⭐ What is Collection Framework?

The **Collection Framework** in Java is a standardized architecture for storing and manipulating groups of objects.
It provides:

- **Interfaces** (Collection, List, Set, Queue, Deque, Map)
- **Classes** (ArrayList, LinkedList, Vector, Stack, HashSet, LinkedHashSet, TreeSet, PriorityQueue, etc.)
- Utility Classes (Collections, Arrays)
- **Algorithms** (Sorting, Searching, Shuffling, Reversing)

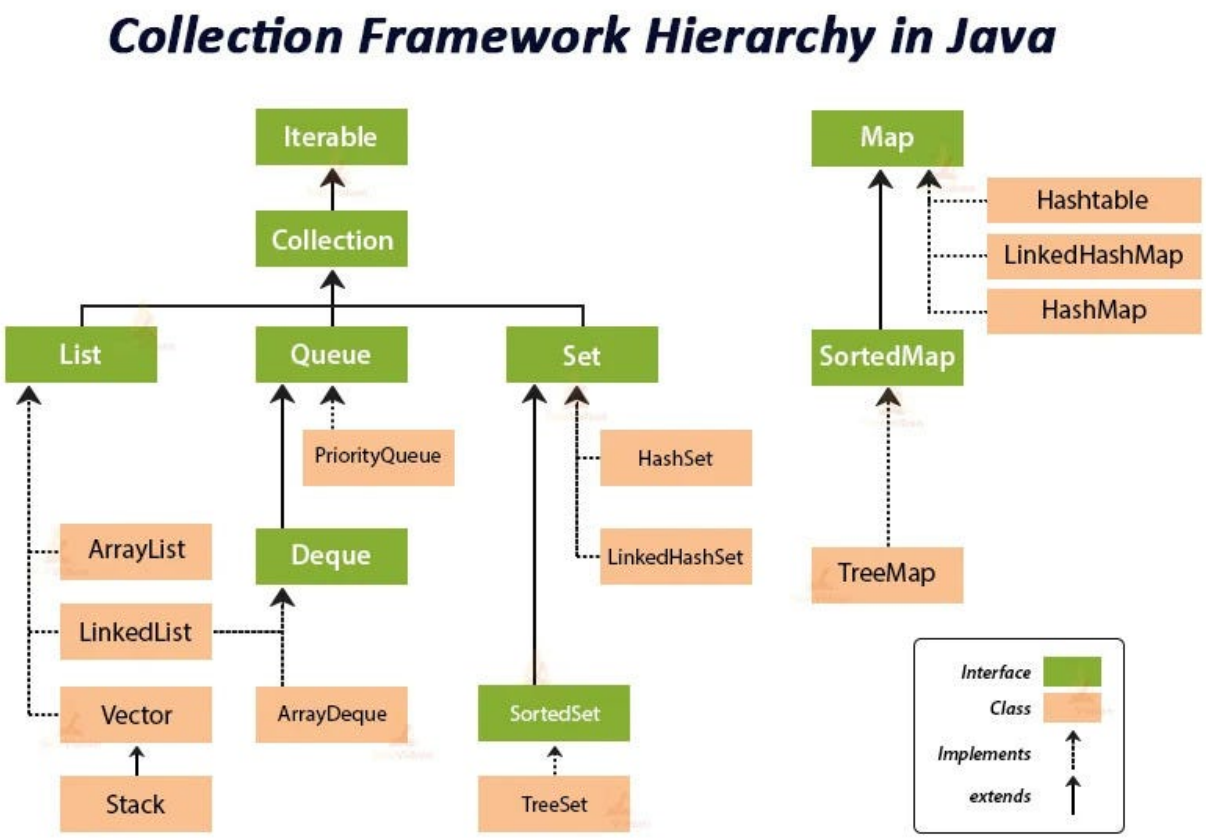
The framework supports operations like insertion, deletion, searching, sorting, updating, iteration, and manipulation of data.

⭐ Difference between Array and ArrayList

Array	ArrayList / Collection
Static size (fixed-length).	Dynamic size (grows/shrinks automatically).
Can store **primitive types** and objects.	Stores **objects only** (wrapper classes for primitives).
Fast and memory efficient for fixed-size data.	More flexible but slightly higher overhead.
Uses direct indexing with a fixed structure.	Built on top of a **dynamic internal array** (`Object[]`).
No built-in methods for manipulation.	Provides many built-in methods (add, remove, contains, etc.).
Supports multi-dimensional arrays.	One-dimensional; can nest lists to simulate multi-dimension.

⭐ Difference between Collection and Collections

Collection	Collections
It is a **interface** in the Collection Framework.	It is a **utility class** in `java.util`.
Represents a group of objects as a single unit.	Provides **static methods** like `sort()`, `reverse()`, `min()`, `max()`, `synchronizedList()`, etc.
Parent of List, Set, Queue.	Works *on* Collection objects.



⭐ List Interface

Definition:

List is a child interface of Collection.

It represents an **ordered collection** that preserves insertion order and allows duplicate elements.

