**1.What is Array?**

Array is indexed collection of fixed number of Homogenous data Elements.

Adv of Arrays is we can represent multiple values by using single variable so that readability of the code is improved.

**2.Limitation of Arrays:**

*1.Arrays are fixed in size*

Once we create an array there is no chance of Increasing/decreasing the size based on our requirement due to this to use array's concept compulsory we should know size in advance which may not possible always.

*2.Array can hold Homogenous data type elements i.e.*

Student[] s=new Student[1000];

s[0]=new Student();

s[1]=new Customer(); //Incompatible Type

*we can solve this problem by*

Object[] a=new Object[1000];

a[0]=new Student();

a[1]=new Customer();

*3.Readymade methods are not available*

For every requirement we have to write code explicitly which increases complexity of programming.

**3.To overcome above problems of arrays we should go for collections concept**

1.Collections are growable based on our requirement.

2.Collections can home Homogenous & Heterogenous data type.

3.Readymade methods are available.

**4.Difference between Array & Collections:**

|  |  |
| --- | --- |
| **Array** | **Collection** |
| Fixed in Size | Growable |
| Not recommended for Memory Management since it wastes memory | Recommended for Memory Management |
| Performance/Speed is Good | Performance/Speed is not good (Slow) |
| No readymade methods are available | Readymade Methods are available |
| It can hold Primitives & Objects | It can hold only primitives. |

**Collection:**

* It is group. If you want to represent group of individual objects as a single entity.
* Technically it is interface.
* It defines most common methods which are applicable for any collection object. It can be considered as set of rules (set of abstract method) which every class need to follow.
* There is no concrete class which implements collection interface directly.

*Collections Framework*: Set of several Classes & Interfaces is called as collection framework. Which can be used to represent group of individual Objects as a single entity.

Collection in C++ is known as Container (To hold Object)



**Difference between List & Set**

|  |  |
| --- | --- |
| **List** | **Set** |
| The list implementation allows us to add duplicate elements. | The set implementation doesn't allow us to add duplicate elements. |
| The insertion order is maintained by the List. | It doesn't maintain the insertion order of elements. |
| List allows us to add any number of null values. | Set allows us to add at least one null value in it. |
| 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(). |
| The List implementation classes are ArrayList, LinkedList and Vector. | The Set implementation classes are HashSet, TreeSet and LinkedHashSet. |

*First 2 differences are important.*

**Insert, Access, remove operation to ArrayList**

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

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

// add elements in the arraylist

animals.add("Cat");

animals.add("Dog");

animals.add("Cow");

System.out.println("ArrayList: " + animals);

// get the element from the arraylist

String str = animals.get(1);

System.out.print("Element at index 1: " + str);

// change the element of the array list

languages.set(2, "JavaScript");

System.out.println("Modified ArrayList: " + languages);

// remove element from index 2

String str = animals.remove(2);

System.out.println("Updated ArrayList: " + animals);

System.out.println("Removed Element: " + str);

}

}

**Insert Elements to HashSet**

import java.util.HashSet;

class Main {

public static void main(String[] args) {

HashSet<Integer> evenNumber = new HashSet<>();

// Using add() method

evenNumber.add(2);

evenNumber.add(4);

evenNumber.add(6);

System.out.println("HashSet: " + evenNumber);

HashSet<Integer> numbers = new HashSet<>();

// Using addAll() method

numbers.addAll(evenNumber);

numbers.add(5);

System.out.println("New HashSet: " + numbers);

}

}

**Example for remove operation***.*

import java.util.HashSet;

class Main {

public static void main(String[] args) {

HashSet<Integer> numbers = new HashSet<>();

numbers.add(2);

numbers.add(5);

numbers.add(6);

System.out.println("HashSet: " + numbers);

// Using remove() method

boolean value1 = numbers.remove(5);

System.out.println("Is 5 removed? " + value1);

boolean value2 = numbers.removeAll(numbers);

System.out.println("Are all elements removed? " + value2);

}

}

**Difference between ArrayList & LinkedList**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| Good for Retrieval | Good for Insertion/Deletion |
| Worst for Insertion/Deletion | Worst for Retrieval |
| Elements will be stored in consecutive memory location Hence retrieval operation will be easy | Elements will not be stored in consecutive memory locations Hence retrieval operation will become complex/difficult |
| Resizable Array | Growable Array | Double linked list |
| 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. |
| ArrayList implements RandomAccess Interface | LinkedList does not implements RandomAccess interface |

**Difference between ArrayList & Vector**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is not synchronized | Vector is synchronized |
| It is not thread safe(Multiple Therad can use it) | It is thread safe(One thread can use at a time) |
| ArrayList increments 50% of current array size if the number of elements exceeds from its capacity. | Vector increments 100% means doubles the array size if the total number of elements exceeds than its capacity. |
| ArrayList is not a legacy class. It is introduced in JDK 1.2 | Vector is a legacy class. (From 1.0 v) |
| ArrayList is fast because it is non-synchronized. | Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object. |
| ArrayList uses the Iterator interface to traverse the elements. | A Vector can use the Iterator interface or Enumeration interface to traverse the elements. |

What is the difference between HashMap and Hashtable?

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |
| 3) | HashMap is not ?thread-safe,? so it is useful for non-threaded applications. | Hashtable is thread-safe, and it can be shared between various threads. |
| 4) | 4) HashMap inherits the AbstractMap class | Hashtable inherits the Dictionary class. |

***Map(I):***

* It is used to represent a group of individual Object as a single Entity as Key & Value Pair then we should go for Map

***Roll No.(Key) Name(Value)***

*101 Durga*

*102 Prem*

*103 Mohit*

* Key & Value both are object
* Duplicate keys are not allowed
* Duplicate Values are possible



1. Implementing HashMap Class

import java.util.Map;

import java.util.HashMap;

class Main {

public static void main(String[] args) {

// Creating a map using the HashMap

Map<String, Integer> numbers = new HashMap<>();

// Insert elements to the map

numbers.put("One", 1);

numbers.put("Two", 2);

System.out.println("Map: " + numbers);

// Access keys of the map

System.out.println("Keys: " + numbers.keySet());

// Access values of the map

System.out.println("Values: " + numbers.values());

// Access entries of the map

System.out.println("Entries: " + numbers.entrySet());

// Remove Elements from the map

int value = numbers.remove("Two");

System.out.println("Removed Value: " + value);

}

}

2. Implementing TreeMap Class

import java.util.Map;

import java.util.TreeMap;

class Main {

public static void main(String[] args) {

// Creating Map using TreeMap

Map<String, Integer> values = new TreeMap<>();

// Insert elements to map

values.put("Second", 2);

values.put("First", 1);

System.out.println("Map using TreeMap: " + values);

// Replacing the values

values.replace("First", 11);

values.replace("Second", 22);

System.out.println("New Map: " + values);

// Remove elements from the map

int removedValue = values.remove("First");

System.out.println("Removed Value: " + removedValue);

}

}

TreeSet & TreeMap > Both need Homogenous Data. All Objects should be inserted according to some sorting order. Hence Comparison Needed & we can compare will be in same object type