(for iterating elements in forward only dir) **ITERATOR** i) boolean hasNext() ii) Object next() iii) void remove()

(root interface of all Collection class) **ITERABLE** i) Iterator<T> iterator() ii) void forEach()

(Collection interface is the foundation of Collection frameworks. It declares methods that every collection class will have)

COLLECTION i) boolean add(E e) ii) boolean addAll(E e) iii) void clear()
iv) boolean contains(Object o) v) boolean containsAll(Collection<?> c)
vi) boolean equals(Object o) vii) boolean isEmpty() viii) boolean removeAll(c)
ix) boolean removeIf(Predicate) x) boolean retainAll(c) xi) int size()
xi) Object[] toArray() xii) Stream<E> stream()

(ordered, ordered, duplicates allowed)

LIST i) add(int index, E) ii) E get(int index)
iii) int indexOf(o) iv) int lastIndexOf(o)
v) remove(int index) remove(o) get(index)
vi) set(index, E) vii) void sort()
viii) List<E> subList(fromIndex, toIndex)
ix) Object[] toArray()

ARRAYLIST (maintains insertion order, random access)
N i) addAll(index, c) ii) clone() (shallow copy)
iii) removeRange(from, to) iv) trimToSize()

LINKEDLIST (doubly LL, duplicates, faster manipulation) DEQUE
N i) addFirst(E e) ii) addLast(E e) iii) element()
iv) descendingIterator() v) get(index) getFirst, getLast.
vi) peek(), peekFirst(), peekLast() vii) poll(), pollF, pollL
viii) pop() ix) push()

VECTOR (implements serializable, cloneable)
S N i) int capacity() ii) E elementAt(index)
iii) insertElementAt(E obj, index)

STACK (LIFO, 5 methods make vector act like Stack)
S N i) boolean empty() ii) E peek()
iii) E pop() iv) E push()
v) int search(o) returns 1-based position of object.

(FIFO, ordered, duplicates allowed)

QUEUE i) add(E e) ii) peek() iii) poll()
iv) element() retrieves, not removes head
v) remove()

PRIORITY QUEUE (Uses Comparator, based on priority heap. The head of the queue is the least element)
i) Comparator() returns comparator implementing the queue or null if according to natural order.

DEQUE (Can be used as a stack/Queue)
Insertion and removal possible at both ends.

ARRAYDEQUE (Faster than arraylist, linkedlist and Stack) concurrent modifications are prohibited by exception.

SET (Unordered set, one null value allowed, unique elements only.)

CLASS HASHSET (Hashing is used to store elements in Hash table, HashMap, one null value.)

LINKEDHASHSET (predictable iteration order, order of insertion is maintained)

SORTEDSET (ascending order, comparable/comparator needs to be specified)
i) first() returns lowest element in the set.
ii) headSet(toElement) element < toElement
iii) tailSet(fromElement) element >= fromEle.
iv) subSet(from, to) from <= element < to.

TREESET: (Based on TreeMap O(log n) time for add, remove, contains, Null not allowed)

MAP (Key-value pair, no duplicate key, replaced Dictionary)
i) clear() ii) compute(key, avaqefunction)
iii) computeIfAbsent / computeIfPresent iv) containsKey(key)
v) containsValue(value) vi) entrySet() (returns set)
vii) equals() viii) forEach() ix) get(key) getOrDefault(k)
x) keySet() xi) merge() xii) put putAll putIfAbsent
xiii) remove(key), remove(key, value) xiii) replace(key, new) replace(key, oldvalue, newvalue) replaceAll(function)
ix) size() x) values() returns a collection view of values.

SORTEDMAP i) headMap() ii) tailMap() iii) subMap.

TREEMAP: (a red-black tree based.

HASHMAP: (Constant time performance (get, put)
NV NK Iteration time proportional to capacity (no of buckets) performance depends on initial capacity & load factor

LINKEDHASHMAP (predictable order of iteration)

HASH TABLE (CLASS)