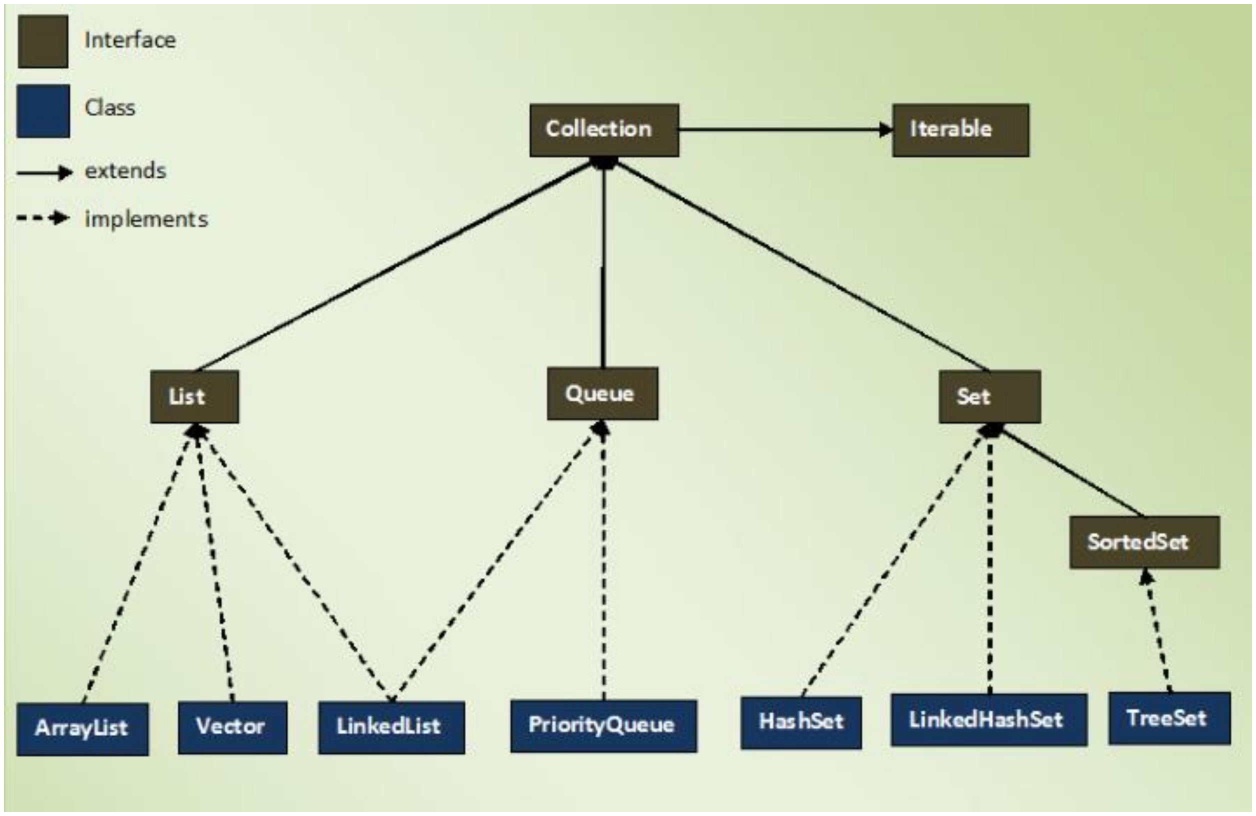
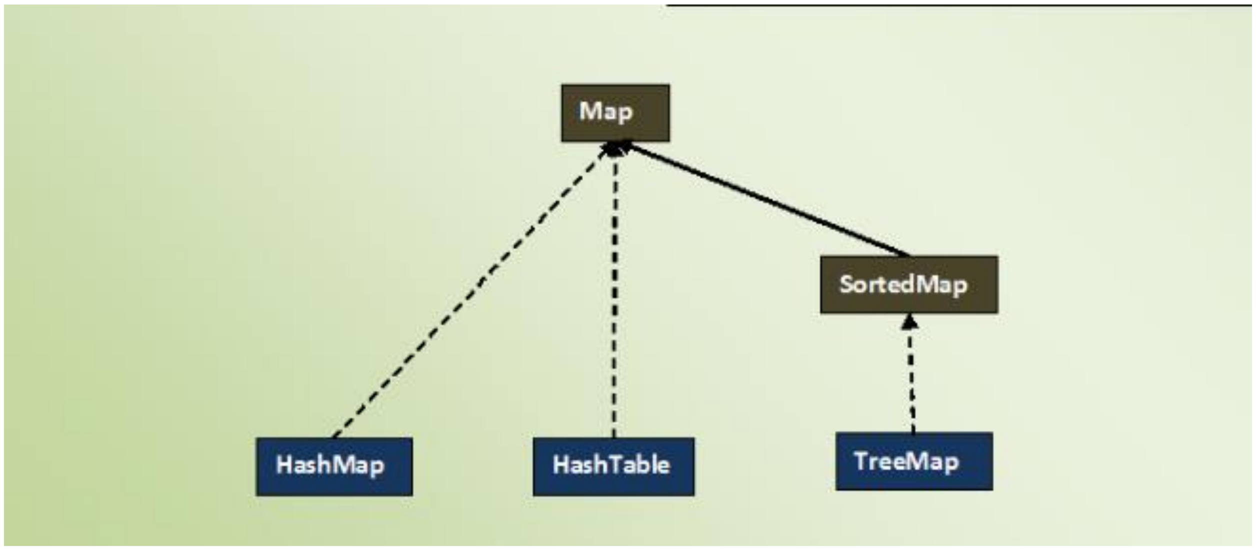
JAVA COLLECTIONS

