Java Collections Framework - Detailed Notes

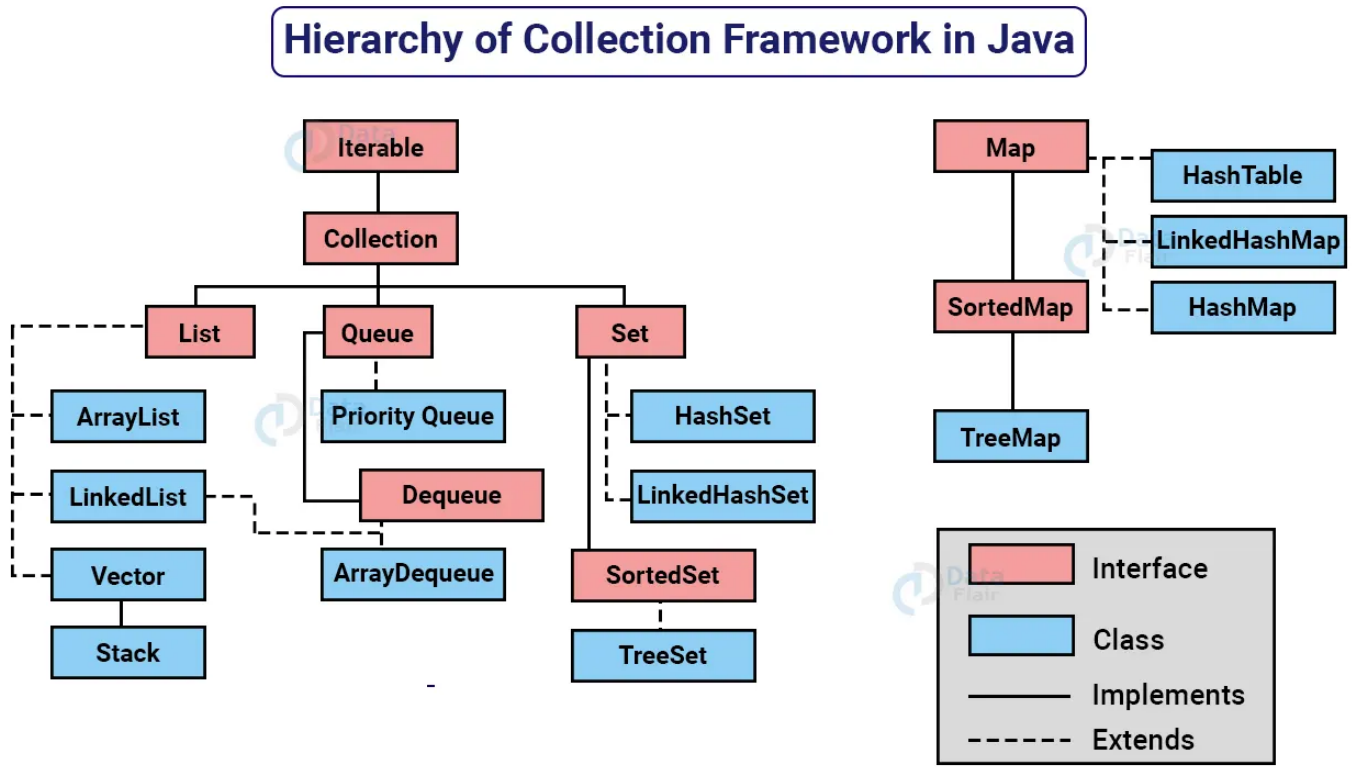
1. Introduction to Collections Framework

The Java Collections Framework is a unified architecture for representing and manipulating collections. It includes interfaces, implementations (classes), and algorithms. Collections are used to store, retrieve, and manipulate data efficiently. It resides in the **java.util** package and supports operations such as searching, sorting, insertion, manipulation, and deletion.

2. Collection Interface

This is the root interface of the Java Collections Framework and defines the core methods common to all collections like add(), remove(), size(), and iterator(). Although you don't directly instantiate Collection, its sub interfaces (List, Set, Queue) are used frequently.

