AGENDA

1·> collections

- list / Queue / set
- implementations

*) **COLLECTION:**

set of objects available to make life easier

eg: implementing DSA

✓ linked list	✓ Queue
✓ Map	✓ Heap
tree ✗	✓ set
Graph ✗	

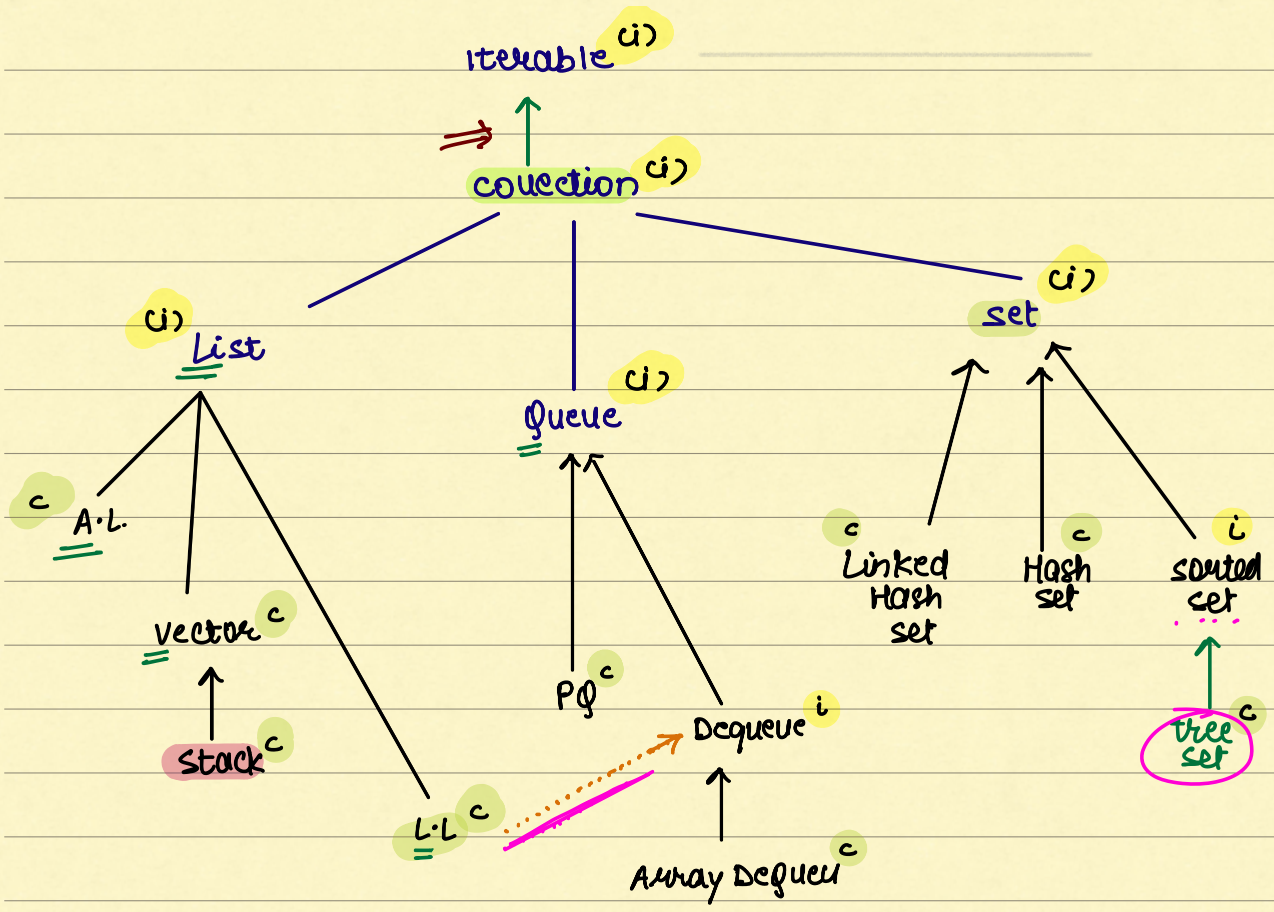
are available in collections framework.

