COLLECTION FRAMEWORK

Arrays	Collections
Arrays are used to store collection of homogeneous(similar) data.	homogeneous(similar) data. Collections is used to store heterogeneous data as well as homogeneous data.
Arrays are fixed in size. int a[]=new int[3];	Collections are growable in size
Arrays does not have any underlying data structure.	Every classes of Collection have data structure.
Arrays does not contains predefine methods(add, sorting, removing, replacing) which makes manipulation of data difficult.	In collection 80% support is by predefine API's(methods).
Memory wise array is not preferred.	Memory wise collections is prefered.
Perfomance wise arays are preferred.	Performance wise collection is not preferred.

It is an Architecture, which provides the group of classes and interfaces, which is used to store multiple objects as single unit.

CollectionFramework--------->concept

Collection---------->Root Interface

Collections---------->class

All the classes and interfaces of CollectionFramework are present in java.util package.

Collection:-

Collection is nothing but a group of objects represents as single unit.

It`s main purpose is to store huge amount of data.

It provides multiple methods to store and manipulate data.

Need of Collection Framework

1. Collections is used to store heterogeneous data as well as homogeneous data.
2. Collections is growable in size.
3. Collection frame work provides many built in methods.

Collection Hierarchy

Iterable interface is the root interface for all the collection classes.

Iterable interface have one abstract method called iterator().

Iterable interface is extended by collection interface.

There are many useful methods are there in collection interface and all the classes which are implementing collection interface can use these methods.

Collection interface is extended by three interfaces, they are

List Interface
Set Interface
Queue Interface

Different classes of collection are,

Array List
Linked List
Vector
These classes implements List Interface
Linked List
Priority Queue
These classes implements Queue Interface
Hash Set
LinkedHash Set
Tree Set
These classes implements Set Interface

Collection (I)

It defines most common methods for all collection objects.

1) isEmpty() check whether Collections empty or not
2) add(Object ref) Used to add objects
3) addAll(Collection ref) Used to copy one collection objects into another
4) size() Provides size of collection(always calculates from 1)
5) remove(Object ref) Used to remove object
6) contains(Object ref) check whether Object is present or not
7) removeAll(Collection ref) Removes whole objects of collection
8) retainAll(Collection ref) Removes those elements which are not present in AL.
9) containsAll(Collection ref) checks whether content of one collection is present in another collection or not.
10) clear() It is used to clear the array

List(I)

List is an Interface which extends Collection Interface
List follows Index Based process.
List allows homogeneous and heterogeneous Objects.
List allows duplicate duplicates
List allows Null objects.
Insertion order is maintained.
List interface contains all methods of Collection interface. And also contains following methods,

1)add(int index, Object ref)
2)addAll (int index, Collection ref)
3)remove(int index)
4)lastindexOf(Object ref)

5)indexOf(Object ref)
6)get(int index)
7)set(int index, Obj)
8)listIterator()

ArrayList()
It is a concrete subclass of List Interface
Characteristics
ArrayList stores Heterogeneous data .
It is possible to add NULL objects in ArrayList.
It allows Duplicate objects.
In ArrayList Insertion Order is preserved
It follows Data Structure as growable size array.
Iterator and ListIterator cursors are used.
Default size of Array List is 10.

It has 3 constructors.
Constructors of ArrayList
ArrayList()
ArrayList(Collection ref)
ArrayList(int initialCapacity)

Formula:- newSize=(oldsize*3/2)+1

ex:-

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		System.out.println(a.isEmpty());
		a.add(2);
		a.add(true);
		a.add(2);
		a.add("Testyantra");
		a.add(5);
		a.add('1');
		System.out.println(a.isEmpty());
		System.out.println(a.size());
		System.out.println(a.get(5));
		System.out.println(a);
	}	}
```

Note:-
We can add elements of one arrayList into another with the help of add() or addAll().
add() method adds the arrayList as a single object.
addAll() adds elements of arrayList as separate objects.

ex:- Adding two ArrayList elements

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		System.out.println(a.isEmpty()); //true
		a.add(2);
		a.add(true);
		a.add(2);
		a.add("Testyantra");
		a.add(5);
		a.add('1');
		System.out.println(a); //[2, true, 2, Testyantra, 5, 1]
		ArrayList a1=new ArrayList(a);
		a1.add(3);
		a1.add(false);
		a1.add(3);
		a1.add("Testyantra");
		a1.add(6);
		a1.add('5');
		System.out.println(a1); //[2, true, 2, Testyantra, 5, 1, 3, false, 3, Testyantra, 6,
5]
	}
}
```

ex2:- Adding two arrays using add()

