Introduction:

1. An array is an indexed collection of fixed no of homogeneous data elements. (or)
2. An array represents a group of elements of same data type.
3. The main advantage of array is we can represent huge no of elements by using single variable. So that readability of the code will be improved.

Limitations of Object[] array:

1. Arrays are fixed in size that is once we created an array there is no chance of increasing (or) decreasing the size based on our requirement hence to use arrays concept 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();//valid
```

```
s[1]=new Customer();//invalid(compile time error)
```

Compile time error:

```
Test.java:7: cannot find symbol
Symbol: class Customer
Location: class Test
s[1]=new Customer();
```

3) But we can resolve this problem by using object type array(Object[]).

Example:

```
Object[] o=new Object[10000];
o[0]=new Student();
o[1]=new Customer();
```

4) Arrays concept is not implemented based on some data structure hence ready-made methods support we can't expert. For every requirement we have to write the code explicitly.

To overcome the above limitations we should go for collections concept.

1. **Collections are growable in nature that is based on our requirement we can increase (or) decrease the size hence memory point of view collections concept is recommended to use.**
2. **Collections can hold both homogeneous and heterogeneous objects.**
3. **Every collection class is implemented based on some standard data structure hence for every requirement ready-made method support is available being a programmer we can use these methods directly without writing the functionality on our own.**

Differences between Arrays and Collections ?

Arrays	Collections
