The **Java Collections Framework** is a powerful and versatile set of interfaces and classes that provide efficient ways to store, manage, and manipulate data.

**Core Interfaces:**

1. **Collection Interface:** The root of the collection hierarchy.
2. **List Interface:** An ordered collection (also known as a sequence).
3. **Set Interface:** A collection that does not allow duplicate elements.
4. **Queue Interface:** A collection designed for holding elements prior to processing.
5. **Deque Interface:** A double-ended queue that allows insertion and removal from both ends.
6. **Map Interface:** An object that maps keys to values (Note: Map is part of the Collections Framework but it doesn’t inherit the collection of the interface).

**Collection**: The collection is considered as the root interface of the collection framework. It provides several classes and interfaces to represent a group of individual objects as a single unit.

The [List](https://www.geeksforgeeks.org/list-interface-java-examples/), [Set](https://www.geeksforgeeks.org/set-in-java/), and [Queue](https://www.geeksforgeeks.org/queue-interface-java/) are the main sub-interfaces of the collection interface. The map interface is also part of the java collection framework, but it doesn’t inherit the collection of the interface. The**add()**,**remove()**, **clear()**, **size()**, and **contains()** are the important methods of the Collection interface. Directly implemented by classes such as ArrayList, HashSet, and LinkedList. Part of the java.util package.

**Collections:** Collections is a utility class present in java.util package. It defines several utility methods like sorting, searching and reversing which is used to operate on collection. It has all static methods. It provides methods like sort(), reverse(),max(), min()

|  |  |  |
| --- | --- | --- |
| **S.No** | **List** | **Set** |
| 1. | The list implementation allows us to add the same or duplicate elements. | The set implementation doesn't allow us to add the same or duplicate elements. |
| 2. | The insertion order is maintained by the List. | It doesn't maintain the insertion order of elements  LinkedHashSet, access is similar to HashSet (hash-based), but it also maintains a **doubly-linked list** to preserve the insertion order. |
| 3. | List allows us to add any number of null values. | Set allows us to add at least one null value in it. |
| 4. | The List implementation classes are LinkedList and ArrayList. | The Set implementation classes are TreeSet, HashSet and LinkedHashSet. |
| 5. | We can get the element of a specified index from the list using the get() method. | We cannot find the element from the Set based on the index because it doesn't provide any get method(). |
| 6. | It is used when we want to frequently access the elements by using the index. | It is used when we want to design a collection of distinct elements. |
| 7. | Access the elements based on index | For HashSet and LinkedHashSet, access is based on the **hashing** mechanism  These sets use the hashCode() method of elements to determine their position in the set |

Note: **TreeSet** accesses elements using **tree-based (sorted) ordering**.

|  |  |
| --- | --- |
| ARRAY | ARRAYLIST |
| An **array** is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location. | The **ArrayList** is a class of Java **Collections** framework. It contains popular classes like **Vector, HashTable**, and **HashMap**. |
| Array is **static** in size. | ArrayList is **dynamic** in size. |
| An array is a **fixed-length** data structure. | ArrayList is a **variable-length** data structure. It can be resized itself when needed. |
| It is mandatory to provide the size of an array while initializing it directly or indirectly. | We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size. |
| It performs **fast** in comparison to ArrayList because of fixed size. | ArrayList is relatively slower because of its dynamic nature |
| An array can store both **objects** and **primitives** type. | We cannot store **primitive** type in ArrayList. It automatically converts primitive type to object. |
| We use **for** loop or **for each** loop to iterate over an array. | We use an **iterator** to iterate over ArrayList. |
| length keyword can give the total size of the array. | ArrayList provides the **size()** method to determine the size of ArrayList. |
| We can add elements in an array by using the **assignment**operator. | Java provides the **add()** method to add elements in the ArrayList. |
| Array can be **multi-dimensional**. | ArrayList is always **single-dimensional**. |

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements.  Each element (node) contains references to the previous and next nodes in the list. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 5) Consumes less memory compared to LinkedList since it stores only the actual elements. | Consumes more memory because, in addition to storing the elements, each node also stores references to the previous and next elements. |

**Map interface**

Map Interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package represents a mapping between a key and a value. Java Map is part of the Collections Framework but it doesn’t inherit the collection of the interface The Map interface allows for efficient data storage in key-value pairs,

A Map cannot contain duplicate keys and each key can map to at most one value

There are two interfaces for implementing Map in Java. They are Map and [SortedMap](https://www.geeksforgeeks.org/sortedmap-java-examples/), and three classes: HashMap, TreeMap, and LinkedHashMap

The Map interface provides several essential methods for managing key-value pairs:

[clear()](https://www.geeksforgeeks.org/map-clear-method-in-java-with-example/),[containsKey(Object)](https://www.geeksforgeeks.org/map-containskey-method-in-java-with-examples/), [containsValue(Object)](https://www.geeksforgeeks.org/map-containsvalue-method-in-java-with-examples/), [entrySet()](https://www.geeksforgeeks.org/map-entryset-method-in-java-with-examples/),[keySet()](https://www.geeksforgeeks.org/map-keyset-method-in-java-with-examples/), V get(Object key), V put(K key, V value)