Detailed Comparison: ArrayList vs. LinkedList vs. HashSet

Introduction

Java provides various collection classes, and choosing the right one is crucial for performance. This document compares ArrayList, LinkedList, and HashSet based on insertion, deletion, search performance, and memory usage.

Performance Comparison Table

Operation	ArrayList	LinkedList	HashSet
Add (Insertion)	O(1) Amortized, O(n) Middle	O(1) Head/Tail, O(n) Middle	O(1) Best, O(n) Worst
Delete (Removal)	O(n) (Shift Elements)	O(1) Head/Tail, O(n) Middle	O(1) Best, O(n) Worst
Search (Contains)	O(n) Linear	O(n) Linear	O(1) Best, O(n) Worst
Memory Usage	Low (Compact)	High (Extra Pointers)	Higher (Hashing Overhead)

ArrayList

ArrayList is backed by a dynamic array, meaning elements are stored in contiguous memory locations. It provides fast random access (O(1) for get()), but inserting or deleting elements in the middle requires shifting elements, making those operations O(n). Best suited for scenarios where frequent random access and append operations are needed.

LinkedList

LinkedList is implemented as a doubly linked list, meaning each element points to both its previous and next elements. Inserting and deleting elements is efficient (O(1) at head/tail), but accessing elements by index is slow (O(n)), as it requires traversal from the head or tail. Useful when there are frequent insertions/deletions but not many random accesses.

HashSet

HashSet is backed by a HashMap, making insertions, deletions, and lookups average O(1) due to hashing. However, in case of hash collisions, performance degrades to O(n). It does not allow duplicate elements and does not maintain any order. Best for fast membership checks when ordering is not important.

Best Use Cases

Fast Random Access	ArrayList
Frequent Insert/Delete at Middle	LinkedList

Fastest Contains Check	HashSet
Unique Elements (No Duplicates)	HashSet
Sorted Order Needed	TreeSet