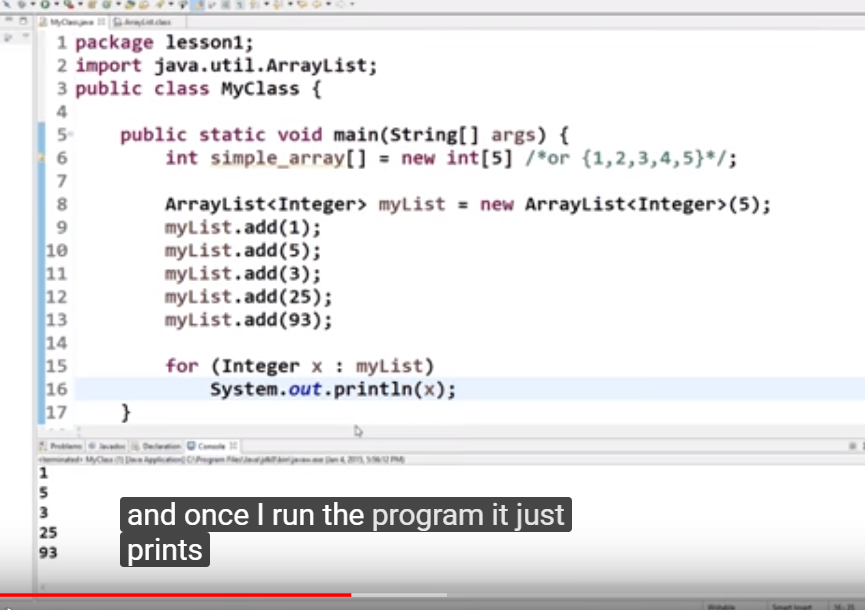
Array List ADT



他可以指定类型<Integer>

虽然事先设定了长度是5，但是实际上可以拓展或减少



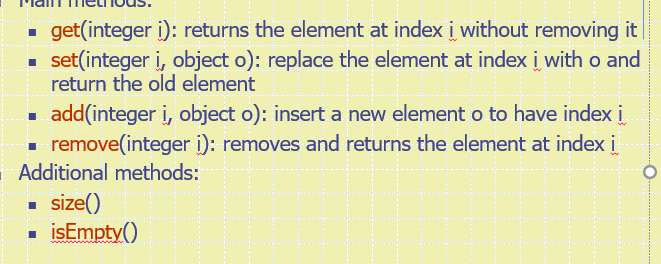
size永远显示这个arraylist实际element，如果我们初始5，实际3，.size给的是3

Array List 这种ADT通过储存任意类型的Object拓展了array的概念

一个element可以进入，插入，或移除通过指定他的index

如果指定的是一个错误index，会throw 一个 exception，比如负数index

自带的method



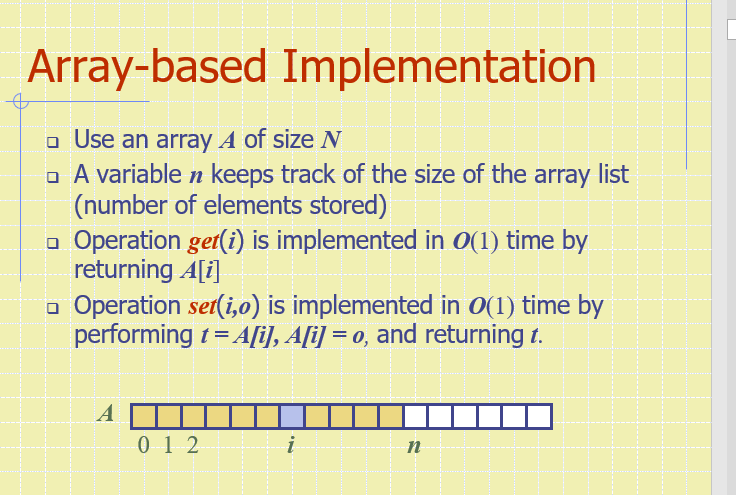
基于array的 implementation

用一个size是N的array A

一个变量n 跟踪这个arraylist的size //存储的element的数量

get(i)这个操作 需要O（1）时间通过return A[i]

