Collections Framework

A collection is an object that groups multiple elements into a single unit. The Collection interface is the foundation upon which the collections framework is built. It declares the core methods that all collections will have. A collections framework is a unified architecture for representing and manipulating collections.

The core collection interfaces encapsulate different types of collections, which are the following.

1. Set
2. List
3. Queue
4. Map
5. SortedSet
6. SortedMap

There are three ways to traverse collections:

1. Aggregate operations
2. for-each construct
3. Iterators

Bulk Operations

1. containsAll — returns true if the target Collection contains all of the elements in the specified Collection.
2. addAll — adds all of the elements in the specified Collection to the target Collection.
3. removeAll — removes from the target Collection all of its elements that are also contained in the specified Collection.
4. retainAll — removes from the target Collection all its elements that are not also contained in the specified Collection. That is, it retains only those elements in the target Collection that are also contained in the specified Collection.
5. clear — removes all elements from the Collection.

Set

Set interface is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited. Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

The Java platform contains three general-purpose Set implementations: HashSet, TreeSet, and LinkedHashSet.

HashSet, which stores its elements in a hash table, is the best-performing implementation; however it makes no guarantees concerning the order of iteration.

TreeSet, which stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashSet.

LinkedHashSet, which is implemented as a hash table with a linked list running through it, orders its elements based on the order in which they were inserted into the set (insertion-order).

Bulk operations are particularly well suited to Sets; when applied, they perform standard set-algebraic operations. Suppose s1 and s2 are sets

1. s1.containsAll(s2) — returns true if s2 is a subset of s1.
2. s1.addAll(s2) — transforms s1 into the union of s1 and s2.
3. s1.retainAll(s2) — transforms s1 into the intersection of s1 and s2.
4. s1.removeAll(s2) — transforms s1 into the set difference of s1 and s2.

List

The List interface is an ordered Collection. Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes operations for the following:

1. Positional access — manipulates elements based on their numerical position in the list. This includes methods such as get, set, add, addAll, and remove.
2. Search — searches for a specified object in the list and returns its numerical position. Search methods include indexOf and lastIndexOf.
3. Iteration — extends Iterator semantics to take advantage of the list's sequential nature. The listIterator methods provide this behavior.
4. Range-view — The sublist method performs arbitrary range operations on the list.

The following algorithms may be used with the List interface.

1. sort — sorts a List using a merge sort algorithm, which provides a fast, stable sort. (A stable sort is one that does not reorder equal elements.)
2. shuffle — randomly permutes the elements in a List.
3. reverse — reverses the order of the elements in a List.
4. rotate — rotates all the elements in a List by a specified distance.
5. swap — swaps the elements at specified positions in a List.
6. replaceAll — replaces all occurrences of one specified value with another.
7. fill — overwrites every element in a List with the specified value.
8. copy — copies the source List into the destination List.
9. binarySearch — searches for an element in an ordered List using the binary search algorithm.
10. indexOfSubList — returns the index of the first sublist of one List that is equal to another.
11. lastIndexOfSubList — returns the index of the last sublist of one List that is equal to another.

Queue

The Queue interface is a collection for holding elements prior to processing. Besides basic Collection operations, queues provide additional insertion, removal, and inspection operations. The Queue interface follows.

1. The add method, which Queue inherits from Collection, inserts an element unless it would violate the queue's capacity restrictions, in which case it throws IllegalStateException.
2. The offer method, which is intended solely for use on bounded queues, differs from add only in that it indicates failure to insert an element by returning false.
3. The remove and poll methods both remove and return the head of the queue. Exactly which element gets removed is a function of the queue's ordering policy. The remove and poll methods differ in their behavior only when the queue is empty. Under these circumstances, remove throws NoSuchElementException, while poll returns null.
4. The element and peek methods return, but do not remove, the head of the queue. They differ from one another in precisely the same fashion as remove and poll: If the queue is empty, element throws NoSuchElementException, while peek returns null.

Map

The Map interface maps unique keys to values. A map cannot contain duplicate keys: Each key can map to at most one value. It models the mathematical function abstraction. The Map interface includes methods for the following

1. Basic operations such as put, get, remove, containsKey, containsValue, size, and empty.
2. Bulk operations such as putAll and clear. The clear operation removes all the mappings from the Map. The putAll operation is the Map analogue of the Collection interface's addAll operation.
3. Collection views such as
4. keyset — the Set of keys contained in the Map.
5. entrySet — the Set of key-value pairs contained in the Map. The Map interface provides a small nested interface called Map.Entry, the type of the elements in this Set.
6. values — The Collection of values contained in the Map. This Collection is not a Set, because multiple keys can map to the same value.

SortedSet

The SortedSet interface extends Set and maintains its elements in ascending order, sorted according to the elements' natural ordering or according to a Comparator provided at SortedSet creation time. In addition to the normal Set operations, the SortedSet interface provides operations for the following.

1. Range view — allows arbitrary range operations on the sorted set
2. Endpoints — returns the first or last element in the sorted set
3. Comparator access — returns the Comparator, if any, used to sort the set

The operations that SortedSet inherits from Set behave identically on sorted sets and normal sets with two exceptions. The Iterator returned by the iterator operation traverses the sorted set in order. The array returned by toArray contains the sorted set's elements in order.

SortedMap

The SortedMap interface is a Map that maintains its entries in ascending order, sorted according to the keys' natural ordering, or according to a Comparator provided at the time of the SortedMap creation. The SortedMap interface provides operations for normal Map operations and for the following.

1. Range view — performs arbitrary range operations on the sorted map
2. Endpoints — returns the first or the last key in the sorted map
3. Comparator access — returns the Comparator, if any, used to sort the map