**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.

**Collections in Java**

**Collections in java** is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

**What is Collection in java?**

Collection represents a single unit of objects i.e. a group.

**What is framework in java?**

* provides readymade architecture.
* represents set of classes and interface.
* is optional.

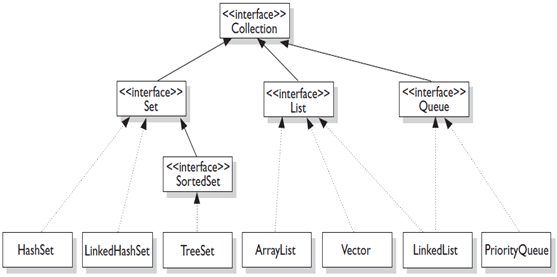
**What is Collection framework?**

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

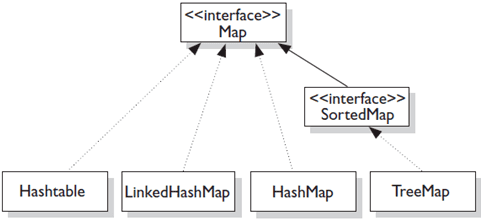
1. Interfaces and its implementations i.e. classes
2. Algorithm

**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.

**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

**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 |

﻿

### Iterator interface

#### Iterator interface provides the facility of iterating the elements in forward direction only.

#### Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

1. **public boolean hasNext()** it returns true if iterator has more elements.
2. **public object next()** it returns the element and moves the cursor pointer to the next element.
3. **public void remove()** it removes the last elements returned by the iterator. It is rarely used.

**Java ListIterator Interface**

ListIterator Interface is used to traverse the element in backward and forward direction.

Commonly used methods of ListIterator Interface:

1. public boolean hasNext();
2. public Object next();
3. public boolean hasPrevious();
4. public Object previous();

### Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(Object element) | is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection c) | is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | is used to delete an element from this collection. |
| 4 | public boolean removeAll(Collection c) | is used to delete all the elements of specified collection from the invoking collection. |
| 5 | public boolean retainAll(Collection c) | is used to delete all the elements of invoking collection except the specified collection. |
| 6 | public int size() | return the total number of elements in the collection. |
| 7 | public void clear() | removes the total no of element from the collection. |
| 8 | public boolean contains(Object element) | is used to search an element. |
| 9 | public boolean containsAll(Collection c) | is used to search the specified collection in this collection. |
| 10 | public Iterator iterator() | returns an iterator. |
| 11 | public Object[] toArray() | converts collection into array. |
| 12 | public boolean isEmpty() | checks if collection is empty. |
| 13 | public boolean equals(Object element) | matches two collection. |
| 14 | public int hashCode() | returns the hashcode number for collection. |

## List

A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. Elements can be inserted or accessed by their position in the list, using a zero-based index.

* [**ArrayList**](http://beginnersbook.com/2014/08/arraylist-in-java/)
* [**LinkedList**](http://beginnersbook.com/2014/08/java-linkedlist-class/)
* [**Vector**](http://beginnersbook.com/2014/08/java-vector-class/)

## Set

A Set is a Collection that cannot contain duplicate elements. There are three main implementations of Set interface: 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).

* [**HashSet**](http://beginnersbook.com/2013/12/hashset-class-in-java-with-example/)
* [**LinkedHashSet**](http://beginnersbook.com/2013/12/linkedhashset-class-in-java-with-example/)
* [**TreeSet**](http://beginnersbook.com/2013/12/treeset-class-in-java-with-example/)

## Map

A Map is an object that maps keys to values. A map cannot contain duplicate keys. There are three main implementations of Map interfaces: HashMap, TreeMap, and LinkedHashMap.  
HashMap: it makes no guarantees concerning the order of iteration  
TreeMap: It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.  
LinkedHashMap: It orders its elements based on the order in which they were inserted into the set (insertion-order).

* [**HashMap**](http://beginnersbook.com/2014/08/java-hashmap-class/)
* [**TreeMap**](http://beginnersbook.com/2013/12/treemap-in-java-with-example/)
* [**LinkedHashMap**](http://beginnersbook.com/2013/12/linkedhashmap-in-java/)

A *collection* is a data structure—actually, an object—that can hold references to other objects. Usually, collections contain references to objects that are all of the same type. The collections-framework interfaces declare the operations to be performed generically on various types of collections.

| **Interface** | **Description** |
| --- | --- |
| Collection | The root interface in the collections hierarchy from which interfaces Set, Queue and List are derived. |
| Set | A collection that does not contain duplicates. |
| List | An ordered collection that can contain duplicate elements. |
| Map | A collection that associates keys to values and cannot contain duplicate keys. |
| Queue | Typically a first-in, first-out collection that models a waiting line; other orders can be specified. |

