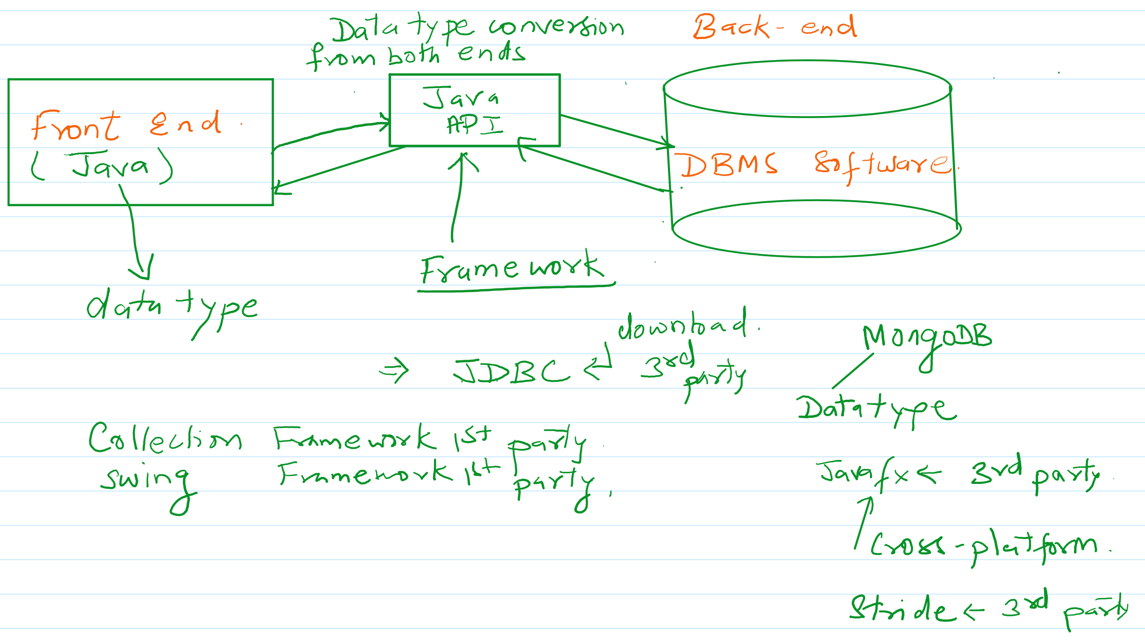
Collection (API)

20 January 2025 18:23

Application Program Interface(API) it is a framework of classes and interfaces. It is used to solve complex problems.



Array

# An array is an indexed collection of fixed number Homogenous data elements.
The main advantage of arrays is, we can represent multiple values, by using single variable. So that readability of code will be improved.

Limitations of Arrays

1. **Arrays are fixed in size. That is, once we create an array, there is no chance of increasing or decreasing the size based on our requirement. due to this to use array concept compulsorily, we should know the size in advance, which may not be possible always.**
2. **Arrays can hold homogeneous data type elements.**

```
Student[] s = new Student[10000];

s[0] = new Student();
for(I = 0; I < 10000; I++){
   s[I] = new Student();
}
```

How to store heterogenous elements in an Array?

```
Object[] A = new Object[10000];

A[0] = new Student();
A[1] = new Customer();
A[2] = new Employee();
```

Array concept is not implemented based on some standard data structure. And hence, ready-made method support is not available for every requirement, we have to write the code explicitly, which increases the complexity of programming.

To overcome above problems of arrays, we should go for Collection concepts.

- **Collections are growable or shrinkable in nature, that is based on our requirement. We can increase or decrease the size.**
- **Collection can hold both homogeneous and heterogeneous objects.**
- **Every collection class is implemented based on some standard data structure. Hence, for every requirement, ready-made method support is available.**
- **Being a programmer, we are responsible to use those methods, and we are not responsible to implement those methods.**

Difference between Arrays and Collection

Arrays	**Collection**
Fixed in size. That is, once we create an array, we can't increase or decrease the size based on our requirements.	**Growable in nature, that is based on our requirement. We can increase or decrease the size**
With respect to memory arrays are not recommended to use.	**With respect to memory collection are recommended to use.**
With respect to performance Arrays recommended to use.	**With respect to performance collection are not recommended to use.**
Arrays can hold only homogeneous data type elements.	**Collection can hold both homogeneous and heterogeneous elements.**
No underlying data structure for arrays. And hence, ready-made method support is not available. for every requirement, we have to write the code explicitly which increases complexity of programming.	**Every collection class is implemented based on some standard data structure. And hence, for every requirement, ready-made methods support is available. Being a programmer, we can use these methods directly, and we are not responsible to implement those methods.**
Arrays can hold both primitives and object types.	**Collection can hold only object types, but not primitives.**

What is Collection?
If we want to represent a group of individual objects 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 objects as a single entity.

