Collection

- Why collection ?
- Why Array ?
- Limitations of an Array ?
- What is collection ?

Why collection:

to store on a single value we go for int x = 10. for two values, int x=10, y=20; if you want to store 10,000 values!

So, To over come this problem Array came into Picture.

Why Array ?

An array is indexed collection of fixed number of homogenous data elements.

Limitations of Array

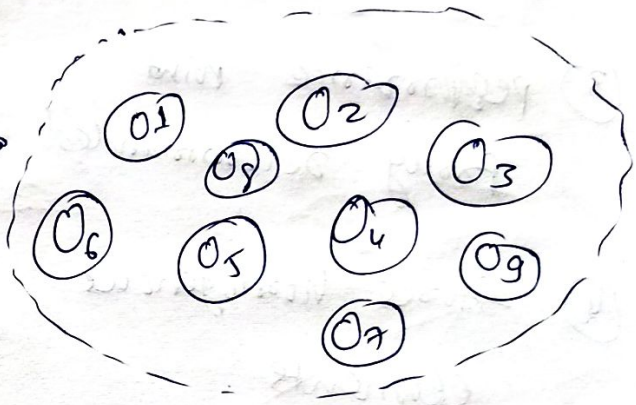
- fixed in size
- Array can store homogenous data elements
- In array ready-made method support is not available.

So, To overcome this limitation collection concept is introduced.

* Collection is one type of containes which stores the group of objects into single variable.
- Group of objects that are stored and manipulated in well manner
- introduced in JDK 1.2

* Collection: group of indusival objects represented as a single Entity

Collection



* Collection framework:

To represent a group of indusival objects as a single entity we need several classes and Interfaces, these are nothing But framework.

* Advantages of collection over the Array

(1) Collection is growable in nature
(2) collection can store heterogenous objects
(3) In collection ready made method support is avalible.

Difference Between Array and Collection

Array	Collection
(1) Fixed size	Growable in size
(2) memory point not good	memory point collection recommended
(3) performance point Array recommanded	performance point collection not recommanded.
(4) Stores homogenous elements	Stores both homogeneous and heterogenous.
(5) NIO, ready made method support	yes, ready made method support avilable.
(6) Array holds Both Primitive and objects	holds objects not Primitives.