*) CLASS HIERARCHY:

$$\begin{tikzcd}
\text{Iterable}^{(i)} \\
\text{Collection}^{(i)} \arrow[u]
\end{tikzcd}$$

- Interface can extend another interface (Many)
- 1 class can ONLY extend 1 class

i → interface
c → class



Dequeue can be implemented using L.L. as well.

*) LIST :

Multiple implementations
- vector
- LL
- Arraylist

① List<int> x= new Arraylist();
② list<Arraylist> y= new Arraylist ();

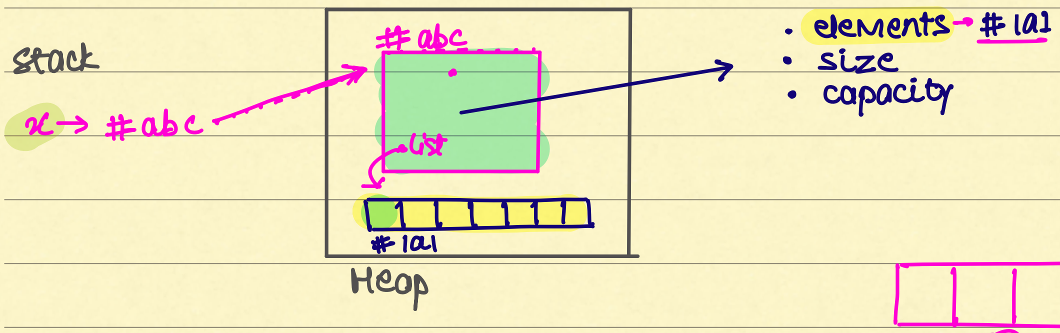
Always preferred.

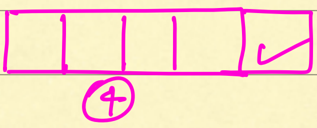
x= new vector(); ✓
y= new vector(); ✗

(i) Arraylist:

syntax:

List<int> x = new Arraylist<>();





① SIZE : No. of elements in list as of Now

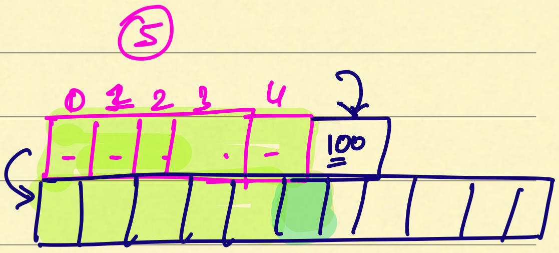
② **CAPACITY:** How many elements it can store...

default = 10

[you can Pass any value]

if size > capacity ??

51st → list.add (100);



a.) creates New list with 2.x. of capacity of old list

b.) elements of old list ⟶ copied to New

c.) old list is G.C

add ( ) ?

O(N)

T.C. of adding element when size = capacity O(N)**

*) **FUNCTIONS:**

1.) set

syntax x.set(idx, value)

eg: x.set(5,100)

0	1	2	3	4	5
1	2	3	4	5	6

Range (idx) ⟶ (0 - size - 1)

2.) get(index)

returns value at index

3.) add (index, value)

index range → 0-size

4.) add(value)

adds at the end of list

(ii) VECTOR:

(dynamic Array)

similar to Arraylist, with (1) difference
∴ This is synchronized

share data b/w threads → use vector
NOT arraylist

vector is thread safe, Arraylist is Not

Disadvantage:
→ slower than Arraylist

(iii) **LINKED LIST**

syntax:

```
list<int> x= new linkedlist();
```

·) similar implementation of L.L.

·) contains head/ size/ tail

$\hookrightarrow$ Generally $\rightarrow$ doubly LL

$\downarrow$

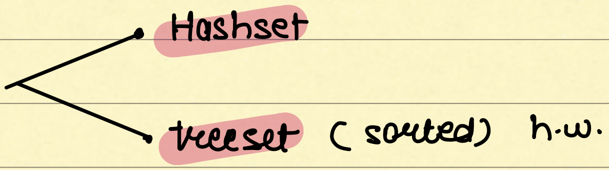
advantage $\rightarrow$ Insert at End as well.

