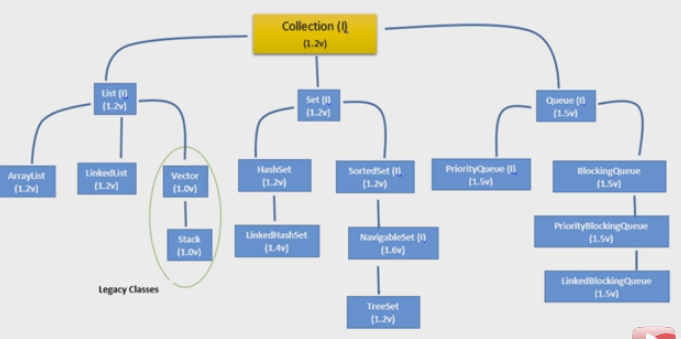
**Collection**



1)group of dissimilar object.size will increase dynamically.

2)collection is a interface but collections is a class.

3)Animal array contains all type of animal like cat,dog,snake ,lizard etc.

4) Root interface in collection hierarchy is Collection interface . Few interviewer may argue that   
Collection interface extends Iterable interface. So iterable should be the root interface. But you should reply iterable interface present in java.lang package not in java.util package .

5)Stack, Properties , Vector,Hashtable and ConurrentHashmap can be used in multi threaded environment because they are synchronized classes (or thread-safe).

**Internal architecture of Arraylist**

1. Whenever we are creating the arraylist object then it will create the object array of size 10. If it is full then it will create the new object array of size =size + (size>>1) and copy all the element of old array to new array(copy reference to new array like shallow copy). Then old array will be garbage collected. So this operation(copy and delete the old array) is overhead.
2. Auto index.
3. Insertion order preserved.
4. duplicate value allowed.
5. Contains null value.
6. Heterogeneous elements are allowed.
7. For retrieval the element we can use for or foreach or iterator.
8. This is fail fast(Iteration and modification concurrently).
9. Works on dynamic array.
10. Implements RandomAccess interface(marker interface) so that we can access any random elements with the same speed.So time complexity of ArrayList is O(1).
11. Implements serializable and cloneable interface because Collections are used to hold and transfer objects from one place to another place through network.
12. Insertion and Deletion is slow because of shifting all the elements. This is fast for searching the elements due to index.
13. Three constructors : new ArrayList(),new ArrayList (int initialCapacity),new ArrayList (Collection c).

**Internal architecture of Vector**

1. Whenever we are creating the Vector object then it will create the object array of size 10. If it is full then it will create the new object array of size =(size\*2) and copy all the element of old array to new array. Then old array will be garbage collected. So this operation(copy and delete the old array) is overhead.
2. It’s implemented on dynamic array.
3. Auto index.
4. Insertion order preserved.
5. duplicate value allowed.
6. Contains null value.
7. Heterogeneous elements are allowed.
8. Implements RandomAccess interface so that we can access any random elements with the same speed.So time complexity of ArrayList is O(1).
9. Implements serializable and cloneable interface because Collections are used to hold and transfer objects from one place to another place through network.
10. Most of the methods are synchronized so we can use in multithreading.
11. Insertion and Deletion is slow because of shifting all the elements. This is fast for searching the elements due to index.
12. Four constructors : new Vector(),new Vector(int initialCapacity),new Vector(int initialCapacity,int incrementalCapacity),new Vector(Collection c).