1) Arrays are fixed in size.	1) Collections are growable in nature.
2) Memory point of view arrays are not recommended to use.	2) Memory point of view collections are highly recommended to use.
3) Performance point of view arrays are recommended to use.	3) Performance point of view collections are not recommended to use.
4) Arrays can hold only homogeneous data type elements.	4) Collections can hold both homogeneous and heterogeneous elements.
5) There is no underlying data structure for arrays and hence there is no	5) Every collection class is implemented based on some standard data structure and hence

readymade method support.	readymade method support is available.
6) Arrays can hold both primitives and object types.	6) Collections can hold only objects but not primitives.

Collection:

If we want to represent a group of objects as single entity then we should go for collections.

Collection framework:

It defines several classes and interfaces to represent a group of objects as a single entity.

Java	C++
Collection	Containers
Collection framework	STL(Standard Template Library)

9(Nine) key interfaces of collection framework:

1. Collection
2. List
3. Set
4. SortedSet
5. NavigableSet
6. Queue
7. Map
8. SortedMap
9. NavigableMap

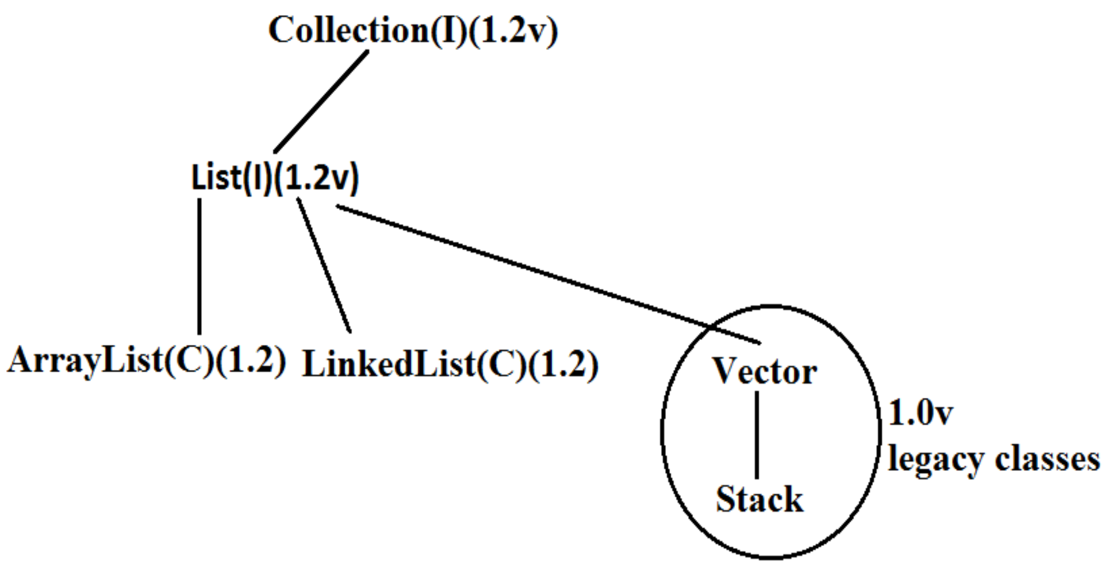
Collection:

1. If we want to represent a group of "individual objects" as a single entity then we should go for collection.
2. In general we can consider collection as root interface of entire collection framework.
3. Collection interface defines the most common methods which can be applicable for any collection object.
4. There is no concrete class which implements Collection interface directly.

List:

1. It is the child interface of Collection.