Collection vs collections

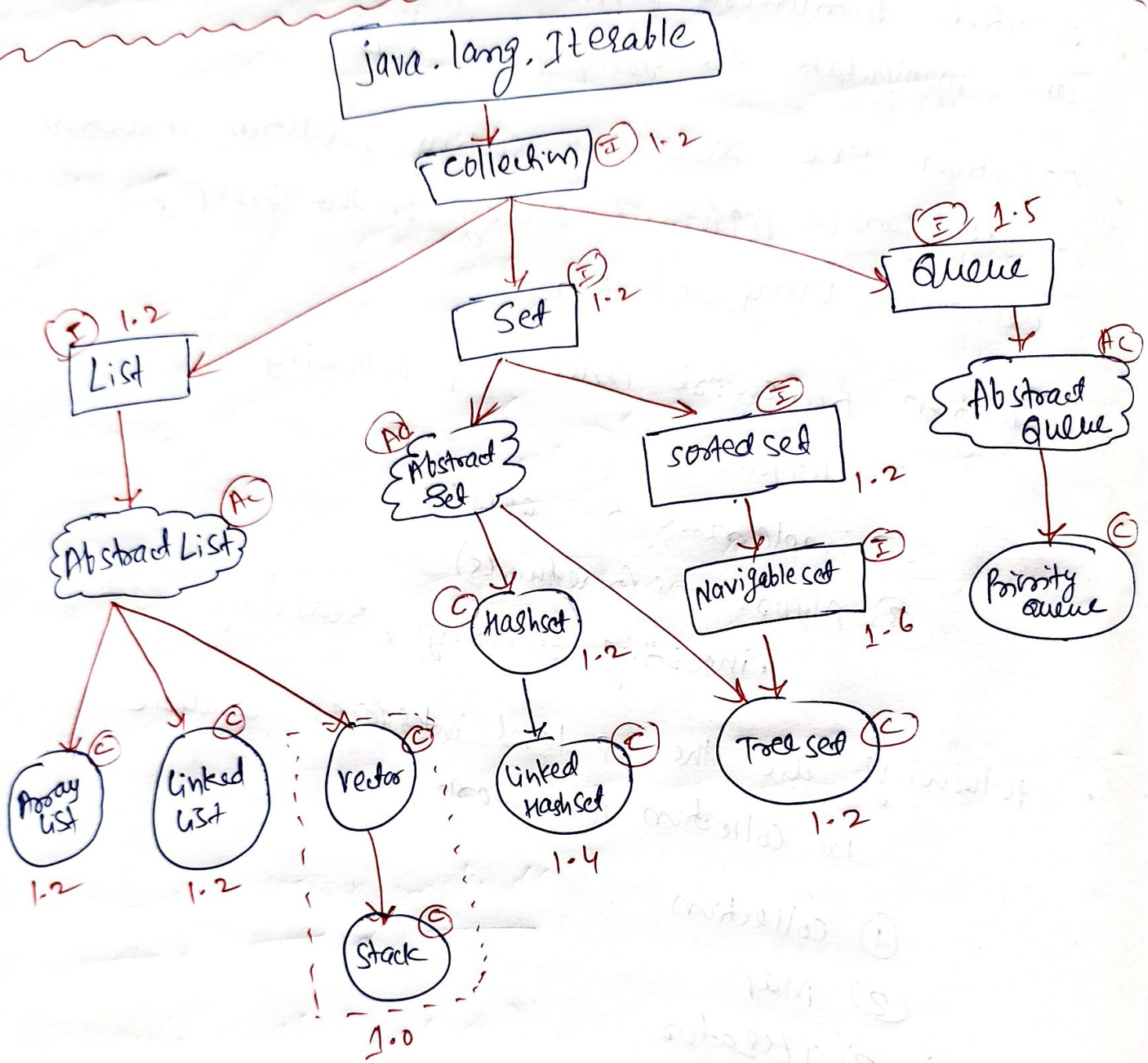
→ Collection is an interface, can be used to represent a group of individual object as a single entity.

→ Collections is an utility class, present in java.util package, to define several utility methods for collections.

Collection framework

- Collection framework provides mechanism to store and manipulate the group of elements
- operations like searching, sorting, deletion, insertion ...etc can be performed easily on the group of elements using collection.
- Collection framework consists of following
  1. Interfaces
  2. implementation classes
  3. Algorithms (methods)
     - inserting, deleting, searching, sorting....
- following are the top level interfaces avalible in collection framework.
  1. collection
  2. Map
  3. Iterator
- Subclasses of collection are used to manage the collection of elements without key-value format
- subclasses of map interface are used to manage the collection of elements with key and value format.
- Iterator is used to access data from the collection.

* Collection Hierarchy *



- collection Interface is root Interface of collection Hierarchy
- There are 9 Interfaces and are follows.

