Need of Collections :
=====================
An array is an indexed colletions of fixed number of homogeneous data elements.
The main advantage of Arrays is we can represent multiple values with a single variable
so that reusability of the code will be improved.

Limitation of Object type arrays :
Arrays are fixed in size i.e. once we created an array with some size there is no chance of increasing
or decreasing it's 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 homogeeous data elements
ex., Student[] s = new Student[100];
s[0] = new Student(); //correct
s[1] = new Customer(); //wrong

But we can resolve this problem by using objects Arrays.
Object[] o = new Object[100];
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 explicity.
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 our requirement we can increase or decrease the size.
- Collections can hold both homogeneous and heterogenous elements.
- Every collections 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 collection :
==========

Arrays :
1) Arrays are fixed in size
2) With respect to memory arrays are not recommended to use.
3) With respect to performance arrays are recommended to use.
4) Arrays can hold only homogeneous data type elements.
5) There is no underlying data structure for arrays and hence readymade method support is not available.

6) Arrays can hold both primitive and object types.

Collections :
1) Collections are growable in nature i.ie. based on our requirement we can increase or decrease the size
2) With respect to memory collections are recommended to use.
3) With respect to performance collections are not recommended to use.
4) Collections can hold both homogeneous and heterogeneous data type elements.
5) Every collections class is implemented based on some standard data structure. Hence readymade method support is availble for every requirement.
6) Collections can hold only objects but not primitive data types.

==========

Collection :
==========
    If we want to represent a group of individual objects as a single entity then we should go for collection.

Collection Framework :
==========
    It defines several classes and interfaces which can be used a group of objects as single entity.

Java	C++
Collection	Container
Collection Framework	STL(Standard Template Library)

Difference between collection and collections :
==========
- Collection is an interface which can be used to represent a group of individual objects as a single entity.
- Collections is an utility class present in java.util.package to define several utility methods(like sorting, searching) for collection objects.

==========

9 Key interfaces of collection framework :
==========
1) Collection (1.2v):
- If we want to represent a group of individual objects as a single entity then we should go for collections.
- Collection interface 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.
- Note : It doesn't contain any method to retrive objects. There is no concrete class which implements collection

class directly.
- Ususally we can use Collection to hold and transfer objects from one place to another place, to provide support for
this requirement every Collection already implements Serializable and Cloneable interfaces.

****
****

2) List (1.2v) (implemented classes : ArrayList-1.2v, LinkedList-1.2v, Vector->Stack-1.0v legacy classes):
- It is a child interface of collection.
- 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.
- we can differentiate duplicates by using index.
- we can preserve insertion order by using index, hence index play very important role in list interface.
- ArrayList and Vector classes implements RandomAccess(It doesn't contain any method & It's a Marker interface. It present in java.util package) interface so that we can access any Random element with the same speed.
- Hence if our frequent operation retrival operation the ArrayList is the best choice.

i) ArrayList (c-1.2v) :
==========
- The underlined data structure resizable array or growable array.
- duplicates are allowed
- insertion order is preserved.
- Heterogenous objects are allowed (except TreeSet & TreeMap everywhere heterogeneous objects are allowed.)
- Null insertion is possible.
- ArrayList is best choice if our frequent operation is retrival operation (because ArrayList implements RandomAccess interfaces)
- ArrayList is the worst choice if our frequent operation is insertion or deletion in the middle (because several shift operation are require)

* Constructor :
---
(1) ArrayList l = new ArrayList()
- creates an empty array list object with default initial capacity 10.
- Once array list reaches its map capacity a new ArrayList will be created with the new capacity = (current capacity * 3/2)+1

(2) ArrayList l = new ArrayList(int intialCapacity)

(3) ArrayList l = new ArrayList(Collection c)

---

ii) LinkedList (c-1.2v):
==========
- The underlying data structure is double linked list.

- insertion order is preserved.
- duplicates are allowed.
- heterogeneous objects are allowed
- null insertion is possible.
- It implements Serilizable and Clonable interfaces but not RandomAccess interface.
- It is the best choice if our frequent operation is insertion or deletion in the middle.
- It is the worst choice if our frequent operation is retrieval.
- usually we can use LinkedList to implement stacks and queues to provide support for this requirement.