2. If we want to represent a group of individual objects as a single entity where "duplicates are allow and insertion order must be preserved" then we should go for List interface.

Diagram:

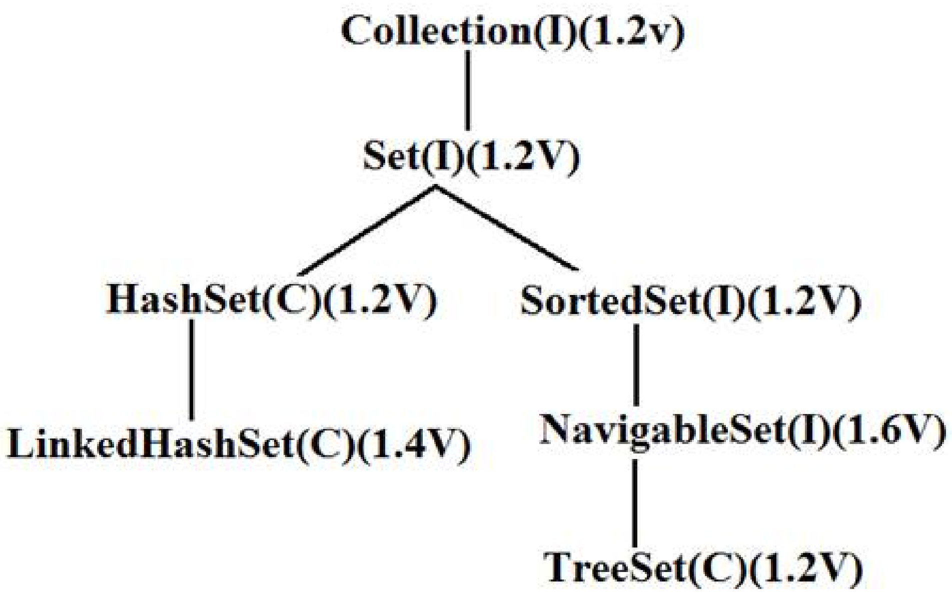


Vector and Stack classes are re-engineered in 1.2 versions to implement List interface.

Set:

1. It is the child interface of Collection.
2. If we want to represent a group of individual objects as single entity "where duplicates are not allow and insertion order is not preserved" then we should go for Set interface.

Diagram:



SortedSet:

1. It is the child interface of Set.
2. If we want to represent a group of individual objects as single entity "where duplicates are not allow but all objects will be insertion according to some sorting order then we should go for SortedSet.
   (or)
3. If we want to represent a group of "unique objects" according to some sorting order then we should go for SortedSet.

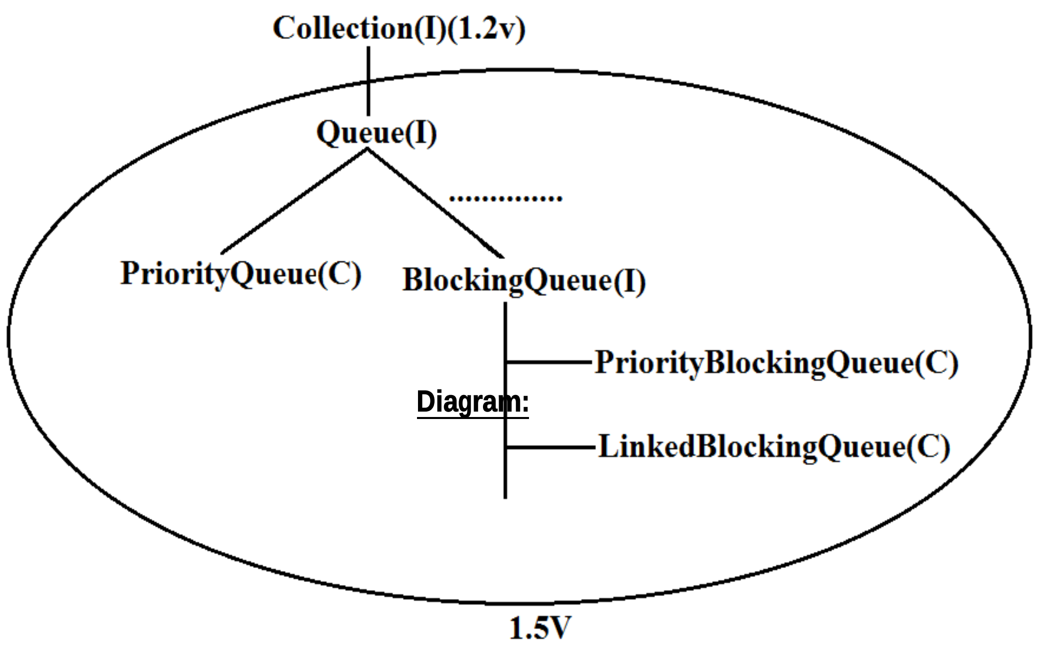
NavigableSet:

1. It is the child interface of SortedSet.
2. It provides several methods for navigation purposes.

Queue:

1. It is the child interface of Collection.
2. If we want to represent a group of individual objects prior to processing then we should go for queue concept.

Diagram:

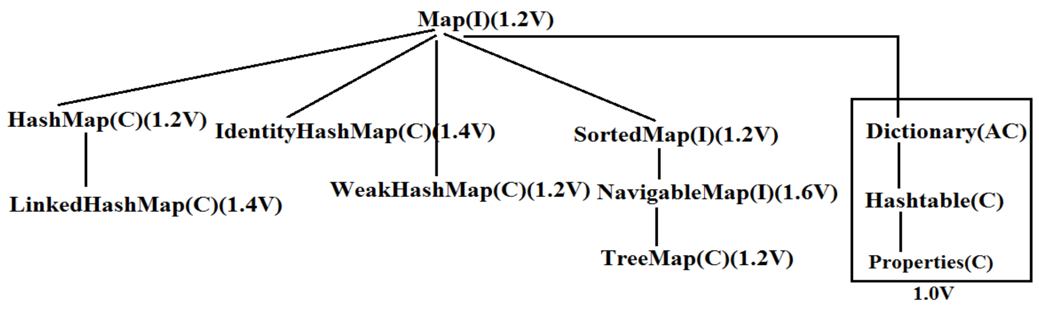


Note: **All the above interfaces (Collection, List, Set, SortedSet, NavigableSet, and Queue) meant for representing a group of individual objects.**
If we want to represent a group of objects as key-value pairs then we should go for Map.

Map:

1. **Map is not child interface of Collection.**
2. **If we want to represent a group of objects as key-value pairs then we should go for Map interface.**
3. **Duplicate keys are not allowed but values can be duplicated.**

