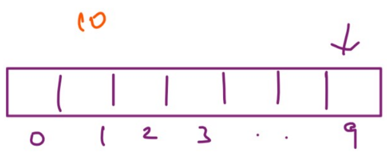
ArrayList

Different types of creating ArrayList() —(constructor)



→ ① new ArrayList( )

② new ArrayList(int capacity)

③ new ArrayList (Collection c)

(1) Creates an internal Array of capacity 10 (by default)

(2) creates an internal Array of capacity as the we passed

(3) creates an ArrayList by replicating any other collection(i.e ArrayList)

Properties:

(1) The underlying data structure is Array

(2) Heterogeneous type of data can be stored

(3) The default capacity of arraylist is 10

(i) if the internal Array used by the ArrayList is full while adding an element, it creates a new array of capacity as follows

$$\boxed{new\_capacity = current\_capacity * \frac{3}{2} + 1}$$

$\textcircled{11}^{th}$ element $\longrightarrow \textcircled{16} \longrightarrow 25$

$\cancel{10}^{5} \times \frac{3}{\cancel{2}} + 1$

$5 \times 3 + 1$

$15 + 1 = \underline{16}$

$16 \times \frac{3}{2} + 1$

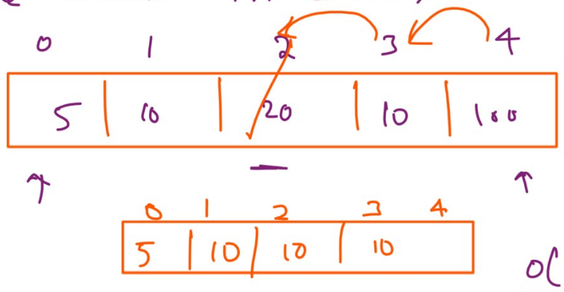
$= 24 + 1 = 25$

④ Mostly used for retrieving an element

Disadvantages

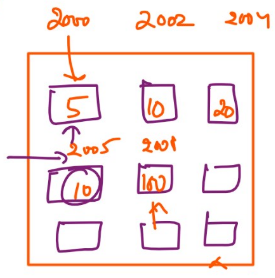
→ Needs contigous memory

→ Adding / Deleting the element in between the ArrayList is time taking



arr. get(3)
↓
10

$\underline{O(1)}$



Sol : LinkedList can come over this problem

Advantages

i) Searching/fetching of an element is fast, any element from arraylist we can get it at one shot O(1)

Linked List

Way to Create :

new LinkedList ( )

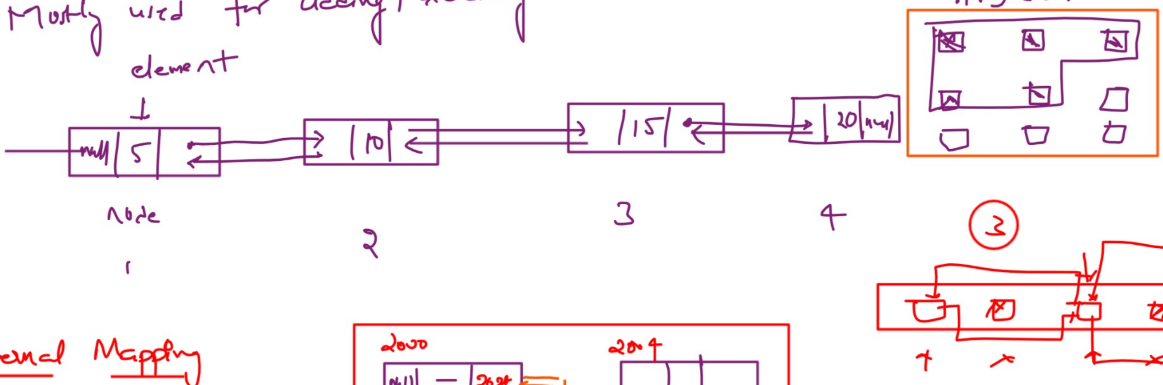
new LinkedList (Collection c)

Properties :

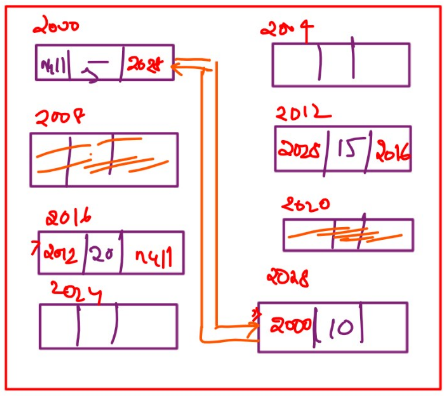
→ The underlying datastructure is doubly linked list