Premkumar Balasubramanian

Collection Framework

Collections are nothing but group of objects stored in well defined manner. Earlier, Arrays are used to represent these group of objects. But, arrays are not re-sizable. size of the arrays are fixed. Size of the arrays can not be changed once they are defined. This causes lots of problem while handling group of objects. To overcome this drawback of arrays, **Collection framework** or simply collections are introduced in java from JDK 1.2.





The entire collection framework is divided into four interfaces.

1) List —> It handles sequential list of objects. **ArrayList**, **Vector** and **LinkedList** classes implement this interface.

2) Queue —> It handles special list of objects in which elements are removed only from the head. **LinkedList** and **PriorityQueue** classes implement this interface.

3) Set —> It handles list of objects which must contain unique element. This interface is implemented by **HashSet** and **LinkedHashSet** classes and extended by **SortedSet** interface which in turn, is implemented by **TreeSet**.

4) Map —> This is the one interface in Collection Framework which is not inherited from Collection interface. It handles group of objects as Key/Value pairs. It is implemented by **HashMap** and **HashTable** classes and extended by **SortedMap** interface which in turn is implemented by **TreeMap**.

ArrayList:

ArrayList, in simple terms, can be defined as re-sizable array. ArrayList is same like normal array but it can grow and shrink dynamically to hold any number of elements. ArrayList is a sequential collection of objects which increases or decreases in size as we add or delete the elements.

In ArrayList, elements are positioned according to **Zero-based index**. That means, elements are inserted from index 0. **Default initial capacity** of an ArrayList is 10. This capacity increases automatically as we add more elements to arraylist. You can also specify initial capacity of an ArrayList while creating it.

Properties Of ArrayList :

- Size of the ArrayList is not fixed. It can increase and decrease dynamically as we add or delete the elements
- ArrayList can have any number of null elements.
- ArrayList can have duplicate elements.

- As ArrayList implements RandomAccess, you can get, set, insert and remove elements of the ArrayList from any arbitrary position.
- Elements are placed according to Zero-based index. That means, first element will be placed at index 0 and last element at index n-1, where ‘n’ is the size of the ArrayList.
- If you know the element, you can retrieve the position of that element.

