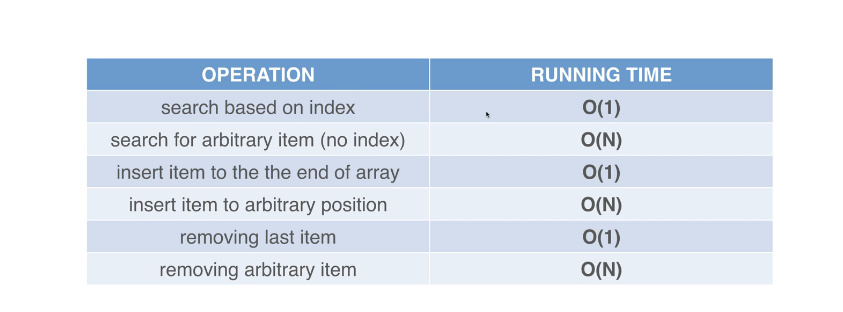
* remove(Object) – sterge primul obiect gasit
* remove(index) – sterge obiectul de la indexul dat
* get(index)
* toArray() – returneaza un array de Object[].
* toArray(array) - pune itemele din list in array creat de noi. Daca acesta nu e destul de mare, se va crea unul nou.
* retainAll(Collection) – retine toate elemntele ce sunt prezente in Collection
* removeAll(Collection) – sterge toate elementele care sunt prezente in ArrayList si in Collection data de noi
* removeIf(Predicate) – sterge itemele care corespund conditiei

**Array/ArrayList Operations**



Deci, daca de ex array este deja plin, dar mai vrem sa adaugam iteme in el, trebuie sa cream unul nou, cu mai multa memorie, si sa copiem toate itemele din vechiul array in el si sa le punem pe cele noi. De asta, nu e prea bine sa dam resize la un array, ca in Java nu putem propriu zis sa marim memoria la un array, doar copia datele intr-un nou array