→ Heterogeneous data is allowed

→ Mostly used for adding / deleting an element into a list O(1)



Internal Mapping



Special Methods (only in Linked List)

addFirst(Object o)

addLast (Object o)

removeFirst( )

removeLast( )

getFirst ( )

getLast ( )

↓ time complexity O(1) O(n)

Advantages

(1) Operations are easy for adding / removing the elements into List (time complexity : O(1))

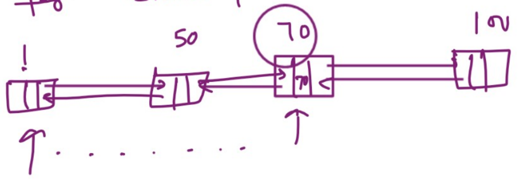
⑤ ↑ size

(2) does not need contiguous memory to store the element

70 → get

Disadvantage:

→ not efficient for retrieval (T.C : O(n))



Vector

Creating

new Vector( )

new Vector(int initialCapacity)

new Vector(Collection c)

} × 2

→ new Vector(int initialCapacity , int incrementalCapacity)

Note:

→ Once the vector is full & if we try to add a new element it creates a new vector where the capacity (current capacity + incremental capacity) which is given by user

0 1 2 3 4

10 × 16 × 24 × 3 ...

0 ① .add (50)

→ By default new capacity 2

new Capacity = current_capacity * 2

= current_capacity + 50

Collection

0 1 2 3 4 5 6 7

← Current capacity = size of element

⑩ ⑤ 25

Q: → 1000 ← 1001 = 1000 * 2 = 2000   1050   950

→ 50 +

→ 500 +

→ 1000

→ 1001

ArrayList

(1) underlined data structure is array

(2) mostly used for retrival operation

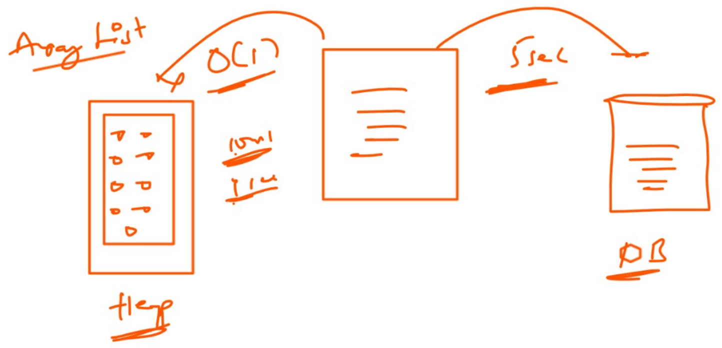
(3) Need contigous memory

Linked List

(1) underlined data structure is doubly linked list

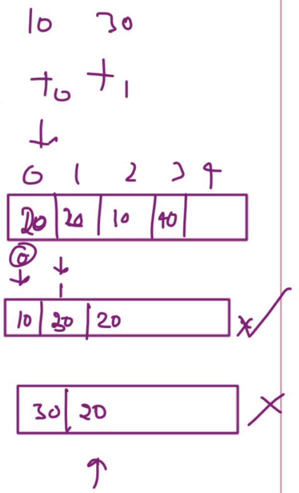
(2) mostly used for adding | removing objects

(3) doesn't need contigous memory



arr
.add (10)

synchronized add () {
}

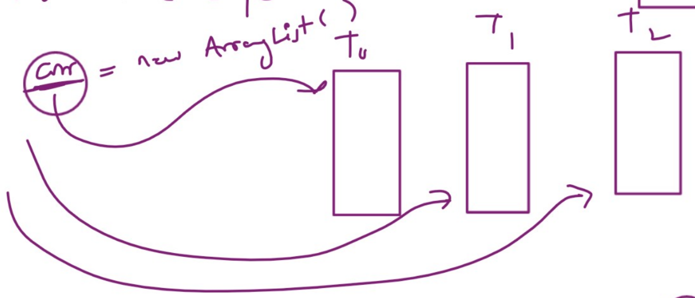


ArrayList

① methods are not synchronized

② Not thread-safe

arr = new ArrayList()



③ It is introduced from 1.2

④ Race-condition will exist

Vector

(1) methods are synchronized

$\boxed{\text{(2) thread safe}}$

for (; ;)
add(10)
add (10)

③ It is considered as (1.0)

④ No race-condition