Java Collection	C++ Container
Collection Framework	→ STL (Standard Template Library)

9 Key intefaces of Collection Framework.

1. Collections. 2. List 3. Set 4. SortedSet

5. NavigableSet 6. Queue. 7. Map 8. SortedMap

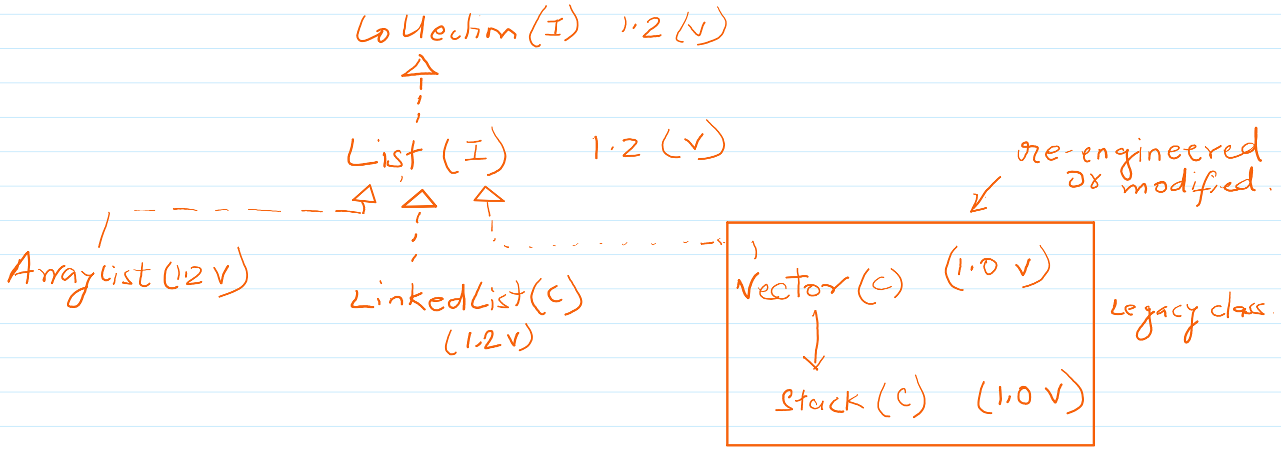
9. NavigableMap.

Collection(I): defines the most common methods which are applicable for any collection object. In general Collection(I) is considered as root(I) of Collection framework. There is no concrete class, which implements Collections(I) directly.

What is the difference between Collection and Collections?
Collection is an interface. If we want to represent a group of individual object in a single

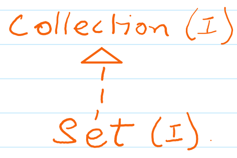
entity, then we should go for Collection interface.

Collections(C) is an utility class present in java.util package to define several utility methods for Collections objects(like sorting, search, etc.)

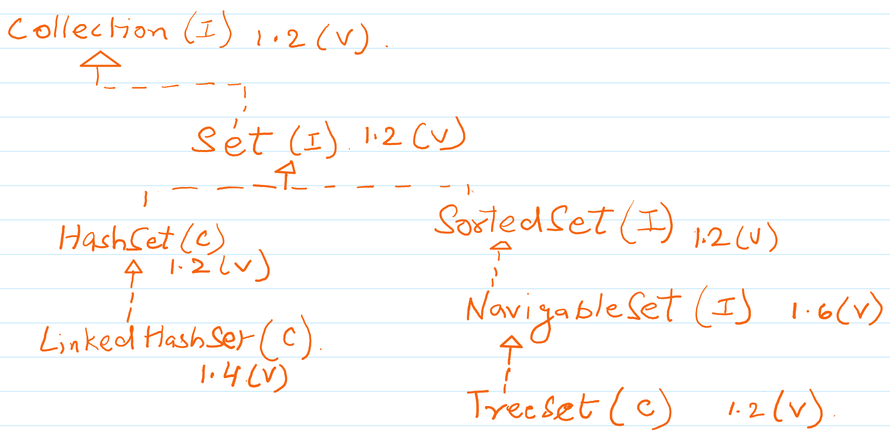


List(I): It is a child interface of Collection(I). If we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order must be preserved, then we should go for list interface.