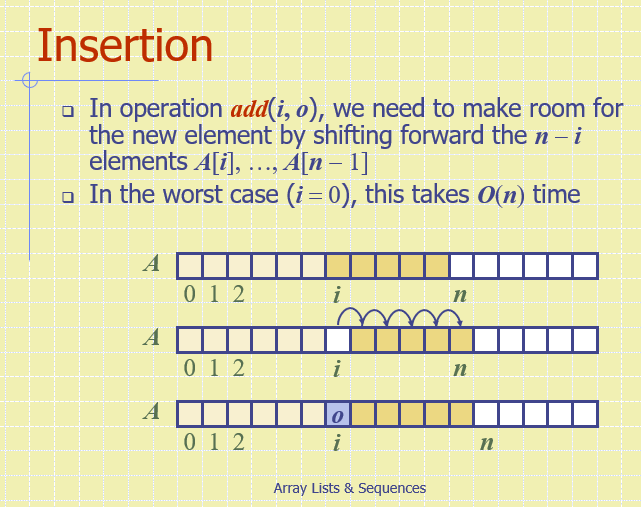
operate set(i,o)这个操作需要O(1)时间通过 t=A[i],A[i]-o,然后return t

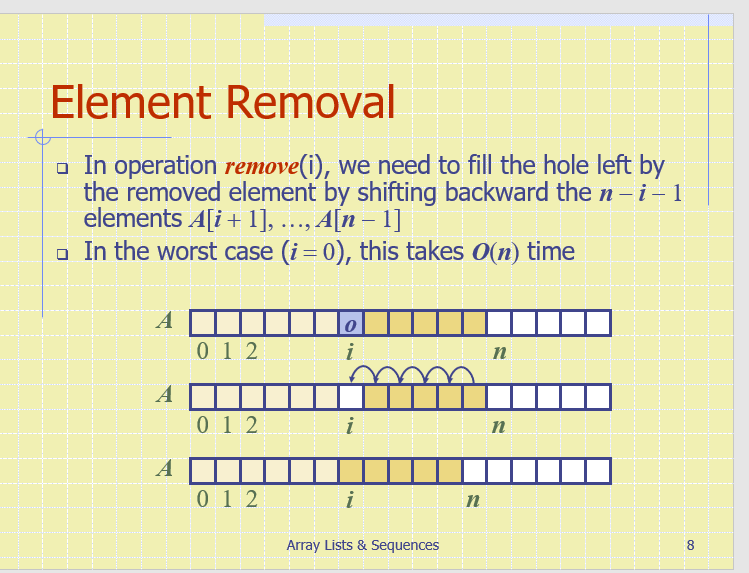


Insertion

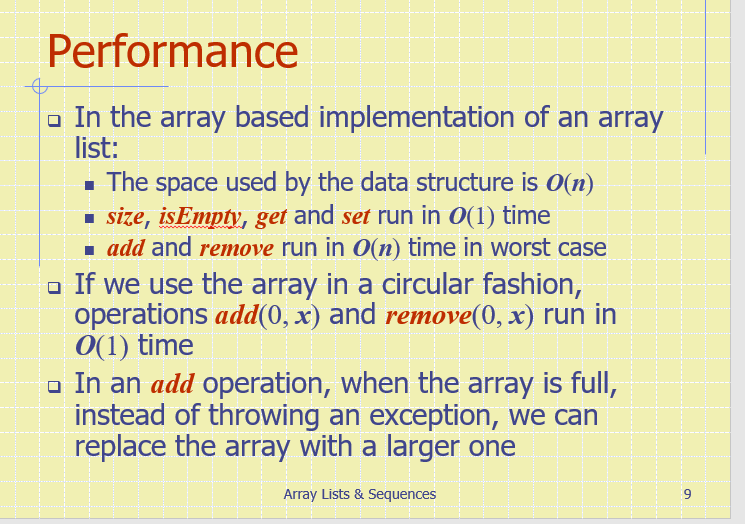
add(I,o)这个操作我们需要插出一个新空间，需要移动n-i个element，即现有的A[i]to A[n-i]

