=> Point to remember for Array & Collection Framework :-

1. -> Array is java language feature inbuilt support provided by Sun Microsystems. We have to develop algorithms to sort or insert or delete etc
-> Collection Framework are API feature. It provides predefined classes and interfaces and methods by which we can easily iterate or delete or sort the elements

2. -> Array can store primitive (int, char etc) and non-primitive (objects) data types
-> Collection Framework can store only nonprimitive data types (objects)

3. -> Array can store only homogeneous data types i.e. array can store only similar type of data
-> Collection Framework can store hetrogeneous data i.e. we can store different type of data

4. -> The size of an array cannot be increased or decreased according to our requirement at runtime
-> The size of collection can be increased or decreased according to our needs

5. -> Array are not good with respect to memory
-> Collection framework are very good with respect to memory

6. -> Arrays are good by performance wise
-> Collection are not good by performance wise

=> What is Collection Framework ?

-> Collection Framework consists of 2 words i.e. Collection and Framework

= Collection is a single entity or an object which contains multiple data

= Framework represents the library

-> Collection framework is the set of classes and interfaces that implement commonly reusable collection data structure

-> Collection framework contains 2 main parts :-

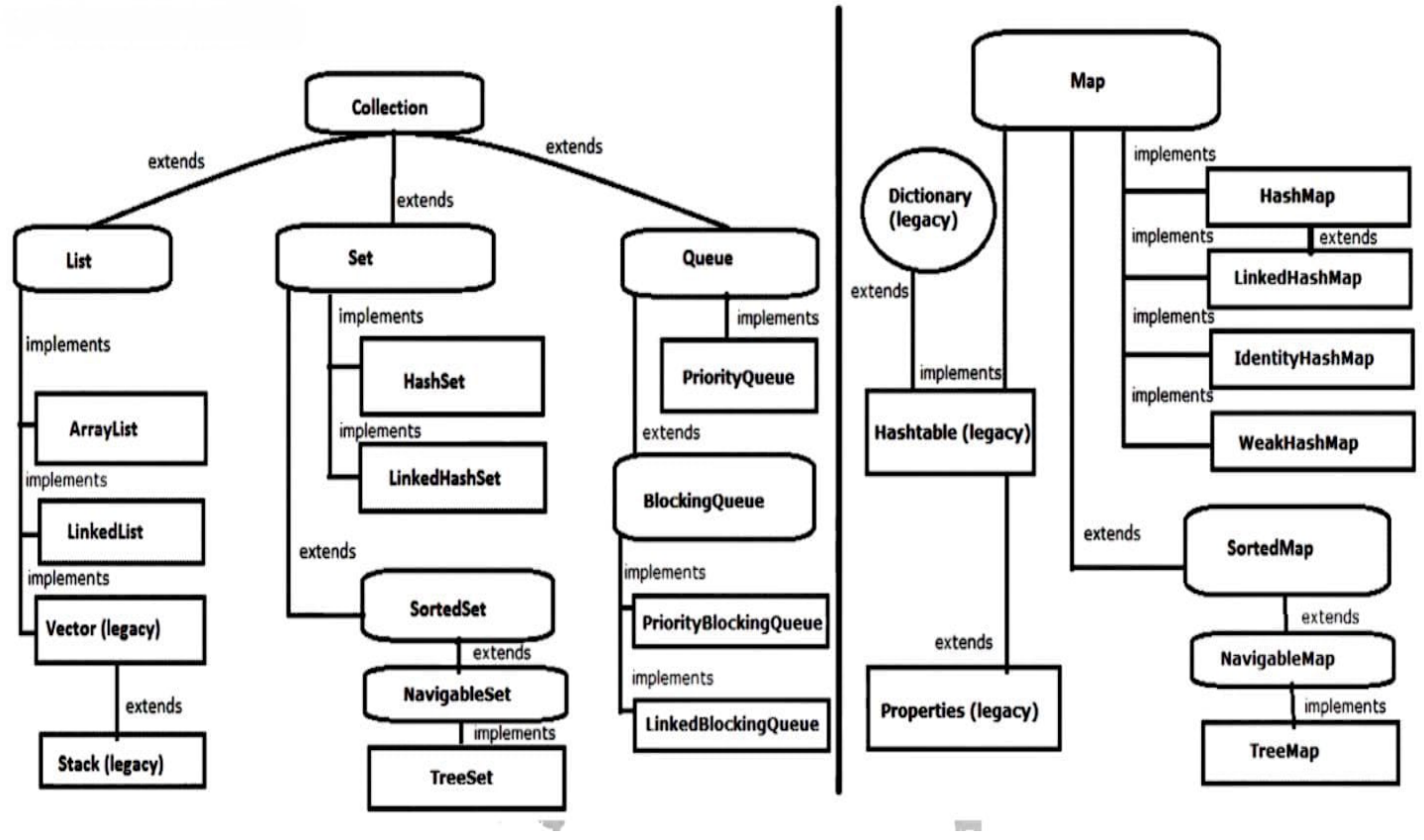
1. java.util.Collection
2. java.util.Map

