**Collections framework**

An array is an indexed collection of fixed number of homogenous data elements.  
The main advantages of arrays are we can represent multiple values with single variables.

**Limitation of Objects type Arrays:**

Arrays are fixed in size i.e. once we created an array with some size there is no change of increasing or decreasing its size based on our requirement. Hence to use arrays compulsory we should know the size in advance which may not possible always.

Arrays can hold only homogenous data elements.  
Ex:   
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();

Arrays concept is not implemented based on some 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.**Collections are growable in nature. i.e. Based on our requirement we can increase or decrease the size.  
Collections can hold both homogeneous and heterogeneous elements.

Heterogeneous means different types of objects.  
  
Each collection class is 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. |
| 1. Write memory arrays are not recommended to use. | 1. Write to memory collections are recommended to use. |
| 1. Write performance arrays are recommended to use | 1. Write performance collections are not recommended to use. |
| 1. Arrays can hold only homogeneous datatype elements. | 1. Collections can hold both homogeneous and heterogeneous elements. |
| 1. There is no underlying data structure for arrays and hence readymade method support is not available. | 1. Every collection class is implemented based on some standard data structure Hence readymade method support is available for every requirement. |
| 1. Arrays can hold both primitives and objects types | 1. Collections can hold only objects but not primitives. |

Collection is a group of individual objects as a single entity.

**What is collection?**If we want to represent a group of individual objects as a single entity then we should go for collection.

**What is collection framework?**It defines several classes and interfaces which can be used a group of objects as single entity.  
  
**9 – Key Interfaces of collection Framework**1) **Collection:**

* if we want to represent a group of individual’s objects as a single entity then we should go for collection.
* Collection interfaces defines the most common methods which are applicable for any collection object.
* In general collection interface is considered as root interface of collection framework.

**Note**: There is no concrete class which implements collection interface directly.

**Difference between collection and collections.**

1. Collection is an interface which can be used to represent a group of individual objects as a single entity.
2. Collections is an utility class present in java.util.package to define several utility methods ( like Sorting, Searching) for collection objects.
3. **list**
4. List is child interface of collection.
5. If we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order preserved then we should go for list.

Diagram:

|  |
| --- |
| Collection(1.2 v) |
| List (1.2) |
| 1. ArrayList (1.2) 2) LinkedList 3) VectorList |
| Under vector list 1) Vector (1.0) |
| Stack (1.0) |

1. **Set:**
2. It is the child interface of collection.
3. If we want to represent a group of individual objects as a single entity where duplicates are not allowed and insertion order not preserved then we should go for set.

**Diagram:**

|  |
| --- |
| Collection (1.2) |
| Set (1.2) |
| HashSet(1.2) |
| LinkedHashSet(1.4) |

**Difference between List & Set**

|  |  |
| --- | --- |
| **List** | **Set** |
| Duplicates are allowed | Duplicates are not allowed |
| Insertion order preserved | Insertion order not preserved. |

**4. SortedSet:**

1) It is the child interface of set.  
2) If we want to represent a group of individual objects as a single entity where duplicates are not allowed but all objects should be inserted according to some sorting order then we should go for SortedSet.

**5. NavigableSet:**

1) It is the child interface of SortedSet if defines several methods for navigation purposes.

Diagram:

|  |
| --- |
| Collection (1.2 v) |
| Set (1.2 v) |
| SortedSet (1.2 v) |
| NavigableSet(1.6 v) |
| TreeSet(1.2 v) |

**6. Queue:**1) It is child interface of collection.  
2) If we want to represent a group of individual objects prior to processing then we should go for Queue.  
  
ex: Before sending a mail all mail id’s we have to store somewhere and in which order we saved in the same order mail’s should be delivered (First in First out) for this requirement Queue concept is the best choice.

|  |
| --- |
| Collection |
| Queue (1.5 v) |
| 1. Priority Queue 2) Blocking Queue (1.5 v) |
| 2 .1)LinkedBlockingQueue (1.5 v) |
| 2.2) PriorityBlockingQueue (1.5 v) |

**Note:**

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

**7. Map** 1) Map is not the child interface of collection  
 2) if we want to represent a group of individual objects as key value pairs then should go for Map.   
Ex:

|  |  |
| --- | --- |
| Roll No | Name |
| 101 | Ganesh |
| 102 | Ravi |
| 103 | Ravi |

Both key and value are objects, duplicated keys are not allowed but values can be duplicated.

**Diagram:**

|  |
| --- |
| **Map (1.2)** |
| 1. HashMap (1.2) 2) WeakHashMap (1.2) 3)IdentityHashMap (1.4) 4) HashTable child of Dictionary (1.0) |
| 1. LinkedHashMap child of HashMap (1.4) |
| 1. Properties child of Hashtable (1.0) |

**8. Sorted Map**1) It is the child interface of map.  
2) If we want to represent a group of key value pairs according to some sorting order of keys then we should go for SortedMap.