最坏的情况下，需要O（n）time,假如你要查在第一个，要把n个元素后移





要移动n-i-1个元素，因为你是I,后面的元素就是n-i-1



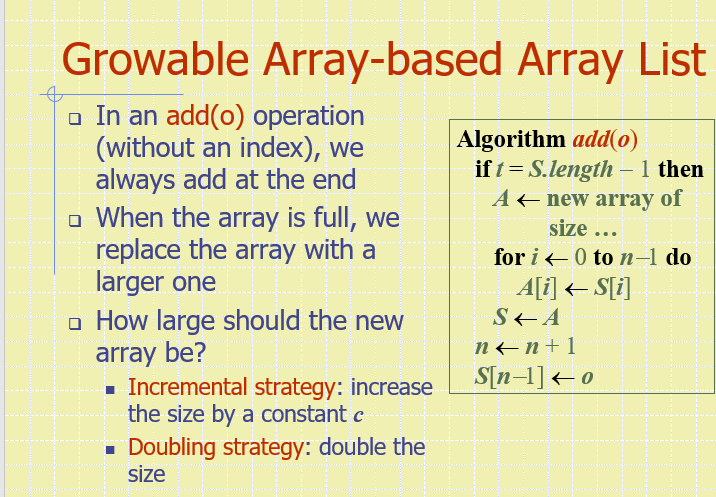
基于array的arraylist需要的space是O（n）

绝大多数操作室O（1）

add和remove这两个最差的是O（n）因为要移动其他元素

如果我们使用循环的方式，addd(0,x),remove(0,x)只需要O（1）次

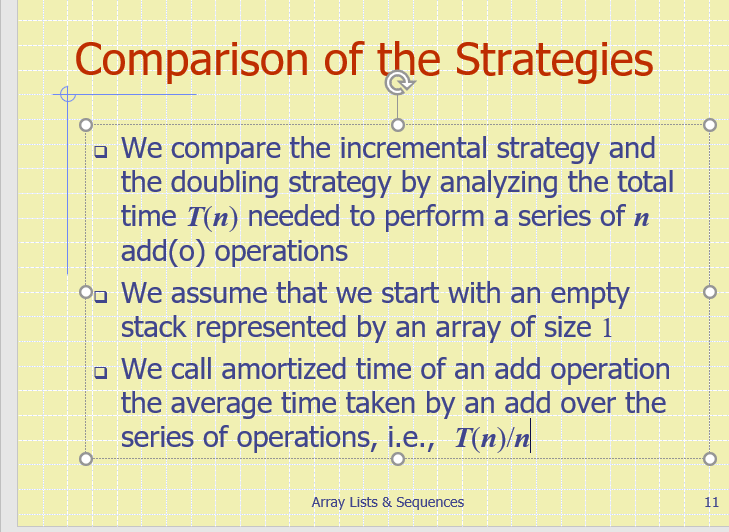
add这个操作，如果array满了，我们可以用更大的array替换



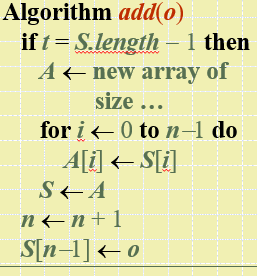
比较incremental strategy两种策略，

现在有个空stack，array长度为1，我们用T(n)的时间进行n次add(o)操作

我们将每一次add操作的时间平均一下

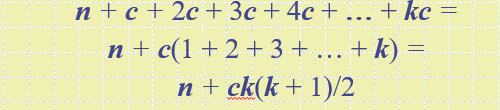


incremental这个策略我们一共需要replace这个array 次，因为一次加的都是c

这个add操作每次需要的时间就是当前array的长度

总共加n个数需要

n是add本身，第一个c是第一次加满，第二个c是第二次加满，总共k次加满

c是常数，所以T（n）是O(n^2) (n/c)^2

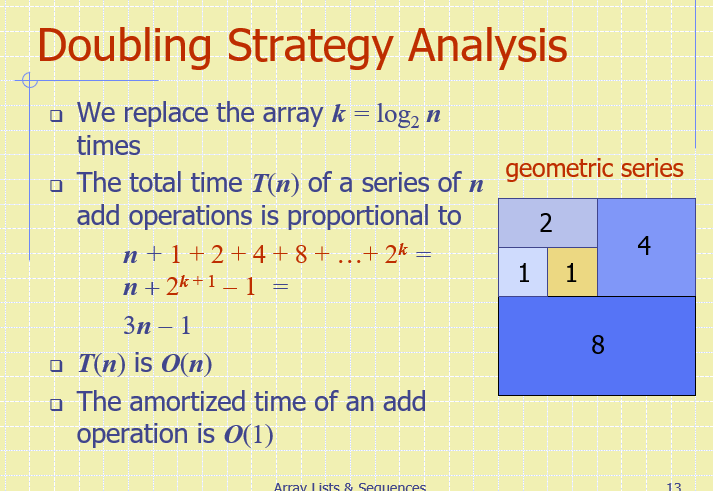
平均time是N^2/N=O(n)