Note: In 1.2 version Vector & Stack classes are Re-engineered or modified to implement List interface.



SET(I): It is a child interface of Collection. If we want to represent a group of individual Object as a single entity where it duplicates are not allowed and insertion order not required to preserve, then we should go for set interface.

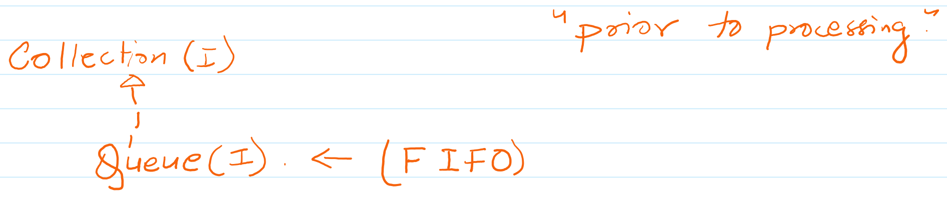


Sorted Set(I): It is the child interface of **Set** interface. If we want to represent a group of individual object as a single entity, where it duplicates are not allowed, and all objects should be inserted according to some sorted order, then we should go for **SortedSet** interface.

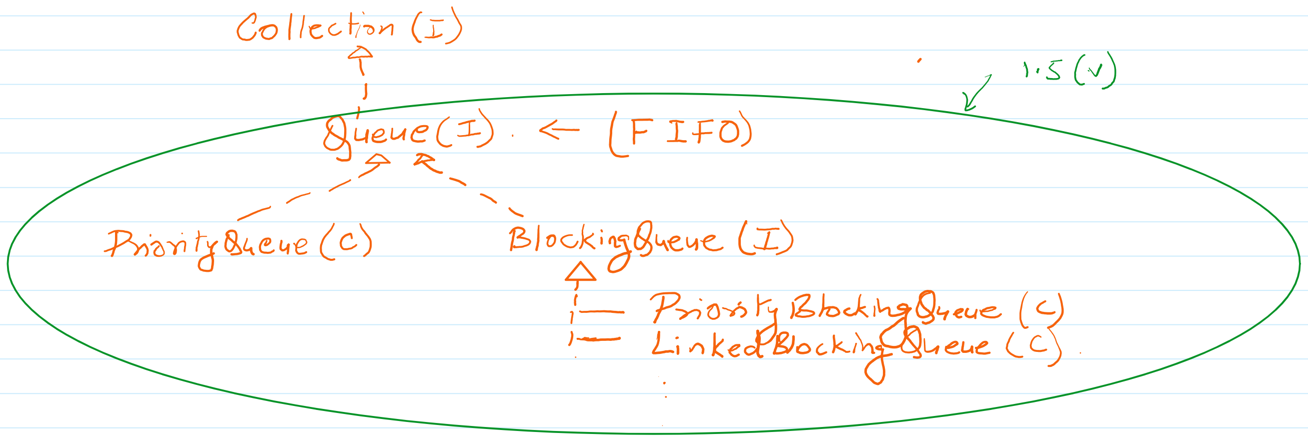
NavigableSet(I): It is a child interface of **SortedSet** interface. It contains several methods for navigation purposes (Scrolling of data).

What is the difference between Set(I) and List(I)

Set(I)	**List(I)**
Duplicates are not allowed.	Duplicates are allowed.
Insertion order is not preserved.	Insertion order is preserved.



