**Java Collection**

Java collection framework is a collection of interfaces and classes which helps in storing and processing the data efficiently. [ It is Dynamic Data Structure, is a framework that provides and architecture to store and manipulate the group of objects, all operation to perform on data as searching, sorting, insertion, manipulation, deletion etc.]

Java collections: **Collection is Interface**

**List:** is an Interface -- is an **ordered(sequence) collection**, List **may contain duplicates**. Allows null values. Elements are accessed by their position. using zero-based index.

**Set:** is an Interface -- is a collection that **cannot contain duplicate elements. Does not maintain insertion order. Allows single null values.**

**Map:** is an Interface -- is an object that **maps keys to values**. A map **cannot contain duplicate keys**

**List:**

There are **three main implementations** of **List Interface, ArrayList, LinkedList** and **Vector.**

**ArrayList** --is a class implements List interface and it is based on an Array data structure, it is widely used because of the functionality and flexibility it offers.

Most developers **choose** **ArrayList over Array**. ArrayList is a **resizable array** implementation of the List interface.

Why ArrayList i better than Array.

**Limitations with arrays** is that it has a **fixed length** where as **ArrayList** can **dynamically grow** and **shrink** after **addition** and **removal of elements**.

ArrayList class enables us to use **predefined methods** of it which makes our task easy.

Syntax :

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

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

**LinkedList** -- is a doubly-linked list implementation of the List and Queue interfaces. LinkedList allows for constant-time insertions or removals using iterators, but searched forward and backword but the time it takes to traverse the list directly proportional to the size of the list

Syntax:

LinkedList< String> slist= new LinkedList<String>();

LinkedList<Integer> ilist=new LinkedList<Integer>();

**Vector** -- implements List interface, like ArrayList it also maintains insertion order but it is rarely used in non-thread environments as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements

Syntax:

Vector object = new Vector (int initialCapacity)

**Set:**

There are **three main implementations** of **Set Interface**, **HashSet, TreeSet** and **LinkedHashSet**.

**HashSet** -- stores its elements in hash table, is best performing implementation.

HashSet does **not guarantee** that the **order will remain constant over time**. This class permits null elements, does not permit duplicates.

**TreeSet** -- stores its elements in a red-black tree, orders its elements based on their values.

TreeSet is similar to HashSet except that it sorts the elements in ascending order while HashSet doesn't maintain any order. TreeSet allows null elements.

Like most of the other collection classes this class is also not synchronized. however it can be synchronized explicitly.

**LinkedHashSet** -- implements as hash table with linked list running through it, orders its elements based on order in which they were inserted into set.

It is similar to HashSet and TreeSet, HashSet **doesn't maintain any kind of order of its elements** and TreeSet sorts elements in ascending order.

LinkedHashSet maintains the insertion order. Elements get sorted in the same sequence in which they have been added to the set.

**Map :**

There are **three main implementations** of **Map Interface, HashMap,TreeMap** and **LinkedHashMap**.

**HashMap** -- is used for **storing Key and Value pairs**. It is denoted as HashMap < Key, Value >

This class makes **no guarantee** as to the **order of the map**. it is not ordered collection which means it does not return the keys and values in the same order in which they are inserted into HashMap.

it makes **no guarantees** concerning the order of **iteration**.

**TreeMap** -- it stores its elements in red-black tree, orders its elements based on their values.

It is sorted according to the natural ordering of its keys.

it substantially slower than HashMap.

**LinkedHashMap** -- it orders its elements based on the order in which they were inserted into the Set is an Hash table and linked list implementation of the Map interface. This implementation differ from HaspMap in that it maintains a doubly-linked list running through all of its entries.

This linked list defines the iteration ordering, which means is normally the order in which Keys were inserted into the map, maintains the insertion order.