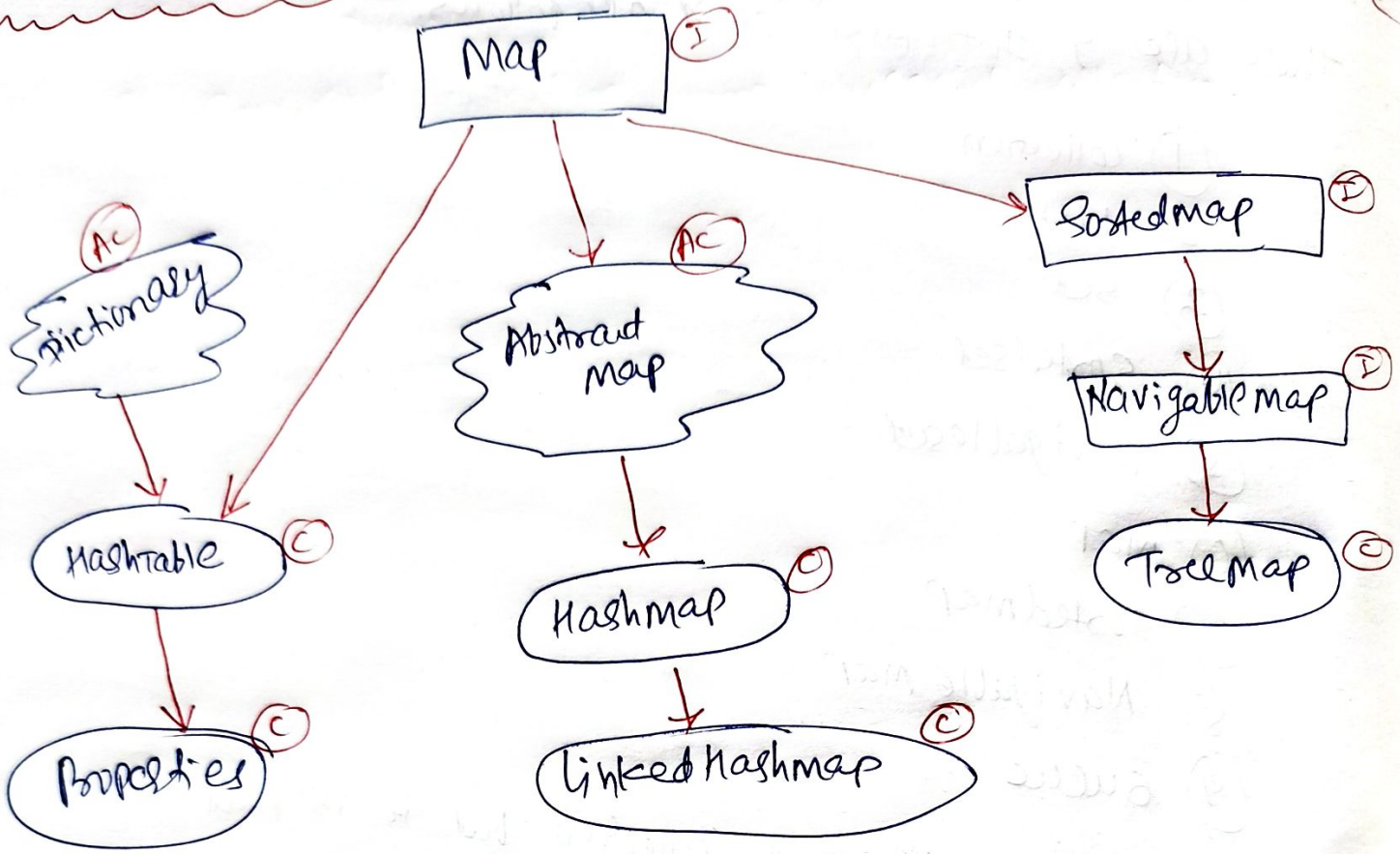
1. collection
2. List
3. Set
4. Sortedset
5. Navigableset
6. Map
7. Sortedmap
8. Navigable map
9. Queue

- map is not child interface of collection but it is part of collection framework.
- collection interface has following 3 sub interfaces

1. List
2. Set
3. Queue

- vector, stack, Dictionary, Hashtable and properties classes are legacy classes and which came in Jdk 1.0 version.
- Queue concept came in 1.5
- Navigableset and Navigable Map Interface came in 1.6 version of JDk.
- LinkedHashset, LinkedHashmap and IdentityHashmap came in Jdk 1.4

Map Interface



Collection Interface methods

1. boolean add (Object obj)

   Adds objects to collection

2. int size ()

   returns no of objects available in collection.

3. boolean isEmpty ()

   checks whether collection is empty or not

(4) boolean contains (object obj)

checks whether collection has specified object or not.