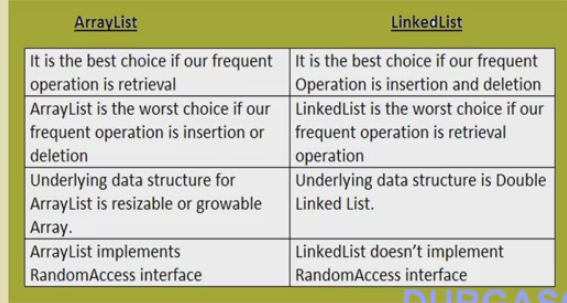
**Internal architecture of LinkedList**

1. It’s implemented on doubly linked list.
2. Default size is 0.
3. Auto index.
4. Insertion order preserved.
5. Insertion order is preserved.
6. duplicate value allowed.
7. Contains null value.
8. Heterogeneous elements are allowed.
9. Not Implements RandomAccess.So time complexity of LinkedList is O(n).
10. Implements serializable and cloneable interface because Collections are used to hold and transfer objects from one place to another place through network.
11. It specifies six specific methods to implement stack and queue that is addFirst(),addLast(),getFirst(),getLast(),removeFirst() and removeLast().
12. Best choice for insertion and deletion in middle.Worst for searching operation.
13. There are two constructors are there.

new LinkedList();new LinkedList(Collection c);

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
|  |  |  |
| Resizable | No | Yes |
|  |  |  |
| Primitives | Yes | No |
|  |  |  |
| Iterating values | for, for each | Iterator , for each |
|  |  |  |
| Length | length variable | size method |
|  |  |  |
| Performance | Fast | Slow in comparision |
|  |  |  |
| Multidimensional | Yes | No |
| Add Elements | Assignment operator | add method |

|  |  |
| --- | --- |
| Arraylist | Linkedlist |
| 1) uses **dynamic array** to store the elements. | uses **doubly linked list** to store the elements. |
| 2)Inserting and deleting the element is slow because all data will shift in memory. | Faster because no bit shifting required in doubly linked list. |
| 3)Act as a list because it implements list only. | Act as a list and queue both because it implements both. |
| 4)Search the data is faster. | 4) Search the data is slower. |
|  |  |
|  |  |

****

3)**Queue-**

**->**FIFO

->not empty space

->index not available.

->duplicate value allowed.

4)**Set-**

->duplicate values not allowed.

->unique value(key of locker).

->index not available.

->union,intersection

5)**Map-**

->is collection of unique key.

->key value paid.

->not related to collection interface.

5)**WeakHashMap**

The key of WeakHashMap has weak reference. If the key has been garbage collected, then the entry in WeakHashMap object will automatically be deleted. It does not happen in normal HashMap. The entry will not be deleted if the key is garbage collected.e.g.

|  |  |  |  |
| --- | --- | --- | --- |
| **List** | **Queue** | **Set** | **MAP** |
| 1)auto index  2)display [].  3)allowed duplicate values  4)dissimilar elements  5)allowed NULL value  6)can use for & foreach loop to access the object.  7)can display member of list through indexwise.  8)add() method return Boolean type(true or false).  9)display according to the index. | 1)not indexed  2)[]  3)allowed  4)similar elements  5)not allowed  6)use only foreach loop and Iterator class.  7)can display in random.  8)same  9)sorting the elements in ascending order & follow FIFO. | 1)not indexed  2)[]  3)not allowed(unique key).  4)dissimilar elements.  5)allowed but tree set not allowed null value.  6)Use only foreach & Iterator class to access the elements.  7)can display in random.  8)same  9)checking the duplicate values & storing. | 1)contain key and data.  2){}  3allowed(but unique key).  4)dissimilar elements.  5)allowed null value but treemap allowed only key1 as a null.  6)through key we can access the elements.  7)according to keyvalue.  8)put() for adding & return null.  7)checking duplicate key &storing in memory. |

