Collection

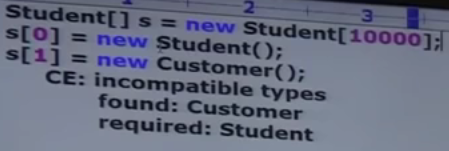
An Array is an indexed collection of fixed number of homogeneous data elements.

The main advantage of Arrays is, we can represent multiple values by using single variable, so that readability of the code would be improved.

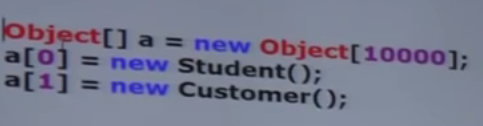
Limitattion of Arrays:

1. Arrays are fixed in size i.e once we creates an array there is no chance of increasing or decreasing the size based on our requirement. Due to this to use arryas concept we should be aware of size in advance which may not be possible always
2. Arrays can hold only homogeneous data type elements

Eg:



Though we can solve this problem by using array of object



1. Arrays concept is not implemented based on standard data structure and hence readymade method support is not available. For every requirement we have to write the code explicitly which increases complexity of programming.

**To overcome problems of arrays we should go for Colections concept**

1. Collections are growable in nature i.e based on our requirement we can increase or decrease the size
2. Collections can hold both homogeneous and heterogeneous elements
3. Every collection class is implemented based on some standard data structure, hence for every requirement readymade method support is available. Being a programmer we are responsible to use those methods and we are not responsible to implement those methods.

**Differences between Arrays and Collections**

|  |  |
| --- | --- |
| Array | Collection |
| Fixed in size i.e. once we creates an array, we can’t increase or decrease the size based on our requirement. | Growable in nature. |
| W.r.t memory Arrays are not recommended to use | Recommended |
| W. r. to performance Arrays are recommended | Not Recommended. |
| Only homogeneous | Homegeneous and Hetrogeneous |
| Not built upon any data structure so readymade methods re not available | Built upon some standard ds, hence many methods are available |
| Array can be of primitive and non primitive (object) data type | Only objects |

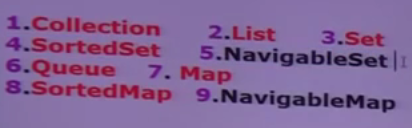
**Collection**

If we want to represent a group of individual object as a single entity then we should go for Collection.

**Collection Framework**

It contains several classes and interfaces which can be used to represent a group of individual object as a single entity.

**9 Key Interfaces of Collection Framework**



1. **Collection (I)**

* If you want to represent a group of individual object then you should go for Collection interface.
* Collection interface defines the most common method which are applicable for any collection object.
* In general collection interface is considered as root interface of collection framework.
* There is no concrete class which implements collection interface directly.

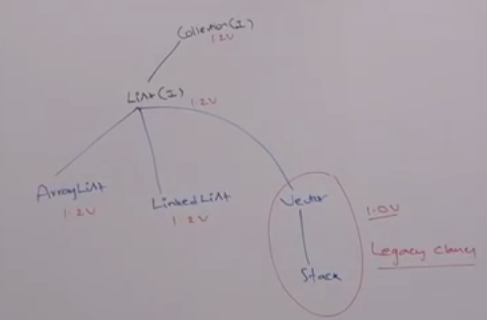
***Difference between Collection and Collections***

Collection is an interface, if we want to represent a group of individual objects as a single entity then we should go for Collection.

Collections is a utility class present in java.util package to define several utility methods for collection objects (like sorting, searching etc.). eg: collections.addAll(). Collections.sort().

1. **List (I)**

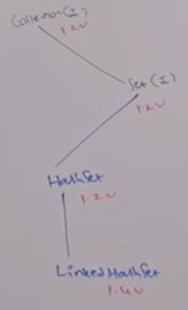
* It is child interface of collection
* If you want to represent a group of individual object as a single entity where **duplicates are allowed and insertion order must be preserved** then we should go for **List**.



Note: In Java 1.2 version Vector and Stack classes are re-engineered to implement List Interface.

1. **Set (I)**

* It is child interface of Collection.
* If we want to represent a group of individual objects as a single entity where duplicates are not allowed and insertion order not required then we should go for Set Interface.

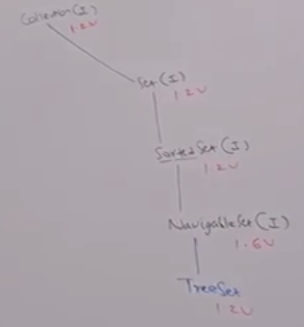


1. **Sorted Set (I)**

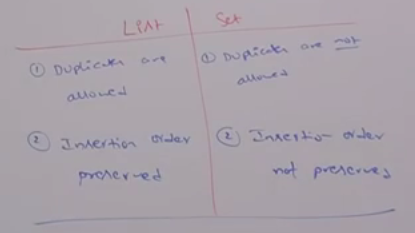
* It is child interface of Set.
* If we want to represent a group of individual objects as a single entity where duplicates are not allowed and all objects should be inserted according to some sorting order, then we should go for Sorted Set.

1. **Navigable Set (I)**

* It is child interface of Sorted Set.
* It contains several methods for navigation purposes.