(5) Iterator iterator()

Returns Iterator subclass object

(6) Object[] toArray()

returns an array containing all of the elements of collection.

(7) boolean remove (object obj)

removes first occurance of specified object from collection.

(8) void clear()

removes all of the elements from collection.

(9) boolean addAll (Collection c)

Adds All the data of specified collection in the current collection.

(10) boolean containsAll (Collection c)

returns true if current collection all the elements of the specified collection.

(11) boolean removeAll (Collection c)

Removes all the matching elements of specified collection from current collection.

(12) boolean retainAll(Collection c)

removes all the non matching elements of specified collection from current collection.

(12) boolean retainAll(Collection c)

removes all the non matching elements of specified collection from current collection.

Cursors (Iterator)

1. Enumeration
2. Iterator
3. ListIterator

① Enumeration 1.0V

- Enumeration is an Interface available in java.util package.
- It is a cursor to access the elements from legacy classes one by one.
- Enumeration interface contains following methods:

① boolean hasMoreElements()
checks whether next element is available or not.

② Object nextElement()
moves the pointer and returns elements from collection.

- Using Enumeration we can access element but we can't remove.
- Forward direction only.

② Iterator 1.2v

- java.lang
- java.util Package

- It is used to access the elements from collection subclasses one by one.
- Invoke the Iterator() method on collection object to get the instance of Iterator.

Ex:
```
Iterator it = col.iterator();
```
col → collection object

- Universal iterator

```
Arraylist a = new Arraylist();
Iterator itr = a.iterator();
```

- Iterator Interface containg following methods:

① boolean hasNext()
checks whether next element is availble or not.

② object next()
moves the pointer and returns elements from collection.

③ void remove()
Removes the current elements from collection.

- Single direction (Forward)
- we can perform read and remove operation on element, can't perform replacement & addition.

③ ListIterator:

- ListIterator is the child Interface of Iterator
- ListIterator provides some Extra methods that can be used only with List subclasses
- ListIterator methods can't be used with set and Queue subclasses.

- By using ListIterator you can:
  - * Access the index of elements
  - * Access the elements in both Forward and Backward direction.
  - * Add, remove and replace the data of List subclasses.
  - * Access the elements starting from the specified index.

Ex:

```
ArrayList al = new ArrayList();
ListIterator listitr = al.listIterator();
```

ListIterator Interface has following methods

1. Public boolean hasNext()
2. Public object next()
3. Public int nextIndex()
4. Public boolean hasPrevious()
5. Public object Previous()
6. Public int PreviousIndex();

(7) public void remove()

(8) public void set (object newvalue)

(9) public void add (object new)

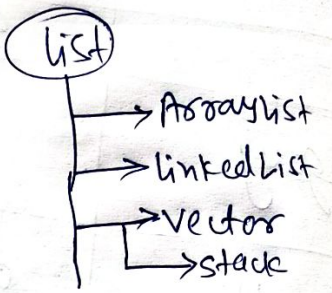
Difference between Enumeration, Iterator & ListIterator

	Enumeration	Iterator	ListIterator
legacy	yes (1.0)	No (1.2)	No (1.2)
Applicable	only for legacy class objects	For Any Collection objects	For only List object
direction	Forward	Forward	Both direction
get it?	By using .elements()	By using .iterator()	By using .listIterator()
methods	hasMoreElements(), nextElements()	next() hasNext() remove()	9 - methods .....

List Interface

- List is an interface, which extends java.util.collection
- List subclasses allow duplicate elements
- List Based on index representation
- order must be preserved
- List interface have following concrete sub classes

① Arraylist
② LinkedList
③ vector
   ◦ Stack



Methods of List Interface

① Object get (int index)
returns the element avaible at specified index.

② Object remove (int index)
removes the element available at specified index.

③ void add (int index, Object obj)
Add/insert the element at specified index.

④ int indexOf (Object obj)
returns the first position of specified object.

