**Collections Framework**

What is a Collection ?

● A collection is simply an object that represents a group of

objects, know as elements.

● What is collection framework – it provides set of classes &

interface, that help in managing group of elements.

**Key interface in collection framework**

● **Collection** : the root interface for all other collection types.

● **List**: An oredered collection that can contain duplicate elements

● .(eg. ArrayList, LinkedList).

● **Set**: A collection that cannot contains duplicate elements

● (eg. HashSet, TreeSet).

● **Queue**: A collection designed for elements prior to processing

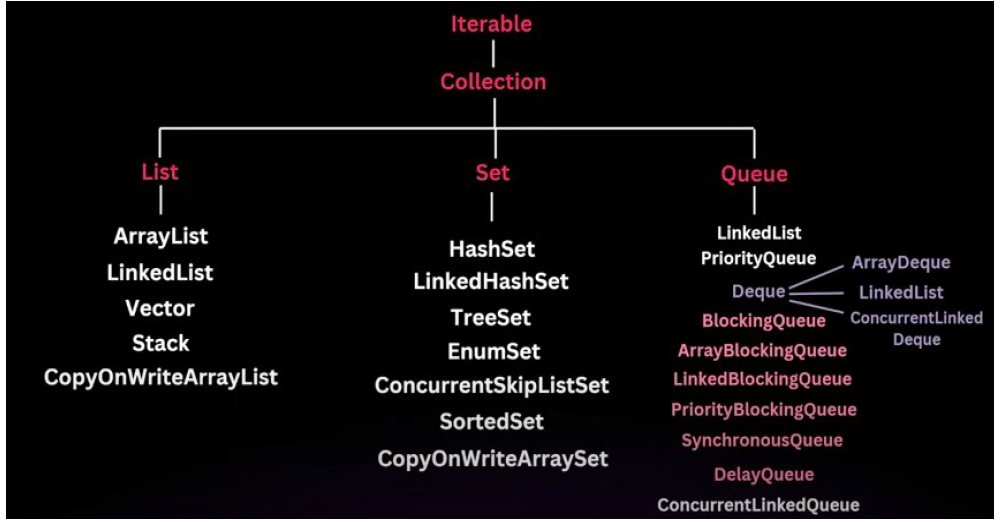
● (eg. PriorityQueue, LinkedList when used as Queue).

● **Deque**:A double-ended queue that allows insertion & removal from both ends

● (eg. ArrayDeque).

● **Map**: An interface that represents a collection of key-value pairs

● (eg. HashMap, TreeMap).

**Hierarchy:**

● **Iterable**: is an root interface of collection hierarchy.

● The collection interface is the root interface of the java collection framework.it

is the most basic interface that defines a group of objects known as elements.

● The collection interface is part of java.util package.

● It is a parent interface that is extended by other collection interfaces like

● List, Set and Queue.

● Since collection is an interface, it cannot be instantiated directly,rather,

● it provides a blueprint for the basic operations that are common to all collections.

● The collection is an interface defines a set of core methods that’s are implemented by all classes that implements the interface.

● These methods are allow for basic operations like adding, removing & checking element is exists in the collection.

**List Interface**

● The List interface is part of java.util package and sub-interface of the Collection interface.

● It provides a way to store an ***oredered collection of elements***.(Known as sequence).

● List can contain **duplicate** elements.

● **When to use List** – When we need ordered collection, In which order elements are coming store

in same order and can have duplicates.

● The List interface is implemented by several classes in the java Collection Framework,

such as ***ArrayList, LinkedList, Vector and Stack.***

***\*\*\*\* Key Features of List Interface***

● **Ordered Preservation**

● **Index based access**

● **Allow duplicates**

**ArrayList**

● An ArrayList is a resizable array, implementation of List interface.

● Unlike arrays in java, which have a fixed size, An arrayList can change its size dynamically as elements are added or removed.

● This flexibility makes it a popular choice when the number of elements in a list is not known in advance.

● **Internal working :** ArrayList can grow & shrink as elements added or removed.

● This dynamic resizing achieved by creating a new array when the

current array is full and copying the elements to the new array.

● \*\* internally, ArrayList is implemented as an array of Object references.

● When you add elements to the ArrayList, you are essentially storing these elements in this internal array.

● \*\* When you create an ArrayList, it has an initial capacity(default 10).

● The capacity refers the size of internal array that can hold elements before needing to resize.

