Java Collection Framework - Complete Overview

1. ArrayList

- Resizable array.
- Maintains insertion order.
- Allows duplicates.
- Fast for index-based access.

Example:

```
ArrayList<String> list = new ArrayList<>();
list.add("Apple");
list.add("Banana");
```

2. LinkedList

- Doubly linked list.
- Maintains insertion order.
- Allows duplicates.
- Fast for insertions/deletions in middle.

Example:

```
LinkedList<String> list = new LinkedList<>();
list.add("Ram");
list.add("Shyam");
```

3. HashSet

- No duplicates allowed.
- No guaranteed order.
- Fast operations using hashing.

Example:

```
HashSet<String> set = new HashSet<>();
```

```
set.add("Dog");
set.add("Cat");
```

4. LinkedHashSet

- No duplicates allowed.
- Maintains insertion order.
- Slower than HashSet.

Example:

```
LinkedHashSet<String> set = new LinkedHashSet<>();
set.add("Delhi");
set.add("Mumbai");
```

5. TreeSet

- No duplicates.
- Sorted order (ascending).
- Slower due to sorting.

Example:

```
TreeSet<Integer> set = new TreeSet<>();
set.add(10);
set.add(20);
```

6. HashMap

- Key-Value pairs.
- Keys are unique.
- Fast lookup by key.
- No order guaranteed.

Example:

```
HashMap<String, String> map = new HashMap<>();
map.put("India", "Delhi");
```

7. LinkedHashMap

- Key-Value pairs.
- Maintains insertion order.
- Keys are unique.

Example:

```
LinkedHashMap<String, Integer> map = new LinkedHashMap<>();
map.put("Alice", 90);
```

8. TreeMap

- Sorted by keys.
- Keys must be comparable.
- No null keys allowed.

Example:

```
TreeMap<Integer, String> map = new TreeMap<>();
map.put(1, "Arya");
```

9. Stack

- Last In First Out (LIFO).
- push(), pop(), peek().

Example:

```
Stack<String> stack = new Stack<>();
stack.push("Java");
stack.pop();
```

10. PriorityQueue (Queue)

- First In First Out (FIFO).

- PriorityQueue sorts automatically.

- poll(), add().

Example:

```
PriorityQueue<Integer> queue = new PriorityQueue<>();
queue.add(30);
queue.poll();
```