|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is non-synchronized which means multiple threads can work on ArrayList at the same time. | Vector is synchronized. This means if one thread is working on Vector, no other thread can get a hold of it. |
| ArrayList grow by half of its size when resized | Vector doubles the size of itself by default when grows |
| ArrayList gives better performance as it is non-synchronized | Vector operations gives poor performance as they are thread-safe. |
| Enumeration returned by ArrayList is not fail-fast. | Enumeration returned by Vector is not fail-fast. |
| ArrayList is not a legacy class. It is introduced in JDK 1.2 | Vector is a legacy class. |

**Adil Ashraf**

**L15-4238**

Diffrence between ArrayList and Vector :

Diffrenece between HashSet And SortedSet :

|  |  |
| --- | --- |
| **HashSet** | **SortedSet** |
| HashSet is an unordered collection containing unique elements. | SortedSet is an ordered set collection |
| Hashset use linear search. | The SortedSet<T> must perform a binary search to find the correct location for the new element. |
| Add and Remove cost is O(n) | Add and Remove cost is O(logn) |
| To access elements you can either use an enumerator or use the built-in function to convert the HashSet into a List and iterate through that. | SortedSet does not include hashing, meaning that it has to do linear searches for lookups. |

Difference between HashSet and TreeSet:

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| HashSet stores the object in random order . There is no guarantee that the element we inserted first in the HashSet will be printed first in the output | Elements are sorted according to the natural ordering of its elements in TreeSet. |
| HashSet can store null object. | TreeSet does not allow null object. |
| HashSet take constant time performance for the basic operations like add, remove contains and size | TreeSet guarantees log(n) time cost for the basic operations (add,remove,contains). |
| HashSet is much faster than TreeSet,as performance time of HashSet is constant against the log time of TreeSet for most operations (add,remove ,contains and size) . | Iteration performance of HashSet mainly depends on the load factor and initial capacity parameters. |
| HashSet is difficult to use. | TreeSet is rich in functionality as compare to HashSet. Functions like pollFirst(),pollLast(),first(),last(),ceiling(),lower() etc. makes TreeSet easier to use than HashSet. |
| HashSet uses equals() method for comparison in java | TreeSet uses compareTo() method for maintaining ordering . |

Difference between Array vs List :

|  |  |
| --- | --- |
| **Array** | **List** |
| Array List is a class which extends Abstract Class. | List is an Interface which extends Collection. |
| Array is static in nature i.e its length is fixed. You can’t change its size once it is created. | Array List is dynamic in nature. ArrayList is also called as dynamic array or re-sizable array. Because, it automatically resizes itself if you try to add elements beyond its capacity. |
| Array can hold both primitive data types (int, float….) as well as objects. | ArrayList can hold only objects. If you try to insert primitive data into ArrayList, data is automatically boxed into corresponding wrapper class. |
| ArrayList provides iterators to iterate through its elements. You can also use for loop or for-each loop to iterate an ArrayList. | But to iterate an array, you have to use either for loop or for-each loop. |

Diffrenece between List vs Set:

|  |  |
| --- | --- |
| **List** | **Set** |
| List is an ordered collection it maintains the insertion order | Set is an unordered collection, it doesn’t maintain any order. |
| List allows duplicates | Set doesn’t allow duplicate elements. |
| List allows any number of null values. | Set can have only a single null value at most. |
| If the requirement is to have only unique values then Set is your best bet as any implementation of Set maintains unique values only. | If there is a need to maintain the insertion order irrespective of the duplicity then List is a best option. |

Diffrenece between NevigableSet vs NevigableMap:

|  |  |
| --- | --- |
| **NevigableSet** | **NevigableMap** |
| A NavigableSet extends the SortedSet interface and as well as NavigableMap interface provides methods for navigating elements over your set. | A NavigableMap extends the SortedMap interface which represents a sorted map. |
| The NavigableSet interface represents a Set that is sorted in terms of a client. A set is an unordered collection of distinct elements (i.e it doesn't store duplicates). | The NavigableMap represents a Map, that is additionally sorted in terms of a client. The Map is a data structure that associates its elements with certain keys so that these elements could be obtained by that keys. |
| Use the NavigableSet when you need an ordered Set and operations it does provide. | NavigableMap when you need an ordered Map and it's functionality. |
| The elements of the NavigableSet interface must be mutually comparable | The elements of the NavigableMap interface must not be mutually comparable |