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		System.out.println(a.isEmpty()); //true
		a.add(2);
		a.add(true);
		a.add("Testyantra");
		System.out.println(a); //[2, true, Testyantra]
		ArrayList a1=new ArrayList(a);
		a1.add(a);
		a1.add(3);
		a1.add(false);
		System.out.println(a1); //[2, true, Testyantra, [2, true, Testyantra], 3, false]
	}
}
```

ex3:- Adding two arrays using addAll()

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		System.out.println(a.isEmpty()); //true
```

```
		a.add(2);
		a.add(true);
		a.add("Testyantra");
		System.out.println(a); //[2, true, Testyantra]
		ArrayList a1=new ArrayList(a);
		a1.addAll(a);
		a1.add(3);
		a1.add(false);
		System.out.println(a1); //[2, true, Testyantra, 2, true, Testyantra, 3, false]
	}
}
```

ex4:- removing object using remove()

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		System.out.println(a.isEmpty()); //true
		a.add(2);
		a.add(true); //true
		a.add("Testyantra");
		a.add("Hyderabad");
		a.add('3');
		System.out.println(a); //[2, true, Testyantra, Hyderabad, 3]
		System.out.println(a.remove(0)); //2
		System.out.println(a.remove(true)); //true
		System.out.println(a); //[Testyantra, Hyderabad, 3]
	}
}
```

ex:- fetching values using for loop

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		System.out.println(a.isEmpty()); //true
		a.add(2);
		a.add(true); //true
		a.add("Testyantra");
		a.add("Hyderabad");
		a.add('3');
		for(int i=0;i<a.size();i++) {
			System.out.println(a.get(i));
		}
	}
}
```

Ex:- Generic ArrayList is used to Store homogeneous values

```
public class CoFrameWork {
```

```
	public static void main(String[] args) {
		ArrayList <Integer>a=new ArrayList<Integer>();
		System.out.println(a.isEmpty()); //true
		a.add(2);
		a.add(true); //CTE
		a.add("Testyantra");//CTE
		a.add("Hyderabad"); //CTE
		a.add('3'); //CTE
		for(int i=0;i<a.size();i++) {
			System.out.println(a.get(i));
		}
	}
}
```

ex2:- Generic ArrayList is used to Store homogeneous values

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList <Integer>a=new ArrayList<Integer>();
		System.out.println(a.isEmpty()); //true
		a.add(2);
		a.add(3);
		a.add(202);
		a.add(69);
		a.add(null);
		System.out.println(a); //[2, 3, 202, 69, null]
		for(int i=0;i<a.size();i++) {
			System.out.println(a.get(i));
		}
	}
}
```

ex:- using for each loop to retrieve the values

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList <Integer>a=new ArrayList<Integer>();
		System.out.println(a.isEmpty()); //true
		a.add(2);
		a.add(3);
		a.add(202);
		a.add(69);
		a.add(null);
		System.out.println(a); //[2, 3, 202, 69, null]
		for (Integer i : a) {
			System.out.println(i);
		}
	}
}
```

contains() :-

it is used to search an element

containsAll() :-

it is used to search the specified collection from the collection

ex:- Using contains() and containsAll()

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList <Integer>a=new ArrayList<Integer>();
		a.add(2);
		a.add(3);
		a.add(202);
		a.add(69);
		a.add(null);
		System.out.println(a.contains(2)); //true
		System.out.println(a); //[2, 3, 202, 69, null]
	}
}
```

ex2:-

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		a.add(2);
		a.add(3);
		a.add(202);
		a.add(69);
		a.add(null);
		a.add('a');
		System.out.println(a); //[2, 3, 202, 69, null]
		ArrayList a1=new ArrayList (a);
		a1.add(a);
		a1.add(null);
		a1.add('a');
		System.out.println(a1);
		System.out.println(a1.containsAll(a)); //true
		System.out.println(a.containsAll(a1)); //false
	}
}
```

Cursors

This is the special characteristics given for collection concepts

Using cursors we can retrieve the objects in collection one by one,

they are of 2 types –

1.Iterator

2.ListIterator

1.Iterator

Iterator is an interface which is used to traverse the list in forward direction.
Basically it provides the privilige to access the objects without using index.

```
interface Iterator {
	public boolean hasNext();
	public object next();
	public void remove();
}
```

hasNext():
It returns true, if there is Object available in collection.

next():
It returns current Object and move the cursor to next Object.