✔ Key Features of List

1. **Maintains Order:** Insertion order is preserved.
2. **Duplicates Allowed:** Multiple identical elements permitted.
3. Index-Based Access: Supports add(index), get(index), set(index), remove(index).
4. **Null Allowed:** List implementations allow null values.
5. Supports Iterators:
    - Iterator (forward)
    - ListIterator (forward + backward)

✔ Implementations of List

1. ArrayList
    - Backed by dynamic array.
    - Fast random access (O(1)).

    - Slower for insert/delete in the middle (O(n)).
    - Not synchronized.

2. LinkedList
    - Backed by doubly linked list.
    - Fast insert/delete operations (O(1) at ends).
    - Slow random access (O(n)).
    - Also implements Queue/Deque.

3. **Vector** (Legacy)
    - Similar to ArrayList but synchronized.
    - Slower due to synchronization.
    - Rarely used today.

4. **Stack** (Legacy)
    - Extends Vector.
    - LIFO structure (push, pop, peek).
    - Modern replacement: ArrayDeque.

⭐ ArrayList

Definition:
ArrayList is a **dynamic array implementation** of the List interface.
It can grow or shrink automatically as elements are added or removed.

✔ Key Features of ArrayList

1. **Dynamic Array:** Automatically resizes.
2. Maintains Insertion Order: Index-based access.
3. **Allows Duplicates:** Same values can be inserted.
4. **Allows Null Values:** Can store one or more nulls.
5. Fast Random Access: get(index) is O(1).
6. **Not Synchronized:** Not thread-safe by default.
    - Thread-safe options:
        - Collections.synchronizedList(list)
        - CopyOnWriteArrayList
7. **Default Capacity:** 10 (auto-expands by 1.5×).
8. Resizable Underlying Array: Uses Object[] internally.

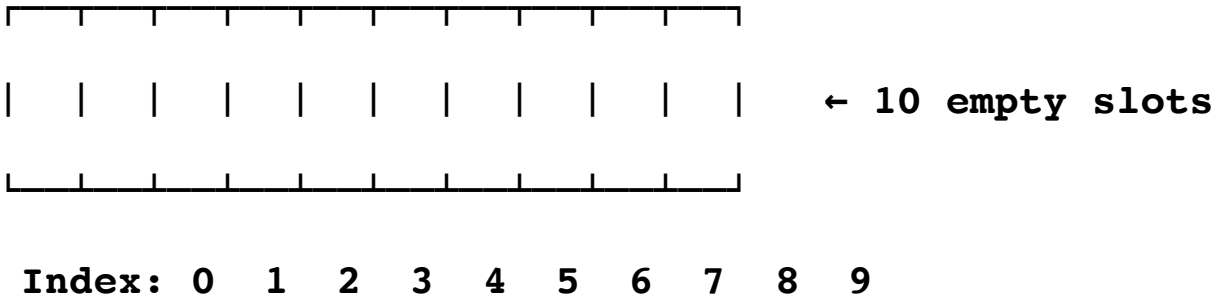
✔ Common Methods in ArrayList

- add(E e)
- add(int index, E e)
- get(int index)
- set(int index, E e)
- remove(int index) / remove(Object o)
- contains(Object o)

- size()
- clear()
- indexOf() / lastIndexOf()
- iterator() / listIterator()
- addAll(Collection c)

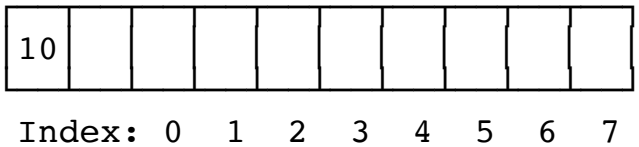
Initially, when we declare `ArrayList<Integer> al = new ArrayList<>();`

al



After first add → `al.add(10);`

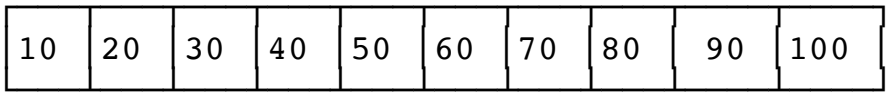
al



After adding all initial elements (10 elements total)

```
al.add(10);
al.add(20);
al.add(30);
al.add(40);
al.add(50);
al.add(60);
al.add(70);
al.add(80);
al.add(90);
al.add(100);
```

al (Initial capacity full)



Now add a new element → `al.add(110);`
➡ Capacity is full → ArrayList **resizes by 50%**
➡ New capacity = **15**

```
al     (Resized by 50%)
┌──┬──┬──┬──┬──┬──┬──┬──┬──┬───┬───┬──┬──┬──┬──┐
│10│20│30│40│50│60│70│80│90│100│110│  │  │  │  │
└──┴──┴──┴──┴──┴──┴──┴──┴──┴───┴───┴──┴──┴──┴──┘
Capacity: 15
Size: 11
```

✔ Default Capacity = 10
✔ Resize happens by 1.5× (50%)
✔ Array grows only when full
✔ After removing elements, capacity does NOT shrink automatically