Sample Pgm:

```
public static void main(String[] args) {
             // TODO Auto-generated method stub
               ArrayList<Integer> list = new ArrayList<>();

              //Adding elements to ArrayList

              list.add(10);

              list.add(20);

              list.add(30);

              list.add(40);
              list.add(40);

              System.out.println(list);     //Output : [10, 20, 30, 40, 40]

              //Retrieving element at index 2

              System.out.println(list.get(2));     //Output : 30

              //Setting value of element at index 2

              list.set(2, 2222);

              System.out.println(list);      //Output : [10, 20, 2222, 40,40]

              //Inserting element at index 1

              list.add(1, 1111);

              System.out.println(list);     //Output : [10, 1111, 20, 2222, 40,40]

              //Removing element from index 3

              list.remove(3);

              System.out.println(list);    //Output : [10, 1111, 20, 40,40]

              list.add(null);
              System.out.println(list);    //Output : [10, 1111, 20, 40,40 , null]
              System.out.println(list.indexOf(20)); //Output : 1
               }
```

LinkedList

LinkedList is an implementation of List interface. Implements all optional list operations, and permits all elements (including null). In addition to implementing the List interface, the LinkedList class provides uniformly named methods to get, remove and insert an element at the beginning and end of the list. These operations allow linked lists to be used as a stack, queue, or double-ended queue.

```java
import java.util.*;
public class LinkedListExample {
     public static void main(String args[]) {

         /* Linked List Declaration */
         LinkedList<String> linkedlist = new LinkedList<String>();

         /*add(String Element) is used for adding
          * the elements to the linked list*/
         linkedlist.add("Item1");
         linkedlist.add("Item5");
         linkedlist.add("Item3");
         linkedlist.add("Item6");
         linkedlist.add("Item2");

         /*Display Linked List Content*/
         System.out.println("Linked List Content: " +linkedlist);

         /*Add First and Last Element*/
         linkedlist.addFirst("First Item");
         linkedlist.addLast("Last Item");
         System.out.println("LinkedList Content after addition: " +linkedlist);

         /*This is how to get and set Values*/
         Object firstvar = linkedlist.get(0);
         System.out.println("First element: " +firstvar);
         linkedlist.set(0, "Changed first item");
         Object firstvar2 = linkedlist.get(0);
         System.out.println("First element after update by set method: " +firstvar2);

         /*Remove first and last element*/
         linkedlist.removeFirst();
         linkedlist.removeLast();
         System.out.println("LinkedList after deletion of first and last element: "
+linkedlist);

         /* Add to a Position and remove from a position*/
         linkedlist.add(0, "Newly added item");
         linkedlist.remove(2);
         System.out.println("Final Content: " +linkedlist);
     }
}
```

```
OUTPUT:
Linked List Content: [Item1, Item5, Item3, Item6, Item2]
LinkedList Content after addition: [First Item, Item1, Item5, Item3, Item6, Item2,
Last Item]
First element: First Item
First element after update by set method: Changed first item
LinkedList after deletion of first and last element: [Item1, Item5, Item3, Item6,
Item2]
Final Content: [Newly added item, Item1, Item3, Item6, Item2]
```

HashMap:

HashMap maintains key and value pairs and often denoted as HashMap<Key, Value> or HashMap<K, V>. HashMap implements Map interface. HashMap is similar to Hashtable with two exceptions – HashMap methods are unsynchornized and it allows null key and null values unlike Hashtable. It is used for maintaining key and value mapping.

