|  |  |  |
| --- | --- | --- |
| **ArrayList vs Vector** | | |
| **SR #** | **ArrayList** | **Vector** |
| 1 | ArrayList implements List interface and maintains insertion order; however internal implementation is different than that of Vector. | Vector implements List interface and maintains order of insertion; however internal implementation is different than that of Array List. |
| 2 | No Synchronization in ArrayList. This is key difference between Arraylist and vector | Synchronization is present in Vector. This is key difference between Arraylist and vector |
| 3 | Due to no synchronization more than one thread can access the ArrayList which makes it less thread safer and more venerable. | Due to synchronization not more than one thread can access the Vector which makes it more thread safer and difficult o exploit. |
| 4 | Fast Access to elements due to no exclusive thread lock and waiting of threads in queue | Slow Access to elements due to exclusive thread locking mechanism and waiting of threads in queue |
| 5 | Dynamic growth and shrinking of size.  ArrayList grows by 50% of current size | Dynamic growth and shrinking of size.  Vector grows by 100% of current size; i.e. doubling the current size. |
| 6 | Use only Iterator as traversal tool. | Use Iterator and enumeration as traversal tool |
| 7 | Arraylist is not a legacy class. Introduced later and should be used if sure about single thread access. | Vector is a legacy class. |
| References:  <https://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html>  <https://www.geeksforgeeks.org/vector-vs-arraylist-java/> | | |

**Advance Programming**

**Assignment -2**

**M Ibrahim Khwaja L14-4156**

|  |  |  |
| --- | --- | --- |
| **Differences Between HashSet & Sorted Set** | | |
| **SR #** | **HashSet** | **SortedSet** |
| 1 | The HashSet is an unordered collection of unique items/objects. | A collection of objects/items that contains no duplicate elements and arranged in orderly manner. |
| 2 | HasSet has contiguous storage and can be accessed directly by key. | SortedSet has contiguous storage can be accessed directly by key. |
| 3 | Time Complexity in terms of accessing data is O(1). Faster access. | Time Complexity in terms of accessing data is O(log n) . Slow access as compare to Sorted Set. |
| 4 | HashSet uses a hash-table as an underlying structure for its implementation. | SortedSet uses a red-black tree a balanced binary tree as an underlying structure for its implementation. |
| 5 | It’s better to use hash set when there is no need of sorted elements. | It’s better to use sorted set when there is need of sorted elements. |
| References: <https://docs.oracle.com/javase/7/docs/api/java/util/SortedSet.html>  <http://geekswithblogs.net/BlackRabbitCoder/archive/2011/06/16/c.net-fundamentals-choosing-the-right-collection-class.aspx> | | |

|  |  |  |
| --- | --- | --- |
| **Differences Between TreeSet & HashSet** | | |
| **SR #** | **TreeSet** | **SortedSet** |
| 1 | TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. | A collection of objects/items that contains no duplicate elements and arranged in orderly manner. |
| 2 | TreeSet does not allow to insert Heterogeneous objects. It will throw classCastException at Runtime if trying to add hetrogeneous objects | SortedSet uses a red-black tree a balanced binary tree as an underlying structure for its implementation. |
| 3 | In TreeSet add, remove and search take O(Log n) where as for printing it takes O(n) times. | Time Complexity in terms of accessing data is O(log n) . Slow access as compare to Sorted Set |
| 4 | All elements inserted into a sorted set must implement the Comparable interface Furthermore, all such elements must be *mutually comparable*  *i.e epm1.compare(emp2)* | All elements inserted into a sorted set must implement the Comparable interface Furthermore, all such elements must be *mutually comparable*  *i.e epm1.compare(emp2)* |
| References: <https://docs.oracle.com/javase/6/docs/api/java/util/TreeSet.html>  <https://docs.oracle.com/javase/6/docs/technotes/guides/collections/index.html> | | |

|  |  |  |
| --- | --- | --- |
| **Differences Between Array & List** | | |
| **SR #** | **Array** | **List** |
| 1 | Array is a fixed length data structure whose length cannot be modified once array object is created. | List is a data structure whose length can be modified once list object is created. |
| 2 | Unlike sets, lists typically allow duplicate elements | Unlike sets, lists typically allow duplicate elements |
| 3 | Array is no growable, it has fixed size. | List is dynamically growable in nature. |
| 4 | Length variable is used to determine the length of the Array. | List uses size () method to determine the size of the ArrayList, it is rather different from determining the length of the Array. |
| 5 | Iterating over an array is faster than iterating over a List. | Iterating over an List is slower than iterating over an Array. |
| References:  <http://www.differencebetween.net/technology/software-technology/difference-between-arraylist-and-vector/>  <https://www.javatpoint.com/java-list> | | |

|  |  |  |
| --- | --- | --- |
| **Differences Between List & Set** | | |
| **SR #** | **List** | **Set** |
| 1 | List is a data structure whose length can be modified once list object is created and can have duplicate elements. | Set is a collection that contains no duplicate elements. |
| 2 | New methods are defined inside List interface. | No new methods are defined inside Set interface, so we have to use Collection interface methods only with Set subclasses. |
| 3 | Use List when order is important | Use sets when all you’re interested in is membership |
| 4 | If an element is added again it, will added. | if same element is added again, there won’t be any compile-time or runtime error, just that add() method returns false. |
| References: <https://docs.oracle.com/javase/8/docs/api/java/util/List.html> | | |

|  |  |  |
| --- | --- | --- |
| **Differences Between NavigiableSet & NavigiableMap** | | |
| **SR #** | **NavigiableSet** | **NavigiableMap** |
| 1 | NavigableSet represents a navigable set in Java Collection Framework. The NavigableSet interface inherits from the SortedSet interface. It behaves like a SortedSet with the exception that we have navigation methods available in addition to the sorting mechanisms of the SortedSet. | NavigableMap is an extension of SortedMap which provides convenient navigation method like lowerKey, floorKey, ceilingKey and higherKey |
| 2 | A NavigableSet can be accessed and traversed in either ascending or descending key order. | A NavigableMap may be accessed and traversed in either ascending or descending key order. |
| 3 | Navigable Set lies under collections. | Unlike Navigable Set Navigable map doesn’t lies under collections. |
| 4 | NavigableSet has different methods like :  NavigableSet lower() method in Java  NavigableSet higher() method in Java  NavigableSet floor() method in Java  NavigableSet subSet() method in Java  NavigableSet ceiling() method in Java  NavigableSet headSet() method in Java | NavigableMap has different methods like:  NavigableMap headMap() in Java  NavigableMap put() Method in Java  NavigableMap firstEntry() method in Java  NavigableMap lowerEntry() method in Java  NavigableMap pollFirstEntry() method in Java  NavigableMap ceilingEntry() method in Java |
| References: <https://www.geeksforgeeks.org/navigableset-java-examples/> | | |

