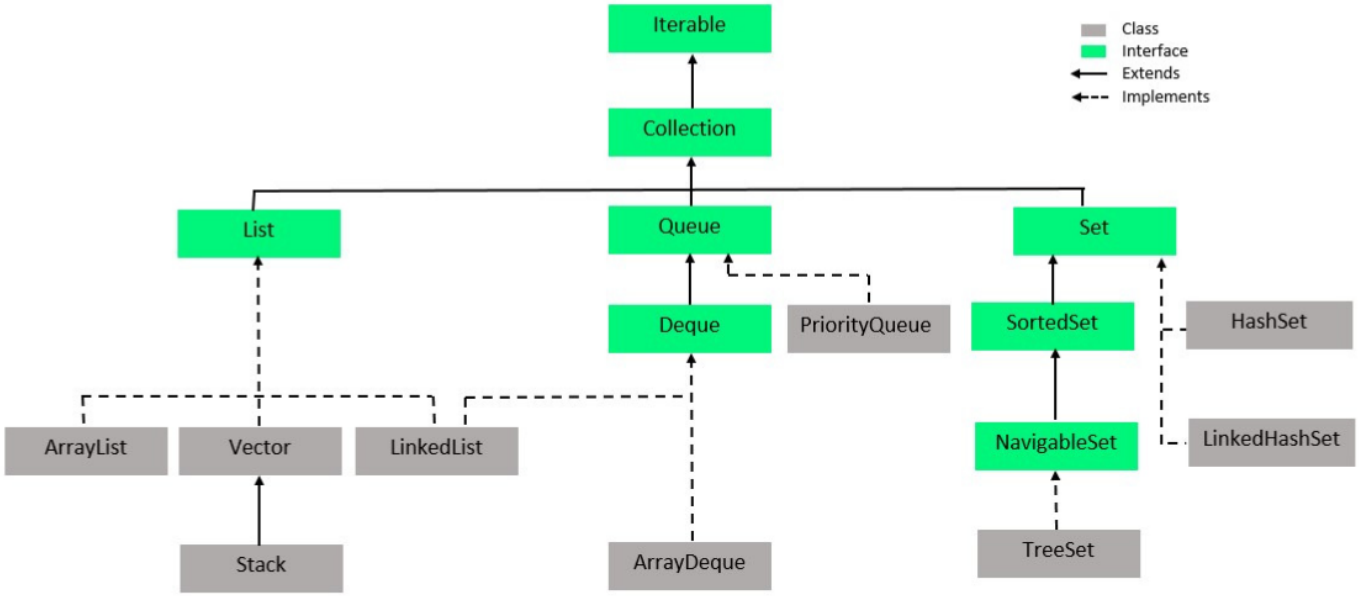
Overview



Purpose: Provides a set of interfaces and classes for storing and manipulating groups of data as a single unit.

Main Interfaces: Collection, List, Set, Queue, Deque, Map.

Collection Types & Scenarios

ArrayList

Use Case: Dynamic arrays, random access.

Benefits: Fast iteration and fast random access.

Drawbacks: Slow insertion and removal (especially at the beginning).

LinkedList

Use Case: Insertions/deletions at any point.

Benefits: Fast insertions and deletions.

Drawbacks: Slow random access.

HashSet

Use Case: Unique elements, hash table.

Benefits: Fast access, ensures uniqueness.

Drawbacks: No order of elements, no duplicates allowed.

LinkedHashSet

Use Case: Unique elements with predictable iteration order.

Benefits: Order of elements maintained.

Drawbacks: Slightly slower than HashSet.

TreeSet

Use Case: Unique sorted elements.

Benefits: Sorted set, good for range queries.

Drawbacks: Slower than HashSet.

HashMap

Use Case: Key-value pairs.

Benefits: Fast lookups.

Drawbacks: No ordering.

LinkedHashMap

Use Case: Key-value pairs with predictable iteration order.

Benefits: Maintains insertion order.

Drawbacks: Slightly slower than HashMap.

TreeMap

Use Case: Key-value pairs in sorted order.

Benefits: Sorted map, good for range queries.

Drawbacks: Slower than HashMap.

PriorityQueue

Use Case: Elements ordered by priority.

Benefits: Efficiently manages priority elements.

Drawbacks: Not suitable for non-comparable elements.

Vector

Use Case: Synchronized alternative to ArrayList.

Benefits: Thread safety.

Drawbacks: More overhead, less efficient than ArrayList.

Stack

Use Case: LIFO data structure.

Benefits: Easy stack operations.

Drawbacks: Outdated, slower than modern alternatives.

Useful Methods

Iterating Over Collections: For-each loop, Iterator, ListIterator, Stream API.

Converting/Casting Collections: Using toArray(new Type[0]), Collections.addAll, Stream API.

Constructors & Initialization: Copy constructors, Collections.unmodifiableCollection, initializing from another collection.

Tips & Tricks

HashSet vs ArrayList: Use HashSet when uniqueness is a priority; ArrayList when order matters.

ArrayList vs LinkedList: Prefer ArrayList for small lists or where random access is frequent.

Map Key Uniqueness: Keys in maps (HashMap, TreeMap) are unique.

Null Elements: LinkedList, HashMap, and TreeMap (only keys) allow null elements; TreeSet and TreeMap (values) do not.

Synchronization: Collections are not synchronized by default. Use Collections.synchronizedList, etc., for thread safety.

Sorting: Use Collections.sort for Lists; TreeSet and TreeMap inherently maintain order.

Use Case Examples

Creating int[] from Collection of Integers: collection.stream().mapToInt(Integer::intValue).toArray();

Initializing HashSet from an ArrayList: new HashSet<>(arrayList);