ArrayList vs Vector

|  |  |
| --- | --- |
| ArrayList | Vector |
| 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 not a legacy class. It is introduced in JDK 1.2. | Vector is a legacy class. |
| ArrayList is fast because it is non-synchronized. | Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object. |
| ArrayList uses the Iterator interface to traverse the elements. | A Vector can use the Iterator interface or Enumeration interface to traverse the elements. |

Array vs List

|  |  |
| --- | --- |
| Array | List |
| Array is a data structure | List is a java interface |
| Array is fixed length | List is variable length when implemented by another class |
| Arrays can store both objects and primitive datatypes as well | List can only store objects. Since Java 5, primitives are automatically converted in objects. |
| We cannot change length of Array once created. | Length can be changed in List |

HashSet vs SortedSet

|  |  |
| --- | --- |
| HashSet | SortedSet |
| HashSet is a Java class | SortedSet is a Java interface |
| HashSet uses Hashtable for storage | Method of storage is specified by the class that implements this interface |
| Stores elements in the order specified by the hashing | The elements are ordered either by using a natural ordering or by using a Comparator. |
| HashSet is best approach when searching is required | SortedSet is best approach when sorting is required |
| Iterator will traverse the HashSet in hasing order | Iterator will traverse the SortedSet in an ascending order |

List vs Set

|  |  |
| --- | --- |
| List | Set |
| HashSet is a Java class | SortedSet is a Java interface |
| HashSet uses Hashtable for storage | Method of storage is specified by the class that implements this interface |
| Stores elements in the order specified by the hashing | The elements are ordered either by using a natural ordering or by using a Comparator. |
| Iterator will traverse the HashSet in hasing order | Iterator will traverse the SortedSet in an ascending order |

TreeSet vs HashSet

|  |  |
| --- | --- |
| TreeSet | HashSet |
| Java TreeSet class maintains ascending order. | HashSet doesn't maintain the insertion order. Elements are inserted on the basis of their hashcode. |
| Doesn’t Allow Null elements | Allows Null elements |
| TreeSet implements NavigableSet which in turn implements SortedSet | HashSet extends AbstractSet which implements Set interface |
| TreeSet is best approach when sorting is required | HashSet is best approach when searching is required |
| Iterator will traverse the SortedSet in an ascending order | Iterator will traverse the HashSet in hasing order |

NavigableSet\_vs\_NavigableMap

|  |  |
| --- | --- |
| NavigableSet | NavigableMap |
| Java TreeSet class maintains ascending order. | HashSet doesn't maintain the insertion order. Elements are inserted on the basis of their hashcode. |
| Doesn’t Allow Null elements | Allows Null elements |
| TreeSet implements NavigableSet which in turn implements SortedSet | HashSet extends AbstractSet which implements Set interface |
| TreeSet is best approach when sorting is required | HashSet is best approach when searching is required |
| Iterator will traverse the SortedSet in an ascending order | Iterator will traverse the HashSet in hasing order |