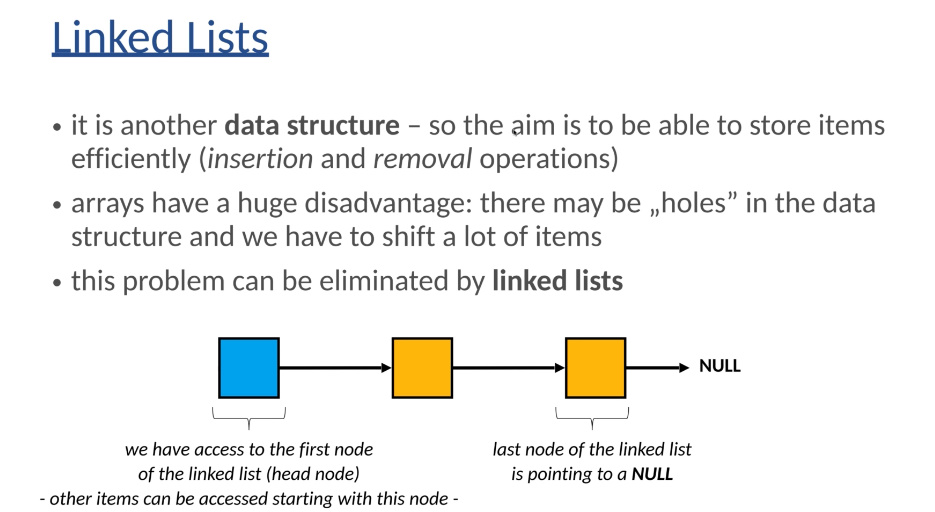
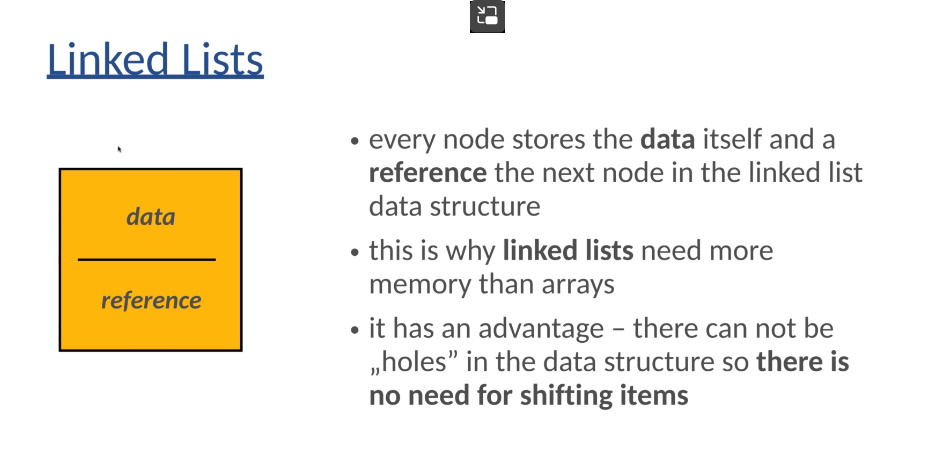
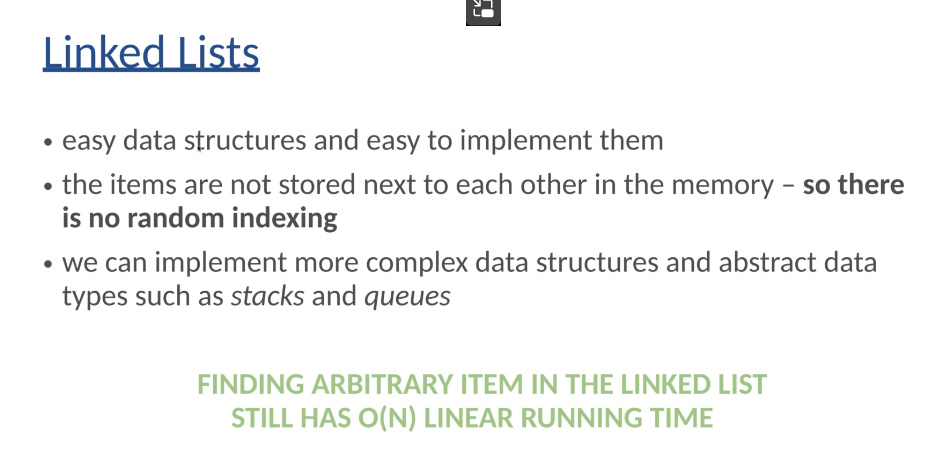
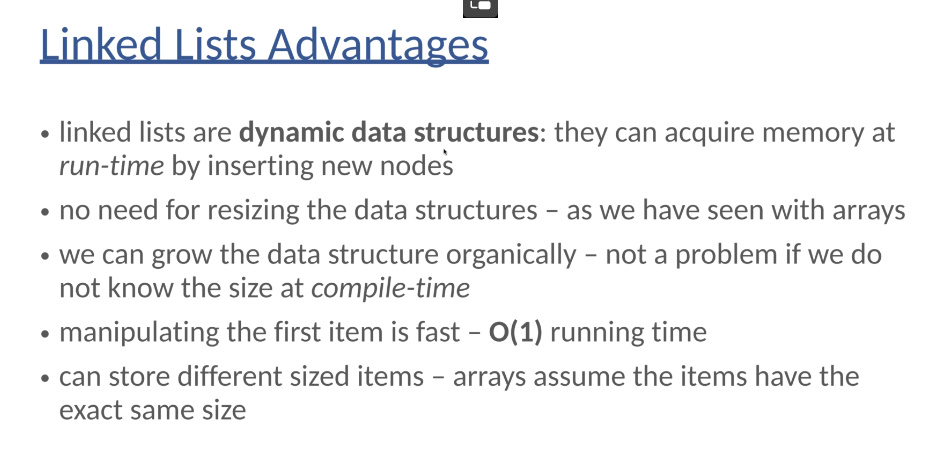
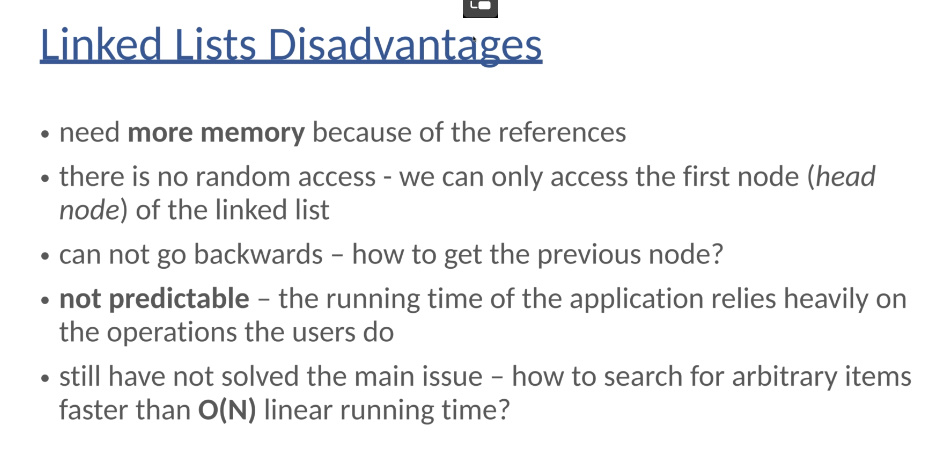
* Aici si e problema la arrays, pe deoparte ne ajuta sa economisim memorie ca specificam capacitatea initiala, dar pe de alta parte, daca trebuie sa marim memoria, trebuie sa cream un alt array si sa copiem toate elementele in el
* Daca cream un array cu un numar foarte mare de iteme, desi nu ne trebuie, dar ca asa ca sa avem rezerva, atunci vom irosi memorie
* ArrayList cam asa si lucreaza. El creaza un array Object[] cu capacitatea initiala de 10, si daca adaugam mai multe elemente decat el poate pastra, pai el va trebui sa creeze un nou array Object[] si sa copie in el vechile elemente. Noul array va avea memoria dubla fata de precedentul.
* Adaugarea sau stergerea unui item intr-un index oarecare, nu ultimul, va face ca toate itemele dupa el sa isi modifice indexul, de asta add() are O(n), adica ele vor fi mutate inainte sau inapoi
* In schimb, daca punem sau stergem iteme la urma, avem O(1)
* Accesarea itemelor dupa index e foarte rapida cu ArrayList, O(1)