ex:-

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		a.add(1);
		a.add('a');
		a.add("Testyantra");
		a.add(0.11);
		Iterator i=a.iterator();
		while(i.hasNext()) {
			System.out.println(i.next());
		}
	}
}
```

ex2:- to remove object from collection in iterator

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		a.add(1);
		a.add('a');
		a.add("Testyantra");
		a.add(0.11);
		Iterator i=a.iterator();
		while(i.hasNext()) {
			Object s = i.next();
			if(s.equals("Testyantra")) {
				i.remove();
				System.out.println("The removed element is Testyantra");
				break;
			}
		}
```

```
		for(int j=0;j<a.size();j++) {
			System.out.println(a.get(j));
		}
	}
}
```

2.ListIterator

ListIterator is an interface which provides the facility to traverse the list in forward as well as backward direction.

```
interface ListIterator {
	public boolean hasNext();
	public object next();
	public void remove();
	public boolean hasPrevious();
	public object previous();
	public void add();
}
```

hasPrevious(): Returns true, if object is available to iterate from previous direction.
previous(): prints the current object and move the cursor to next object in previous direction.
hasNext(): It returns true, if there is Object available.
next(): It returns current Object and move cursor to next Object.

ex:- using hasNext and next

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		a.add(1);
		a.add('a');
		a.add("Testyantra");
		a.add(0.11);
		ListIterator i=a.listIterator();
		while(i.hasNext()) {
			System.out.println(i.next());
		}
	}
}
```

ex2:-

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
```

```
                a.add(1);
                a.add('a');
                a.add("Testyantra");
                a.add(0.11);
                ListIterator i=a.listIterator();
                while(i.hasNext()) {
                        System.out.println(i.next());
                }
                while(i.hasPrevious()) {
                        Object s = i.previous();
                        if(s.equals(1)) {
                                i.remove();
                                System.out.println("The removed object is 1");
                                break;
                        }
                }
                for(Object o:a) {
                        System.out.println(o);
                }
        }
}
```

Differences between Iterator and ListIterator

Iterator	ListIterator
It is an interface used to retrieve objects in forward direction.	It is an interface used to retrieve objects in forward as well as backward direction.
It is used for all collection classes	It is only used for classes which implements ListInterface - Arraylist, LinkedList, Vector
Iterator object, we will get by using iterator()	Listiterato object, we will get by using listIterator()
Using iterator interface methods, we can traverse only in forward direction i.e hasNext(), next()	Using ListIterator methods, we can traverse in both forward and backward direction hasNext(), next(), hasPrevious(), previous()

removeAll() :- It is used to remove elements from the current ArrayList, It removes those elements which are present in the ArrayList given as argument.

ex:-

```
public class CoFrameWork {
        public static void main(String[] args) {
                ArrayList a=new ArrayList();
                a.add(1);
                a.add('a');
                a.add("Testyantra");
                a.add(0.11);
                ArrayList a1=new ArrayList();
                a1.add(1);
```

```
                a1.add('a');
                a1.add(0.11);
                a1.add("s");
                a.removeAll(a1);
                System.out.println(a);
        }
}
```

retainAll() :- It retains those elements in the arraylist that area also present in the specified collection.

ex:-

```
public class CoFrameWork {
        public static void main(String[] args) {
                ArrayList a=new ArrayList();
                a.add(1);
                a.add('a');
                a.add("Testyantra");
                a.add(0.11);
                ArrayList a1=new ArrayList();
                a1.add(1);
                a1.add('a');
                a1.add(0.11);
                a1.add("s");
                a.retainAll(a1);
                System.out.println(a);
        }
}
```

Sorting an arrayList :- With the help of collections class we can sort ArrayList.

ex:-

```
public class CoFrameWork {
        public static void main(String[] args) {
                ArrayList a=new ArrayList();
                a.add(1);
                a.add(6);
                a.add(3);
                a.add(11);
                Collections.sort(a);
                System.out.println(a);
                Collections.reverse(a);
                System.out.println(a);
        }
}
```

Index and lastIndex of objects :-

ex:-

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		a.add(1);
		a.add(6);
		a.add(3);
		a.add(11);
		a.add(1);
		System.out.println(a.indexOf(1)); //0
		System.out.println(a.lastIndexOf(1)); //4
	}
}
```

set() :- It is used to replace the element present at specified index with the given element.
ex:-

```
public class CoFrameWork {
	public static void main(String[] args) {
		ArrayList a=new ArrayList();
		a.add(1);
		a.add(6);
		a.add(3);
		a.add(11);
		a.add(1);
		a.set(3, 5);
		System.out.println(a);
	}
}
```

Drawbacks of ArrayList :-

ArrayList is fast for accessing a specific elements but if you are adding the element after specifying the size then new block will be created and all the old elements will be copied in the new block because of which the performance will reduce.