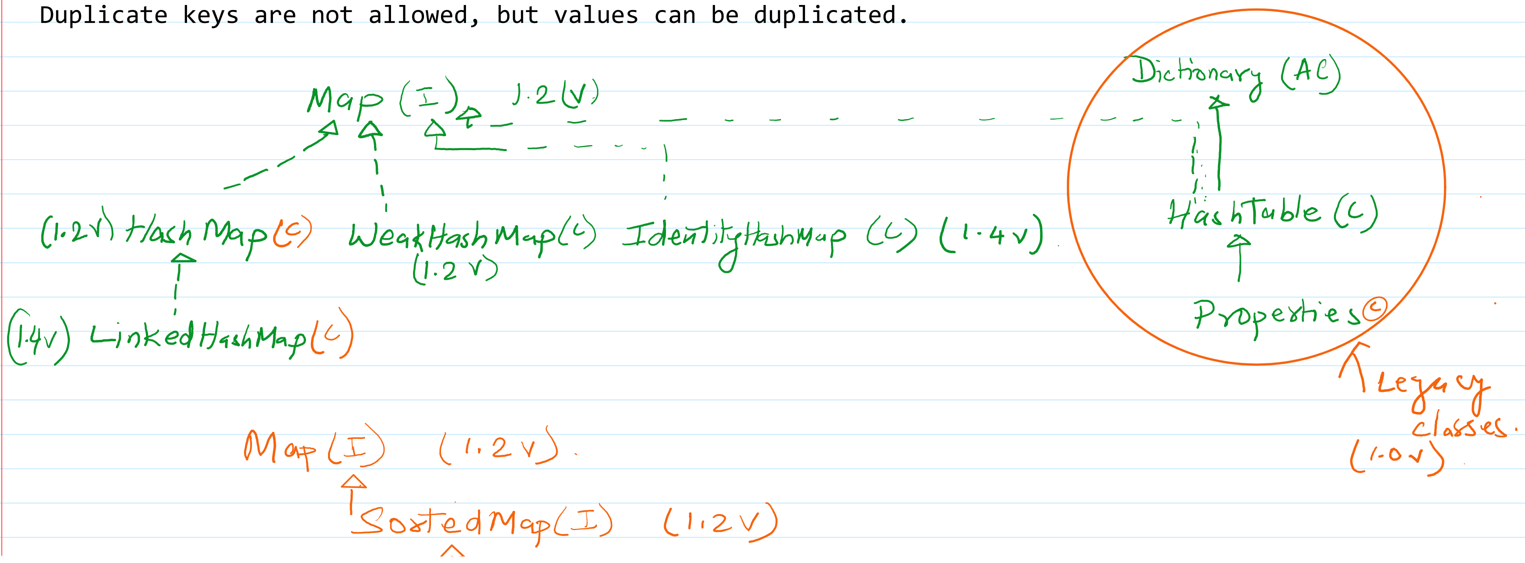
Queue(I): Queue is a child interface of collection interface. If we want to represent a group of individual "prior to processing" then we should go for Queue. Usually queue follows FIFI order. But based on our requirement, we can implement our own priority order also. Example, before sending a mail, all mail-ids have to store in some data structure in which order we added mail-id, the same ordering mail should be delivered for this requirement. Queue is the best choice.

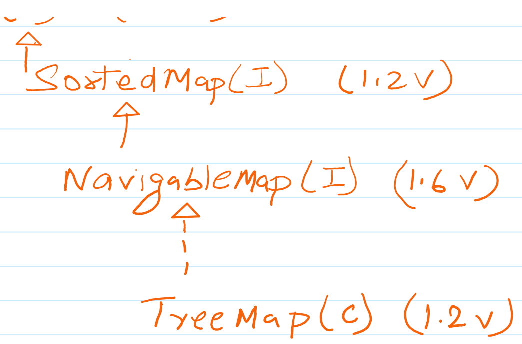


Note: All the above interfaces (Collection, List, Set, SortedSet, NavigableSet. Queue) ment for representing a group of individual objects.

If we want to represent a group of objects as key-value pair, then we should go for Map interface.