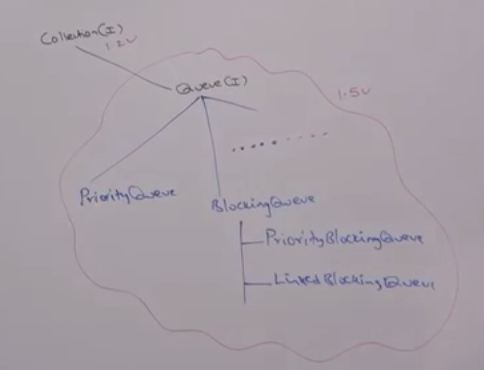
***Differences between List and Set***



1. **Queue (I)**

* It is child interface of collection.
* If we want to represent a group of individual objects prior to processessing then we should go for Queue.
* Usually Queue follows FIFO order, but based on our requirement we can implement our own priority order also.

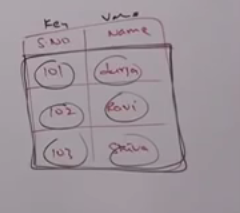
Ex. Before sending a mail all mail ids need to be stored in some data structure. In which order we added mail ids, in the same order only mail should be delivered. For this requirement queue is best choice.



Note: All the above interfaces (Collection, List, Set, Sorted Set, Navigable Set and Queue) meant up for representing a group of individual object. If want to represent a group of objects as key-value pairs then we should for Map.

1. **Map (I)**

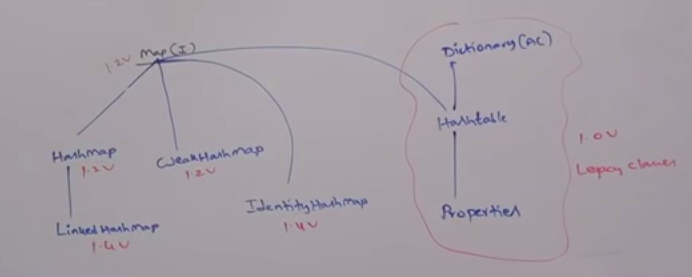
* *Map is not child interface of Collection Interface.*
* If you want to represent a group of objects as key-value pairs then we should go for Map.



Both key and value are objects only.

Duplicates keys are not allowed but values can be duplicated.

Various implementation classes of Map Interface

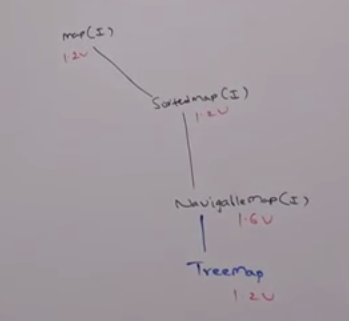


1. **Sorted Map**

* It is child interface of Map.
* If we want to represent a group of key-value pairs according to some sorting order of keys then we should go for Sorted Map.
* In Sorted Map, the sorting should be based on Keys but not based on value.

1. **Navigable Map**

* It is child interface of Sorted Map
* It defines several methods for navigation purposes.



My conclusions

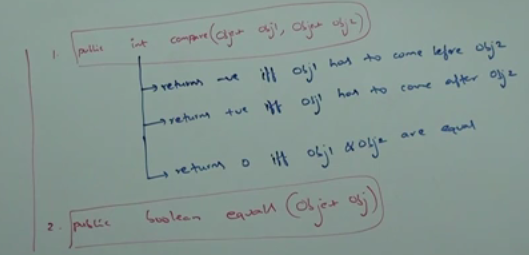
|  |  |
| --- | --- |
| **List** | **Preserve the insertion order and duplicates are allowed** |
| ArrayList | Used when collection is intended for frequent search operation |
| LinkedList | Used when collection is intended for frequent insertion and deletion |
|  |  |
| **Set** | **Doesn't preserve insertion order and duplicates are not allowed** |
| HashSet | Doesn't preserve insertion order and duplicates are not allowed |
| TreeSet | Data is stored in sorted order. |
| LinkedHashSet | Insertion order is preserved but still duplicate is not allowed. |
|  |  |
| **Map** | **Key Value Pair Data Structure** |
| HashMap | Doesn't preserve insertion order and duplicates keys are not allowed. |
| TreeMap | Collection is sorted on the basis of Key. |
| LinkedHashMap | Insertion order is preserved and still duplicate key is not allowed. |

***=>Tree is for sorted ones and Linked is for preserving insertion order.***

**Comparator**

It is present in java.util package and it defines two methods:

* compare and equals method



Whenever we are implementing ***Comparator*** interface then we should provide implantation only for ***compare*** method and we are not required to provide implementation for ***equals*** method because it is already available to our class from Object class through inheritance.

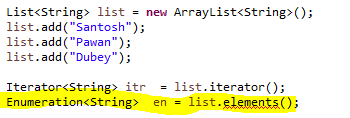
Write a program to insert integer object in to TreeSet where the sorting order is descending order.

**Iterator vs Enumeration**

Both are used to traverse the collection.

With iterator element can be removed but with enumeration we can’t remove.

Enumeration is applicable for legacy collection classes only but iterator is applicable to all.

Enumeration is not allowed in list.

Enumeration and Iterator,both are allowed in Vactor.

**If custom object is to be used as key in map**

HashCode and equals method need to be overridden

http://www.javamadesoeasy.com/2015/02/overriding-equals-and-hashcode-method.html