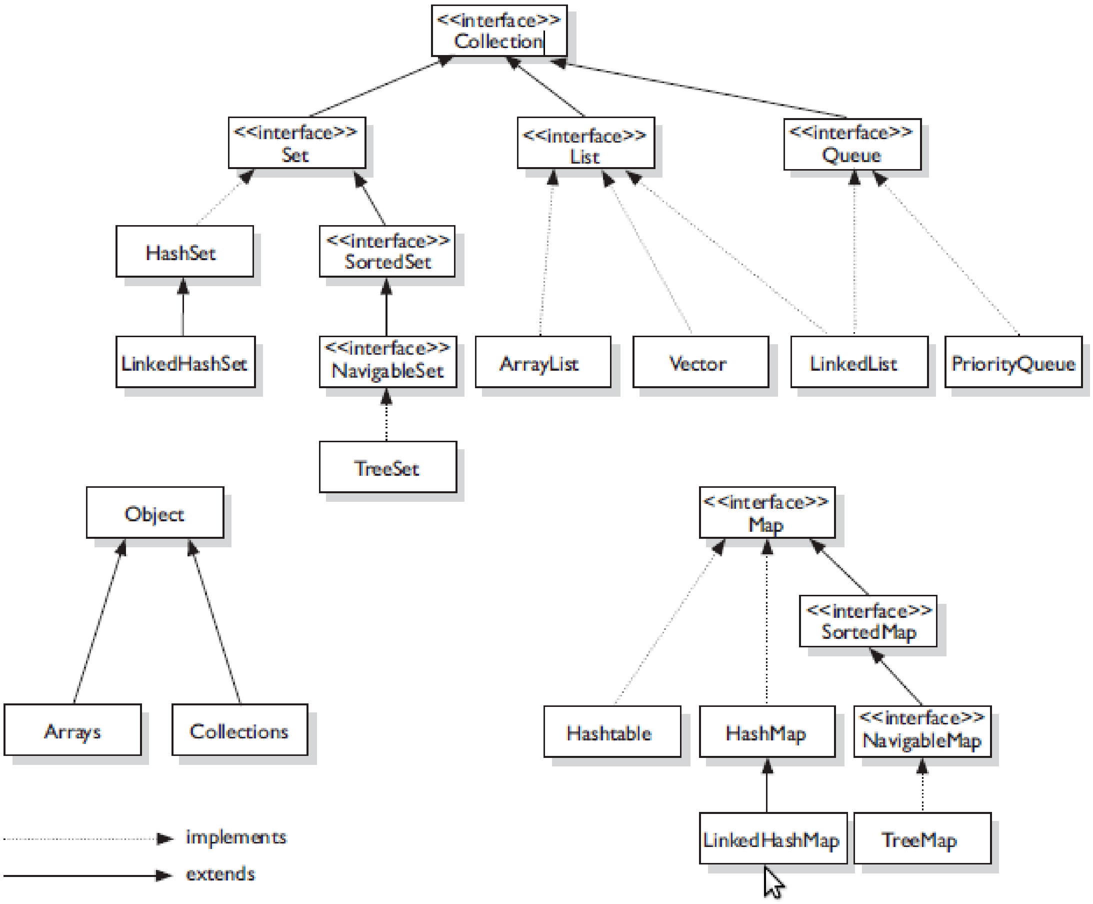
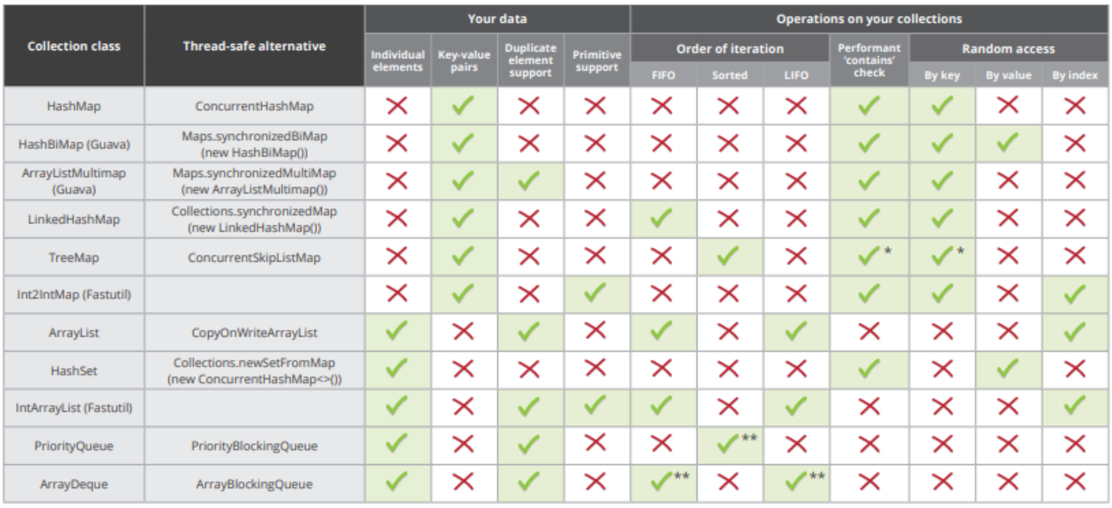
Java Collections Cheat Sheet

