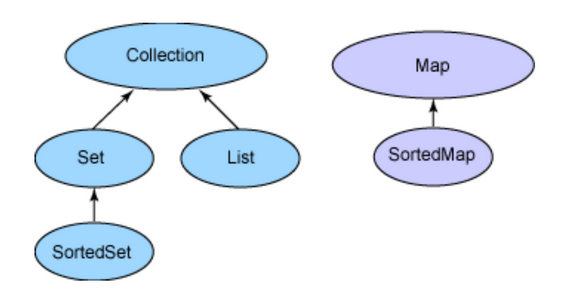
Collections in java are used to store multiple element of same or different elements, and allows you to manipulate these elements in the collection.

They can dynamically grow and shrink, which is their advantage over arrays. Collections

provide methods to add objects to a collection, remove objects from it, check if an object is

present in it, retrieve objects from it, and iterate through it.

**Inheritance hierarchy of the core collection interface**



Collection is the root interface, from which the Set and List interfaces extend. The Map

interface does not extend the Collection interface. Also, do not confuse the Collection

interface with the Collections class, which holds static utility methods for collections.

interfaces.

### 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. |

### 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.

The List interface represents ordered collection while Set cannot contain duplicate elements. The Map interface matches unique keys to values. SortedSet holds elements in sorted order while SortedMap orders the mapping in the sorted order of keys. The classes that implement these interfaces are listed

**Set** HashSet TreeSet LinkedHashSet

**List** Vector ArrayList LinkedList

**Map** HashMap HashTable TreeMap LinkedHashMap

**Set classes:-**

The classes implementing the Set interface do not allow duplicate elements.

A **HashSet** is not ordered or sorted. This class offers constant time performance for basic

operations like add and remove.

**TreeSet** arranges the elements in ascending element order, sorted according to the natural order of the elements.

A **LinkedHashSet** is an ordered HashSet, which gives the elements in the order of

**Ex:-**

LinkedHashSet linkSet = new LinkedHashSet();

linkSet.add("mango");

linkSet.add("apple");

linkSet.add("mango");

linkSet.add("banana");

Iterator i = linkSet.iterator();

while(i.hasNext())

System.out.print(i.next()); // Prints "mango apple banana"

**List classes:-**

A List is an ordered collection, which allows positional access and search.

The classes implementing List are ordered by index position. An ArrayList enables fast

iteration and constant speed positional access.

A Vector is similar to ArrayList, only slower because it is synchronized.

**LinkedList** allows fast insertion and deletion at the beginning or end. It is commonly used for implementing stacks and queues. For instance:

ArrayList list = new ArrayList();

list.add("mango");

list.add("apple");

list.add("mango");

list.add("banana");

Iterator i = list.iterator();

while(i.hasNext())

System.out.print(i.next()); // Prints "mango apple mango banana"

**Map classes:-**

The classes implementing the Map interface map unique keys to specific values.

The **HashMap** class is not sorted or ordered. It allows one null key and many null values.

**Hashtable** is similar to HashMap, but does not allow null keys and values. It's also slower

than HashMap because it is synchronized.

**LinkedHashMap** class iterates by insertion or last accessed order. It allows one null key and many null values.

**TreeMap** is a map in ascending key order, sorted according to the natural order for the key's

class.

import java.util.\*;

class TestCollection13{

public static void main(String args[]){

HashMap<Integer,String> hm=new HashMap<Integer,String>();

hm.put(100,"Amit");

hm.put(101,"Vijay");

hm.put(102,"Rahul");

for(Map.Entry m:hm.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

}

}