# **Java Collections**

This document covers the differences of the following:

1. ArrayList vs Vector
2. HashSet vs SortedSet
3. TreeSet vs HashSet
4. Array vs List
5. List vs Set
6. NavigableSet vs NavigableMap

**ArrayList vs Vector**

Both are derived classes of class ‘List’ and represent group of individual objects as a single entity where *duplicates are allowed,* and insertion order is *preserved*.

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is not synchronized. That means multiple threads can work on arrayList at the same time. | Vector is synchronized. That means only one thread at a time can access the code. |
| 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 exceed its capacity. |
| ArrayList is not a legacy class. It was introduced in JDK 1.2. | Vector is a legacy class. It was introduced in JDK 1.0. |
| 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 of enumeration to traverse the elements. |

**HashSet vs SortedSet**

Both are derived classes of class ‘Set’ and represent group of individual objects as a single entity *where duplicates are NOT allowed and NO insertion order is preserved.*

|  |  |
| --- | --- |
| **HashSet** | **SortedSet** |
| The **HashSet<T>** is an unordered collection of unique items. This means that the collection cannot have duplicates and no order is maintained. | SortedSet is an ordered set collection. Collection cannot have duplicates but the order is maintained. |
| HashSet uses a **hash-table.** | SortedSet uses a **red-back tree** which is a balanced binary tree. |
| It has the standard collection operations *Add, Remove and Contains*, but since it uses a hash-based implementation, these operations are O (1). | As it does not include hashing, meaning that it must do linear searches for lookups. Therefore, the SortedSet is much slower. It is O (log n). |
| There is contiguous storage in HashSet. | Storage is not contiguous in SortedSet. |

**TreeSet vs HashSet**

Both are derived classes of class Set and represent group of individual objects as a single entity *where duplicates are NOT allowed and NO insertion order is preserved.*

|  |  |
| --- | --- |
| **TreeSet** | **HashSet** |
| Main feature of TreeSet is sorting without duplicates. | HashSet is just general-purpose collection for storing object. Use it as default Set implementation if you need a fast, duplicate free collection. |
| Backed up by NavigableMap and by default it uses TreeMap. | Implemented using HashMap. |
| TreeSet maintains order of elements. | HashSet does not maintain any order. |
| TreeSet is bit slower because of sorting operation. | HashSet is fastest among Set’s children. |
| It provides guaranteed O(log n) time for common operations like add, remove and contains. | It offers constant time performance, that is, O(1) for add, contains and remove given hash function uniformly distribute elements in bucket. |
| TreeSet does not allow null and throws [java.lang.NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html) when you will insert null into it. | Allows null. |
| TreeSet uses *compareTo()* for maintaining order. So it should be consistent with *equals()* method. | HashSet uses *equals()* method for comparison. |

**Array vs List**

An array (something like int[]) is a **built in type** while List is a regular class part of the **Java standard library.**

|  |  |
| --- | --- |
| **Array** | **List** |
| An array is basic functionality provided by Java. | List is part of collection framework in Java. |
| Array members are accessed using [] | List has a set of methods to access elements and modify them. |
| It has fixed size. Its mandatory to specify size at the time of creation. | It is growable in nature. You don not have to specify size of list while creating its object. |
| It provides *length* variable which denotes length of Array. | List has *size()* method to calculate size. |
| It can contain both primitive data types as well as objects of a class depending upon its definition. | It supports only object entries, not primitive data types. |
| In case of primitive data types, actual values are stored in contiguous locations. But if objects are defined, allocation is like List then. | Its members are always referencing to objects at different memory locations. |
| We cannot use Generics along with Array because its instance knows what type it can hold and throws ArrayStoreException. | Allows Generics to ensure type-safety. |
| Use *assignment operator* to store element in Array. | Use *add()* method to insert element in ArrayList. |
| Immutability cannot be achieved. | It can be achieved using *unmodifiable* keyword. |

**List vs Set**

List and Set both are interfaces. They both extend Collection interface in Java.

|  |  |
| --- | --- |
| **List** | **Set** |
| It is an ordered collection and maintains insertion order. | It is an unordered collection and does not maintain insertion order though it has *LinkedHashSet* as an implementation that maintains order. |
| It allows duplicates. | Does not allow duplicates. It would replace the existing value if you try to insert a duplicate element. |
| It allows multiple null values. | It allows only a single null value. |
| *ListIterator* can be used to traverse a list in both forward and backward direction. | We use *Iterator* to traverse a Set. |
| List collection has one legacy class, i.e., Vector. | Set collection does not have any legacy class. |
| Can be accessed by index. | Cannot be accessed through index. |
| Implementations include ArrayList, LinkedList etc. | Implementations include HashSet, TreeSet, LinkedHashSet etc. |

**NavigableSet vs NavigableMap**

Both are used for traversal.

|  |  |
| --- | --- |
| **NavigableSet** | **NavigableMap** |
| Inherits from SortedSet. | Extension of SortedMap. |
| It can navigate the Set in reverse order compared to the order defined in SortedSet. | It also provides ways to create a Sub Map from existing Map in Java. |
| Methods:   * Lower(E e) * Floor(E e) * Ceiling(E e) * Higher(E e) | Methods:   * lowerKey(Object key) * floorKey(Object key) * ceilingKey(Object Key) * higherKey(Object key) |