Doubling策略

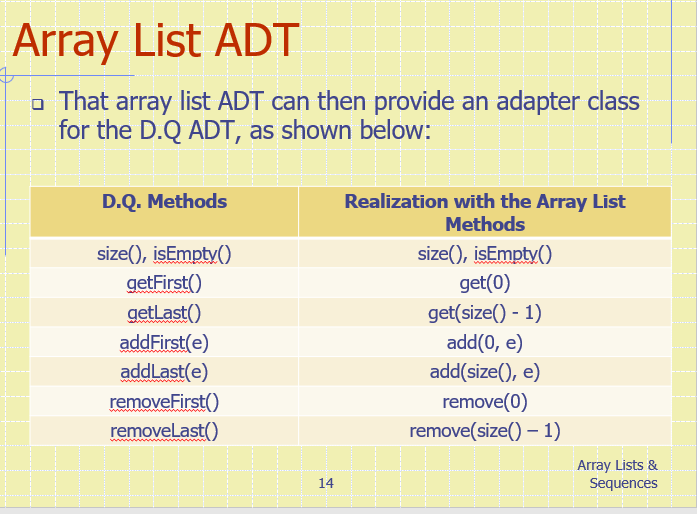
总共需要replace 次，因为2^n=k

总共时间是3n-1

每个平均操作是O（1）



array list ADT也可以通过double-sided queue ADT替代



Node List ADT

对于linked list来说使用node也许更有效，虽然有时会导致隐私泄露

node list ADT 抽象了具体的linked list data structure，让他们的元素可以通过position 进入

换句话来说，list是一组nodes是一组的集合比如Linked node，而node list的node们被加上了position标记用来记录他们的位置，这些Position线性排列，

也就是说，与其return这些node的pointer，which导致泄露， node list ADT 利用另一种ADT，叫做Position ADT, 用来定位list里的element

Position ADT,

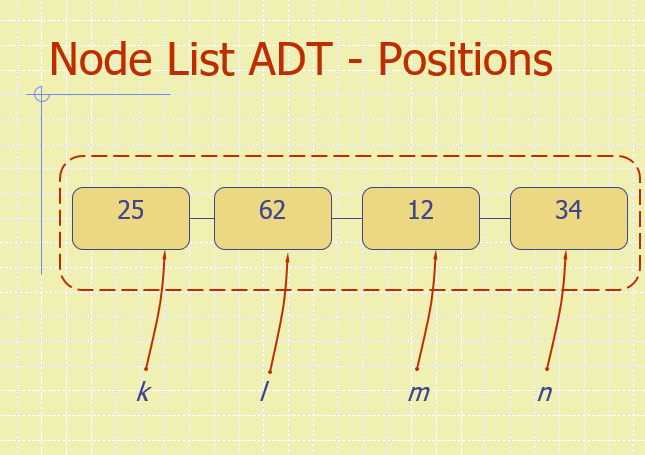
支持的method是可以return这个position的element

一个position 通常是被相对relatively描述的，与他的另据有关

比如说，在一个llist中，每一个Position b都夹杂在position a与positionc中，除了b是第一个Position or最后一个position

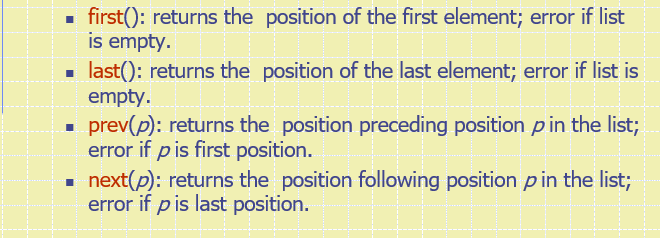
LIst中的Position不会改变即使这个position上的element被改变或者交换了，这个