|  |  |
| --- | --- |
| **HASHMAP** | **HASHTABLE** |
| 1)not synchronized. | 1)synchronized. |
| 2)permits null value. | 2)not permits null keys and values. |
| 3)Hashmap object values are iterated by using iterator . | 3)uses enumerator to iterate the values of HashTable. |
| 4)faster | 4)slower. |
| 5)Hashmap is a subclass of Map. | 5) Hashtable is a subclass of Dictionary class |
| 3)not contains elements() method. | 3)contains elements() of enumeration type. |
| 4)not contains keys() method. | 4)contains keys() method |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **HASHTABLE** | **CONCURRENT HASHMAP** |
| 1)HashTable uses the same lock for its implementation. | 1)The [ConcurrentHashMap](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html) uses a reader-writer lock to allow parallel reads, and uses segmentation to allow parallel writes. |
| slow, any operation blocks whole Map. So a new operation has to wait for finish of previous one. | Faster than Hashtable, as locks are applied on different segments and any two operations across different segments are parallel. |
| Hashtable lock's home's main door. | ConcurrentHashMap lock's each room instead of main door. |
| multiple threads can not read simultaneously.Reason, the get() method of Hashtable is synchronized. | multiple threads can read simultaneouslyin ConcurrentHashMap by using volatile keyword. |

|  |  |
| --- | --- |
| **HASHMAP** | **CONCURRENT HASHMAP** |
| 1)not synchronized. | 1)synchronized. |
| 2)permits null value. | 2)Not permits null value. |
| 3) faster | 3)slower. |
| 3) HashMap only slightly better in Single threaded environment. | 3) ConcurrentHashMap is more scalable and performs better in multi-threaded environment |
| **ARRAYLIST** | **VECTOR** |
| 1)methods of arraylist is not synchronized and faster. | 1)synchronized and slower. |
| 2)increases by half of its size. | 2)vector doubles the size of its array when its size is increased. |
| 3)not costlier performance. | 3)costlier performance. |
| 4)no default size(10). | 3)it has default size(10). |
| 5)don’t define any increment size. | 4)vector does with setSize() method. |

**ArrayList**

1)a arraylist,vector & linkedList class implements list interface.list interface inherits from collection interface.

2)ArrayList class follow the following features.

->inside ArrayList we can add any types of data.each elements of list are of object types.each elements are auto indexed first element 0 index,second element 1st index etc.

->in ArrayList class,toString() methods are overrided and display the values of arraylist.if values won’t be there then display [] bracket.

->add(Object obj) method takes argument type object is used to add(append) into the list.the methods return Boolean type(true or false).

->Use reverse () method to reverse the list.

Collections.reverse(listobject);

->Use asList() method to convert thearray of strings into the list.

String[] wordArray = {"Love Yourself" , "Alive is Awesome" , "Be in present"};

List wordList = Arrays.asList(wordArray);

->synchronizedCollection() is used to convert from collection to synchronizecollection.

Collections.synchronizedCollection(Collection collectionObj)

->unmodifiableMap() method is used to convert a collection as read only.

Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)//read only

->**Adding**-Arraylist checks the capacity of the ArrayList , before adding the lement. ensureCapacity()  determines what is the current size of occupied elements and what is the maximum size of the array. If size of the  filled elements (including the new element to be added to the ArrayList class) is greater than the  maximum size of the array then increase the size of array. But the size of the array can not be increased dynamically. So what happens internally is new Array is created with capacity  
 int newCapacity = (oldCapacity \* 3)/2 + 1;

->Searching-  
  
also, data from the old array is copied into the new array.

->inside ArrayList we can insert duplicate values & null values.

->while adding a number(primitive data type) of the collection type then primitive type converted into object type using wrapper class.later the converted object is upcasted to object type.

->Collections class provides numerous static methods to perform following operation.

->it doesn’t override getClass(),wait(),notify() methods of object class.

1)searching

2)sorting

3)suffling

4)reversing the elements of collection type.

->most of the classes and interfaces of collection API’s are available in java.util package.

5)it can be synchronized using the java collections framework utility class and then arraylist itself can be used in place of vector.

**CopyOnWriteArrayList**

1)This is a thread-safe collection class as they allow collections to be modified while iterating . The iterator is fail-safe that is it will not throw ConcurrentModificationException.

**VectorList**

1)all are same but vector list contain synchronize method.

