# Collection Framework.

$$\begin{tikzcd}
 & \text{Iterable} & \\
 & \text{Collection} \arrow[u] & \\
\text{List} \arrow[ur, dash] & \text{Queue} \arrow[u] & \text{Set} \arrow[ul, dash] \arrow[r] & \text{Interfaces.}
\end{tikzcd}$$

Map is interface but it ~~[illegible]~~ is not extending iterable.

Methods : add, size, remove, iterate, addAll, removeAll, clear

→ List Interface.

$$\begin{tikzcd}
 & \text{List} & \\
\text{ArrayList} \arrow[ur, dash] & \text{LinkedList} \arrow[u, dash] & \text{Vector} \arrow[ul, dash] \\
 & & \text{Stack.} \arrow[u]
\end{tikzcd}$$

→ Queue Interface.

$$\begin{tikzcd}
 & \text{Queue.} & \\
\text{PriorityQueue} \arrow[ur, dash] & \text{LinkedList} \arrow[u, dash] & \text{Deque} \arrow[ul, dash] \\
 & & \text{ArrayQueue} \arrow[u]
\end{tikzcd}$$

→ Set Interface.

$$\begin{tikzcd}
 & \text{Set} & \\
\text{HashSet} \arrow[ur, dash] & \text{LinkedHashSet} \arrow[u, dash] & \text{Sorted set} \arrow[ul, dash] \\
 & & \text{TreeSet.} \arrow[u]
\end{tikzcd}$$

→ Map Interface.

$$\begin{tikzcd}
 & & \text{Map} & \\
\text{HashMap.} \arrow[urr, dash] & \text{LinkedHashMap.} \arrow[ur, dash] & \text{SortedMap} \arrow[u, dash] & \text{Hash Table.} \arrow[ul, dash] \\
 & & \text{TreeMap} \arrow[u] &
\end{tikzcd}$$

Collection Framework. -Riddhi Dutta.

→ Lists:-
- Lists are collections that maintain their ele in order & can contain duplicates
- elements in a list are ordered
- Each ele are position based, starting from index 0.

List Methods:-
any generic type
- E get (int index)
- E set (int idx, E ele)
- void add(int idx, E ele)
- boolean addAll
- E remove (int idx)

→ ArrayList, LinkedList, Vector. → List is implemented by

1) ArrayList: → add, set, indexOf (ele), subList(1, 4) ↳ idxes
↳ Dynamic Array
↳ if u change this arList then Original arList will also change

2) Vector class:- → It is legacy class.
↳ same as arrayList but vector is thread safe.

3) LinkedList: → It also implements Deque & List.

→ ArrayList vs LinkedList (VVimp) ☆.

| ArrayList | LinkedList |
|---|---|
| position based ↳ O(1) | u need to traverse (O(N)) |
| insertion & delitions ↳ O(N) | → O(1) |

→ Boxing & Unboxing (wrapper class).

ListIterator:
↳ hasNext(), hasPrevious(), next(), previous().
(next: first it will print & then curser will go to next index.)
(previous: first curser will go to prev index & then prints.)

It has 2 methods:
ListIterator<E> listIterator()
" " listIterator (int idx)

→ You can traverse in either of direction.

→ To convert arraylist to array
Integer ar[] = ar.toArray (new Integer[0]) ↳ size of array (any size u can pass, generally we will pass as 0).

- Arrays.asList(), Collections.binarySearch()

Queue Interface:-

→ add(), offer(), poll(), remove(), peek(), element().

add(): gives exception if queue is full
offer(): no exception
poll(): return null if that ele is not present.
remove(): exception

Implementations:- ~~Dele~~ Deque, Priority Queue, LinkedList.

→ LinkedList implements deque interface & deque interface extends Queue interface

→ For Deque use ArrayDeque.

```
Deque<Integer> dq=new ArrayDeque<>();
```

→ For Queue use LinkedList.

```
Queue<Integer> q=new LinkedList<>();
```

Priority Queue:-

→ they are not sorted.

→ don't iterate over priority queue. (its not good choice)

→ Comparable (Natural Ordering) & comparator → Total Ordering.

Comparable ↓ it should be implemented by class itself

comparator ↳ it should be implemented by comparator class

→ Lambda fns.

```
PriorityQueue<Integer> pq =new PriorityQueue<>(
                                   (a, b) → b-a );
```

Set Interface:-

↳ Collection of unique elements.

→ retainAll - gives intersection of 2 sets.

→ removeAll - removes intersection ele.

→ addAll - union.

→ no orderings. So for ordering we use LinkedHashSet.

→ Sorted Set

→ Navigable Set Interface extends sorted set interface.

↓

pollFirst(), pollLast(), ceiling (E e), floor (E e), higher(E e), lower (E e).

→ TreeSet.

~~TreeSet~~

Map.

put , get, remove, containsKey , containsValue , size, isEmpty.

→ Values in map can be duplicate, but key should be unique.