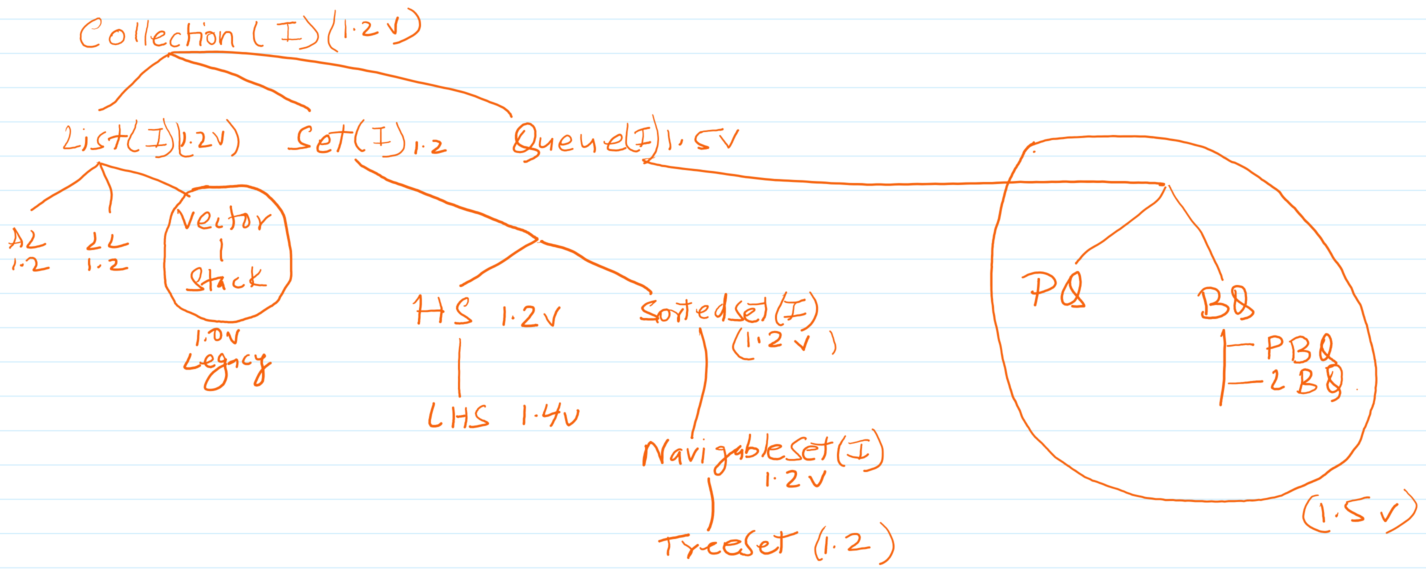
Map(I): Map is not child interface of collection. If we want to represent a group of object as key value pairs, then we should go for map interface. both key and value are objects only. Duplicate keys are not allowed, but values can be duplicated.

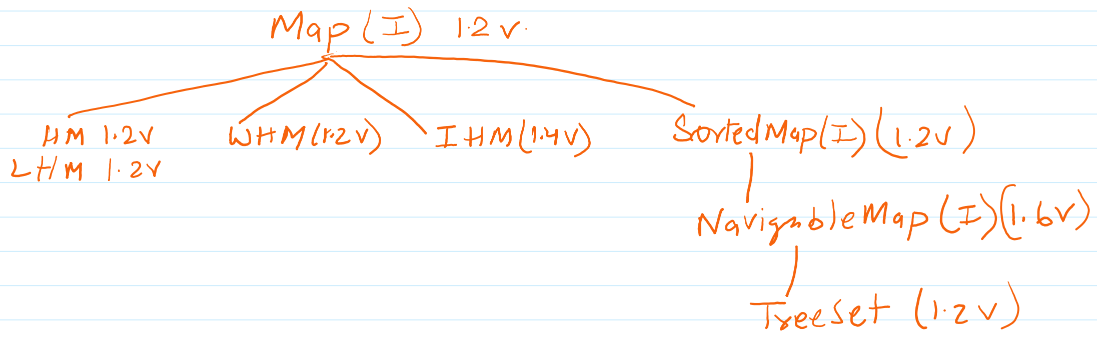




SortedMap(I): It is a child interface of **Map** interface. If we want to represent a group of key-value pairs according to some sorting order of keys, then we should go for **SortedMap** interface. In sorted map interface, the shorting should be based on key, but not based on value.

NavigableMap(I): It is a child interface of **SortedMap** interface. It defines several methods for navigation purpose.





The following are legacy characters present in collection framework:
Enumeration interface.
Dictionary abstract class.
Vector class.
Stack class.
Hashtable class.
Properties class.

Sorting:
Comparable(I): Default natural sorting order(Ascending order).
Comparator(I): Customized sorting order.

Cursors:
Enumeration(I)
Iterator(I)
ListIteraor(I)

Utility Classes:
Collections
Arrays