It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap. It neither does any kind of sorting to the stored keys and Values. You must need to import java.util.HashMap or its super class in order to use the HashMap class and methods.

```java
public static void main(String args[]) {

      /* This is how to declare HashMap */
      HashMap<Integer, String> hmap = new HashMap<Integer, String>();

      /*Adding elements to HashMap*/
      hmap.put(12, "Chaitanya");
      hmap.put(2, "Rahul");
      hmap.put(7, "Singh");
      hmap.put(49, "Ajeet");
      hmap.put(3, "Anuj");

      /* Display content using Iterator*/
      Set set = hmap.entrySet();
      Iterator iterator = set.iterator();
      while(iterator.hasNext()) {
         Map.Entry mentry = (Map.Entry)iterator.next();
         System.out.print("key is: "+ mentry.getKey() + " & Value is: ");
         System.out.println(mentry.getValue());
      }

      /* Get values based on key*/
      String var= hmap.get(2);
      System.out.println("Value at index 2 is: "+var);
      /* Remove values based on key*/
      hmap.remove(3);
      System.out.println("Map key and values after removal:");
      Set set2 = hmap.entrySet();
      Iterator iterator2 = set2.iterator();
      while(iterator2.hasNext()) {
          Map.Entry mentry2 = (Map.Entry)iterator2.next();
          System.out.print("Key is: "+mentry2.getKey() + " & Value is: ");
          System.out.println(mentry2.getValue());
       }

   }
}
```

OUTPUT:

```
key is: 49 & Value is: Ajeet
key is: 2 & Value is: Rahul
key is: 3 & Value is: Anuj
key is: 7 & Value is: Singh
key is: 12 & Value is: Chaitanya
Value at index 2 is: Rahul
Map key and values after removal:
Key is: 49 & Value is: Ajeet
Key is: 2 & Value is: Rahul
Key is: 7 & Value is: Singh
Key is: 12 & Value is: Chaitanya
```

HashSet:

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class is not synchronized. However it can be synchronized explicitly like this: `Set s = Collections.synchronizedSet(new HashSet(...));`

Points to Note about HashSet:

1. HashSet doesn’t maintain any order, the elements would be returned in any random order.
2. HashSet doesn’t allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.
3. HashSet allows null values however if you insert more than one nulls it would still return only one null value.

4. HashSet is non-synchronized.
5. The iterator returned by this class is fail-fast which means iterator would throw ConcurrentModificationException if HashSet has been modified after creation of iterator, by any means except iterator's own remove method.

```
  public class HashSetExample {
                     public static void main(String args[]) {
                        // HashSet declaration
                        HashSet<String> hset =
                                 new HashSet<String>();

                        // Adding elements to the HashSet
                        hset.add("Apple");
                        hset.add("Mango");
                        hset.add("Grapes");
                        hset.add("Orange");
                        hset.add("Fig");
                        //Addition of duplicate elements
                        hset.add("Apple");
                        hset.add("Mango");
                        //Addition of null values
                        hset.add(null);
                        hset.add(null);

                        //Displaying HashSet elements
                        System.out.println(hset);
                      }

                  }
```

OUTPUT:

```
[null, Mango, Grapes, Apple, Orange, Fig]
```

TreeSet :

TreeSet is similar to HashSet except that it sorts the elements in the ascending order while HashSet doesn't maintain any order. TreeSet allows null element but like HashSet it doesn't allow. Like most of the other collection classes this class is also not synchronized, however it can be synchronized explicitly like this: SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));

```
public class TreeSetExample {
                  public static void main(String args[]) {
                      // TreeSet of String Type
                      TreeSet<String> tset = new TreeSet<String>();

                      // Adding elements to TreeSet<String>
                      tset.add("ABC");
                      tset.add("String");
                      tset.add("Test");
```

```
                tset.add("Pen");
                tset.add("Ink");
                tset.add("Jack");

                //Displaying TreeSet
                System.out.println(tset);

                // TreeSet of Integer Type
                TreeSet<Integer> tset2 = new TreeSet<Integer>();

                // Adding elements to TreeSet<Integer>
                tset2.add(88);
                tset2.add(7);
                tset2.add(101);
                tset2.add(0);
                tset2.add(3);
                tset2.add(222);
                System.out.println(tset2);
           }

        }
```

OUTPUT

```
[ABC, Ink, Jack, Pen, String, Test]
[0, 3, 7, 88, 101, 222]
```