→ HashMap is not thread safe , Hash Table are thread safe

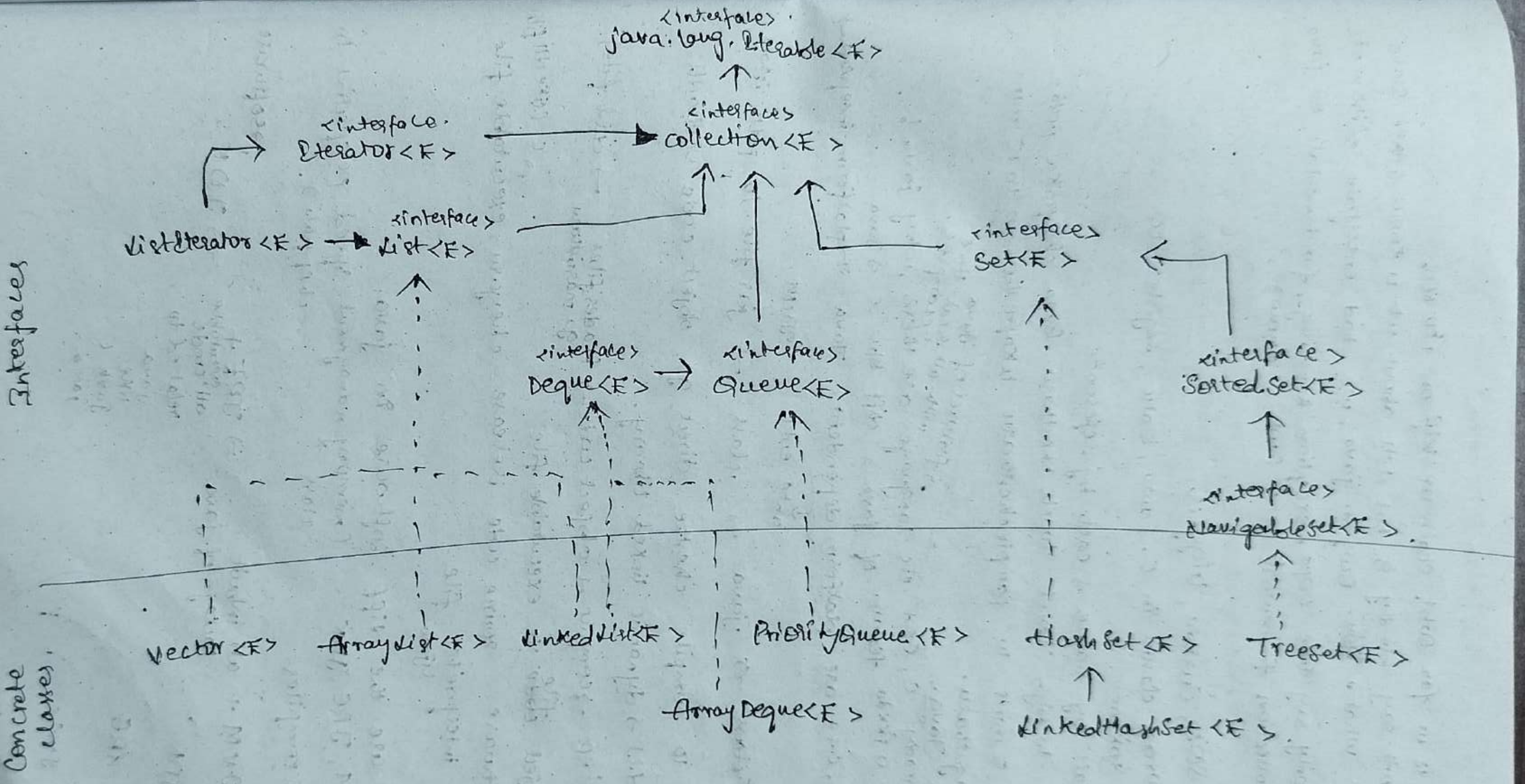
→ computeIfAbsent (

→ As we cannot traverse or iterate on map, we need to convert into set.

```
Set <Map.Entry <String, Integer>> entrySet = map.entrySet();
for (Map.Entry<String, Integer> entry : entrySet) {
    s.o.pln (entry.getKey () + "," + entry.getValue());
}
```

```
→ set<String> keySets = map.keySet();
for (String key : keySets) {
    S.o.pln (key + "," + map.get(key));
}
```

$$\begin{tikzcd}
\begin{array}{c}\langle\text{interface}\rangle\\ \text{Map.Entry}\langle K,V\rangle\end{array} \arrow[r] & \begin{array}{c}\langle\text{interface}\rangle\\ \text{Map}\langle K,V\rangle\end{array} & & \\
 & & \begin{array}{c}\langle\text{interface}\rangle\\ \text{SortedMap}\langle K,V\rangle\end{array} \arrow[ul] & \\
 & & \begin{array}{c}\langle\text{interface}\rangle\\ \text{NavigableMap}\langle K,V\rangle\end{array} \arrow[u] & \\
 & \text{HashMap}\langle K,V\rangle \arrow[uuu, dashed] & \text{TreeMap}\langle K,V\rangle \arrow[u, dashed] & \text{HashTable}\langle K,V\rangle \arrow[uuul, dashed] \\
 & \text{LinkedHashMap}\langle K,V\rangle \arrow[u] & &
\end{tikzcd}$$


Interfaces
Concrete classes.
<interface>
java.lang.Iterable<E>
<interfaces>
Collection<E>
<interface.
Iterator<E>
ListIterator<E>
<interface>
List<E>
<interface>
Set<E>
<interface>
Deque<E>
<interfaces>
Queue<E>
<interface>
SortedSet<E>
<interface>
NavigableSet<E>
Vector<E>
ArrayList<E>
LinkedList<E>
PriorityQueue<E>
ArrayDeque<E>
HashSet<E>
TreeSet<E>
LinkedHashSet<E>