**Diagram :**

|  |
| --- |
| Map (1.2) |
| SortedMap(1.2) |

**9. Navigable Map**

1) It is the child interface of sorted map; it defines several utlity methods for navigation purpose.

|  |
| --- |
| Map (1.2) |
| SortedMap (1.2) |
| Navigable Map (1.6) |
| TreeMap(1.2) |

1. **Collection Interfaces:**
2. If we want to represent a group of individual objects as a single entity then we should go for collection.
3. In general collection interface is considered as root interface of collection framework.
4. Collection interface defines the most common methods which are applicable for any collection object.

**Importance methods of collection interface**

1. Boolean add(Object o)
2. Boolean addAll(Collection C)
3. Boolean remove (Object o)
4. Boolean removeAll ( Collection c)
5. Boolean retainAll( Collection 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()

**Note:** Collection interface doesn’t contain any method to retrieve objects there is no concrete class which implements collection class directly.

1. **List Interface:**
2. It is the child interface of collection.
3. If we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order must be preserved then we should go for list.
4. We can differentiate duplicates by using index.
5. We can preserve insertion order by using index, hence index play very important role in list interface.

**List interface specific methods**

Void add (int index, Object o)  
Boolean addAll(int index, Collection c)  
object get(int index)  
object remove(int index)  
object set(int index, Object new)  
int IndexOf(Object o)  
int lastIndexOf(Object o)  
ListIterator listIterator();

**ArrayList:** 1) The underlined data structure resizable array or growable array.  
2) Duplicates are allowed.  
3) Insertion order is preserved.  
4) Heterogeneous objects are allowed [expect TreeSet & TreeMap everywhere heterogeneous objects are allowed]  
5) Null insertion is possible.  
  
**ArrayList Constructor**1) ArrayList al = new ArrayList()  
Creates an empty ArrayList object with default initial capacity 10. Once Array List reaches its map capacity a new ArrayList will be created with new capacity = ( currenctCapacity \*3/2)+1  
2) ArrayList al = new ArrayList(int initialCapacity);  
3) ArrayList al = new ArrayList(Collection c);  
  
ExCode:  
**import** java.util.\*;

**class** arrayList {

**public** **static** **void** main(String[] args) {

ArrayList l = **new** ArrayList();

l.add("A");

l.add(10);

l.add("A");

l.add(**null**);

System.***out***.println(l);

l.remove(2);

System.***out***.println(l);

l.add(2,"Ganesh");

l.add("Raviraj");

System.***out***.println(l);

}

}

**Note:**

1. Usually we can use Collections to hold and transfer Objects from one to place to another place, to provide support for this requirement every collection already implements serialization and cloneable interfaces.
2. ArrayList and Vector classes implements RandomAccess Interface so that we can access any Random elements with the same speed.
3. Hence if our frequent operation is retrieval operation then ArrayList is the best choice.

**RandomAccess:**

1. Present in java.util package.
2. It doesn’t contain any methods and it is a Maker interface.

**ArrayList:**

ArrayList l1 = new ArrayList();  
LinkedList l2 = new LinkedList();  
System.out.println(l1 instance of Serializable) True  
System.out.println(l2 instance of Cloneable) True  
System.out.println(l1 instance of RamdomAccess) True  
System.out.println(l2 instance of RamdomAccess) false

**ArrayList:**

1. ArrayList is best choice if our frequent operation is retrieval operation (Because ArrayList implements RandomAccess interfaces)
2. ArrayList is the worst choice if our frequent operation is insertion or deletion in the middle (Because several shift operation are require)

**Difference between ArrayList & Vector**

|  |  |
| --- | --- |
| ArrayList | Vector |
| 1. Every method present arraylist is non-synchronize | 1. Every method present in LinkedList is synchronize |
| 1. At a time multiple threads are allowed to operate on arrayList Object and hence ArrayList is not thread safe | 1. At a time only one thread is allowed to operate on Vector Object is thread safe. |
| 1. Thread are not required to wait to operate on ArrayList, Hence relatively performance is high | 1. Threads are required to wait to operate on Vector Object and hence relatively performance is low. |
| 1. Introduced in 1.2 version and it is non-legacy class | 1. Introduced 1.0 version and it is a legacy class. |

**How to get synchronized version of arrayList Object?**

1. By default ArrayList is object is non-synchronized but we can get synchronized version of ArrayList by using Collection class synchronizedList () method.

Public static List synchronizedList(List l)

Non-Synchronized  
 ArrayList l1 = new ArrayList();

Synchronized  
 List l = Collections.synchronizedList (l1);

\* Similarly we can get Synchronized version of Set, Map Objects by using the following methods of Collection class.

Public static Set synchronizedSet (Set s);  
 Public static Set synchronizedMap (Map m);