# Collections Framework

A set of Collection classes and interfaces is called a collection framework.

A collection is an object that represents a group of objects.

Arranging data in different formats is called data structures.

All collection classes and interfaces are part of java.util package.

The collection framework helps to increase programming speed and quality.

## Core Interfaces

Here is the list of the core collection interfaces:

* Collection: It is a root interface in a one-dimensional collection hierarchy.
* List: it extends the Collection interface and it allows duplicate elements.
* Set: it extends the collection interface and it does not allow duplicate elements.
* Map: It is a root interface in a two-dimensional collection hierarchy, it contains key and value parts. It does not allow duplicate keys but values may be duplicated.

**A screenshot of a computer

Description automatically generated with medium confidence**

* SortedSet: A SortedSet is a set in which elements are sorted.
* SortedMap: A sortedmap is a map in which key and value pairs are sorted based on the keys.
* NavigableSet: it is used to navigate elements in a set.
* NavigableMap: it is used to navigate key and value pairs in a map.
* Queue: it is called as first in first out list, it allows insertion at rear end and deletion at front end.
* Deque: it stands for double-ended queue. It allows insertion and deletion on both sides.

These interfaces are the foundation of the collection framework.

**Timeline

Description automatically generated**

## General Purpose Implementation

Core collection interfaces implementation classes are called general-purpose implementations.

* ArrayList
* LinkedList
* HashSet
* LinkedHashSet
* TreeSet
* HashMap
* LinkedHashMap
* TreeMap
* PriorityQueue
* ArrayDeQue

### ArrayList<E>

It is an array representation of the List implementation class.

It allows duplicate elements because it implements list interface.

It allows null values.

It supports both heterogeneous and homogeneous elements.

The default initial capacity of ArrayList is 10, the Load factor of ArrayList is 100%.

**Graphical user interface, text, application, table

Description automatically generated**

Refer Code Snippet: ArrayListExample1\_Class

As demonstrated in ArrayListExample1\_Class, ArrayList class is very unsafe as it allows different types of data.

We have to generalize it using generics to have safe or same type of data.

Refer Code Snippet: ArrayListExample2\_Class

As demonstrated in ArrayListExample2\_Class, ArrayList<Integer> class is generalized to accept only integer, this is safer as the user is not allowed to provide other than integers.

Refer Code Snippet: ArrayListExample3\_Class

As demonstrated in ArrayListExample3\_Class, ArrayList<employee> class is generalized to accept only employee type.

How ArrayList<E> Elements Shuffle works

Diagram

Description automatically generated

### LinkedList<E>

It is a linked representation of List implementation class.

It allows duplicate elements because it implements List interface.

It allows null values also.

It occupies more memory because data is stored in nodes.

* Left link field: it contains the previous node address.
* Data field: it contains data.
* Right Link field: it contains next node address.

Diagram

Description automatically generated with medium confidence

Refer Code Snippet: LinkedListExample1\_Class

How LinkedList<E> Elements Shuffle works.

Diagram

Description automatically generated

### Difference between ArrayList<E> and LinkedList<E>

**Table

Description automatically generated with medium confidence**

### HashSet<E>

* Insertion order is not preserved and not sorted.
* Null values are allowed.
* Duplicate values are removed.

Refer Code Snippet: HashSetExample1\_Class

### LinkedHashSet<E>

* Insertion order is preserved.
* Null values are allowed.
* Duplicate values are removed.

Refer Code Snippet: LinkedHashSetExample1\_Class

### TreeSet<E>

* Insertion order is not preserved.
* Elements are sorted.
* Null values are not allowed, Null pointer exception.
* Duplicate values are removed.

Refer Code Snippet: TreeSetExample1\_Class

### HashMap<E,V>

* It is a two-dimensional collection class and it maintains key and value pairs.
* It is an implementation of hashing technique with array representation.
* It does not allow duplicate keys but values can be duplicated.
* It allows one null key and many null values.
* It is unordered map.
* It will not preserve insertion order.
* Initial capacity is 16 and load factor is 75%.

Refer Code Snippet: HashMapExample1\_Class

### LinkedHashMap<E,V>

* It is a two-dimensional collection class and it maintains key and value pairs.
* It is an implementation of hashing technique with linked representation.
* It does not allow duplicate keys but values can be duplicated.
* It allows one null key and many null values.
* It is an ordered map.
* Insertion order is preserved.
* Initial capacity is 16 and load factor is 75%.

Refer Code Snippet: LinkedHashMapExample1\_Class

### TreeMap<E,V>

* It is a two-dimensional collection class and it maintains key and value pairs.
* It is an implementation of the binary tree technique with linked representation.
* It does not allow duplicate keys but values can be duplicated.
* It does not allow null key but it allows null values.
* It is a sorted map.
* Insertion order is not preserved.
* Initial capacity is 16 and load factor is 75%.

Refer Code Snippet: HashTreeExample1\_Class

### Difference between Set and Map implementation classes

**Text

Description automatically generated**

### PriorityQueue<E>

* Elements are inserted at rear end and deleted at front end.
* In priority queue least valued element has highest priority.
* It allows duplicate elements
* It does not allow null value
* Initial capacity is 11.
* Load factor is 100%
* Internally it uses heap tree data structure. In this tree each tree is greater than its child node.
* Offer() or add() is used to insert the elements
* Poll() or remove() is used to remove the element.
* Insertion order is not preserved.

Refer Code Snippet: PriorityQueueExample1\_Class

### ArrayDeque<E>

* It is an implementation of double ended queue data structure with array representation.
* It allows both insertion and deletion at both ends because it is implements deque interface.
* It allows duplicate elements
* It does not allow null value
* Initial capacity is 16
* Load factor is 100%

Refer Code Snippet: ArrayDeQueueExample1\_Class

## Utility Collections

* Iterator(Interface)
* ListIterator(interface)
* Map.Entry(Here Entry is an inner interface of MAP Interface)
* Scanner(Class)
* Collections(class)
* Comparator(interface)

### Iterator(interface)

It is used to iterate elements of a collection.

Refer Code Snippet: IteratorInterfaceExample1\_Class

Elements of a collection can also be iterated with enhanced for loop. Refer same example.

### ListIterator(interface)

It is also same as above but it works with ArrayList and LinkedList. It supports both forward and backward directions.

### Map.Entry

Entry is an inner interface of MAP interface. It is used to get the keys and values of MAP While iterating two dimensional collections.

Refer Code Snippet: MAPEntryExample1\_Class

### Scanner Class

This is used to enter the input from console.

Refer Code Snippet: ScannerClassExample1\_Class