1. Collection(Interface):
    - Collection represents Group of individual object as a single entity.

    Common Methods:
    - Boolean add(Object o)
    - Boolean addAll(Collection c)
    - Boolean remove(Object o)
    - Boolean removeAll(Collection c)
    - Boolean retainAll(Collection c)//Remove all objects except the objects present in collection c.
    - Void clear()
    - Boolean isEmpty();
    - Boolean containsAll(Collection C)
    - Int size()
    - Object [] toArray()
    - Iterator iterator()

The above methods are by default present for all the classes and interfaces which are present under collection interface.

2. **List:**
    - Child interface of Collection interface.
    - Index based collection of group of individual objects represented as a single entity where duplicates are allowed and insertion order is preserved.
    - We can differentiate duplicate objects with the help of index.
    - In below example 0,1,2,3,4,5,6 are the indexes of that collection and index represents the objects present in it.

    Eg:

0	1	2	3	4	5
T	U	S	A	R	R

Implementation classes for List Interface are:

- ArrayList(Implements List)
- LinkedList(Implements List)
- Vector(Implements List)(Synchronized verson of ArrayList)
- Stack(extends vector)

Methods present in List interface:

- 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)//Replace the object present in specific method and return new object.
- Int indexOf(object o)//it will return the object present in first index
- Int lastIndexOf(object o)
- ListIterator listIterator()

Array List:

- It is the child class of List Interface.
- If we want to represent a group of individual object as a single entity where duplicates are allowed and insertion order is preserved then we will go for array list.
- Under lying datastructure is growable or resizeable Array.
- ArrayList can hold heterogeneous objects.
- Deafult capacity is 10.
- After getting filled the default capacity a new array List will be created and the capacity will be (current capacity*3/2)+1.
- Array list is not thread safe and can contain null value.
- We can get Synchronized version of ArrayList by using the SynchronizedList() method present in Collections class.

E.g:

```
ArrayList <Integer> l2=new ArrayList<Integer>();//Non
Synchronized version
          l2.add(10);
          l2.add(11);
          System.out.println(l2);//[10, 11] Not thread safe
          List<Integer>
list=Collections.synchronizedList(l2);//Synchronized
version
          System.out.println(list);//[10, 11]//Thread Safe
```

- **Useually we use collections to hold and transfer objects from one location to another location to provide support for this requirement every collection class implements serialaizable and clonable interface.**
- ArrayList implements serialaizable,cloneable and RandomAccess interface.
- As Arraylist implements RandomAccess interface due to which we can access any random element with same speed.
- ArrayList is best choice if our frequent operation is retrival operation ArrayList is the best choice because ArrayList implements Random Access.
- If our frequent operation is insertion and deletion from the middle then ArrayList is the worst choice.

Random Access Interface (Not a part of Collection):

- It is a marker interface(Empty Interface) present in java.util package.
- Required ability will be provided by jvm.

Constructors present in ArrayList:

- ArrayList <Integer> l=new ArrayList <Integer>();//Will create an empty array list with default capacity 10.

- ArrayList l=new ArrayList(int initialcapacity)//it will create a empty arraylist object with specified initial capacity.
- ArrayList l=new ArrayList(Collection c)//Will create an equivalent arraylist for the given collection.

E.g:

```
public static void main(String[] args) {
        ArrayList <Integer> l=new ArrayList <Integer>();
        l.add(25);//adding values to arraylist
        l.add(20);//adding values to arraylist
        l.add(25);//adding values to arraylist
        l.add(22);//adding values to arraylist
        l.add(21);//adding values to arraylist
        System.out.println(l);//[25, 20, 25, 22, 21]
        ArrayList <Integer> l1=new ArrayList<Integer>();
        l1.add(30);
        l1.add(35);
        l1.add(32);
        l1.add(33);
        l1.add(35);
        System.out.println(l1);//[30, 35, 32, 33, 35]
        l.addAll(l1);//Added the array list l1 into l
        System.out.println(l);//[25, 20, 25, 22, 21, 30,
35, 32, 33, 35]
        System.out.println(l.get(3));//Finding the value
present in index 3 from Array list object
        System.out.println(l.remove(1));//20 is removed
from index 1.

    }
```