* LinkedList class defines following specific methods :

```
void addFirst();
void addLast();
Object getFirst();
Object getLast();
Object removeFirst();
Object removeLast();
```

* Constructor :

```
(1) LinkedList l = new LinkedList();
(2) LinkedList l = new LinkedList(Collection c);
```

---

iii) Vector (1.0v):

- The underlying data structure for the vector is resizable array or growable array.
- duplicate objects are allowed.
- insertion order is preserved.
- null insertion is possible.
- heterogeneous objects are allowed
- It implemented Serializable, Cloneable and RandomAccess interfaces.
- It is thread safe. (synchronized method)
- Best choice if the frequent operation is retrieval.

* methods :

* Constructor :

```
(1) Vector v = new Vector(); //default capacity = 10    //new capacity = 2*current capacity
(2) Vector v = new Vector(int initialCapacity)
(3) Vector v = new Vector(Collection c)
(4) Vector v = new Vector(int initialCapacity, int incrementalCapacity)
//we can specify our require incremental Capcity
for ex., we want after 10000 initialCpacity 5 elements then we can speficy incremental capacity.
```

---

iv) Stack :

- It is a child class of Vector
- specially designed class for Last in First Out(LIFO)

* Methods :
push(Object o)
pop();
peek(); //to return top of the stack without removing
empty();
search(Object o);   //it's return offset : offset means from the top of stack what is the number of element. if element is not there it will return -1
empty();

* Constructor :
---
(1) Stack s = new Stack();

****
****

3) Set (1.2v) (implemented classes : HashSet-1.2v, LinkedHashSet-1.4v):
- It is a child interface of collection.
- 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.
- Set interface doesn't contain any new methods. So we have to use only Collection interface methods.
- Implemented class :
i) HashSet :
==========
- The underlying data structure is HashTable.
- Duplicates are not allowed. If we are trying to insert duplicates, we won't get any compilation or runtime erros.
add() method simply returns false.
- Insertion order is not preserved and all objects will be inserted based on hash-code of objects.
- Heterogenous objects are allowed.
- 'null' insertion is possible.
- implements Serilizable and Clonable interfaces but not RandomeAccess .
- It is the best choice if our frequent operation is search.
- Constructor :
HashSet h = new HashSet(); //default capacity=16 and default Fill Retio 0.75
HashSet h = new HashSet(int initialCapacity); //you can set initial capacity and default fill retio 0.75
HashSet h = new HashSet(int initialCapacity, float loadFactor); //you can set initial capcaity and load factor also
HashSet h = new HashSet(Collect c);

Load factor/ Fill Ratio :
    After loading the how much factor a new HashSet object will be created that factor is called as Load factor or Fiill Ratio.

ii) LinkedHashSet :
==========
- It is the child class of HashSet
- introduced in 1.4 version
- it is the best choice to develop cache based applications, where duplicates are not allowed and insertion order must be preserved.
- it is exactly same as HashSet except the following differences.
HashSet :
i) the underlying data structure is Hashtable
ii) insertion order is not preserved.
iii) introduced in 1.2 verrsion

LinkedHashSet :
i) the underlying data structure is Hashtable + LinkedList (that is hybrid data structure)
ii) insertion order is preserved.
iii) introduced in 1.4 verrsion

****
****

4) SortedSet (1.2v) :
- It is a child interface of Set.
- If we want to represent a group of individual objects 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.
- specific Methods :
i) Object first(); //return first element
ii) Object last(); return last elements
iii) SortedSet headSet(Object obj); //return the SortedSet whose elements are < obj
iv) SortedSet tallSet(Object obj); //return the SortedSet whose elements are >= obj
v) SortedSet subSet(Object obj1, Object obj2); //return the SortedSet whose elements are >= obj1 and < obj2;
6) Comparator comparator();   //return comparator object that describes underlyiing sorting technique.
If we are using default natural sorting order then we will get null.

Note :
* Default natural sorting order for numbers Ascending order and for String alphabetical order.
* we can apply the above methos only on sorted set implemented class objects. That is on the Treeset object.

- Implemented class :

TreeSet :
=========
- The underlying data structure for treeset is balanced tree.
- duplicate objects are not allowed.
- Insertion order not preserved, but all objects will be inserted according to some sorting order.
- Heterogeneous objects are not allowed. If we are trying to insert heterogeneous objects then we will get runtime exception saying classcastexception.
- null insertion is not allowed.
- Constructor :
i) TreeSet t = new TreeSet();   //element inserted according to default natural sorting order.
i) TreeSet t = new TreeSet(Comparator c);   //elements will be inserted according to customized sorting
i) TreeSet t = new TreeSet(Collection c);
i) TreeSet t = new TreeSet(SortedSet s);
- Methods :

Note :
* If we are depending on default natural sorting order then objects should be homogeneous and comparable. Otherwise we will get runtime exception saying ClassCastException.
* An object is said to be comparable if and only if the corresponding class implements java.lang.comparable interface.
* String class and all wrapper classes already implements comparable interface. But StringBuffer doesn't implement comparable interface. Hence if we trying to add StringBufferObject then we will get ClassCastException.