-> "9 key interfaces" of Collection Framework

1. Collection
2. List
3. Set
4. SortedSet
5. NavigableSet
6. Queue
7. Map
8. SortedMap
9. NavigableMap

-> In Collection, we can store the data directly but in Map we can store the data in key-value pairs

-> Hierarchy of Collection Framework :-



=> Collection :-

-> Collection is an interface which is present in java.util package

-> Syntax : public interface Collection<E> extends Iterable<E> { - }

-> Collection was introduced in JDK 1.2 version

-> Collection is an object which is used to represent a group of individual objects as a single unit

-> Collection interface is the root interface of Collection Framework

-> There is no concrete class which implements the Collection interface directly but there are interfaces which inherit the Collection interface i.e. List, Set & Queue

-> Hierarchy of Collection interface :-

-> Collection interface contains most common methods which are applicable for any collection object

-> Methods of Collection Interface :-

1. boolean add(Object obj);
2. boolean addAll(Collection c);
3. boolean remove(Object obj);
4. boolean removeAll(Collection c);
5. default boolean removeIf(-) { - }
6. boolean retainAll(Collection c);
7. void clear();
8. boolean contains(Object obj);
9. boolean containsAll(Collection c);
10. boolean isEmpty();
11. int size();
12. Iterator iterator();
13. Object toArray();
14. boolean equals(Object obj);
15. int hashCode();

=> What is difference between Collection & Collections

1. -> Collection is an interface
   -> Collections is a utility class

2. -> Collection is an object which is used to represent a group of individual objects as a single unit
   -> Collections defines several utility methods that are used to operate on collection objects like sorting, searching etc

3. -> Collection interface contains default, abstract methods and static methods
   -> Collections class contains only static methods

=> What is Utility Class in Java ?

-> Utility class is also known as helper class which cannot be instantiated
-> Utility class contains only static methods
-> Examples are Arrays, Collections

-> How we can create utility class :-

1. declare the class as public and final
2. we have to declare private constructor to prevent object creation
3. class should contain only static methods and does not contain abstract methods

=> What is Utility Methods ?

-> Utility methods perform common, often reused methods.
-> Utility methods are always static type
-> Examples are sorting, searching, methods performing string manipulation, methods connecting to databases etc

=> List Interface :-

-List is a interface which is present in java.util package

-List is the child interface of Collection interface

Syntax : public interface List extends Collection

{ - }

-List was introduced in JDK 1.2 version

-Hierarchy of List interface :-

-Properties of List Interface :-

1. List is an index based Data Structure which means that first element will be inserted at 0 index position

2. List can store different data types or heterogeneous elements

3. We can store duplicate elements in the List

4. We can store any number of null values in the List

5. List follows the insertion order which means the sequence in which we are inserting the elements, in the same sequence we can retrieve the elements

6. List does not follow the sorting order

=> Methods of List Interface :-

1. List contains all the methods of Collection interface
2. void add(int index, Object obj);

3. boolean addAll(int index, Collection c);

4. Object get(int index);

5. Object remove(int index);

7. 6. Object set(int index, Object newobj); //set method is used to replace the object at given index position

8. 7. int indexOf(Object obj); //it will return the index position of provided object and if object is not found then it will return -1

9. int lastIndexOf(Object obj);

=> ArrayList :-

-ArrayList is an implemented class of List interface which is present in java.util package

-Syntax : **public class ArrayList extends AbstractList implements List, RandomAccess, Cloneable, Serializable**

-The underline Data-Structure of ArrayList is resizable array or growable array

-ArrayList was introduced in JDK 1.2 version

-Properties of ArrayList :-

1. ArrayList is an index based Data Structure which means that first element will be inserted at 0 index position
2. ArrayList can store different data types elements or hetrogeneous elements
3. We can store duplicate elements in the ArrayList
4. We can store any number of null values in the ArrayList
5. ArrayList follows the insertion order which means the sequence in which we are inserting the elements, in the same sequence we can retrieve the elements
6. ArrayList does not follow the sorting order (above properties are same as List interface)
7. ArrayList is non-synchronized collection because ArrayList does not contain any synchronized method
8. ArrayList allows more than one thread at one time
9. ArrayList allows parallel execution
10. ArrayList reduces the execution time which in turn makes the application fast
11. ArrayList is not threadsafe
12. ArrayList does not gurantee for data consistency

-Working of an ArrayList :-

1. When we create default ArrayList, a new ArrayList with initial capacity 10 is created (but size is 0)
2. When the ArrayList capacity is full, a new ArrayList will be created with new capacity.

The new Capacity is calculated by this formula:-

(CurrentCapacity * 3 / 2) + 1

3. Then all the elements will be copied into the new ArrayList (and due this this reason performance of an ArrayList decreases)
4. When new ArrayList is created automatically, then reference variable will point to the new ArrayList
5. Then old ArrayList object will be not referenced by any reference and then garbage collection will delete that object

Note : There is no way by which we can find the capacity of an ArrayList

=>Contructors of ArrayList :-

1. ArrayList al=new ArrayList();

- In this arraylist, an ArrayList collection object is created whose capacity is 10

2. ArrayList al=new ArrayList(int initialCapacity);

- In this arraylist, an ArrayList object is created with provided initialCapacity

3. ArrayList al=new ArrayList(Collection c);

-In this arraylist, another collection object is copied into new arraylist object

=>When we should use ArrayList ?