position只会被destroy当他所指的element被明确的移走时//就是改变element没事，删除有事

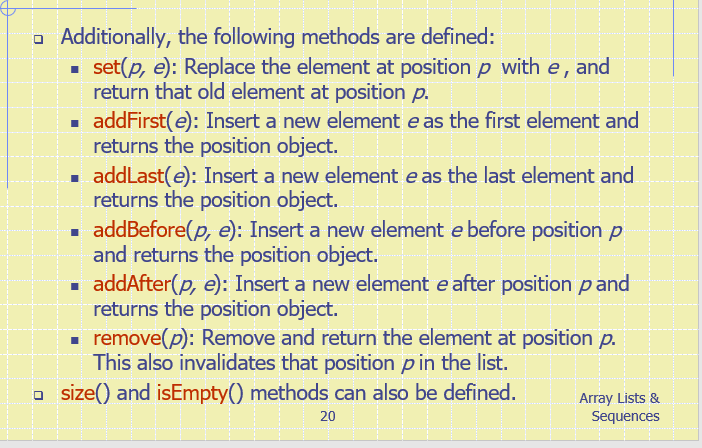


K l m n 是独立的，25变成29，k还是k

通过使用position，node list ADT可以使用以下method

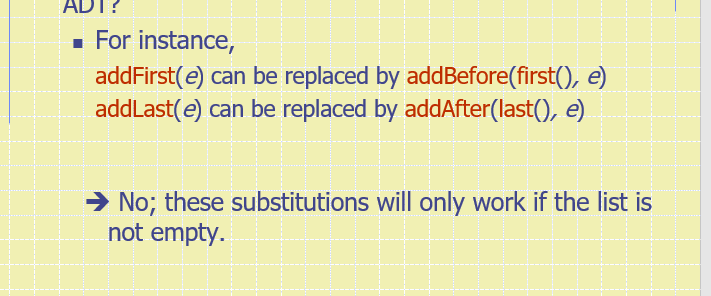


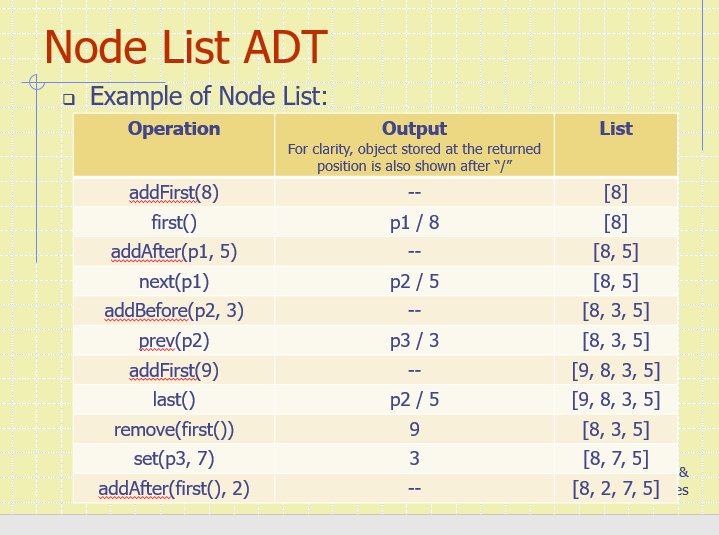
注意着几个方法都是return position



这些方法才是改变element，

这些method没有重复的

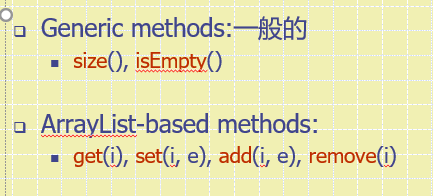
无法替代，因为后两者只有在list不是empty的时候可以使用

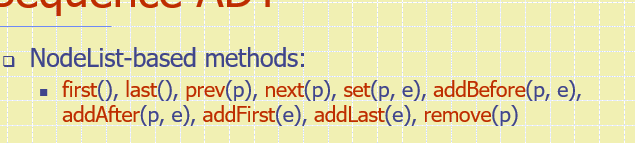


Sequence ADT

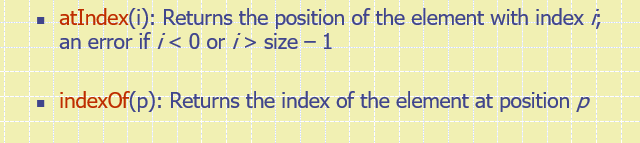
sequence是一种ADT，支持所有的DQ method,arraylist method, nodelist method

element通过index或者position aceessed

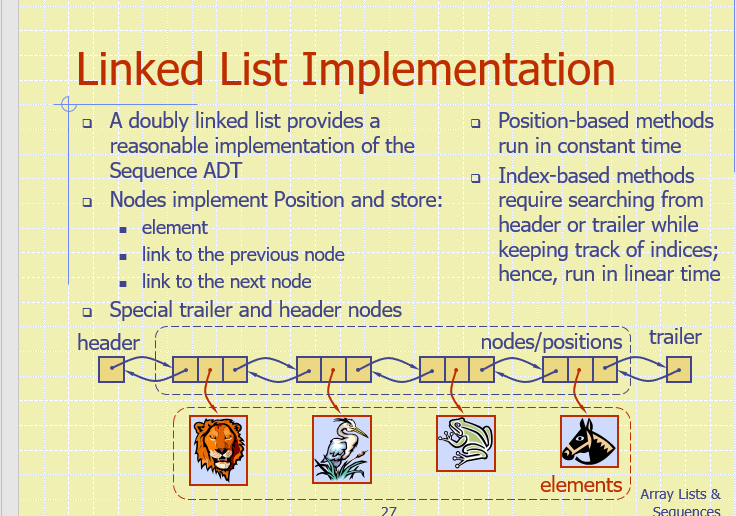




这个ADT由两个bridging method，提供了位置与 index的联系



一个Sequence ADT可以用 双链表DQ来构建



node相当于position，并且这些node存储元素，之前Node的Link，下一个node的Link