**Queue**

1. the priority queue class implements the queue interface(1.5).
2. if we want to represent a group of individual objects prior to processing then we should go for queue.
3. Ex:before sending sms message all mobile nos we have to store in some data structure. in which order we added mobile no in the same order only message should be sent.for ths FiFO requirements queue is the best choice.
4. Usually queue follows FiFO order but based on our requirements we can implement our own priority order also(priority queue).
5. From 1.5 verson onwards linkedilst class also implements queue interface.
6. Linedlist based implementation of queue always follows FiFO order.
7. Queue interface specific methods.

* Offer(o) : add element
* poll() : return and return head element.return null for empty.
* remove() : return and return head element.return excepton(nosuchelementexcepton) for empty.
* peek() : return head element without deletng. return null for empty.
* element() : return head element without deleting. return exception((nosuchelementexcepton)) for empty.

**Priorty Queue**

1. Prior to processing according to some priority then we should go for PQ.
2. Priority can be DNSO or CSO.
3. Duplicates not allowed.
4. Null not allowed also for empty.
5. For DNSO homogeneous and comparable But CSO both object are allowed.
6. IO not preserved that s based on priority(some system is not supporting thread priorities and priority queue).
7. Constructors

* New PO();//11DNSO
* New PO(int intalcapcty);
* New PO(int intalcapcty,Comparator c);
* New PO(SortedSet s);
* New PO(Colecton c);

**BlockingQueue**

1. BlockingQueue implements the java.util.Queue interface . BlockingQueue supports operations that wait for the queue to become non-empty when retrieving an element , and wait for space to become available in the queue when storing an element .
2. It does not accept null elements.
3. Blocking queues are primarily designed for the producer-consumer problems.
4. BlockingQueue implementations are thread-safe and can also be used in inter-thread communications.

**HashMap Class**

1)map is a collection of key value face.inside the map the data or object are added by using keys.key can be any type of object.

2)key should be unique,value can be duplicate.

3)we are adding the data in map through put() method then after adding the data will return **null**.

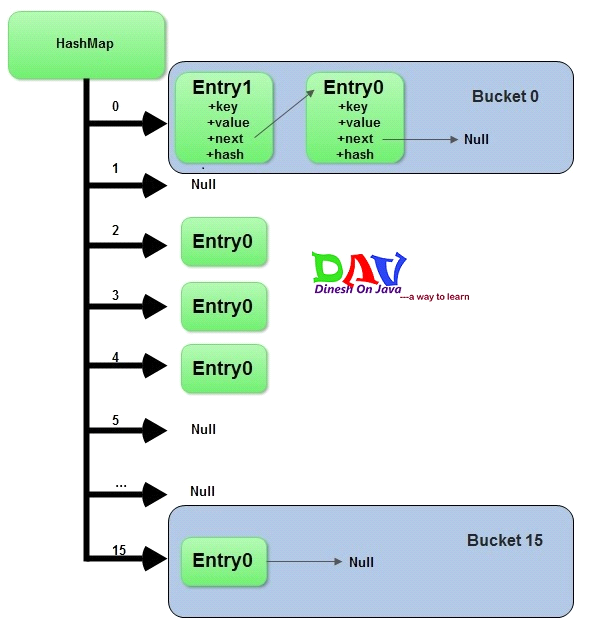
4)toString() method is overrided in such way that ,empty hashmap will display {}.

5)the elements from the map is read with the help of keys.

6)maps are used whenever the data has to be stored with respect to another data.

HashMap hashMap = new HashMap();

Array is gets created with size 16 and default 0.75 load balance.