⑤ int lastIndexOf (Object obj)

⑥ ~~int~~ boolean addAll (int index, Collection col)
adds all the element of collection at specified index.

(7) List subList (int startIndex, int endIndex)
returns Portion of list.

(8) ListIterator listIterator()

(9) ListIterator listIterator(int startIndex)

(10) Object set(int index, Object obj)
replaces the object avaible at specified index.
and return old value.

ArrayList

- The underlying data structure is growable array or resizable array or dynamic Array.
- Random Access, duplicate allowed, insertion order preserved.
- Heterogenous elements can be stored
- "Null" value can be inserted.
- By default implements Serializable, Clonable and Random Access interfaces.
- default size of arraylist is 10 locations.
- If you want more Retrival operation best is Arraylist.

Constructors

(1) Arraylist l = new ArrayList();
(2) ArrayList l = new ArrayList(int initsize);
(3) ArrayList l = new ArrayList(Collection c);

New capacity = (current capacity * 3/2) + 1;

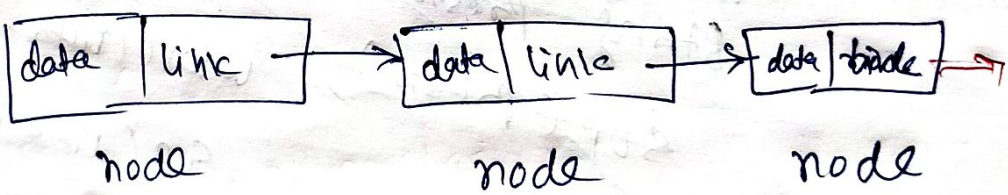
Ex:- New capacity = (10 * 3/2) + 1 = 16

After all 10 locations fields are stored, new capacity will be 16.

∴ capacity will be ⇒ 10, 16, 25, 39 .....

Linked List

- The underlying data structure is doubly linked list
- duplicates allowed, insertion order preserved
- null insertion allowed
- Heterogenous objects are allowed
- implements Serializable, Clonable but not RandomAccess Interfaces.
- Linked list is best choice if your frequent operations are insertion or deletion in middle
- No capacity concept, not stored in continuous memory location.
- Every element separate object with data part and address part called node.



- Nodes can't be Accessed directly instead we need to start from Head (first node) & follow through the link to reach node we wish to access (So, retrival is not Best).

Constructor of LinkedList

(1) LinkedList ll = new LinkedList();

(2) LinkedList ll = new LinkedList(Collection c);

LinkedList Specific methods

(1) void addFirst(Object obj)
add object at first position

(2) void addLast(Object obj)
add at last position

(3) Object getFirst()
returns the first object from list

(4) Object getLast()
returns last object from list

(5) Object removeLast()
removes first object from the last

(6) Object removeFirst()
removes the last object from the list.

Example:

```
class LinkedListExample {

psvm(-)

{

    LinkedList ll = new LinkedList();

     ll.add("A");

     ll.add("B");
     ll.add("C");
     ll.add("D");
      sop(ll);
     ll.addFirst("1");

     ll.addLast("4");

    sop( ll.getFirst());

    sop(ll.getLast());

     ll.removeFirst();

       sop(ll);

            ll.removeLast();

              sop(ll);

    }

}
```

* Vector

- vector allows duplicates and it is implemented as a resizable array
- vector elements will be stored internally using index notation.
- vector methods are synchronized
- it is a legacy class and not in use.

* stack:

- stack is subclass of vector
- stack methods are synchronized
- important operations like peek(), pop(), search(), and other operation are same as vector.
- It is legacy class and not in used.

arraylist	array
(1) Size is growable	fixed in size
(2) Iterator allows random Access of elements	looping problem with the element access.
(3) No need to specify size of array	Need to specify the size
(4) Primitives can't be stored in arraylist	can be stored.
(5) No multidimensional	Multi-dimensional
(6) size() method is used to get size of element in list	length is used to find size of the array.

