Here’s a **comprehensive comparison table** covering the differences between all major Java data structure implementations:

**Java Data Structures - Comparison Table**

| **Category** | **Implementation** | **Ordering** | **Duplicates Allowed** | **Thread-Safe** | **Performance (Avg. Case)** | **Best For** |
| --- | --- | --- | --- | --- | --- | --- |
| **List** | ArrayList | ✅ Yes (Insertion Order) | ✅ Yes | ❌ No | O(1) access, O(n) insert/delete | Fast random access |
|  | LinkedList | ✅ Yes (Insertion Order) | ✅ Yes | ❌ No | O(n) access, O(1) insert/delete at head/tail | Frequent insertions/deletions |
|  | Vector | ✅ Yes (Insertion Order) | ✅ Yes | ✅ Yes (Synchronized) | O(1) access, O(n) insert/delete | Thread-safe ArrayList alternative |
|  | CopyOnWriteArrayList | ✅ Yes (Insertion Order) | ✅ Yes | ✅ Yes (Copy-on-Write) | O(n) write, O(1) read | Read-heavy operations in multi-threading |
| **Queue** | LinkedList (Queue mode) | ✅ Yes (FIFO) | ✅ Yes | ❌ No | O(1) insert/delete, O(n) access | General queue usage |
|  | PriorityQueue | ❌ No (Sorted Order) | ✅ Yes | ❌ No | O(log n) insert/delete | Priority-based tasks |
|  | ArrayDeque | ✅ Yes (Insertion Order) | ✅ Yes | ❌ No | O(1) insert/delete | Faster stack/queue alternative |
| **Stack** | Stack | ✅ Yes (LIFO) | ✅ Yes | ✅ Yes (Synchronized) | O(1) push/pop | Legacy stack (use ArrayDeque instead) |
|  | ArrayDeque | ✅ Yes (LIFO) | ✅ Yes | ❌ No | O(1) push/pop | Preferred stack alternative |
| **Set** | HashSet | ❌ No | ❌ No | ❌ No | O(1) insert/delete | Unique items, fast lookup |
|  | LinkedHashSet | ✅ Yes (Insertion Order) | ❌ No | ❌ No | O(1) insert/delete | Ordered unique elements |
|  | TreeSet | ✅ Yes (Sorted Order) | ❌ No | ❌ No | O(log n) insert/delete | Sorted unique elements |
|  | CopyOnWriteArraySet | ✅ Yes (Insertion Order) | ❌ No | ✅ Yes (Copy-on-Write) | O(n) insert, O(1) read | Read-heavy concurrent set |
| **Map** | HashMap | ❌ No | ❌ No (Unique Keys) | ❌ No | O(1) insert/delete | Fast key-value lookups |
|  | LinkedHashMap | ✅ Yes (Insertion Order) | ❌ No (Unique Keys) | ❌ No | O(1) insert/delete | Ordered key-value pairs |
|  | TreeMap | ✅ Yes (Sorted Order) | ❌ No (Unique Keys) | ❌ No | O(log n) insert/delete | Sorted key-value pairs |
|  | ConcurrentHashMap | ❌ No | ❌ No (Unique Keys) | ✅ Yes (Thread-Safe) | O(1) insert/delete | Fast thread-safe map |
|  | Hashtable | ❌ No | ❌ No (Unique Keys) | ✅ Yes (Synchronized) | O(1) insert/delete | Legacy thread-safe map |
| **Concurrent Queues** | ConcurrentLinkedQueue | ✅ Yes (FIFO) | ✅ Yes | ✅ Yes (Non-blocking) | O(1) insert/delete | Thread-safe queue |
|  | BlockingQueue (ArrayBlockingQueue, LinkedBlockingQueue) | ✅ Yes (FIFO) | ✅ Yes | ✅ Yes (Blocking) | O(1) insert/delete | Producer-consumer scenarios |
| **Graph** | Adjacency List | ❌ No | ✅ Yes | ❌ No | O(1) add edge, O(V+E) traverse | Sparse graphs |
|  | Adjacency Matrix | ❌ No | ✅ Yes | ❌ No | O(1) edge check, O(V²) space | Dense graphs |

**Which One Should You Use?**

* **Fast lookup with unique elements?** → HashSet or HashMap
* **Maintain insertion order?** → LinkedHashSet or LinkedHashMap
* **Need sorting?** → TreeSet or TreeMap
* **Thread safety required?** → CopyOnWriteArrayList, ConcurrentHashMap, BlockingQueue
* **Fast queue operations?** → ArrayDeque (better than Stack and LinkedList)
* **Priority-based tasks?** → PriorityQueue

Would you like code examples for any specific structure? 🚀