**Adding a new key-value pair**

* Calculate ***hashcode*** for the key
* Calculate position hash % (arrayLength-1)) where element should be placed(bucket number) if no entry are present there.
* If entry are present(this is a case of collision) then equals method will check the key of both the entry. If key will be same then replace the value with current value otherwise add the entry with previous entry via linked list. But it jdk 8 it will add via balanced tree.
* We have to override equals() and hashcode() method because if I will not override equals() method then it will call the equals() method of the object class and it will do comparison between reference and duplicates will be added in same bucket.
* If we will not override hashcode() then for two key it will give the different hashcode value and it will store into different bucket. Equals() is always called after hashcode() method while adding and removing the elements.
* While retrieving the element hashcode will use to reach desired bucket and equals is used to fetch desired element.
* Key1.equals(key2) =true and key1.hashCode() != key2.hashCode()
* If two two objects are not equal and their hashcode is same that is called collision.
* In one bucket we can keep 2^32 key.

**TreeMap**

1)storing key/value pairs in sorted order.

**ConcurrentHashMap**

1. public ConcurrentHashMap (int initialCapacity, float loadFactor, int concurrencyLevel)

So the above line creates a new, empty map with the specified initial capacity, load factor and concurrency level.

1. where,Important Parameters to consider from ConcurrentHashMap Constructor:

initialCapacity - the initial capacity. The implementation performs internal sizing to accommodate this many elements.

concurrencyLevel - the estimated number of concurrently updating threads. The implementation performs internal sizing to try to accommodate this many threads.

1. In the ConcurrentHashMap Api , you will find the following constants.

static final int DEFAULT\_INITIAL\_CAPACITY = 16;

static final int DEFAULT\_CONCURRENCY\_LEVEL = 16;

initial capacity parameter and concurrency level parameters of ConcurrentHashMap constructor (or Object) are set to 16 by default.

1. Thus, instead of a map wide lock, ConcurrentHashMap maintains a list of 16 locks by default ( number of locks equal to the initial capacity , which is by default 16) each of which is used to lock on a single bucket of the Map.This indicates that 16 threads (number of threads equal to the concurrency level , which is by default 16) can modify the collection at the same time , given ,each thread works on different bucket. So unlike hashtable, we perform any sort of operation ( update ,delete ,read ,create) without locking on entire map in ConcurrentHashMap.

**IdentityHashMap**

1)IdentityHashMap is a class present in java.util package. It implements the Map interface with a hash table , using reference equality instead of object equality when comparing keys and values.In other words , in IdentityHashMap two keys k1 and k2 are considered equal if only if (k1==k2).

2)IdentityHashMap is not synchronized.

**HashSet Class**

1)set is a collection of unique elements.

2)it is not indexed collection.

3)we can add only one null value.

4)whenever set is printed, the elements are displayed in random order.

5)in order to iterator each element of the set an iterator object is used.

**EnumSet**

1)EnumSet is a specialized Set implementation for use with enum types. All of the elements in an enum set must come from a single enum type that is specified explicitly or implicitly , when the set is created.

2)The iterator never throws ConcurrentModificationException and is weakly consistent.

3)Advantage over HashSet:

All basic operations of EnumSet execute in constant time . It is most likely to be much faster than HashSet counterparts.

**Iterator interface**

consist of following three abstract method.

->next(),hashnext(),remove()

->next() method iterator moves to next() element.

->hashNext() method returns true or false based on the next object.if next element is present then returns true otherwise false.

->remove() method is used to delete the object from the set.

**ListIterator interface**

1)used for move the pointer forward & backward in both direction to get both elements.we can use in list.

2)contains five methods.

->next(),hashNext(),remove(),hasPrevious(),previous(),remove(),set(),add()

3)The main difference between Iterator and Enumeration is that Iterator has remove() method while Enumeration doesn't.Hence , using Iterator we can manipulate objects by adding and removing the objects from the collections. Enumeration behaves like a read only interface as it can only traverse the objects and fetch it .

4)UnsupportedOperationException will be thrown when the requested operation is not supported.if you call add() or remove() method on the readOnly collection . We know readOnly collection can not be modified . Hence , UnsupportedOperationException will be thrown.