**ArrayList**

* ArrayList are 3 constructori:
* ArrayList() – are capacitatea initiala la array egala cu 10
* ArrayLisy(Collection) – creaza un ArrayList ce are in el un array cu toate elementele din colectia data
* ArrayList(int size) – creaza un arraylist cu un array gol, dar cu size oferit de noi
* Deci, daca stim ca vom stoca foarte multe iteme, nu e de loc o idee buna sa lasam default size de 10, caci adesea Java va trebui sa tot creeze noi arrays cu ArrayList, ceea ce e foarte costisitor
* ArrayList e rapid daca citim iteme, adaugam/stergem ultimul item.
* **Metode specifice:**
* iterator(index) – returneaza un iterator cu elementele incepand de la index
* removeRange(from,to) – sterge itemele de la from pana la to, dar to este exclusiv

**LinkedList**

* **LinkedList implementeaza Doubly Linked List Data Structure**
* Deci, deoarece e o Doubly Linked List, putem manipula inceputul si finalul ei
* LinkedList e mai des folosit ca ArrayList
* 
* Deci, ArrayList nu e prea buna la adauga si sterge iteme, si mai ales ca se creaza adesea un nou array, cand nu mai incape in cel vechi, cu un size dublu fata de cel precedent, si se poate intampla asa ca sa se ocupe multa memorie degeaba
* Daca chiar si e suficienta memorie, inserarea sau stergerea unui item(inafra de ultimul) face ca toate celelalte elemente sa fie mutate din pozitia lor.
* LinkedList sunt mult mai rapide daca inseram iteme si le stergem(nu ultimul doar)
* 
* LinkedList necesita mai multa memorie ca ArrayList
* 
* Gasirea unui item dupa index e O(N), si aici e mai rau ca la ArrayList
* Dar, inserarea unui item la inceput e mai rapida, O(1)
* LinkedList e foarte bun daca lucram cu primul/ultimul item sau inseram iteme, dar nu e prea bun daca accesam multe iteme si le modificam.
* 
* 



* Cel mai mare dezavantaj deci e accesarea itemelor, care e O(n), caci trebuie sa parcurgem nodurile de la head pana la acel item
* **Constructors**:

LinkedList()

LinkedList(Collection)

* **Metode Specifice(+ cele de la Deque)**
* addFirst(item)
* addLast(item)
* descendingIterator() – returneaza un iterator ce merge de la ultimul element la primul
* getFirst()
* getLast()
* offer(item) – adauga elementul la urma, ca tail, exact ca add()
* offerFirst(item) – adauga elementul ca head, adica primul, ca addFirst()
* offerLast(item) – adauga itemul la urma, ca offer()
* peek() – returneaza primul element
* peekFirst() – ca peek
* peekLast() – returneaza tail
* poll() – returneaza si sterge primul item
* pollFirst() – ca poll
* pollLast() – returneaza si sterge ultimul item
* pop() – returneaza si sterge primul item, ca la Stack creat cu LinkedList
* push(Item) – adauga un item ca head, ca si la Stack creat cu LinkedList
* remove() – returneaza si sterge Head
* removeFirst() – sterge si returneaza primul element
* removeLast()
* element() – returneaza HEAD