Comparable Interface :
=======================
- This interface present in java.lang package. It contains only one method CompareTo()
public int compareTo(Object obj)
Example :
obj1.compareTo(obj2)
return -ve : if obj1 has to come before obj2
return +ve : if obj1 has to come after obj2
return 0 : if obj1 and obj2 are equal.

```
System.out.println("A".compareTo("Z")); //-25 -ve
System.out.println("Z".compareTo("B")); //24    +ve
System.out.println("A".compareTo("A")); //0 0
System.out.println("A".compareTo(null));    //NullPointerException
```

If we depending on default natural sorting order internally JVM will call CompareTo() method
will inserting objects to the TreeSet. Hence the objects should be Comparable.

```
TreeSet t = new TreeSet();
```

```
t.add("B");
t.add("Z"); //Z.compareTo(A); +ve
t.add("A"); //A.compareTo(B); -ve
```

Note :
- If we are not satisfied with default natural sorting order or inf the default natural sorting order is not already available then we can define our own customized sorting by using Comparator.
- Comparable ment for default natural sorting order where as comparator ment for customized sorting order.

Comparator Interface :
==========
- We can use comparator to define our own sorting (Customized sorting).
- Comparator interface present in java.util package.
- It defines two methods. compare and equlas.

1) public int compare(Object obj1, Object obj2)
i) return -ve if obje1 has to come before obj2
ii) return +ve if obj1 has to come after obj2
iii) return 0 if obj1 and obj2 are equal
2) public boolean equals();

- when ever we are implementing Comparator interface, compulsory we should provide implementation for compare() method.
- And implementing equals() method is optional, because it is already available in every java class from object class through inheritance.

****

5) NavigableSet (1.6v) (Implemented classes : TreeSet-1.2v):
- It is a child interface of SortedSet.
- It defines seral methods for navigation purposes.

****

6) Queue (1.5v) (Implemented classed : PriorityQueue, BlockingQueue-> LinkedBlockingQueue-PriorityBlockingQueue 1.5v):
- It is a child interface of Collection.
- 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.

****

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 objects as a key value pairs then we should go for Map Interface.

****
****

7) Map (1.2v) (Implemented classes : HashMap->LinkedHashMap-1.2v, WeakHashMap-1.2v, IdentityHashMap-1.4v, HashTable-1.0v):
- MAp is not the child interface of Collection.
- If we want to represent a group of individual objects as key value pairs then we should go for map.
- Both key and value are objects, duplicated key are not allowed but values can be duplicated.

****
****

8) SortedMap (1.2v) :
- It is the child interface of map.
- If we want to represent a grouup of key values pairs according to some sorting order of keys then we should go for sortedMap.

****
****

9) NavigableMap (1.6v) (Implemented class : TreeMap-1.2v) :
- It is the child interface of SortedMap.
- It defines several utility method for navigation purpose.

==========
==========

Difference between list and set :
==========

List :
1) Duplicates are allowed.
2) Insertion order preserved.

Set :
1) Duplicates are not allowed.
2) Insertion order not preserved.

==========
==========

Difference between ArrayList and Vector class :
==========

ArrayList :

1) Every method present ArrayList is non-synchronized
2) At a time multiple threads are allowed to operate on ArrayList Object and hence ArrayList is not thread safe.
3) Threads are not required to wait to operate on ArrayList, hence relatively performance is high.
4) Introduces in 1.2 version and it is non legacy class.

Vector :
1) Every method present (LinkedList)Vector synchronize
2) At a time only one thread are allowed to operate on Vector Object and hence Vector is thread safe.
3) Threads are required to wait to operate on Vector object , hence relatively performance is low.
4) Introduces in 1.0 version and it is legacy class.

==========
==========

How to get synchronized version of ArrayList Object :
==========

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

```
public static List synchronizedList(List l);
ex.,
ArrayList l1 = new ArrayList();
List l = Collections.synchronizedList(l1);
```

Note : similarly we can get Synchronized version of Set, Map objects by using the following method of Collections class

```
public static Set synchronizedSet(Set s);
public static Map synchronizedMap(Map m);
```

==========
==========

Difference between ArrayList and LinkedList :
==========

ArrayList :
1) It is the best choice if our frequent operation is retrieval.
2) It is the worst choice if our frequent operation is insertion or deletion.
3) It is a underlying data structure for ArrayList is resizable of growable Array.
4) It implements RandomAccess interface

LinkedList :
1) It is the best choice if our frequent operation is insertion and deletion.
2) It is the worst choice if our frequent operation is retrieval.

