Java Collections Cheat Sheet

ArrayList

Use When: Fast access by index, frequent reads.

Avoid When: Frequent inserts/deletes in the middle or beginning.

Use Case: Storing student names or item lists.

Time Complexities: get(index): O(1), add(element): O(1) amortized, add/remove(index): O(n), contains: O(n)

LinkedList

Use When: Frequent inserts/deletes at beginning/middle.

Avoid When: Random access is needed.

Use Case: Browser history, undo operations.

Time Complexities: get(index): O(n), add/removeFirst/Last(): O(1), contains: O(n)

HashSet

Use When: Unique values, no ordering needed.

Avoid When: Sorting or ordering is required.

Use Case: Store unique user IDs.

Time Complexities: add/remove/contains: O(1) avg, O(n) worst

LinkedHashSet

Use When: Unique values + insertion order.

Avoid When: Memory usage is critical.

Use Case: Cache where order matters.

Time Complexities: Same as HashSet + maintains insertion order

TreeSet

Use When: Sorted, unique elements.

Avoid When: Sorting isn't needed.

Use Case: Sorted leaderboard.

Time Complexities: add/remove/contains: O(log n)

HashMap

Use When: Fast key-value access without ordering.

Avoid When: Ordered keys or values are needed.

Use Case: Store user profiles (username to object).

Time Complexities: get/put/remove: O(1) avg, O(n) worst

LinkedHashMap

Use When: Maintain insertion/access order.

Avoid When: Don't care about ordering.

Use Case: LRU cache implementation.

Time Complexities: Same as HashMap + maintains order

TreeMap

Use When: Sorted keys required.

Avoid When: No sorting needed.

Use Case: Sorted timeline data.

Time Complexities: get/put/remove: O(log n)

ConcurrentHashMap

Use When: Thread-safe map access.

Avoid When: Single-threaded environment.

Use Case: Caching in multithreaded services.

Time Complexities: get/put: O(1) avg

PriorityQueue

Use When: Min/Max heap with priority logic.

Avoid When: Need constant-time insertion/removal.

Use Case: Job scheduler, Dijkstra's algorithm.

Time Complexities: add/remove: O(log n), peek: O(1)

ArrayDeque

Use When: Fast stack/queue operations.

Avoid When: Need random access.

Use Case: Undo/Redo stack, task queues.

Time Complexities: add/removeFirst/Last: O(1)

Stack (Legacy)

Use When: Maintaining legacy code.

Avoid When: Starting fresh; prefer Deque.

Use Case: Backtracking in legacy code.

Time Complexities: push/pop/peek: O(1)