Diagram:



SortedMap:

1. It is the child interface of Map.
2. If we want to represent a group of objects as key value pairs "according to some sorting order of keys" then we should go for SortedMap.

NavigableMap:

1) It is the child interface of SortedMap and defines several methods for navigation purposes.

<u>What is the difference between Collection and Collections ?</u>

"Collection is an "interface" which can be used to represent a group of objects as a single entity. Whereas "Collections is an utility class" present in java.util package to define several utility methods for Collection objects.

Collection-------------------interface
Collections-----------------class

In collection framework the following are legacy characters.

1. Enumeration(I)
2. Dictionary(AC)
3. Vector(C)
4. Stack(C)
5. Hashtable(C)
6. Properties(C)

<u>Diagram:</u>

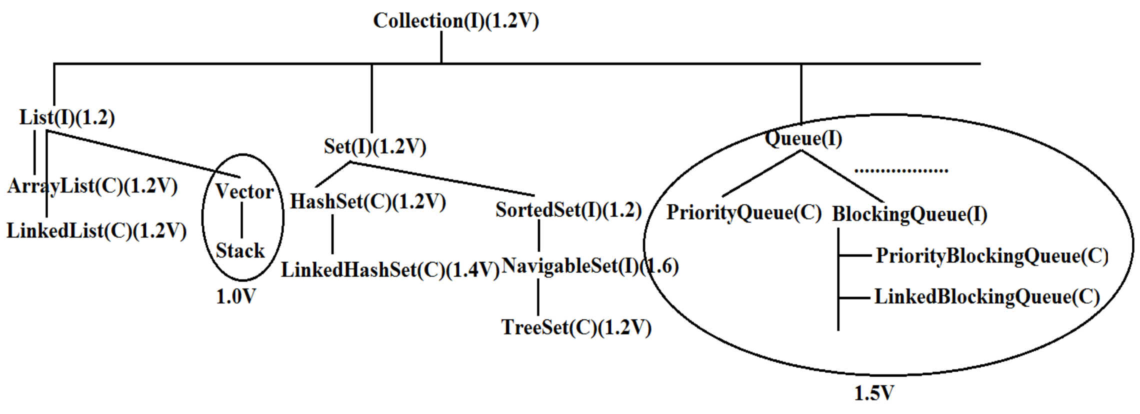
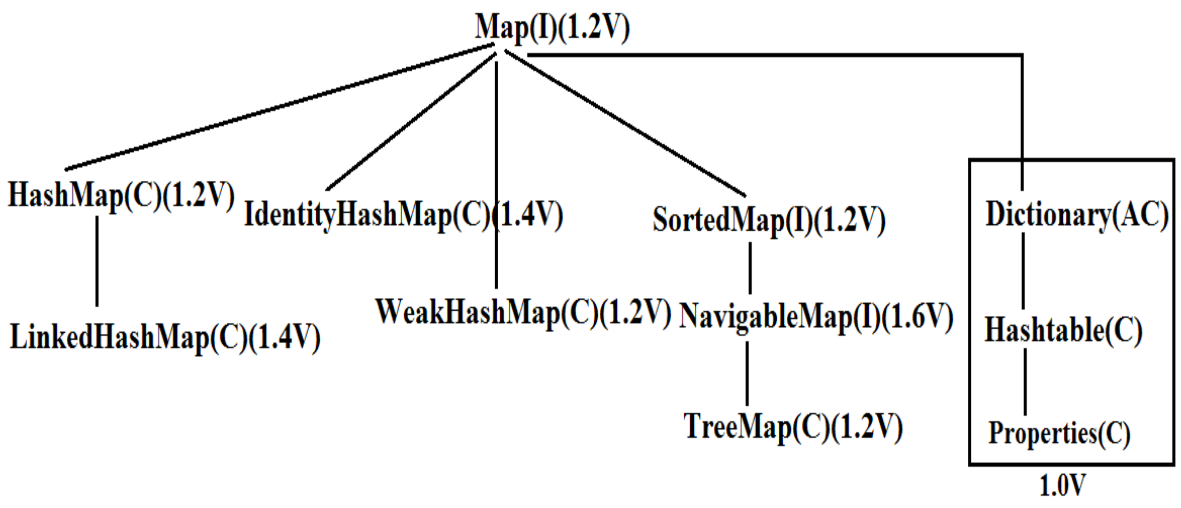


Diagram:



Collection interface:

- If we want to represent a group of individual objects as a single entity then we should go for Collection interface. This interface defines the most common general methods which can be applicable for any Collection object.
- The following is the list of methods present in Collection interface.
    1. boolean add(Object o);
    2. boolean addAll(Collection c);
    3. boolean remove(Object o);
    4. boolean removeAll(Object o);
    5. boolean retainAll(Collection c);
       To remove all objects except those present in c.
    6. Void clear();
    7. boolean contains(Object o);
    8. boolean containsAll(Collection c);
    9. boolean isEmpty();
    10. Int size();
    11. Object[] toArray();
    12. Iterator iterator();