= When we use retrival operation mostly (Retrival operation is fast in case of ArrayList because it implements RandomAccess interface)

=>When we should not use ArrayList ?

= When we have mostly insertion or deletion operation, then we should not use ArrayList

```
public class CollectionDemo {
    public static void main(String[] args) {
        ArrayList list1 = new ArrayList() ;
        list1.add(10) ;
        list1.add(20) ;
        list1.add(30) ;

        ArrayList list2 = new ArrayList() ;
        list2.add("Arun") ;
        list2.add("Bhavek") ;

        // list1.add(list2) ;        //   [10, 20, 30, [Arun, Bhavek]]
        // list1.addAll(list2) ;    //    [10, 20, 30, Arun, Bhavek]
         list1.addAll(2,list2) ;   //    [10, 20, Arun, Bhavek, 30]

        System.out.println(list1);
    }
}
```

==========

```
public class CollectionDemo {
    public static void main(String[] args) {
        ArrayList list = new ArrayList() ;
        list.add(10) ;
        list.add(20) ;
        list.add("Arun") ;
        list.add(30) ;
        //list.remove(10) ;  // give error since treat it as an index
        list.remove("Arun") ;

        System.out.println(list);
    }
}
```

==========

```
public class CollectionDemo {
    public static void main(String[] args) {
        ArrayList list1 = new ArrayList() ;
        list1.add(10) ;
        list1.add(20) ;
        list1.add(30) ;
        list1.add(40) ;

        ArrayList list2 = new ArrayList() ;
        list2.add(10) ;
        list2.add(20) ;

         list1.removeAll(list2) ;

        System.out.println(list1);
        System.out.println(list2);
    }
}
```

```
==========================================================
```

```
public class CollectionDemo {
    public static void main(String[] args) {
        ArrayList list1 = new ArrayList() ;
        list1.add(10) ;
        list1.add(20) ;
        list1.add(30) ;
        list1.add(40) ;
        ArrayList list2 = new ArrayList() ;
        list2.add(10) ;
        list2.add(20) ;

         list1.retainAll(list2) ;

        System.out.println(list1);  // [10, 20]
        System.out.println(list2);
    }
}
```

```
==========================================================
```

```
public class CollectionDemo {
    public static void main(String[] args) {
        ArrayList list1 = new ArrayList() ;
        list1.add(10) ;
        list1.add(20) ;
        list1.add(30) ;
        list1.add(40) ;

        ArrayList list2 = new ArrayList() ;
        list2.add(10) ;
        list2.add(20) ;

        System.out.println(list1.contains(20));
         System.out.println(list1.containsAll(list2));

         list1.clear();
         System.out.println(list1.isEmpty());
    }
}
```

==========

```
public class CollectionDemo {
    public static void main(String[] args) {
        ArrayList list1 = new ArrayList() ;
        list1.add(10) ;
        list1.add(20) ;
        list1.add(30) ;
        list1.add(40) ;

        Object[] obj = list1.toArray() ;

        for(int i=0 ; i< obj.length ; i++){
            System.out.println(obj[i]);
        }
        //---------------------------------------
        for(Object o:obj){
            System.out.println(o);
        }
    }
}
```

==========

```
public class CollectionDemo {
    public static void main(String[] args) {
        ArrayList list = new ArrayList() ;
        list.add(10) ;
        list.add(20) ;
        list.add(30) ;
        list.add(40) ;
        list.add(20) ;

        System.out.println(list.get(2));
        System.out.println(list.indexOf(20));
        System.out.println(list.lastIndexOf(20));
    }
}
```

==========

=> RandomAccess interface :-

-> RandomAccess interface is a marker interface that means it does not contain any methods or fields (variables)

-> The purpose of RandomAccess interface is to retrieve any random element in collection object at the same speed. For example we have collection object having 1 crore elements, we have to search 3rd element or middle element or last element then it will search with the same speed

-> There are only 2 classes which inherits the RandomAccess interface

1. ArrayList
2. Vector

=> Cloneable Interface :-

-> Cloneable interface is also a marker interface

-> It was introduced in JDK 1.0 version

-> It is used to clone the object without using the new keyword

```
public class Test implements Cloneable{
    int no ;
    String name ;
    Test(){                          // non parameterized Constructor

    }
    Test(int no, String name){      // parameterized Constructor
     this.no = no ;
     this.name = name ;
    }

    public static void main(String[] args) throws CloneNotSupportedException{
        Test t1 = new Test(10, "Arun") ;
        Test t2 = (Test)t1.clone() ;
        System.out.println(t2.no);
        System.out.println(t2.name);
    }
}
```