**Diff between Fail Fast and Fail safe Iterator**

|  |  |  |
| --- | --- | --- |
|  | FailFast | Fail safe |
| Throw ConcurrentModification Exception | Yes | No |
| Clone object | No | Yes |
| Memory Overhead | No | Yes |
| Examples | ArrayList,Vector,HashSet HashMap | CopyOnWriteArrayList,  ConcurrentHashMap |

* Dictionary is an abstract class that represents a key/value storage repository and operates much like Map.
* Given a key and value, you can store the value in a Dictionary object. Once the value is stored, you can retrieve it by using its key. Thus, like a map, a dictionary can be thought of as a list of key/value pairs.
* The abstract methods defined by Dictionary are listed below −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **Enumeration elements( )**  Returns an enumeration of the values contained in the dictionary. |
| 2 | **Object get(Object key)**  Returns the object that contains the value associated with the key. If the key is not in the dictionary, a null object is returned. |
| 3 | **boolean isEmpty( )**  Returns true if the dictionary is empty, and returns false if it contains at least one key. |
| 4 | **Enumeration keys( )**  Returns an enumeration of the keys contained in the dictionary. |
| 5 | **Object put(Object key, Object value)**  Inserts a key and its value into the dictionary. Returns null if the key is not already in the dictionary; returns the previous value associated with the key if the key is already in the dictionary. |
| 6 | **Object remove(Object key)**  Removes the key and its value. Returns the value associated with the key. If the key is not in the dictionary, a null is returned. |
| 7 | **int size( )**  Returns the number of entries in the dictionary. |

**Hashtable**

if two different keys have the same hash value then it lead to hash -collision. A bucket of type linkedlist used to hold the different keys of same hash value.

**Need of Collection Framework**

1. fixed number of homogeneous data elements.
2. we can represent multiple values with a single variable.
3. So that reusability of the codewill be improved.

**Limitations of Object type Arrays:**  
  
1) fixed in size i.e. once we created an array with some size there is no chance of increasing its size based on our requirement. Hence to use arrays compulsory we should know the size in advance which may not possible always.

2) Arrays can hold only homogeneous data elements

**Example:**  
  
Student[] s=new Student[10000];  
s[0]=new Student; (correct)  
s[1]=new Customer(); (wrong)

But We can resolve this problem by using object Arrays.  
Object[] o=new Object[10000];  
o[0]=new Student();  
o[1]=new Customer();

1. No standard data structure hence readymade method support is not available for every requirement we have to write the code explicitly. Which is complexity of programming.

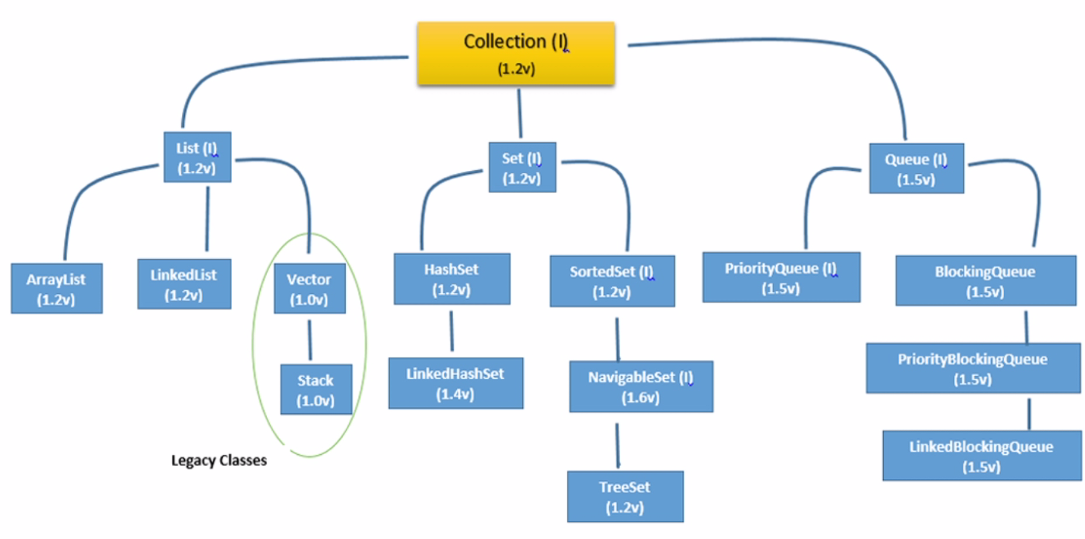
To overcome the above limitations of Arrays we should go for Collections.