3) It is a underlying data structure is double linked list.
4) It doesn't implements RandomAccess interface

==========
==========

Cursor :
=========

If we want to retrieve objects one by one from the collection, then we should go for curosrs.
There are three types of cursors.

1) Enumeration :
==========
- introduces in 1.0v
- we can use this to get objects one by one from the old Collection Objects (Legacy Collections)
- we can create Enumeration Object by using elements() method of Vector class.
public Enumeration elements();
ex., Enumeration e = v.elements();

- Methods :
i) public boolean hasMoreElements();
ii) public Object nextElement();

- Limitations :
* This concept is applicable only for leagacy classes and hence it is not a universal cursor.
* By using Enumeration we can get only read access and we can't perform remove operation.
Note : To overcome above limitations of Enumeration we should go for Iterator.

2) Iterator :
==========
- we can apply Iterator concept for any collection object hence it is universal cursor.
- By using Iterator we can perform both read and remove operations.
- We can create Iterator object by using iterator() method of Collection interface.
public Iterator iterator();
ex., Iterator itr = C.iterator();
C - Collection object.
- Methods :
i) public boolean hasNext();
ii) public object next();
iii) public void remove();
- Limitation :
* By using Enumeration and Iterator we can move only towards forward direction and

we can't move to the backward direction and hence these are single direction cursors.
* By using Iterator we can perform only read and remove operations and we can't perform replacement of new objects.
Note : to overcome above limitaions of iterator we should go for ListIterator.

3) ListIterator :
==========
- By using this we can move either to the forward direction or to the backward direction and hence ListIterator is bidirectional cursor.
- By using ListIterator we can perform replacement and addition of new objects in addition to read and remove operations.
- It is the child interface of iterator and hence all methods of iterator by default available to ListIterator.
- It is the most powerful cursor but its limitation is, it is applicable only for List implemented class objects and it is not a universal cursor.
- We can create ListIterator object by using listIterator() method of List Interface.
public ListIterator listIterator();
ex., ListItrator itr = l.listIterator();
l - any list object
- Methods :
i) forward direction :
public boolean hasNext();
public void next();
public int nextIndex();

ii) Backward direction :
public boolean hasPrevisou;
public void previous
public int previousIndex;

iii) other capability methods :
public void remove();
public void set(Object new)
public void add(Object new)

Difference between Enumeration, Iterator, ListIterator :
==========
Enumeration :
i) Applicable for : only legacy classes
ii) Movement : only forward direction(single direction)
iii) Accessibility : only read access
iv) how to get it? : by using elements() method of Vector class
v) methods : 2 methods (hasMoreElements(), nextElement())
vi) Is it legacy : "yes" (1.0v)

Iterator :
i) Applicable for : any collection classes
ii) Movement : only forward direction(single direction)
iii) Accessibility : both read and remove access

iv) how to get it? : by using iterator() method of Collection interface
v) methods : 3 methods (hasNext(), next(), remove())
vi) Is it legacy : "no" (1.2v)

ListIterator :
i) Applicable for : only List classes
ii) Movement : both forward and backward direction (bidirectional)
iii) Accessibility : read, remove, replace and addition of new objects access
iv) how to get it? : by using listIterator() method of Vector class
v) methods : 9 methods
vi) Is it legacy : "no" (1.2v)

==========

* Some Tricky Interview Questions on ArrayList *
==========

1) List Object creation scenario
ArrayList arr = new ArrayList<String>();   //it's not preferable because it's a tightly coupled
List<String> list - new ArrayList<>(); //it's a preferable

2) How can I write Custome ArrayList where I don't want to allow duplicate?
=>
```
public class CustomArrayList extends ArrayList {
    @Override
    public boolean add(Object o) {
        if(this.contains(o)) {
            return true;
        }
        else {
            return super.add(o);
        }
    }

    public static void main(String[] args) {
        CustomArrayList list = new CustomArrayList();
        list.add(1);
        list.add(1);
        list.add(2);
        list.add(5);
        System.out.println(list);   //[1, 2, 5]
    }
}
```

3) Why set doesn't allow duplicate Element ?

Note : If we will trying to add primitive data type in set then it will not allowed
but if we will try to other object in set then it will allow duplicate to avoid this we have to
override equal and hashcode methods in our custome object.