Hierarchy of Collection Interface



3. List Interface - Ordered Collection with Duplicates

List is an ordered collection that can contain duplicate elements. It allows positional access using indices and maintains insertion order. Suitable when you need to maintain the sequence of insertion.

Example:

List<String> list = new ArrayList<>(); list.add("A");
list.add("B"); list.add("A");

Java Collections Framework - Detailed Notes

3.1 ArrayList

Backed by a dynamically resizing array, ArrayList is preferred for frequent read and random access operations. It resizes itself by 50% when full.
Example:
List<Integer> list = new ArrayList<>(); list.add(10);
list.add(20);

3.2 LinkedList

Implements a doubly linked list. Provides better performance than ArrayList for add/remove operations in the middle of the list.
Example:
List<String> names = new LinkedList<>(); names.add("Alice");
names.add("Bob");

3.3 Vector

A legacy thread-safe list. All methods are synchronized, making it slower in non-threaded environments.
Example:
Vector<String> v = new Vector<>(); v.add("X");
v.add("Y");

3.4 Stack

Extends Vector and represents a LIFO structure. Supports push(), pop(), peek().
Example:
Stack<Integer> stack = new Stack<>(); stack.push(1);
stack.pop();

4. Set Interface - No Duplicates Allowed

A Set is a collection that doesn't allow duplicate elements. It models the mathematical set abstraction.
Example:
Set<String> set = new HashSet<>();
set.add("A"); set.add("B"); set.add("A"); // "A" won't be added twice

4.1 HashSet

Implements Set using a HashMap internally. Offers constant time performance for basic operations, assuming good hash function.
Example:
Set<Integer> nums = new HashSet<>(); nums.add(1);
nums.add(2);

4.2 LinkedHashSet

Extends HashSet and maintains insertion order using a linked list. Suitable when order preservation is needed.
Example:

Set<String> orderedSet = new LinkedHashSet<>(); orderedSet.add("One");
orderedSet.add("Two");

4.3 TreeSet

Implements NavigableSet using a Red-Black Tree. Maintains elements in sorted order.
Example:
Set<Integer> sortedSet = new TreeSet<>(); sortedSet.add(10);
sortedSet.add(5);

5. Queue Interface

Represents a collection designed for holding elements prior to processing. Commonly used in BFS, scheduling.
Example:
Queue<String> queue = new LinkedList<>(); queue.add("Task1");
queue.poll();

6. Map Interface - Key-Value Pair Collection

Map is not a true child of Collection. It stores key-value pairs with unique keys. It allows null values and one null key in most implementations.
Example:
Map<String, Integer> map = new HashMap<>(); map.put("A",
100); map.put("B", 200);

6.1 HashMap

Uses an array of buckets and hashing. From Java 8, switches to a tree if many hash collisions occur.
Example:
Map<String, String> capitals = new HashMap<>(); capitals.put("India",
"Delhi");

6.2 LinkedHashMap

Maintains insertion order. Ideal when predictable iteration order is needed.
Example:
Map<String, Integer> linkedMap = new LinkedHashMap<>(); linkedMap.put("One",
1); linkedMap.put("Two", 2);

6.3 TreeMap

Implements NavigableMap, stores keys in sorted order using Red-Black Tree. Doesn't allow null keys.
Example:
Map<Integer, String> treeMap = new TreeMap<>(); treeMap.put(1,
"A"); treeMap.put(2, "B");

6.4 Hashtable

Legacy synchronized map. Doesn't allow null keys or values. Slower compared to modern concurrent maps.
Example:

Hashtable<Integer, String> table = new Hashtable<>();
table.put(1,"X"); table.put(2, "Y");

6.5 ConcurrentHashMap

Thread-safe implementation allowing high concurrency. Doesn't allow null keys or values. Segmented locks (Java 7), or bin-level locking (Java 8+).

Example:

Map<String, Integer> conMap = new ConcurrentHashMap<>(); conMap.put("Count",
10);