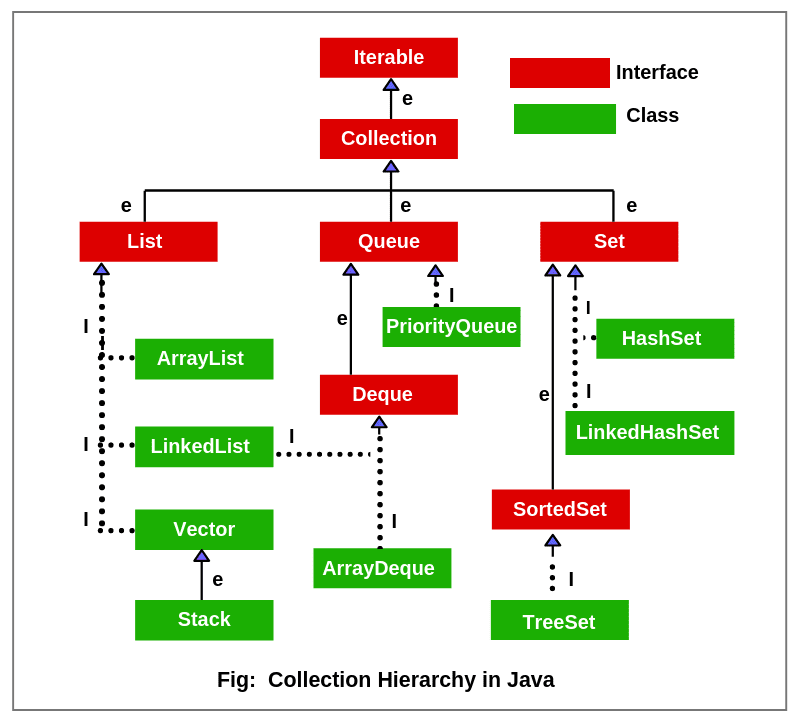
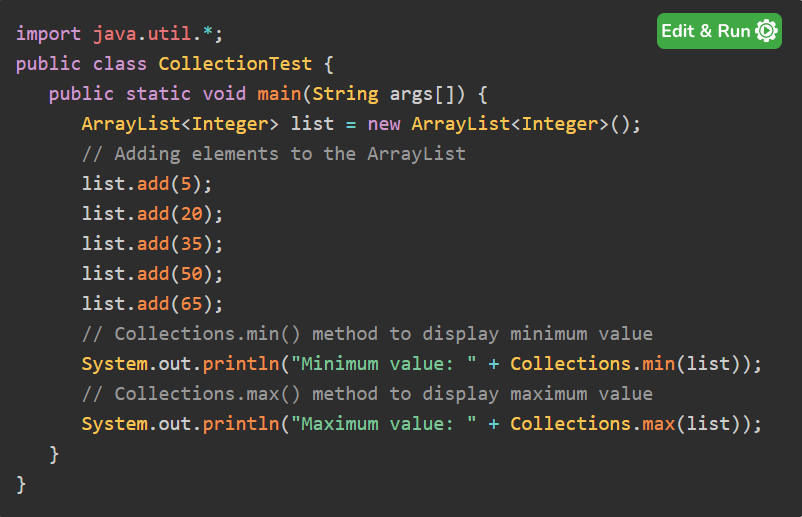
**Collections**

# **Collection vs Collections in Java with Example:**

* The Collection is an interface whereas Collections is a utility class in Java. The Set, List, and Queue are some of the subinterfaces of Collection interface, a Map interface is also part of the Collections Framework, but it doesn't inherit Collection interface.
* The important methods of Collection interface are add(), remove(), size(), clear() etc and the Collections class contains only static methods like sort(), min(), max(), fill(), copy(), reverse() etc.

| **Collection** | **Collections** |
| --- | --- |
| It is an interface. | It is a utility class. |
| It is used to represent a group of individual objects as a single unit. | It defines several utility methods that are used to operate on collection. |
| The Collection is an interface that contains a static method since java8. The Interface can also contain abstract and default methods. | It contains only static methods. |





1. **List**
2. **Linked list**
   1. Singly linked list
   2. Doubly linked list (similar to Deque)

Uses:

* Frequent add/remove during iteration

Note: linked list supports all LIFO & FIFO operations in O(1) = constant time.