(iv) **STACK:**

similarly, has all basic stack func

$$\begin{tikzcd}
 & \text{vector} \\
\text{stack} \arrow[ur, dash] \arrow[r, dash] \arrow[dr, dash] & \text{LL} \\
 & \text{Arraylist}
\end{tikzcd}$$

*)· SET INTERFACE



(i) Hashset

·) No duplicate allowed.

Syntax:

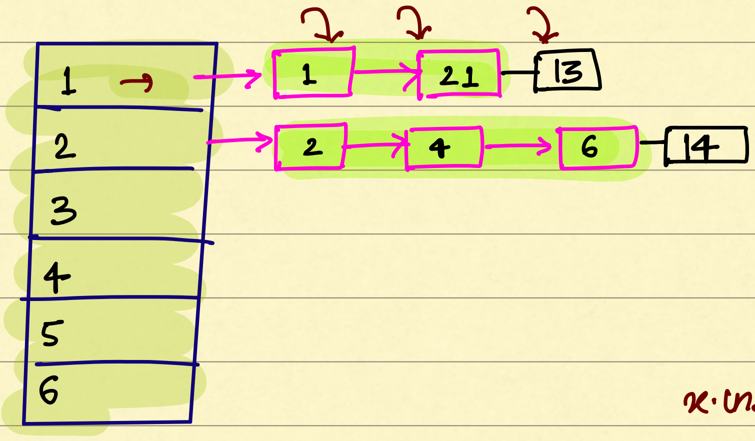
```
Set<int> x= new Hashset();
```

Methods:

1. add
2. remove
3. size
4. contains

Avg. T·C → O(1)

*) HOW HASHSET IS IMPLEMENTED:



x·insert (13)

$$13 \% \overset{6}{(N)} = 1$$

$$14 \% 6 = 2$$

x·ins(13) ⟶

$13 \% 6 = \boxed{1}$

- ·) Every Index is a New L·L
- ·) Each Idx is called bucket

x·insert (10);

1·) Hashcode of 10 is calculated

$10 \% N$

$(N) \longrightarrow$ No. of buckets

2·) value (10) goes to bucket which was calculated above

Handling duplicates:

- ·> Find Hashcode of No
- ·> check in LL, if exists OR NO

Note: HashMap and Hashset Almost same

$\downarrow$ Has (k,v) $\quad\quad \downarrow$ only (k)

A) QUEUE INTERFACE

Queue<int> q = new Queue();
pq = new PQ();

Methods:

add
remove
Poll
offer

Add v/s offer:

Add → throws exception when (q) is full
Offer → returns false; don't throw Exception

-> remove() → Always Removes head of queue
-> remove(i) → Possible, this method come from collection framework

Poll → returns false; don't throw exception; when Empty.
∴ remove() → can throw exception

*) ITERABLE v/s ITERATOR:

both of them are interfaces.

Iterable: has only (1) method - iterator();

interface with single func

return type of iterator(); ⟶ Iterator

Iterator: interface that has ② methods

Object next() //next element

boolean hasNext(); // whether element present OR NOT

```
for (i=0; i<n; i++) {
    a.get(i);
}
```

*) WHY REQUIRED:

```
LinkedList l= new LinkedList (10,20,30,40);
```

forEach ⟹ 1st
```
for (int i: list) {
    ......① i++; // element
}
```
// ONLY syntactical sugar

if youre able to use this loop → Means your class is Implementing Iterable

Purpose: support to clients who want to write Enhanced for loop.

Note: Every collection class implements Iterable

Internally:

- Hard work of iterating is done by iterator
- iterable is just used for classification.

- LinkedList internally has 1 more internal class LinkedList Iterator implements Iterator

  This internal class takes care of iterator.

  - Has Methods for iterator
    hasNext()
    next()

2nd USING Iterator:

```
list.iterator();
while ( it.hasNext() ) {
    ....
}
```

[1st code Above is converted internally to 2nd code]

✓ internally iterator → iterable & we just write iterable

DEMO