Java Collections (Cheat Sheet)

Java Cheat Sheet: ArrayList & LinkedList Methods

📌 Use this as a quick reference while coding!

1️⃣ ArrayList Methods 🚀

(ArrayList is a **dynamic array**, good for fast random access)

Method	Description	Example
`add(E e)`	Adds an element to the end	`list.add("Apple");`
`add(int index, E e)`	Adds an element at a specific index	`list.add(1, "Banana");`
`get(int index)`	Retrieves an element at index	`list.get(2);`
`set(int index, E e)`	Replaces element at index	`list.set(0, "Mango");`
`remove(int index)`	Removes element at index	`list.remove(1);`
`remove(Object o)`	Removes first occurrence of object	`list.remove("Apple");`
`size()`	Returns number of elements	`list.size();`
`contains(Object o)`	Checks if list has an element	`list.contains("Mango");`
`indexOf(Object o)`	Returns index of first occurrence	`list.indexOf("Apple");`
`isEmpty()`	Checks if list is empty	`list.isEmpty();`
`clear()`	Removes all elements	`list.clear();`
`sort(Comparator c)`	Sorts elements	`Collections.sort(list);`
`reverse()`	Reverses the list	`Collections.reverse(list);`

2️⃣ LinkedList Methods 🔗

(LinkedList is **better for frequent insertions/deletions**)

Method	Description	Example
`add(E e)`	Adds element at end	`list.add("John");`
`addFirst(E e)`	Adds element at start	`list.addFirst("Alice");`

`addLast(E e)`	Adds element at end	`list.addLast("Bob");`
`get(int index)`	Retrieves element at index	`list.get(2);`
`getFirst()`	Retrieves first element	`list.getFirst();`
`getLast()`	Retrieves last element	`list.getLast();`
`remove(int index)`	Removes element at index	`list.remove(1);`
`removeFirst()`	Removes first element	`list.removeFirst();`
`removeLast()`	Removes last element	`list.removeLast();`
`size()`	Returns number of elements	`list.size();`
`contains(Object o)`	Checks if element exists	`list.contains("Alice");`
`isEmpty()`	Checks if list is empty	`list.isEmpty();`
`clear()`	Removes all elements	`list.clear();`

🔷 When to Use What?

✅ Use `ArrayList` when:

- You need **fast access** (`get()`, `set()`).
- The number of elements is **mostly fixed**.

✅ Use `LinkedList` when:

- You have **frequent insertions/deletions** (`addFirst()`, `removeFirst()`).
- You need a **queue or stack-like behavior**.

💡 **Tip:** If you're unsure, **start with** `ArrayList` and switch to `LinkedList` only if performance issues arise! 🚀🔥

1️⃣ HashSet (Unique Elements, No Order)

A `HashSet` stores **unique elements** in **no specific order**.

Method	Description	Example	Time Complexity
`add(E e)`	Adds an element	`set.add(10);`	O(1)
`remove(Object o)`	Removes an element	`set.remove(5);`	O(1)
`contains(Object o)`	Checks if an element exists	`set.contains(3);`	O(1)
`size()`	Returns number of elements	`set.size();`	O(1)
`clear()`	Removes all elements	`set.clear();`	O(1)

📌 Comparison of Sets:

- `HashSet` – Fastest, no order.
- `LinkedHashSet` – Maintains insertion order.
- `TreeSet` – Sorted order (O(log n) operations).

2️⃣ HashMap (Key-Value Pair, No Order)

A `HashMap` stores **key-value pairs** for fast retrieval.

Method	Description	Example	Time Complexity
`put(K key, V value)`	Adds a key-value pair	`map.put("A", 100);`	O(1)
`get(Object key)`	Retrieves a value	`map.get("A");`	O(1)
`containsKey(Object key)`	Checks for a key	`map.containsKey("B");`	O(1)
`remove(Object key)`	Removes a key-value pair	`map.remove("A");`	O(1)
`keySet()`	Returns all keys	`map.keySet();`	O(n)
`values()`	Returns all values	`map.values();`	O(n)

📌 Comparison of Maps:

- `HashMap` – Unordered, fastest lookup.
- `LinkedHashMap` – Maintains insertion order.
- `TreeMap` – Stores keys in **sorted order** (O(log n) operations).

3️⃣ Queue (FIFO – First In, First Out)

A `Queue` processes elements **in order** (like a line at a store).

Method	Description	Example	Time Complexity
`add(E e)`	Adds an element (throws exception if full)	`queue.add(5);`	O(1)
`offer(E e)`	Adds an element (returns `false` if full)	`queue.offer(7);`	O(1)
`poll()`	Removes and returns the first element	`queue.poll();`	O(1)

`peek()`	Retrieves first element without removing	`queue.peek();`	O(1)
`size()`	Returns number of elements	`queue.size();`	O(1)

📌 **Comparison of Queues:**

- `Queue (LinkedList)` – Standard FIFO queue.
- `PriorityQueue` – Orders elements by **priority** instead of FIFO.

4️⃣ PriorityQueue (Ordered Processing)

A `PriorityQueue` **processes higher-priority elements first.**

Method	Description	Example	Time Complexity
`add(E e)`	Adds an element with priority	`pq.add(10);`	O(log n)
`poll()`	Retrieves & removes highest priority element	`pq.poll();`	O(log n)
`peek()`	Retrieves highest priority element without removing	`pq.peek();`	O(1)
`size()`	Returns number of elements	`pq.size();`	O(1)

📌 **Types of Priority Queues:**

- **Min-Heap (Default)** → Smallest values processed first.
- **Max-Heap** → Largest values first (**use** `Comparator.reverseOrder()`).

🚀 Quick Comparison Table

Collection	Uniqueness	Order	Best Use Case
`HashSet`	✅ Unique elements	❌ No order	Removing duplicates
`LinkedHashSet`	✅ Unique elements	✅ Insertion order	Maintaining uniqueness + order
`TreeSet`	✅ Unique elements	✅ Sorted order	Sorted unique elements
`HashMap`	✅ Unique keys	❌ No order	Fast key-value lookup
`LinkedHashMap`	✅ Unique keys	✅ Insertion order	Ordered key-value storage
`TreeMap`	✅ Unique keys	✅ Sorted order	Sorted key-value pairs

`Queue (LinkedList)`	❌ Duplicates allowed	✅ FIFO order	Customer service queue
`PriorityQueue`	❌ Duplicates allowed	✅ Priority order	Task scheduling