**Creating an ArrayList**

*//Default constructor, creates an ArrayList with initial capacity of 10*

ArrayList<String> list = new ArrayList<>();

*//creating an ArrayList with specified initial capacity*

ArrayList<String> listWithCapacity = new ArrayList<>(20);

*//creating an ArrayList from another Collection*

List<String> anotherList = Arrays.*asList*("Apple", "Banana", "Orange");

ArrayList<String> listFromCollection = new ArrayList<>(anotherList);

**Adding Elements**

ArrayList<String> list = new ArrayList<>();

*//Adding elements to the end of the list*

list.add("Apple");

list.add("Banana");

*//Adding element at a specific index*

list.add(1, "Orange"); *// Orange will be added at index 1, Shifting "Banana" to index2.*

*ArrayList<Integer> list = new ArrayList<>();*

*list.add(1);*

*list.add(2);*

*list.add(3);*

*list.add(0, 0);*

*System.out.println(list); op: [0, 1, 2, 3]*

*//addAll*

*List.addAll(...);*

**Accessing Elements**

*ArrayList<String> fruits = new ArrayList<>();*

*fruits.add("Apple");*

*fruits.add("Banana");*

*fruits.add("Orange");*

*//remove by index*

*fruits.remove(1); // removes the element at index 1("Apple")*

*//remove by Value*

*fruits.remove("Apple"); // removes apple from the list*

ArrayList<Integer> list = new ArrayList<>();

*// add elements*

list.add(1); *// 0 index*

list.add(5); *// 1*

list.add(30); *// 2*

System.*out*.println(list);

*//size check*

System.*out*.println(list.size());

*//get*

System.*out*.println(list.get(0));

*// iterate*

for (int i = 0; i < list.size(); i++) {

System.*out*.println(list.get(i));

}

for (int e: list) {

System.*out*.println(e);

}

*//contains - exists*

System.*out*.println(list.contains(5));

System.*out*.println(list.contains(50));

ArrayList<Integer> list = new ArrayList<>();

*// add elements*

list.add(1); *// 0 index*

list.add(5); *// 1*

list.add(80); *// 2*

*// list.remove(2);*

*// add at specific index*

list.add(2, 50);

*// update specific value at index*

list.set(1, 20);

for (int e: list) {

System.*out*.println(e);

}

System.*out*.println(list.size()); *//0*

*// ArrayList size is 0, but its capacity is 10 (internal array)*

*// size different capacity different,*

*// what is capacity ?*

*// capacity - inside ArrayList array size*

*public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable {*

*private static final int DEFAULT\_CAPACITY = 10;*

*transient Object[] elementData; // array to store elements.*

*private int size;*

*public ArrayList(int initialCapacity) {*

*if (initialCapacity > 0) {*

*this.elementData = new Object[initialCapacity];*

***\*\* Adding Elements***

***When we add an element to the ArrayList, the following steps occur***

***Check capacity:*** Before adding the new element, ArrayList checks if there is enough space in

the internal array(**elementData**).if the array is full, it needs to be resized.

***Resize if neccessary:***if the internal array is full, the ArrayList will create a new array with larger

capacity(usually 1.5 times the current array) and copy the elements from the old array to new

array.

***Add the element:***The new element is then added to the internal array at appropriate index, and

the size is incremented.

***Resizing the Array***

● ***Initial capacity:*** By default the initial capacity is 10.This mean the internal Array

can hold 10 elements before it need to grow.

● ***Growth factor:*** When the internal is full, a new array is created with 1.5 times the

old array.

● ***Copying elements:*** When resizing occurs, all the elements from the old array

copied to new array, which is o(n) operation, where n is no.of elements in the

ArrayList.

**Removing element**

● ***Check Bound:*** The ArrayList first checks if the index within the valid range.

● ***Remove the element:***The element is removed,and all elements to the right of

the removed element are shifted one position to the left to fill the gap.

● ***Reduce size:*** The size is decremented by 1.

**Check for elements**

boolean hasApple = fruits.contains("Apple"); *// returns true if Apple is present*

**Sorting an ArrayList**

*ArrayList<Integer> list = new ArrayList<>();*

*list.add(10);*

*list.add(3);*

*list.add(7);*

*list.add(11);*

*list.add(5);*

*Collections.sort(list); // Sort in natural order*

*System.out.println(list);*

**Access element**

String fruit = fruits.get(0); // Returns “Apple”

**Remove elements**

//Remove by index

list.remove(1); // Remove the element at index 1 – (3)

// Remove by value

List.remove(Integer.valueOf(3)); // Removes value 3

**List Sorting:**

ArrayList<Integer> list = new ArrayList<>();

Collections.sort(list); // default sorting