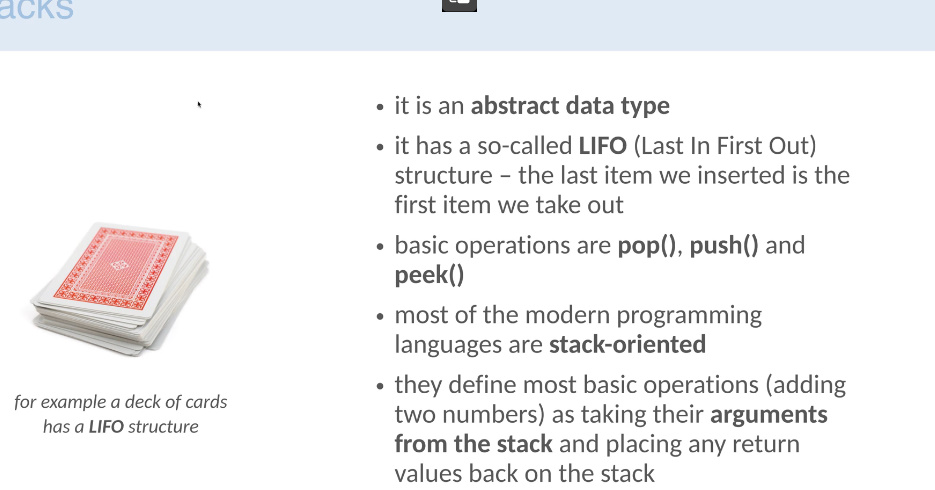
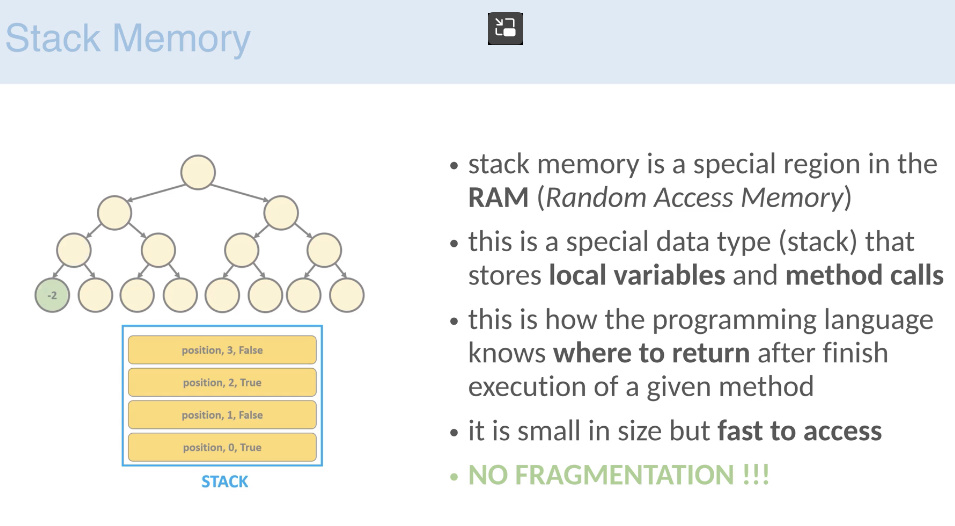
**Vector**

* E Thread Safe
* **Nu Extinde ArrayList**
* Totusi, synchronization e scumpa, deoarece daca de ex avem un for() care adauga iteme intr-un array, acel thread va trebui sa dea lock la Monitor lock si apoi unlock, si apoi iar si iar si iar, si asta e scump.
* Vector nu mai e folosit aproape de loc, de aceea poate fi ignorat.
* vector are un protected field:

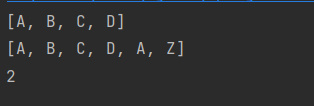
int capacityIncreament

* Vector, default, ca si ArrayLisy, isi dupleaza capacitatea daca nu ajunge spatiu in array, dar daca noi vom seta un capacityIncrement, spatiul lui va creset cu cat am setat noi acel capacityIncreament

**Stack**

* Extinde Vector, deci se bazeaza pe array
* Se bazeaza pe LIFO
* 
* Ultimul element e primul care e sters mereu
* Spre deosebire de ArrayList, putem accesa ultimul item
* 
* Are doar un singur constructor: Stack()
* Metode:
* pop() – returneaza si sterge ultimul element, adica Top
* push(Item) – adauga elementul la urma, si el devine Top, adica elementul desupra la toate
* peek() – returneaza ultimul element, adica Top
* empty() – returneaza true daca exista Top sau nu
* search(Item) – cauta itemul de la dreapta la stanga, adica de la urma la inceput, si returneaza deci cel mai aproape iteme de Top. Atentie! Incepe de la 1,nu de la 0, asa cum Top este 1, ce e dupa Top 2 si tot asa.

public static void main(String[] args) {  
Stack<String> list = new Stack<>();  
list.push("A");  
list.push("B");  
list.push("C");  
list.push("D");  
System.*out*.println(list);  
list.push("A");  
list.push("Z");  
System.*out*.println(list);  
 System.*out*.println(list.search("A"));  
}





**ArrayList vs LinkedList**

addFirst