By Omar Shammout

Collections Hierarchy



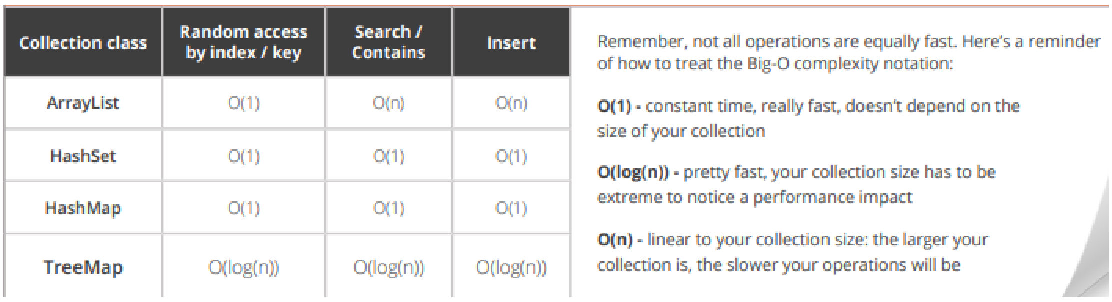


Collection class	Thread-safe alternative	Your data				Operations on your collections						
		Individual elements	Key-value pairs	Duplicate element support	Primitive support	Order of iteration			Performant 'contains' check	Random access		
						FIFO	Sorted	LIFO		By key	By value	By index
HashMap	ConcurrentHashMap	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
HashBiMap (Guava)	Maps.synchronizedBiMap (new HashBiMap())	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗
ArrayListMultimap (Guava)	Maps.synchronizedMultiMap (new ArrayListMultimap())	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗
LinkedHashMap	Collections.synchronizedMap (new LinkedHashMap())	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗
TreeMap	ConcurrentSkipListMap	✗	✓	✗	✗	✗	✓	✗	✓*	✓*	✗	✗
Int2IntMap (Fastutil)		✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	✓
ArrayList	CopyOnWriteArrayList	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓
HashSet	Collections.newSetFromMap (new ConcurrentHashMap<>())	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
IntArrayList (Fastutil)		✓	✗	✓	✓	✓	✗	✓	✗	✗	✗	✓
PriorityQueue	PriorityBlockingQueue	✓	✗	✓	✗	✗	✓**	✗	✗	✗	✗	✗
ArrayDeque	ArrayBlockingQueue	✓	✗	✓	✗	✓**	✗	✓**	✗	✗	✗	✗

**O(log(n)) complexity while all others are O(1) – constant time*

*** when using Queue interface methods: offer() / poll()*

Collections' Operations Complexity



Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	O(1)	O(n)	O(n)
HashSet	O(1)	O(1)	O(1)
HashMap	O(1)	O(1)	O(1)
TreeMap	O(log(n))	O(log(n))	O(log(n))

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

O(1) - constant time, really fast, doesn't depend on the size of your collection

O(log(n)) - pretty fast, your collection size has to be extreme to notice a performance impact

O(n) - linear to your collection size: the larger your collection is, the slower your operations will be

Collections Classes

***Map :** - An object that associates keys to values (e.g., SSN ⇒ Person).

- Keys and values must be objects.