1. **Queue**

* Useful when manipulating head and tail
* Tail → add. Head → remove / retrieve
* FIFO
* Can have nulls, but generally not supported → so better inserting null
* No indexed access

**Methods - Throws exception - returns special value**

Insert - add(e) - offer(e)

Remove - remove() - poll()

Inspect - element() - peek()

**3.1. Deque (double ended queue)**

* Extends queue
* Models FIFO & LIFO
* Deque implementations:
  + - arrayDeque
    - linkedList
    - concurrentLinkedDeque
    - linkedBlockingDeque

**Array deque:**

* A resizable array implementation of a deque interface.
* Models FIFO & LIFO
* Ways of implementations :- Array deque() or arrayDeque(int) or ArrayDeque(Collection)
* Unlike linkedList → nulls are prohibited and does not implement list interface

1. **Hash table**

* Implements an associative array
* <key,value> → <name,phone> = mapping
* Also known dictionary

Key operations:

* Insert <key,value>
* Search by key
* Remove by key

Characteristics:

* No duplicate key
* Duplicate values is fine
* 1 key → 1 value
* Nulls:
  + 1 null key, null values are fine
  + No nulls at all (some implementations)

**Note:** Hash function is applied on key. Hash function properties → quickly locate bucket, uniformly disperse elements.

Applications:

* Database indexing
* noSql databases
* Switch statements

1. **Set & HashSet**

**Set:**

* No duplicates
* Useful when uniqueness & fast lookup matters

**HashSet:**

* Hash table implementation of set interface
* Internally uses hashMap
* Key = element, value = new Object()
* Permits one null element

Use cases:

* Rapid lookup, insertion, deletion
* Insertion order is not important
* Better for removeAll() & retainAll()

1. **LinkedHashSet**

* Preserves insertion order → doubly linked list
* Extends HashSet - nearly as fast too
* Rapid lookup, insertion, deletion
* Permits one null element
* Internally uses linked hashMap

1. **SortedSet & NavigableSet**
2. **Tree Set**

* Red black tree → based implementation of navigableSet interface
* Internally uses treeMap
* Key = element, value = new Object()
* Elements are unique and sorted
* Fast lookup → add/remove/retains

**Note**:

* comparable → natural ordering, comparator → we can define.
* Comparable – java.lang.comparable, comparator – java.util.comparator
* If Comparable interface implemented then **compareTo** method should be implemented
* (note: string class also implements comparable)
* In natural ordering elements itself will decide how to sort.

1. **Map interface**

* Implements an associative array
* No duplicates key, duplicate values are fine
* 1 key → 1 value
* Null key & null values → impl dependent
* abstractMap
* Basic operation:
  + put(k,v),
  + get(object key),
  + remove(object key)
  + Boolean containsKey(object key)
  + Boolean containsValue(object value)
  + Int size()
  + Boolean isEmpty()
* Bulk operation:
  + Void putAll(Map <? Extends k, ?extends v> m)
  + Void clear();
* Collection view:
  + Set(k) keySet();
  + collection<v> values();
  + set<Map.Entry <k,v>> entrySet(); → used for iterating

Note:

Public interface Entry{

K getkey()

V getvalue()

V setvalue(v value)

}

Entry is a Nested interface within map interface.

**Note:** keys should be immutable

1. **Hash Map**

* Insertion order is not preserved
* Permits null values and one null key
* Not sync unlike hashtable

1. **Linked HashMap**

* Hash map & linked list implementation of map interface
* Preserves insertion order - doubly linked list
* Extends hashMap
* Permit null values and one null key
* Not sync

LinkedHashMap used to implement LRU cache → Least Recently Used Cache

1. **Sorted Map**

* Map capabilities + data sorted by keys
* Natural ordering or comparator → by keys

**Operations: (similar to sortedset)**

**Range view**

* SortedMap<k,v> subMap(k fromkey, k tokey);
* SortedMap<k,v> headMap( k tokey);
* SortedMap<k,v> tailMap(k fromkey);

**Endpoints**

* K firstkey()
* K lastkey()

**Comparator access**

* comparator<?super E>comparator();

**Collection view operations (backed by up original map, not permit add operation)**

* set<k>keyset();
* collection<v> values();
* set<Map.entry<k,v>> entryset();