**Name: Ahmed Ijaz**

**Roll no: l16-4114**

**References:** [**https://www.geeksforgeeks.org/**](https://www.geeksforgeeks.org/)

**https://stackoverflow.com/**

**Assignment 2**

**Q No 1:**

**Array List Vs Vector**

* **Synchronization :** Vector is **synchronized**, which means only one thread at a time can access the code, while arrayList is**not synchronized**, which means multiple threads can work on arrayList at the same time. For example, if one thread is performing an add operation, then there can be another thread performing a remove operation in a multithreading environment.  
  If multiple threads access arrayList concurrently, then we must synchronize the block of the code which modifies the list structurally, or alternatively allow simple element modifications. Structural modification means addition or deletion of element(s) from the list. Setting the value of an existing element is not a structural modification.
* **Performance:** **ArrayList is faster**, since it is non-synchronized, while vector operations give slower performance since they are synchronized (thread-safe). 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.
* **Data Growth:**ArrayList and Vector **both grow and shrink dynamically**to maintain optimal use of storage – but the way they resize is different. ArrayList increments 50% of the current array size if the number of elements exceeds its capacity, while vector increments 100% – essentially doubling the current array size.
* **Traversal:**Vector can use both enumeration and iterators for traversing over elements of vector while ArrayList can only use **Iterator** for traversing.

**Q No 2**

**Hash Set Vs Sorted Set**

Both **HashSet** and **SortedSet** are implementing data structure set ) which is a data structure holding unique elements.

The main difference between them is the underlying data structure they use to store data. HashSet uses a **hash-table** while SortedSet uses a **red-back tree** which is a balanced binary tree.

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 while the complexity of SortedSet is log(N) meaning depend on the size of input it will do the basic operations logarithmic. for example if the size of your input data is 1,000 then then the program does the basic operations in 10 steps and if it is 1,000,000 the program does the basic operations in 20 steps.

Conclusion: Use HashSet **if you don't need the elements to be sorted** otherwise use SortedHash. It means Using HashSet is **preferable** unless you need sorting.

**Q No 3**

**Hash Set Vs Tree Set**

* **Speed and internal implementation**  
  [HashSet](http://www.geeksforgeeks.org/hashset-in-java/) : For operations like search, insert and delete. It takes constant time for these operations on average. HashSet is faster than TreeSet. HashSet is Implemented using a [hash table](https://www.geeksforgeeks.org/hashing-set-1-introduction/).

[TreeSet](https://www.geeksforgeeks.org/treeset-in-java-with-examples/) : TreeSet takes O(Log n) for search, insert and delete which is higher than HashSet. But TreeSet keeps sorted data. Also, it supports operations like higher() (Returns least higher element), floor(), ceiling(), etc. These operations are also O(Log n) in TreeSet and not supported in HashSet. TreeSet is implemented using a Self Balancing Binary Search Tree ([Red-Black Tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/)). TreeSet is backed by TreeMap in Java.

* **Ordering**  
  Elements in HashSet are not ordered. TreeSet maintains objects in Sorted order defined by either Comparable or Comparator method in Java. TreeSet elements are sorted in ascending order by default. It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc.
* **Null Object**  
  HashSet allows null object. TreeSet doesn’t allow null Object and throw NullPointerException, Why, because TreeSet uses compareTo() method to compare keys and compareTo() will throw java.lang.NullPointerException.
* **Comparison**  
  HashSet uses [equals()](https://www.geeksforgeeks.org/overriding-equals-method-in-java/) method to compare two object in Set and for detecting duplicates. TreeSet uses compareTo() method for same purpose.  
  If equals() and compareTo() are not consistent, i.e. for two equal object equals should return true while compareTo() should return zero, than it will break contract of Set interface and will allow duplicates in Set implementations like TreeSet

**Q No 4**

**Array Vs List**

It is rare, in reality, that you would want to use an array. Definitely use a List<T> any time you want to add/remove data, since resizing arrays is expensive. If you know the data is fixed length, and you want to micro-optimise for some **very specific** reason (after benchmarking), then an array may be useful.

List<T> offers a lot more functionality than an array (although LINQ evens it up a bit), and is almost always the right choice. Except for params arguments, of course. ;-p

As a counter - List<T> is one-dimensional; where-as you have have rectangular (etc) arrays like int[,] or string[,,] - but there are other ways of modelling such data (if you need) in an object model.

You have to resize an Array but in case of List no resizing is required manually.

**Q No 5**

**List Vs Set**

**List<E>:**

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.

[**Set<E>:**](http://docs.oracle.com/javase/8/docs/api/java/util/Set.html)

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.

**Q NO 6**

**NavigableMap Vs NavigableSet**

NavigableMap is an extension of [SortedMap](http://www.contribute.geeksforgeeks.org/sortedmap-in-java-with-examples/" \t "_blank) which provides convenient navigation method like lowerKey, floorKey, ceilingKey and higherKey, and along with these popular navigation method it also provide ways to create a Sub Map from existing Map in Java e.g. headMap whose keys are less than specified key, tailMap whose keys are greater than specified key and a subMap which is strictly contains keys which falls between toKey and fromKey.

An example class that implements NavigableMap is [TreeMap](http://www.contribute.geeksforgeeks.org/hashmap-treemap-java/" \t "_blank).

**Methods** of NavigableMap:

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.
8. **tailMap(object fromKey, boolean inclusive)** : Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey.

NavigableSet represents a navigable set in [Java Collection Framework](https://www.geeksforgeeks.org/collections-in-java-2/). The NavigableSet interface inherits from the [SortedSet interface](https://www.geeksforgeeks.org/sortedset-java-examples/" \t "_blank). It behaves like a SortedSet with the exception that we have navigation methods available in addition to the sorting mechanisms of the SortedSet. For example, NavigableSet interface can navigate the set in reverse order compared to the order defined in SortedSet.

The classes that implement this interface are, [TreeSet](http://quiz.geeksforgeeks.org/treeset-in-java/) and [ConcurrentSkipListSet](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentSkipListSet.html)

**Methods** of NavigableSet (Not in SortedSet):

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.