There is no concrete class which implements Collection interface directly.

List interface:

- It is the child interface of Collection.
- If we want to represent a group of individual objects as a single entity where duplicates are allow and insertion order is preserved. Then we should go for List.
- We can differentiate duplicate objects and we can maintain insertion order by means of index hence "index play very important role in List".

List interface defines the following specific methods.

1. **boolean add(int index,Object o);**
2. **boolean addAll(int index,Collectio c);**
3. **Object get(int index);**
4. **Object remove(int index);**
5. **Object set(int index,Object new);//to replace**
6. **Int indexOf(Object o);**
   Returns index of first occurrence of "o".
7. **Int lastIndexOf(Object o);**
8. **ListIterator listIterator();**

ArrayList:

1. **The underlying data structure is resizable array (or) growable array.**
2. **Duplicate objects are allowed.**
3. **Insertion order preserved.**
4. **Heterogeneous objects are allowed.(except TreeSet , TreeMap every where heterogenious objects are allowed)**
5. **Null insertion is possible.**

Constructors:

1) ArrayList a=new ArrayList();

Creates an empty ArrayList object with default initial capacity "10" if ArrayList reaches its max capacity then a new ArrayList object will be created with

New capacity=(current capacity*3/2)+1

2) ArrayList a=new ArrayList(int initialcapacity);

Creates an empty ArrayList object with the specified initial capacity.

3) ArrayList a=new ArrayList(collection c);

Creates an equivalent ArrayList object for the given Collection that is this constructor meant for inter conversation between collection objects. That is to dance between collection objects.

Demo program for ArrayList:

