

**Java Collection Framework**

**Introduction**

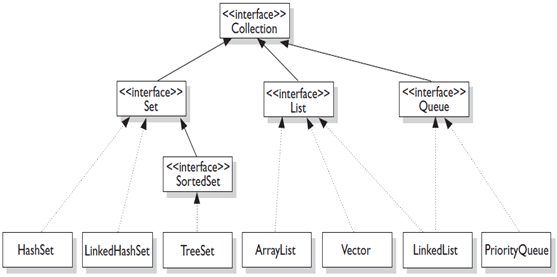
A data structure is a collection of data organized in some fashion. The structure not only stores data but also supports operations for accessing and manipulating the data. The java.util package contains one of Java’s most powerful subsystems: The Collections Framework. The Collections Framework is a sophisticated hierarchy of interfaces and classes that provide state-of-the-art technology for managing groups of objects.

You can perform following activity using Java collection frame work,

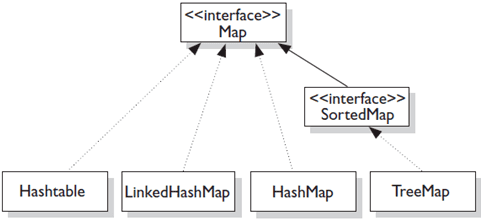
* Add objects to collection
* Remove objects from collection
* Search for an object in collection
* Retrieve/get object from collection
* Iterate through collection for business specific functionality.

**Key Interfaces and classes of collection frame work**

* **collection** (lowercase c): It represents any of the data structures in which objects are stored and iterated over.
* **Collection** (capital C): It is actually the java.util.Collection interface from which Set, List, and Queue extend.
* **Collections** (capital C and ends with s): It is the java.util.Collections class that holds a pile of static utility methods for use with collections.



There are some other classes in collection framework which does not extend Collection Interface they implement Map interface.



**We can say collection has in 4 basic flavors as below,**

* **Lists:**  
  The List interface extends Collection to define an ordered collection with duplicates allowed. The List interface adds position-oriented operations, as well as a new list iterator that enables the user to traverse the list bi-directionally. ArrayList, LinkedList and vector are classes implementing List interface.
* **Sets:**  
  The Set interface extends the Collection interface. It will make sure that an instance of Set contains no duplicate elements. The concrete class implements hashcode and equals methods to make sure uniqueness of objects. Three concrete classes of Set are HashSet, LinkedHashSet and TreeSet.
* **Maps:**  
  A map is a container that stores the elements along with the keys. The keys are like indexes. In List, the indexes are integers. In Map, the keys can be any objects. A map cannot contain duplicate keys. Each key maps to one value. A key and its corresponding value from an entry, which is actually stored in a map. HashMap, HashTable,TreeMap and LinkedHashMap are classes implementing Map interface.
* **Queues:**  
  A queue is a first-in, first-out data structure. Elements are appended to the end of the queue and are removed from the beginning of the queue. In a priority queue, elements are assigned priorities. When accessing elements, the element with the highest priority is removed first.

We can have sub flavors of collection classes like sorted, unsorted, ordered and unordered.

**Ordered:** When a collection is ordered, it means you can iterate through the collection in a specific (not-random) order

**Sorted:** A sorted collection means that the order of objects in the collection is determined according to some rule or rules, known as the sorting order. A sort order has nothing to do with when an object was added to the collection, or when was the last time it was accessed, or what "position" it was added at.

**Iterator Interface**

Iterator enables you to cycle through a collection, obtaining or removing elements. ListIterator extends Iterator to allow bidirectional traversal of a list, and the modification of elements.

An Iterator is an object that's associated with a specific collection. It let's you loop through the collection step by step. There are two important Iterator methods.

* **boolean hasNext():** Returns true if there is at least one more element in the collection being traversed. Invoking has Next() does NOT move you to the next element of the collection.
* **object next() :** This method returns the next object in the collection, AND moves you forward to the element after the element just returned.

We will see example of use of Iterator Interface in LinkedHashSet and HashMap Tutorial.

**Comparator Interface**

The Comparator interface gives you the capability to sort a given collection any number of different ways. The other handy thing about the Comparator interface is that you can use it to sort instances of any class—even classes you can't modify—unlike the Comparable interface, which forces you to change the class whose instances you want to sort.

The Comparator interface is also very easy to implement, having only one method, compare().The Comparator.compare() method returns an int.

**Method Signature**

int compare(objOne, objTwo)

Compare() method returns

* negative if objOne < objTwo
* zero if objOne == objTwo
* positive if objOne > objTwo

We will see Java program using this method in TreeMap/TreeSet Tutorial.

**Summary**

Below table summarize the discussion on collection framework.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Collection Class name | Ordered | Sorted |
| Map | Hashtable | No | No |
| HashMap | No | No |
| TreeMap | Sorted | By natural order or custom order |
| LinkedHashMap | By insertion order or last access order | No |
| Set | HashSet | No | No |
| TreeSet | Sorted | By natural order or custom order |
| LinkedHashSet | By insertion order | No |
| List | ArrayList | Indexed | No |
| Vector | Indexed | No |
| LinkedList | Indexed | No |
| Priority queue | Sorted | By to-do order |