To know more information about the framework,then we need to know the specification(interface)
9 key interfaces of Collection framework

1. Collection(I)
2. List(I)
3. Set(I)
4. SortedSet(I)
5. NavigableSet(I)
6. Queue(I)

MAP(I) → We will see in future lecture

7. Map(I)
8. SortedMap(I)
9. NavigableMap(I)

ArrayList(C)

1. DataStructure: GrowableArray /Sizeable Array
2. Duplicates are allowed through index
3. insertion order is preserved through index
4. Heterogenous objects are allowed.
5. null insertion is also possible.

Constructors

a. ArrayList al=new ArrayList()
Creates an empty ArrayList with the capacity to 10.
    a. if the capacity is filled with 10, then what is the new capacity?
        new capacity=> (currentcapacity * 3/2)+1
        so new capacity is =>16,25,38,.....
    b. if we create an ArrayList in the above mentioned order then it would result in performance issue.
    c. To resolve this problem create an ArrayList using the 2nd way approach.

b. ArrayList al=>new ArrayList(int initialCapacity)
c. ArrayList l=>new ArrayList(Collection c)
    It is used to create an equivalent ArrayList Object based on the Collection Object

When to use ArrayList and when not to use?

ArrayList => it is best suited if our frequent operation is "retrieval operation",because it implements RandomAccess interface.

ArrayList => it is the worst choice if our frequent operation is "insert/deletion" in the middle because it should perform so many shift operations.To resolve this problem we should use "LinkedList".

LinkedList

- Memory management is done effectively if we work with LinkedList.
- memory is not given in continuous fashion.

1. DataStructure is :: doubly linked list
2. heterogenous objects are allowed
3. null insertion is possible
4. duplicates are allowed

Usage

1. If our frequent operation is insertion/deletion in the middle then we need to opt for "LinkedList".

```
LinkedList l=new LinkedList();
  l.add(a);
  l.add(10);
  l.add(z);
 l.add(2,'a');
 l.remove(3);
```

2. LinkedList is the worst choice if our frequent operation is retrieval operation

Constructors

a. LinkedList l=new LinkedList();
   It creates an empty LinkedList object.

b. LinkedList l=new LinkedList(Collection c);
   To convert any Collection object to LinkedList.

ArrayDeque

- The ArrayDeque class implements the Deque interface.
- It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.
- ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

PriorityQueue

- The PriorityQueue class implements the Queue interface.
- It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

TreeSet

- Underlying Data Structure: BalancedTree
- duplicates : not allowed
- insertion order : not preserved
- heterogeneous element: not possible,if we try to do it would result in "ClassCastException".
- null insertion : possible only once
- Implements Serializable and Cloneable interface,but not RandomAccess.
- All Objects will be inserted based on "some sorting order" or "customised sorting order".

Constructor

TreeSet t=new TreeSet();//All objects will be inserted based on some default natural sorting order.

TreeSet t=new TreeSet(Comparator); //All objects will be inserted based on some customized sorting order.

Set

- It is the 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 is Not Preserved then we should go for Set.
- Set Interface doesn't contain any New Methods and Hence we have to Use Only Collection Interface Methods

HashSet

1. Duplicates are not allowed,if we try to add it would not throw any error rather it would return false.
2. Internal DataStructure: Hashtable
3. null insertion is possible.
4. heterogeneous data elements can be added.
5. If our frequent operation is search, then the best choice is HashSet.
6. It implements Serializable,Cloneable, but not random access.

Constructors

HashSet s=new HashSet(); Default initial capacity is 16
Default FillRation/load factor is 0.75

Note: In case of ArrayList, default capacity is 10, after filling the complete capacity then a new ArrayList would be created.
In the case of HashSet, after filling 75% of the ratio only new HashSet will be created.

HashSet s=new HashSet(int initialCapacity);//specified capacity with default fill ration=0.75
HashSet s=new HashSet(int initaliCapacity,float fillRatio)
HashSet s=new HashSet(Collection c);

LinkedHashSet

- It is the child class of "HashSet".
- DataStructure: HashTable + linkedlist
- duplicates : not allowed
- insertion order: preserved
- null allowed : yes

All the constructors and methods which are a part of HashSet will be a part of "LinkedHashSet",but except "insertion order will be preserved".

Difference b/w HashSet and LinkedHashSet

HashSet => underlying data structure is "HasTable"
LinkedHashSet => underlying data structure is a combination of "Hashtable + "linkedlist" .
HashSet=> Duplicates are not allowed and insertion order is not preserved
LinkedHashSet => Duplicates are not allowed,but insertion order is preserved.
HashSet => 1.2V
LinkedHashSet => 1.4v

The 3 Cursors of Java

- If we want to get Objects One by One from the Collection then we should go for Cursors.
- There are 3 Types of Cursors Available in Java.

1. Enumeration
2. Iterator
3. ListIterator

1. Enumeration:

We can Use Enumeration to get Objects One by One from the Collection.
We can Create Enumeration Object by using elements().