# **Advanced Programming**

# **Assignment No: 2**

# **Java-Collections**

**Name: Mehrin Athar**

**Roll No: L154040**

## **ArrayList vs. Vector**

|  |  |
| --- | --- |
| **Vector** | **ArrayList** |
| Vector is a legacy class. | ArrayList is not a legacy class |
| Vector is synchronized means only one thread working on a vector at a time. | ArrayList is not synchronized hence multiple threads may be manipulating it at a time. |
| The default size in vectors, if not mentioned is 10. | By default, if size of ArrayList not given, an empty ArrayList is created which can be grown later on. |
| Vectors are slow since they are synchronized. | ArrayList is faster than vectors because no synchronization mechanisms are being employed. |
| Vectors are thread safe i.e. in a multithreaded environment, vectors ensure that no unwanted interaction or manipulation of the shared data structure is being done. | ArrayList don’t ensure thread safety. |
| Vectors define a value by which it grows, if growing size is not specified in Vector constructor, by default upon increase in total number of elements, its size is doubled. | ArrayList don’t define the growing /increment size |

***Legacy class:***

*Early versions of java did not include Collections framework. Instead it defined several classes and one interface to store objects. When collection came these classes still were able to support the Collection interfaces.*

## **Array vs. List**

|  |  |
| --- | --- |
| **Array** | **List** |
| Array’s direct superclass is Object. | List is child interface of Collection. |
| Array is fixed in size, its size must be specified for allocation. | Lists are dynamic and are extendable in nature and size may not be specified at creation time. |
| Array is a basic functionality for usage by java using [ ] operator and doesn’t come with already there manipulation methods. | List later implemented by lower classes like ArrayList provide a number of methods for accessing and manipulating elements. |
| Based on the definition of Array, it can support both primitive data types like int, char etc. along with objects of some specific custom class. | Lists don’t support primitive data types, they only support objects. |
| In case of arrays, elements are stored at contiguous memory locations. | Elements of a List are references to objects at different memory locations. |

## **HashSet vs. SortedSet**

|  |  |
| --- | --- |
| **HashSet** | **SortedSet** |
| HashSet uses HashTable for storing elements which in turn is a Map. | SortedSet uses balanced binary tree as the underlying data structure to store data. |
| In HashSet, memory is contiguously allocated. | In SortedSet memory is not contiguous. |
| Find an element is HashSet takes constant amount of time as it uses HashTable for storage. | SortedSet is not so efficient in lookup and takes linear amount of time. |
| In HashSet, elements are not in a sorted order. | SortedSet renders the ability to have sorted ordering of elements. |

## **TreeSet vs. HashSet**

|  |  |
| --- | --- |
| **TreeSet** | **HashSet** |
| TreeSet uses some tree data structure for storage of elements. | HashSet uses HashTable for storage purposes. |
| TreeSet uses a tree data structure hence it’s add, remove and contains functions take more time i.e. log (n), than a HashSet which takes constant time. | HashTable is much faster than TreeSet and search can be performed in constant amount of time. |
| TreeSet provides ordering guarantee as it implements the Set interface, by default its gives the order as it occurs or using comparator ascending or descending. | HashSet does not guarantee ordering of elements over the time. |
| TreeSet provides methods to manipulate ordered set like first() to get lowest, first element etc. | HashSet provides no such utilities to handle ordered sets. |

## **List vs. Set**

|  |  |
| --- | --- |
| **List** | **Set** |
| List can contain duplicate elements. | Set does not allow duplicate elements. |
| List maintains order of insertion of elements over the time. | Set could maintain order of elements but it depends upon implementation like HashSet does not guarantee order while SortedSet does. |
| Lists allow positional access using get method where index can be specified. | Sets don’t allow positional access i.e. accessing a particular index element. |
| List allows null values | Set doesn’t allow null values. |

## **NavigableSet vs. NavigableMap**

|  |  |
| --- | --- |
| **NavigableSet** | **NavigableMap** |
| NavigableSet interface inherits from SortedSet interface. | NavigableMap extends from SortedMap. |
| It does not provide methods to create sub maps from existing map. | It allows ways to create sub map from existing map using methods like subMap, tailMap and headMap. |
| NavigableSet stores only one object i.e. Key. | NavigableMap uses two objects i.e. Key and Value. |
| TreeSet implements NavigableSet interface. | TreeMap implements NavigableMap interface. |
| Duplicates values are not allowed in it. | Duplicate values are allowed here. |