**Khawaja zeeshaur**

**L164383**

**Assignment 2**

**Difference between ArrayList and vectors:**

|  |  |
| --- | --- |
| **Array List** | **Vectors** |
| ArrayList is not synchronized. | Vector is synchronized. |
| 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 fast because it is non-synchronized. | Vector is slow because it is synchronized. If one thread works on a vector, it has acquired a lock on it, which forces any other thread wanting to work on it to have to wait until the lock is released. |
| ArrayList uses the Iterator interface to traverse the elements. | A Vector can use the Iterator interface or Enumeration interface to traverse the elements. |

**Difference between HashSet and SortedSet:**

|  |  |
| --- | --- |
| **HashSet** | **SortedSet** |
| HashSet uses a hash-table to store data. | SortedSet uses a red-back tree which is a balanced binary tree to store data. |
| The HashSet which uses a hash table does the basic operations (i.e. Add, Remove, Search) faster than SortedSet as the complexity of HashSet is O(1) meaning it will do basic operations independent of the size of input data in a constant period of time | SortedSet is log(N) meaning depend on the size of input it will do the basic operations logarithmic. |
| HashSet is an unordered collection containing unique elements. | SortedSet is an ordered set collection. You have many elements you need to store, and you want to store them in a sorted order and also eliminate all duplicates from the data structure. |
| Since there is no Built-in Sort Method, enumerating the elements in a sorted order forces you to copy the items to a different collection (like a List<T>) and sort the resulting list. Sort is typically an O(n log n) operation. | The sorted set ensures that the elements in the set are always in sorted order. Every Add operation places the new element in the correct location in the set. That means Add is an O(log n) operation. |

**Difference between TreeSet and HashSet:**

|  |  |
| --- | --- |
| **TreeSet** | **HashSet** |
| Gives log (n) time cost performance, O (log (n)). | Gives constant time performance, O(1) on operational methods like add(), remove() and contains(). |
| Returns the elements in ascending order or as per the Comparator supplied. Performance low due to added cost of sorting. | Does not return the elements in the same way added |
| No such parameters are available as tree is always balanced | To optimize memory usage Initial capacity and load factor can be set |
| Apart add and remove methods, also exist overloaded subSet(), headSet() and tailSet() | only add() and remove() methods exist |
| Should be preferred only when sorting order is required | Due to higher performance, it should be preferred (when sorting is not much needed) |
| Does not accept null object (throws NullPointerException) | Accepts null object |
| Uses compareTo() method | Uses equals() method |

**Difference between Array and List:**

|  |  |
| --- | --- |
| **Array** | **List** |
| An array is basic functionality provided by Java. | List is part of collection framework in Java. Therefore array members are accessed using [], while ArrayList has a set of methods to access elements and modify them. |
| Array is collection of homogeneous elements. | List is collection of heterogeneous elements. |
| For Array memory allocated is static and continuous. | For List memory allocated is dynamic and Random. |
| User need not have to keep in track of next memory allocation. | User has to keep in Track of next location where memory is allocated. |
| bounds checking is removed | bounds checking is performed |
| Simple fixed sized arrays that we create in Java. | One need not to mention the size of list while creating its object. Even if we specify some initial capacity, we can add more elements. |
| Array can contain both primitive data types as well as objects of a class depending on the definition of the array. | List only supports object entries, not the primitive data types. |

**Difference between Set and List:**

|  |  |
| --- | --- |
| **Set** | **List** |
| A collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. As implied by its name, this interface models the mathematical set abstraction. | An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list. |
| The Set classes do not maintain insertion order. They may optionally impose a specific order (as with SortedSet), but typically have an implementation-defined order based on some hash function (as with HashSet). Since Sets are accessed by key, duplicates are not allowed. | All of the List classes maintain the order of insertion. They use different implementations based on performance and other characteristics (e.g. ArrayList for speed of access of a specific index, LinkedList for simply maintaining order). Since there is no key, duplicates are allowed. |
| Set interface does not have any legacy class. | List interface has one legacy class called Vector. |
| we can use iterator() method to access Set class elements. | The listIterator() method can only be used to cycle through the elements within List Classes |
| Set interface does not have its own method so Set uses Collection interface methods only. | List interface has its own methods defined. |

**Difference between NavigableSet and NavigableMap:**

|  |  |
| --- | --- |
| **NavigableSet** | **NavigableMap** |
| 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. NavigableMap offers methods for obtaining the elements of the map relative to other ones, in the context of a sorted collection. |
| 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 those keys. |