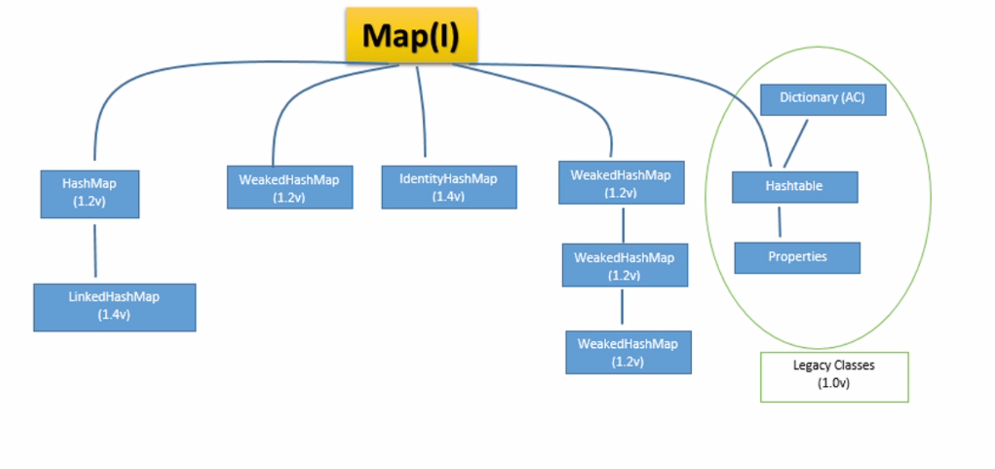
1. Collections are growbable in nature. i.e. Based on our requirment we can increase (or) Decrease the size.
2. Hold both homogeneous & Heterogeneous elements.
3. Implemented based on some standard data structure. Hence readymade method support is available for every requirement. Being a programmer we have to use this method and we are not responsible to provide implementation.

**Difference between Arrays and Collections?**

|  |  |
| --- | --- |
| **Arrays** | **Collections** |
| **1.** Arrays are fixed in size. | **1.**Collections are growable in nature. i.e. based on our  requirement we can increase or decrease the size. |
| **2.** Wrt memory arrays are not recommended to use. | **2.** Wrt to memory collections are recommended to use. |
| **3.** Wrt Performance Arrays are recommended to use. | **3.** Wrt Performance Collections are not recommended to use. |
| **4.** Array can hold only homogeneous datatype eleements. | **4.**Collections can hold both homogeneous and heterogeneous  elements. |
| **5.**There is no underlying data structure for arrays and hence readymade method support is not available. | **5.** Every Collections class is implemented based on some  standard data structure. Hence readymade method support  is available for every requirement. |
| **6.** Array can hold both primitives and object types | **6.** Collections can hold only objects but not primitives. |

**Overview of Collection framework**





**Sorting:**  
  
1. Comparable(I)  
2. Comparator (I)   
  
**Cursors:**  
  
1.Enumeration(I)   
2. Iterator(I)   
3. ListIterator(I)  
  
**Utility Classes:**  
  
1. Collections   
2. Arrays.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| add(o) | addAll(c) |  |  |
| remove(o) | rAll(c) |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| add(i,o) | addAll((I,o) |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |