--------------------------------------------------------------------------------------------------------------------------------------

Need for Collections:  
-Due to limitations of arrays.  
1. Arrays are fixed length, so no flexibility.  
2. Arrays can hold only homogeneous elements.  
3. No underlying data structure concept, so we can’t expect readymade method support can’t be expected. (like add(), isEmpty()).

--------------------------------------------------------------------------------------------------------------------------------------

Array concept is more recommended to use when the size is known beforehand.  
This is because the cost of functionality and memory-flexibility comes at the cost of performance.  
Consider the example of an array and arraylist(AL) of size 10.  
To perform insertion, the elements are simply inserted into the cells in arrays. (Cannot insert beyond max capacity).  
For AL, once max capacity is exceeded, another AL of bigger size is created, all elements from original AL is copied into new AL, new element is inserted at the end, then reference variable is reassigned to new AL, and finally, old AL goes to garbage collection. This asks a lot of the performance, especially when the number of objects in AL is very large.

|  |  |
| --- | --- |
| **Arrays** | **Collections** |
| Fixed size | Growable in nature |
| Not recommended for memory usage | Recommended for memory usage |
| Recommended for performance usage | Not recommended for performance usage |
| Can only support homogeneous elements | Can support homo and heterogeneous elements |
| No underlying data structures for arrays | All collections are based on some Data structures concept. |

--------------------------------------------------------------------------------------------------------------------------------------

**We go for Collection to represent a group of individual objects as a single entity. In order to do this, several Classes and Interfaces are required. These Classes and Interfaces are defined inside Collection Framework.**In java, we have Collection and Collections Framework. Correspondingly, in c++ we have Container and STL (Standard Template Library).

--------------------------------------------------------------------------------------------------------------------------------------

9 key interfaces :

1. Collection interface ->

If we want to represent group of individual objects as single entity, go for Collection. It defines most common methods (add, remove) which are applicable for any Collection object.   
It is the root interface of Collection framework.  
Collection vs Collections:

Collection is an interface used to represent individual objects as single entity. Collections is a utility class which defines several methods for collection object, like Collections.sort().

1. List ->

List is an interface which is used when duplicates are allowed, and insertion order must be preserved.  
List is child interface of Collection interface.

Implementation classes for List are:

1. Arraylist
2. Linkedlist
3. Vector

Collection, List, Arraylist and Linkedlist came in 1.2 Version.  
Vector and Stack came in 1.0 Version. Hence they are called Legacy Classes.

3. Set ->  
 Set is an interface where duplicates are not allowed, and insertion order isn’t preserved.  
 Set is child interface of Collection interface.  
 Implementation classes for Set are:  
 1. HashSet  
 2. Linked HashSet   
 Set and HashSet came in 1.2 version.  
 Linked HashSet came in 1.4 version

.

List vs Set:

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

4. SortedSet ->  
 In Set, duplicates are not allowed and insertion order isn’t preserved.  
 If we want a Set where insertion order is preserved, then we use SortedSet.  
 SortedSet is child interface of Set.  
 Thus, if we want to represent group of individual objects where duplicates are not allowed but insertion order is preserved, then we use SortedSet.

SortedSet came in 1.2 Version.

5. NavigableSet ->  
 It is a chain interface of SortedSet.  
 It defines several methods for navigation purposes.   
 Implementation class for NavigableSet:  
 1. TreeSet

NavigableSet came in 1.6 Version.  
 TreeSet came in 1.2 Version. (Modified in 1.6 to implement NavigableSet).

6. Queue ->  
 We use Queues when we want to represent group of individual objects prior to processing.  
 It is child interface of Collection.  
It’s an ordered list of objects in which elements are inserted at the end and elements are deleted from start of list, following FIFO principle.

Implementation classes for queue:  
 1. PriorityQueue  
 2. BlockingQueue  
 (i) LinkedBlockingQueue  
 (ii) PriorityBlockingQueue

Whole of Queue concept, including all Implementation classes came in 1.5 Version.

All above interfaces (Collection, List, Set, SortedSet, NavigableSet, Queue) are meant for representing a group of individual objects.  
If we want to represent objects as key-value pairs, we need to go for Map interface.

7. Map ->  
Map is the mirror half of Collection.   
It is not a chain interface of Collection.  
Used when we want to represent objects as key value pairs.  
Example- Roll number-Student pair.   
**Key can not be duplicate, value can.**   
Implementation classes of Map:  
1. HaspMap  
2.Linked HashMap  
3. Weak HashMap  
4. Identity HashMap  
5. HashTable (Child of Dictionary)  
6. Properties  
Map, Hashmap, WeakHashMap came in Version 1.2  
LinkedHashMap, IdentityHashMap came in Version 1.4  
Dictionary, HashTable, Properties came inversion 1.0. Thus, Legacy classes

8. SortedMap ->  
Represent key value pairs according to some sorting order.   
It’s a child interface of Map.

9. NavigableMap ->  
Child interface of SortedMap.  
Defines several utility methods for SortedMaps.  
Implementation class of NavigableMap:  
1. TreeMap  
Map, SortedMap and TreeMap came in 1.2 Version.  
NavigableMap came in 1.6 Version.