Arraylist	linked list
(1) Index representation	Node representation
(2) insertion deletion are expensive	insertion and deletion operations are inexpensive
(3) Random Access	No Random Access
(4) default size 10	no size concept
(5) occupies less memory	occupies more memory
(6) implements only List Interface	implements Deque and List Interfaces.

Array list	vector
(1) non-Synchronized methods	Synchronized methods
(2) Not thread safe	thread safe
(3) Performance High	Performance low
(4) non-legacy class	Legacy class
(5) supports Iterator() and ListIterator()	supports Enumeration, ListIterator ListIterators.
(6) No subclass	stack is a subclass

Set Interface

→ Set do not allow duplicate elements
→ Set Interface doesn't have any Extra methods.
→ Order not preserved
→ iterator used But not ListListes.
→ Set Interface has following Concrete Sub classes

- Hashset
- LinkedHashset
- Treeset

Hashset 1.2V

→ duplicates not allowed, order not preserved
→ Heterogenous objects are allowed
→ null insertion possible only once (duplicate null not allowed)
→ Hashset uses Hashmap (HashTable) internally to store elements.
→ It is fast for searching and retriving elements.
→ default size of Hashset is 16

LinkedHashset 1.4V

→ extends Hashset
→ data structure used doubly linklist + HashTable
→ insertion order preserved and duplicate not allowed
→ introduced in JDK 1.4V
→ used in implementation of cache memory.

Tree set

1.2V

- extends abstractset
- implements NavigableSet, clonable, Serializable.
- faster access and Retrival
- No insertion order preserved, elements are in sorted order.
- duplicates not allowed
- elements are arranged in ascending order
- Heterogenous elements are not allowed.
- Null not allowed
- Methods are not synchronized (No threadsafe)
- All methods form collection, sortedset, Navigable set Interfaces are used with TreeSet.
- Random access of elements not possible.

Constructors of Treeset

(1) Treeset t = new Treeset();

(2) Treeset t = new Treeset(collection c);

(3) Treeset t = new Treeset(comparator comp);

passing customized sorting order specified by comparator object

(4) Treeset t = new Treeset(sortedset s);

converting sorted set to Tree set

★ Methods of TreeSet implemented ★

Note: All the methods from collection Interface are used with TreeSet.

In addition to that the methods from SortedSet are as follows:

(1) E first() : first (lowest) element will be returned.

(2) E last() : returns the last (highest) element currently in this sorted set.

(3) headSet(E) ; it returns the group of elements that are less than the specified element E.

(4) tailSet(E fromElement) ; It returns a set of elements that are greater than or equal to the specified element.

(5) subset(from, to) ; returns the subset of elements inclusive "from" and exclusive "To".

(6) and many more methods.....!!

Explore more

☺

Methods from NavigableSet used with TreeSet

(1) E lower(E e) : it returns first occurance of lowest element than e or null there is no such element.

(2) E floor(E e) : It returns first occurance of lowest element than of e or equal to e

(3) ceiling(E e) : returns the element which is equal to e or greater than e.

(4) higher(E e) : It returns the closest greatest element of the specified element from the set or null there is no such element.

(5) pollFirst() ; it is used to retrieve and remove the lowest (first) element.

(6) pollLast() : It is used to retrieve and remove the highest (last) element.

(7) descendingSet() : It returns the elements in reverse order.

and many more methods.....!!

Explore more

:)

difference Between Hashset ; LinkedHashset , TreeSet

Parameter	Hashset	LinkedHashset	TreeSet
Duplicate	X	X	X
Ordering	unorder	insertion order	sorted order
Null value	Allowed	allowed	Not allowed
Heterogenous objects	Allowed	Allowed	Not allowed
data Structure used internally	HashTable	HashTable + LinkedList	Balanced Tree
versions (JDK)	1.2	1.4	1.2
Usage?	No order Needed	Insertion order Needed	Some Sorted order Ascending or decending.