- Keys must be unique.

- Only one value per key.

***List :** - Can contain duplicate elements.

- Insertion order is preserved.

- User can define insertion point.

- Elements can be accessed by position.

- Augments Collection interface.

***Set :** - Contains no methods other than those inherited from Collection.

- add()has restriction that no duplicate elements are allowed.

Collections Implementations

Map →

♣ HashMap: - implements Map.

-No order.

♣ LinkedHashMap extends HashMap : - Insertion order.

♣ TreeMap implements SortedMap : - Ascending key order.

Queue →

♣ LinkedList : - head is the first element of the list.

- FIFO: Fist-In-First-Out.

♣ PriorityQueue : - head is the smallest element.

Set →

♣ SortedSet : - No duplicate elements.

♣ HashSet implements Set : - Hash tables as internal data structure (faster).

♣ LinkedHashSet extends HashSet : - Elements are traversed by iterator according to the insertion order.

♣ TreeSet implements SortedSet : - R-B trees as internal data structure (computationally expensive).

-Depending on the constructor used they require different implementation of the custom ordering.

♣ TreeSet() : - Natural ordering (elements must be implementations of Comparable).

♣ TreeSet(Comparator c) : - Ordering is according to the comparator rules, instead of natural ordering.

Common Methods

Collection Interface → ♣int size()   ♣ boolean isEmpty()

♣ boolean contains(Object element)

♣ boolean containsAll(Collection c)

♣ boolean add(Object element)   ♣ boolean addAll(Collection c)

♣ boolean remove(Object element)   ♣ boolean removeAll(Collection c)

♣ void clear()   ♣ Object[] toArray()   ♣ Iterator iterator()

1) ArrayList → ♣Object get(int index) ♣Object set(int index, Object element)

♣void add(int index, Object element) ♣ Object remove(int index)

♣boolean addAll(int index, Collection c) ♣ int indexOf(Object o)

♣ int lastIndexOf(Object o) ♣ List subList(int fromIndex, int toIndex)

2) LinkedList → ♣void addFirst(Object o) ♣void addLast(Object o) ♣Object getFirst()

♣Object getLast() ♣Object removeFirst() ♣Object removeLast()

3) Map → ♣ Object put(Object key, Object value) ♣ Object get(Object key)

♣ Object remove(Object key) ♣ boolean containsKey(Object key)

♣ boolean containsValue(Object value) ♣ public Set keySet()

♣ public Collection values() ♣ int size() ♣ boolean isEmpty() ♣ void clear()

4) Queue → ♣peek() ♣poll()

Conclusion

- All **lists allow duplicates**; no sets or maps allow duplicates.
- All **list elements are ordered**, i.e., it maintains the order of insertion. So does all sets except **HashSet** and all maps except **HashMap** and HashTable.
- No collections are **sorted** except **TreeSet** and **TreeMap**.
- Except **Vector** and **HashTable**, no other collection is thread safe.

What And How to Choose?

Java has dozens of collection classes and interfaces. Below are some of the considerations that may help you to choose one.

- If you need to access data by index, consider using ArrayList.
- If you need to often insert or remove data in/from a collection, a LinkedList should be a good choice
- If you need a collection that doesn't allow duplicate elements, use one of the collections that implements Set interface. For fast access use HashSet. For sorted set use TreeSet.
- For storing key/value pairs use a collection that implements the Map interface; e.g., HashMap or HashTable.

- If you need a collection for a fast search that remains fast regardless of the size of the data set use HashSet.

Resources:

-https://en.proft.me

-https://www.linkedin.com/pulse/java-collections-table-cheat-sheet-apala-sengupta

-https://softeng.polito.it

-https://www.geeksforgeeks.org