**Assignment**

**Name: Moeed Raza Khan (16L-4187)**

1. **ArrayList VS Vector**

Both vector and arraylist preserves the insertion order and implements the list interface. But still there are many differences in both of them, some of the differences are:

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| Arraylist will increase its size by 50 percent if a new element has been added to it | If even a single element is added to it when it has reached its full capacity it will increase its size by doubling the size of its array. |
| Multiple threads can work on arrayList on same time which means arraylist is not synchronized | Whereas, only a single thread at a time can access the code in vector which means it is synchronized. |
| Arraylist is much faster compared to vector because no thread acquires lock while working in the arrayList | Vector are much slower due to synchronization because the thread working on a vector acquires lock which means other threads have to wait for the lock to open. |
| Traversal can be done only using iterator. | Traversal can be done using Both enumeration and iterator. |
| ArrayList is newer and faster | Vector is old |
|  |  |

1. **HashSet Vs SortedSet**

|  |  |
| --- | --- |
| **HashSet** | **SortedSet** |
| HashSet is a class that implements the interface | SortedSet is an Interface that extends Set interface |
| Objects are inserted based on their hashcode | All elements of a SortedSet must implement the comparable interface |
| HashSet uses hashtable data structure | Some of the methods of sortedSet are comparator(),first(),headset(),last(). |
|  |  |
|  |  |
|  |  |

1. **HashSet Vs TreeSet**

Both of them implements Set interface so both of them does not allow duplicates to store

**Differences are:**

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| It offers constant time for the search, add and remove operations. | It offers log (n) time for the search, add and remove operations. |
| It does not maintain any order of elements | It maintains ascending order by default |
| Hashset allows null | It does now allow null and throws null pointer exception |
| Hashset uses hashmap internally to store its element | TreeSet uses TreeMap internally to store its element |
| HashSet uses equals() method to compare the element to remove duplicates | TreeSet uses compare() or compareTo() methods to compare the element to remove duplicates |
|  |  |

1. **Array Vs List**

|  |  |
| --- | --- |
| **Array** | **List** |
| Group of elements which have a similar data type | List is a child interface of collection, meaning it extends collection interface |
| Array is directly inherited from the class object | It represents group of single object as a single entity |
| Size of an array must be specified by only an int value | Duplication is allowed in List |
| Array makes the code optimized as we can retrieve or sort the data efficiently | It preserves the insertion order |
|  | As it is an interface it can be implemented using ArrayList, Linkedlist and vector |

1. **List Vs Set**

List and Set both are interfaces. They both extends Collection interface.

|  |  |
| --- | --- |
| **List** | **Set** |
| List allows duplicate elements | All elements of set must be unique |
| List interface can be implemented by LinkedList, ArrayList and vector | Set interface can be implemented by HashSet, Linked HashSet and TreeSet |
| Order is maintained in all the implementations of the list | Order is maintained in only few of the implementations of the set such as TreeSet |
| List allows any number of null values | Set can have a single null value at most |
|  |  |
|  |  |
|  |  |

1. **NavigableSet vs NavigableMap**

|  |  |
| --- | --- |
| **NavigableSet** | **NavigableMap** |
| Child interface of SortedSet which is a child interface of Set which is a child interface of collection | Child interface of SortedMap which is a child interface of Map |
| Navigation method also available in addition to the sorting methods | Navigation method also available in addition to the sorting methods |
| Classes that implement it are [TreeSet](http://quiz.geeksforgeeks.org/treeset-in-java/) and [ConcurrentSkipListSet](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentSkipListSet.html) | One of the Class that implement it is TreeMap |
| Some of the methods are   1. **Lower(E e) :** Returns the greatest element in this set which is less than the given element or NULL if there is no such element. 2. **Floor(E e ) :** Returns the greatest element in this set which is less than or equal to given element or NULL if there is no such element. 3. **Ceiling(E e) :** Returns the least element in this set which is greater than or equal to given element or NULL if there is no such element. 4. **Higher(E e) :** Returns the least element in this set which is greater than the given element or NULL if there is no such element. 5. **pollFirst() :** Retrieve and remove the first least element. Or return null if there is no such element. 6. **pollLast() :** Retrieve and remove the last highest element. Or return null if there is no such element. | Some of the methods are:   1. **lowerKey(Object key) :** Returns the greatest key strictly less than the given key, or if there is no such key. 2. **floorKey(Object key) :** Returns the greatest key less than or equal to the given key, or if there is no such key. 3. **ceilingKey(Object key) :** Returns the least key greater than or equal to the given key, or if there is no such key. 4. **higherKey(Object key) :** Returns the least key strictly greater than the given key, or if there is no such key. 5. **descendingMap() :** Returns a reverse order view of the mappings contained in this map. 6. **headMap(object toKey, boolean inclusive) :** Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey. 7. **subMap(object fromKey, boolean fromInclusive, object toKey, boolean toInclusive) :** Returns a view of the portion of this map whose keys range from fromKey to toKey. |