**Advanced Programming**

**Assignment-II**



**Collections**

**Submitted to:** Mr. Ahmad Shahwaiz

**Submitted by**:

Muhammad Zain Jaffery 15L-4125

# **Differences**

**ArrayList vs Vector**

|  |  |  |
| --- | --- | --- |
|  | ArrayList | Vector |
| 1 | Size increased by 50%. | Size increased by 100% (i.e. double). |
| 2 | Since JDK 1.2 | Since JDK 1.0 |
| 3 | Used less memory. | Uses more memory. |
| 4 | Not Thread Safe (i.e. not synchronized). | Thread Safe (i.e. synchronized). |
| 5 | Fast. | Slow. |

**Note**: ArrayList should be preferred over Vectors in non-thread environment.

**HashSet vs SortedSet**

|  |  |  |
| --- | --- | --- |
|  | HashSet | SortedSet |
| 1 | Class | Interface |
| 2 | No insertion order is preserved | Insertion order is preserved. |
| 3 | Data structure used behind HashSet is Hash table. | Does not use Hash table. |
| 4 | Search operation available for sorting. | No search operation. As it is already sorted |
| 5 | By default load factor=0.75 (i.e. a new Hashset is created when 75% is filled) | Does not work on any load factor measures. |
| 6 | Insertion/deletion/search in O(1) | Insertion/deletion/search depends upon the sorting algorithm implemented behind. |

Note: if we try to add duplicate values in sets, the compiler does not show any compile time/run time error rather it returns false in the .add(value) method.

Note: hashset is the best choice if our frequent operation is search operation.

**HashSet vs TreeSet**

|  |  |  |
| --- | --- | --- |
|  | HashSet | TreeSet |
| 1 | For operations like search, insert and delete. It takes constant time for these operations on average | TreeSet takes O(Log n) for search, insert and delete which is higher than HashSet. |
| 2 | No insertion order is preserved | TreeSet keeps sorted (ordered) data. |
| 3 | Data structure used behind HashSet is Hash table. | TreeSet is implemented using a Self Balancing Binary Search Tree (Red-Black Tree) |
| 4 | Search operation available for sorting. | it supports operations like higher() (Returns least higher element), floor(), ceiling(), etc |
| 5 | HashSet allows null object | TreeSet does not allows null object. As it has to compare the objects. |

**HashSet vs TreeSet**

|  |  |  |
| --- | --- | --- |
|  | HashSet | TreeSet |
| 1 | HashSet is Implemented using a hash table. Elements are not ordered. The add, remove, and contains methods have constant time complexity O(1). | TreeSet is implemented using a tree structure(red-black tree in algorithm book). The elements in a set are sorted, but the add, remove, and contains methods has time complexity O(log (n)) |
| 2 | Hash set not preserved sorted order. | Treeset preserved sorted order. |
| 3 | HashSet is backed by HashMap. | TreeSet is backed by TreeMap in Java. |

**List vs Set**

|  |  |  |
| --- | --- | --- |
|  | List | Set |
| 1 | Lists generally allow duplicate objects. Lists must be ordered, and are therefore accessible by index. | Sets do not allow duplicate objects. Most implementations are unordered, but it is implementation specific. |
| 2 | Implementation classes include: Array List, LinkedList, Vector | Implementation classes include: HashSet (unordered), Linked HashSet (ordered),Tree Set (ordered by natural order or by provided comparator) |

**NavigableSet vs NavigableMap**

|  |  |  |
| --- | --- | --- |
|  | NavigableSet | NavigableMap |
| 1 | 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. |
| 2 | For example, it has pollFirst() method that retrieves and removes the lowest element in the set | For instance, it has ceilingEntry(K obj) method that returns an entry with the lowest key that is greater or equal to the object passed as an argument. Or the lastEntry() method, that returns an entry with the greatest key, and so on. |
| 3 | 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. |