```
import java.util.*;
class ArrayListDemo
{
      public static void main(String[] args)
      {
            ArrayList a=new ArrayList();
            a.add("A");
            a.add(10);
            a.add("A");
            a.add(null);
            System.out.println(a);//[A, 10, A, null]
            a.remove(2);
            System.out.println(a);//[A, 10, null]
            a.add(2,"m");
            a.add("n");
            System.out.println(a);//[A, 10, m, null, n]
            }
}
```

- **Usually we can use collection to hold and transfer objects from one tier to another tier. To provide support for this requirement every Collection class already implements Serializable and Cloneable interfaces.**
- **ArrayList and Vector classes implements RandomAccess interface so that any random element we can access with the same speed. Hence ArrayList is the best choice of "retrival operation".**
- **RandomAccess interface present in util package and doesn't contain any methods. It is a marker interface.**

Example :

```
ArrayList a1=new ArrayList();
LinkedList a2=new LinkedList();

System.out.println(a1 instanceof Serializable ); //true
System.out.println(a2 instanceof Clonable); //true

System.out.println(a1 instanceof RandomAccess); //true
System.out.println(a2 instanceof RandomAccess); //false
```

<u>Differences between ArrayList and Vector ?</u>

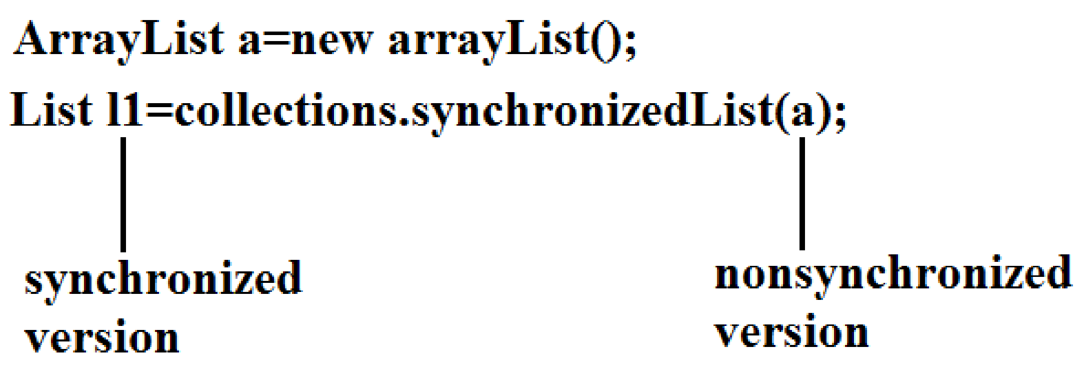
ArrayList	**Vector**
1) No method is synchronized	**1) Every method is synchronized**
2) At a time multiple Threads are allow to operate on ArrayList object and hence ArrayList object is not Thread safe.	**2) At a time only one Thread is allow to operate on Vector object and hence Vector object is Thread safe.**
3) Relatively performance is high because Threads are not required to wait.	**3) Relatively performance is low because Threads are required to wait.**

4) It is non legacy and introduced in 1.2v	4) It is legacy and introduced in 1.0v

Getting synchronized version of ArrayList object:

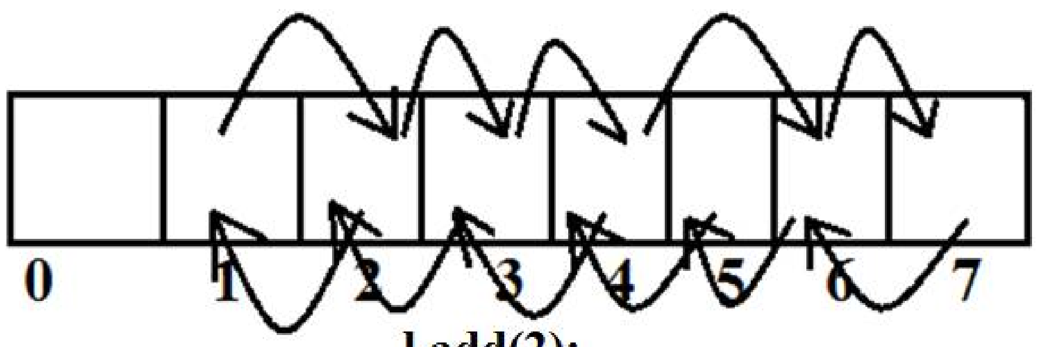
- Collections class defines the following method to return synchronized version of List.
  Public static List synchronizedList(list l);

- Example:



```
ArrayList a=new arrayList();
List l1=collections.synchronizedList(a);
```

- Similarly we can get synchronized version of Set and Map objects by using the following methods.
  1) public static Set synchronizedSet(Set s);
  2) public static Map synchronizedMap(Map m);
- ArrayList is the best choice if our frequent operation is retrieval.
- ArrayList is the worst choice if our frequent operation is insertion (or) deletion in the middle because it requires several internal shift operations.
  Diagram:



```
l.add(2);
l.remove(3);
```

LinkedList: