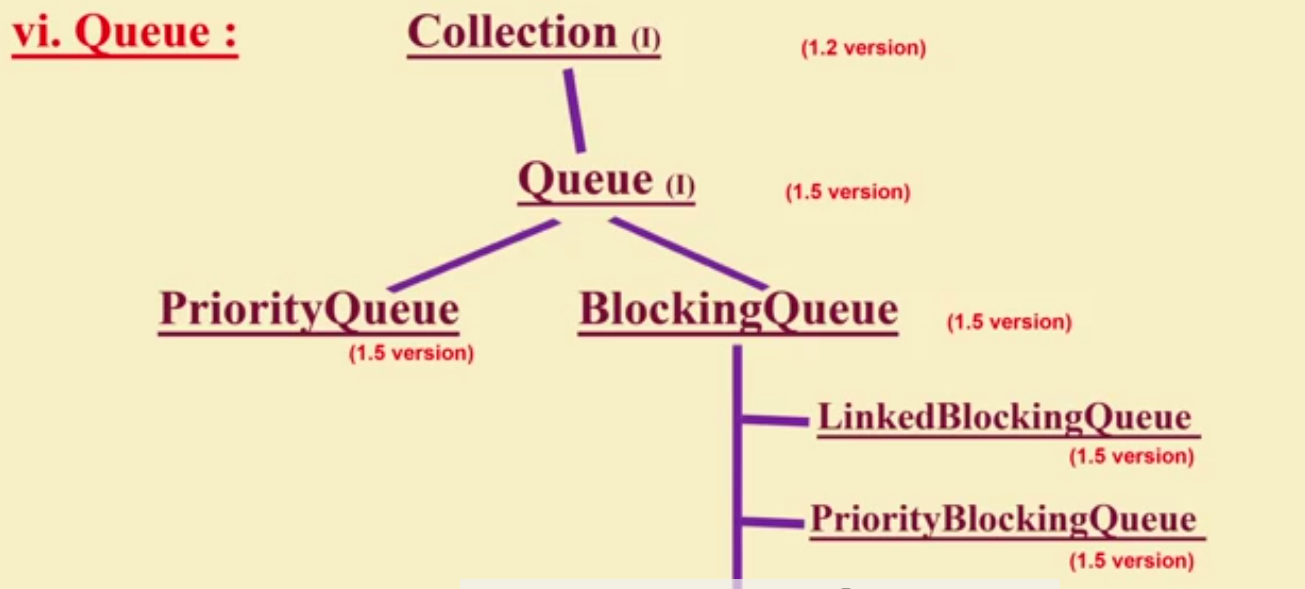
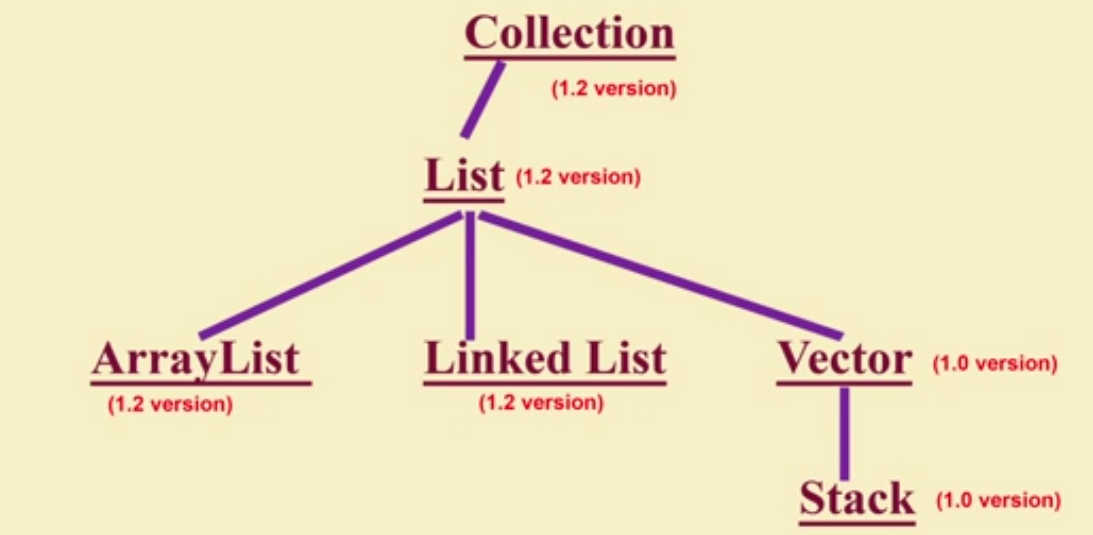
# Collection Framework

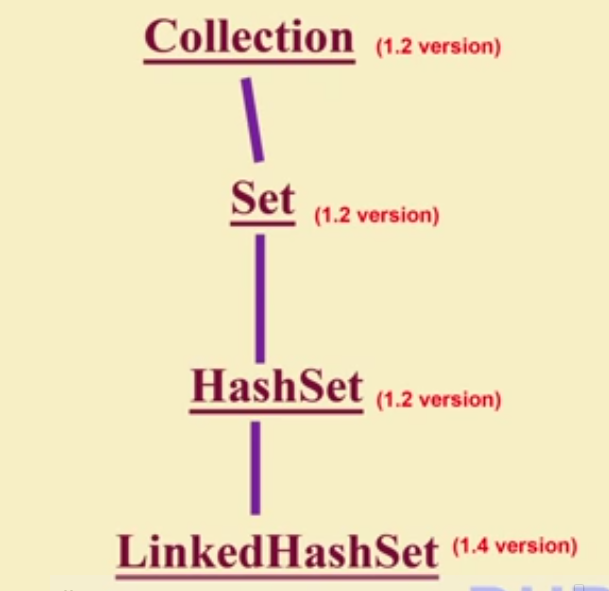
1. Need for collection
   1. Arrays are fixed in size
   2. WRT memory not recommended
   3. homogeneous
   4. no standard underlying data structure used
   5. Performance wise best
   6. Can hold both primitive and objects.
2. Collections
   1. Growable in size
   2. WRT memory recommended
   3. hold both homo and heterogeneous data
   4. build on collection data structure
   5. WRT performance takes time during extension
   6. Hold only objects
3. Difference between collection and collections
   1. Collection is an interface
   2. Collections in utility class present in java.util contains methods to search , sort and other manipulation in collection objects
4. Collection - if we want to represent a group of individual objects in single entity then we need to go to collection.
5. Collection Framework - It defines several classes and interfaces a group of objects as single entity.
6. 9 key interface of collection Framework
   1. if we want to represent group of individual objects in a single entity we will go for collection
   2. Collection interface defines the most common method which is applicable for most collections.
   3. Collection interface is considered as root interface of collection framework
   4. **LIST:** List is the child interface of collection.
   5. If we want to represent a group of individual objects in single entity where duplicates are allowed and insertion order is maintained then go for list
   6. **SET:** it is child interface of collection.
   7. If we want to represent a group of individual objects in single entity where duplicates are not allowed and insertion order need not to be maintained then go for set
   8. **Sorted Set:** Child interface of set
   9. If we want to represent a group of individual objects in a single entity where duplicates are not allowed and elements should be inserted in some sorting order then go for Sorted set.
   10. **Navigable set:** It is a child interface of sorted set. It defines several method for navigation purpose. (**NavigableSet adds Navigation methods like descendingIterator() and descendingSet(), ceiling(), floor(), higher(), lower(), headSet(), tailSet(), subSet(), pollFirst() and pollLast().)**
   11. **Queue:**It is a child interface of collection.
   12. if we want to represent group of individual objects prior to processing then we should go with queue.
   13. **Map:** is the child interface of collection
   14. If we want to represent group of individual Objects as Key map value then we will go with map
   15. SortedMap: Child interface of map
   16. If we want to represent a group of key value pair in some sorted order of keys, then we will go with sorted map.
   17. **NavigableMap:** child interface of sorted map, defines several method for navigation purpose.



<https://examples.javacodegeeks.com/core-java/util/queue/java-queue-example/>







1. Difference bet List and set :

|  |  |
| --- | --- |
| **List** | **Set** |
| Insertion order preserved | Insertion order not preserved |
| Duplicates allowed | Duplicates not allowed |

1. Difference bet Array List and Linked list :

|  |  |
| --- | --- |
| **Array List** | **Linked List** |
| Best for retrieval operation | Best for insertion and deletion at middle |
| Underlying data structure is resizable | Underlying data structure is double linked list |
| Implements randomAccess interface | Does not implements random access interface |
| Add(), Add(index,object), clear(), get(index), get(object), Contains(object), remove(object), remove(index),iterator etc | AddFirst(), AddLast(),getFirst(),getLast(),removeFirst(),removeLast(),removeFirstOccurence(object), iterator,desendingIterator, etc |

1. Difference bet Array List and Vector :

|  |  |
| --- | --- |
| **Array List** | **Vector** |
| All method is not Synchronized | All methods are synchronized |
| Multi thread accessible | Only single thread is accessible |
| High performance, since thread are not allowed to wait | Low performance, since thread are allowed to wait |
| Introduces in 1.2 so it is not a legacy class | Introduced in 1.0, so it is a legacy class |

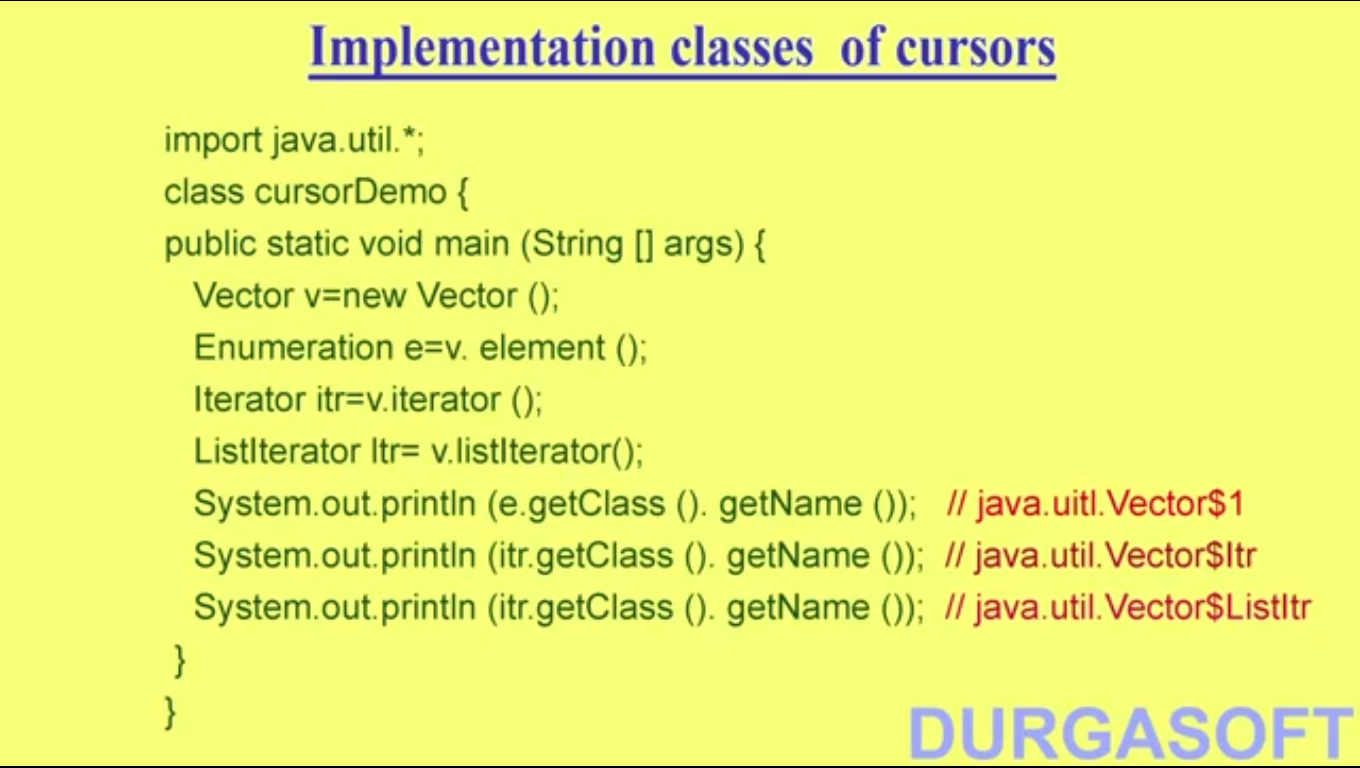
1. Difference bet Comparable and Comparator:

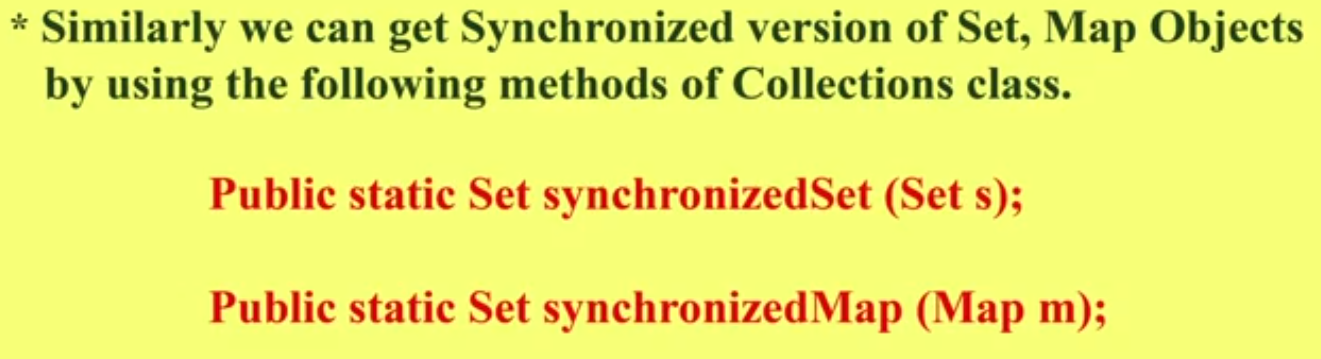
|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| Default sorting order | Customised sorting order |
| Java.lang.package | Java.util.package |
| CompareTo() | Compare() and equals() |
| All wraperclassess | Colator and rule based collator |
| http://www.javatpoint.com/Comparable-interface-in-collection-framework | http://www.javatpoint.com/Comparator-interface-in-collection-framework |

1. Cursor: Retrieve objects one by one from collection then we should go with cursor
   1. Enumeration - introduced in 1.0 version.used to get the elements of legacy collections. enum has two methods:(hasmoreElements,nextElements).
      1. Limitation1: Available for only legacy classess
      2. Limitation2: cannot perform remove operation
   2. iterator -
      1. Available for all the collection classess so it is called universal cursor.
      2. can perform both Retrive and remove operations
      3. Method available (hasNext,next,remove)
      4. Limitation1: it can move only in forward direction,
      5. Limitation2 : using iterator we can only read and remove objects, we cannot replace objects.
   3. Listiterator
      1. It is bidirectional iterator available only for List objects
      2. by using list iterator we can read, remove, replace and create new objects
      3. methods available(hasNext,next,nextIndex, hasprevious,previous,previousindex,remove,set,add).
      4. Limitations: List iteratator is available only for List interface.



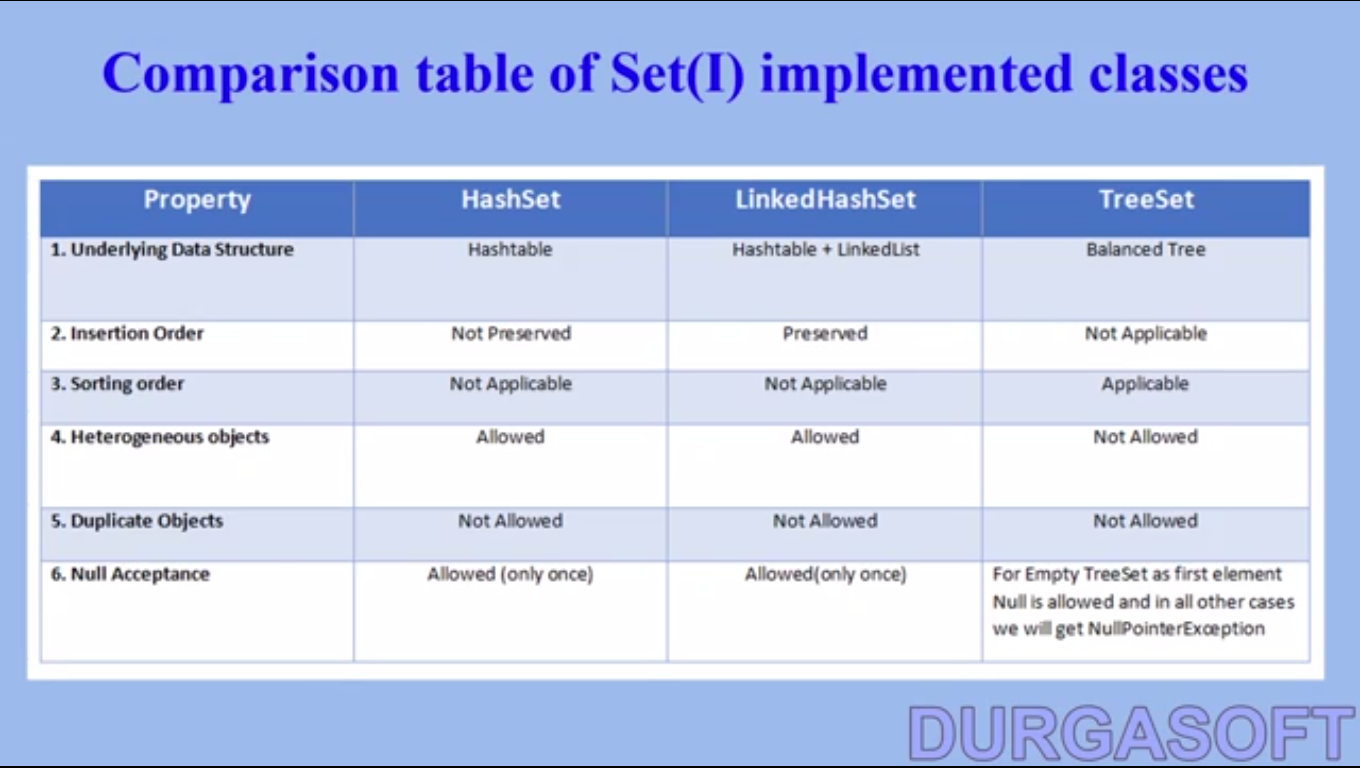
1. Enumeration, iterator and ListIterator are Interfaces. we cannot create an object for this cursors. This we get the corresponding implemented class objects.

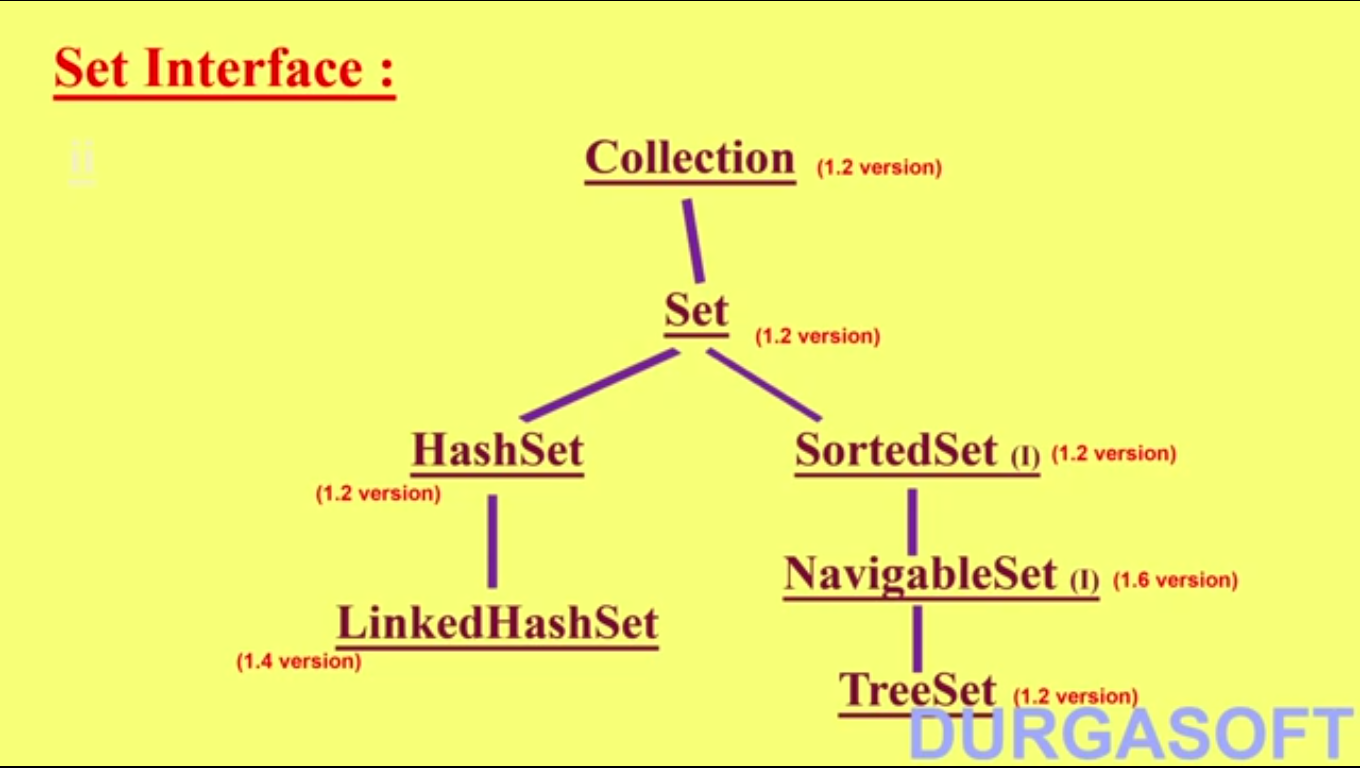


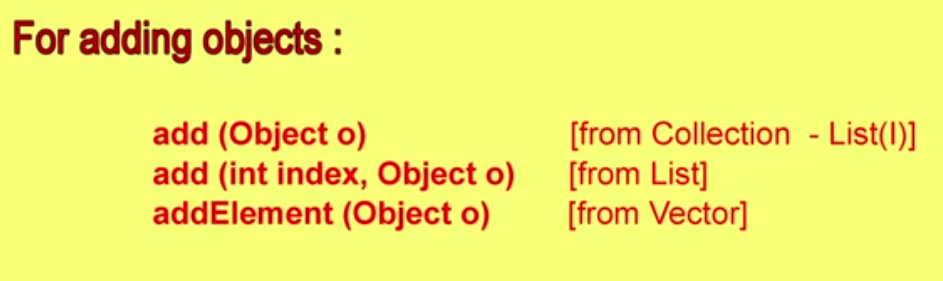
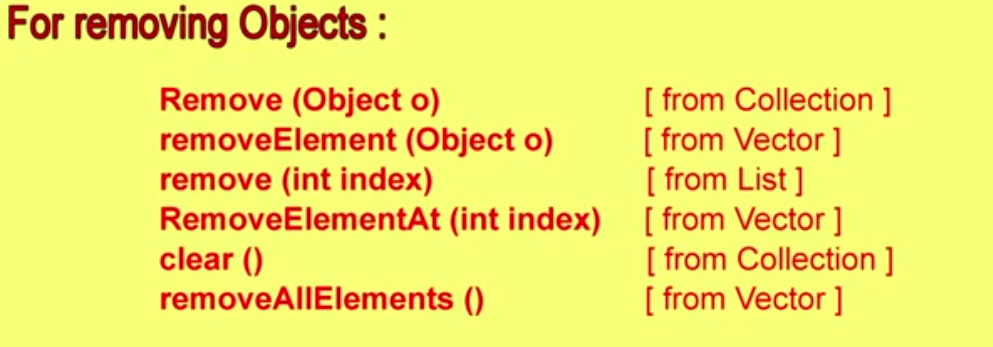
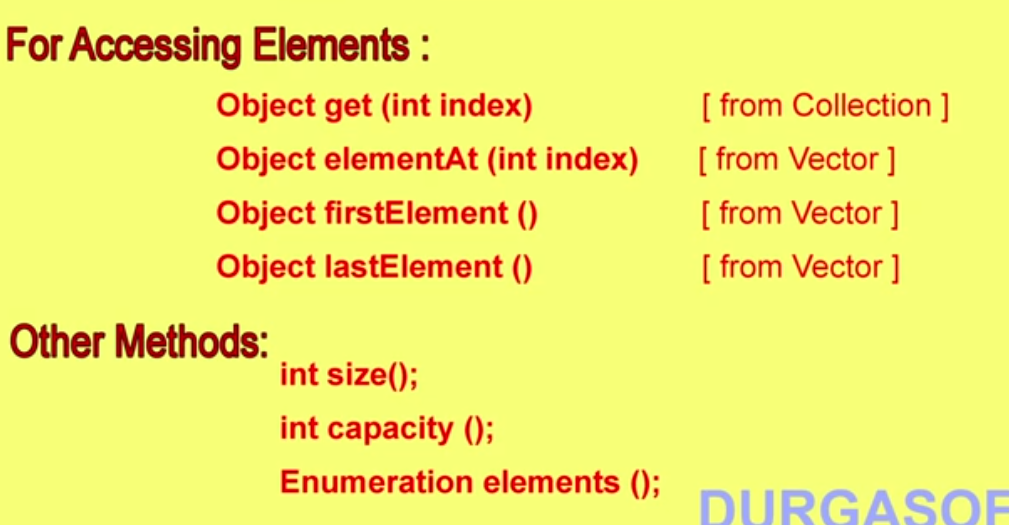
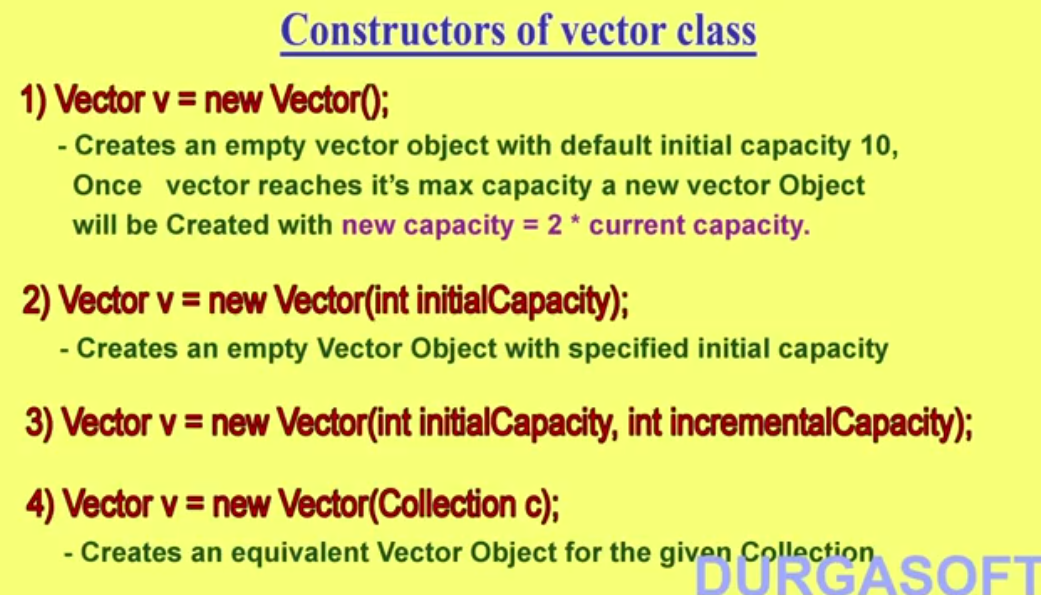
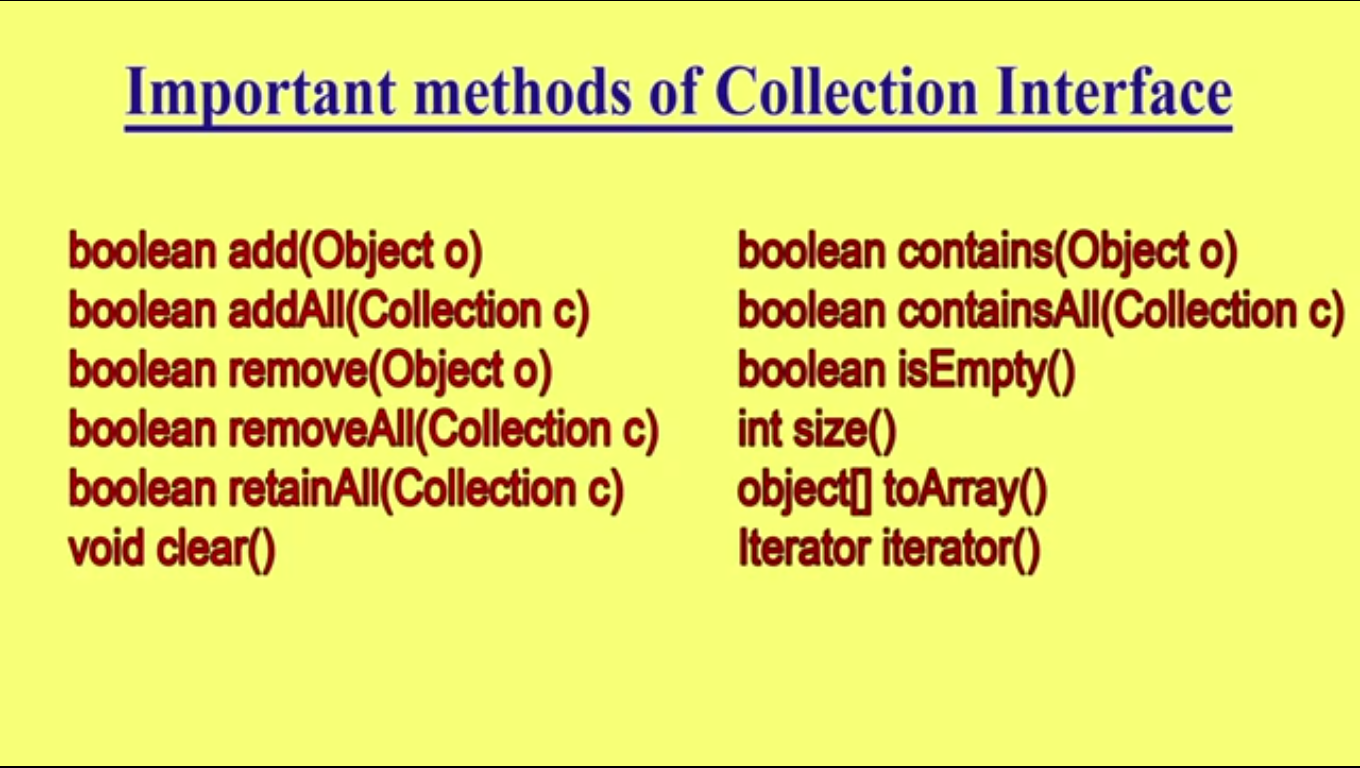
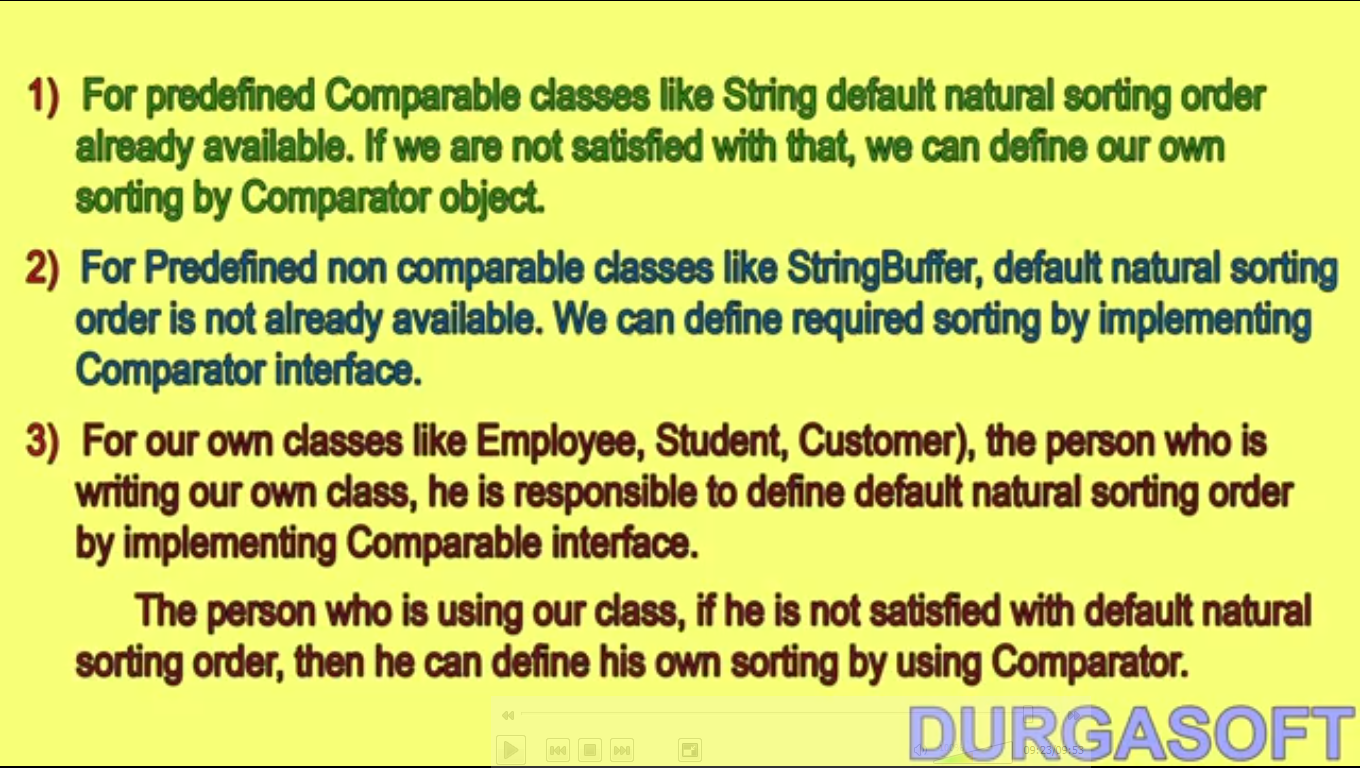
1. Get the synchronized version of ArrayList:
   1. By default array list is non Syncronized
   2. by using the method **Collections.SyncronizedList(AL)** method we can make a syncronizedlist.
   3. 
   4. 
2. Linked List :
   1. Underlying data structure in double linked list
   2. Insertion order is preserved
   3. Duplicates are allowed
   4. heterogeneous objects are allowed
   5. null insertion is possible
   6. LL implements only serializable and Cloneable,but not random access Interface.
   7. LL is the best choice for frequent insertion and deletion in the middle.
   8. LL is the worst choice for frequent retrivals
   9. We use LL to create stack and queue, because LL had the following methods.
      1. AddFirst(), AddLast(),getFirst(),getLast(),removeFirst(),removeLast().
   10. Constructors :
       1. New LL()
       2. New LL(Collection)
3. Hashset
   1. Set is the child interface of collection
   2. If we want to represent a group of elements where duplicates are not allowed and insertion order is not preserved then we should go for set.
   3. Set interface do not contains any new methods, so we need to use only collection interface methods.
   4. HashSet
      1. Underlying terminology is hashtable
      2. Duplicates are not allowed, if added false is returned
      3. Insertion order is not preserved, all objects are added WRT hash code of the objects
      4. Heterogeneous objects are allowed
      5. NULL insertion is possible
      6. implements Serializable and Cloneable.
      7. Hashset is our best choice if our best operation is search
   5. Constructors of hashset
      1. new Hashset() - creates an empty hashset with capacity 16 and fill ratio 0.75
      2. new Hashset(int x) - creates an empty hashset with capacity x and fill ratio 0.75
      3. new Hashset(int x, float y) - creates an empty hashset with capacity x and fill ratio y
      4. new Hashset(Collection c ) - for inter conversion bet collection objects
   6. Fill Ratio/ Load factor - it is the percentage of elements in an hashset WRT to its capacity at which a new hashset object will be created
4. Linked hashset
   1. it is child class of hashset
   2. Introduced in version 1.4
   3. Exactly same as hashset except the following

|  |  |
| --- | --- |
| * + 1. **Hashset** | **Linked Hash set** |
| * + 1. Insertion order is not preserved | Insertion order is preserved |
| * + 1. underlying data structure is hashtable | Underlying data structure is hashtable and linked list |
| * + 1. Introduced in version 1.2 | Introduced in version 1.4 |

* 1. Best to develop cache based applications

1. Ordered Set or treeset
   1. it is the child interface of set
   2. if we want to represent a group of individual objects according to some sorting order and duplicates are not allowed then go for sorted set.
   3. Six method of sorted set
      1. First - return s the first element
      2. last - returns the last elements
      3. headset(Object) - returns the sorted set whose elements are <obj
      4. tailset(object) - returns the sorted set whose elements are >= obj
      5. subset(object,object) - returns the set whose values are bet the objandobj
      6. compatator - returns the underlying comparble object
2. Comparison bet the set interface
   1. 



1. Vector class
   1. Underlying DS is Resizable array
   2. Duplicates are allowed
   3. NULL are allowed
   4. Insertion order is preserved
   5. heterogeneous objects are allowed
   6. Vector class implements Serializable, Cloneable, and Random access interface
   7. Most of the methods present in vector are Synchronized, Hence thread safe
   8. Best choice if frequent operation is retrieval
   9. Vector methods -
   10. 
   11. 
   12. 
   13. 
2. Stack Class
   1. Stack is the child of vector class
   2. LIFO
   3. Method -
      1. Push, - Add element to stack
      2. Pop, - remove element from stack
      3. Peek() - Get the element from stack without removing
      4. empty() - check the stack is empty
      5. search() - return the offset of the element in the stack(offset is the number when the searched element will be removed)
3. OVERVIEW
   1. 
4. Tree Set
   1. No Duplicates
   2. Insertion order not presered
   3. hetroGenious not allowed for Natural sorting
   4. Sorting Order not preserved
   5. Null Acceptance - only once at the beginning
   6. Constructors
      1. New TreeSet()
      2. New Treeset(Comparator c) - sorting according to Comparator
      3. New Treeset(Collection C)
      4. new TreeSet(SortedSet S)
   7. All the objects should implements comparable objects for customised sorting
   8. Comparable
      1. it contains only one method - CompareTo()
      2. obj1.CompareTo(Obj2)
         1. negative if obj1 comes before obj2
         2. positive if obj1 comes after obj2
         3. 0 if both are equal
   9. Comparator Interface
      1. We can use comparator to use our customised sorting
      2. in Java.Util Package
      3. Two mehods (Compare and Equals)
      4. Whenever implementing comparator method,
   10. Comparator and Comparable
   11. 

|  |  |  |
| --- | --- | --- |
| **HashMap** | **HashTable** | **Hashset** |
| * A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class. * It contains only unique elements. * It may have one null key and multiple null values. * It maintains no order. | * A Hashtable is an array of list.Each list is known as a bucket.The position of bucket is identified by calling the hashcode() method.A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class. * It contains only unique elements. * It may have not have any null key or value. * It is synchronized. | * uses hashtable to store the elements.It extends AbstractSet class and implements Set interface. * contains unique elements only. |
|  |  |  |

**What is Concurrent Modification ?**  
  
When one or more thread is iterating over the collection, in between, one thread changes the structure of the collection (either adding the element to the collection or by deleting the element in the collection or by updating the value at particular position in the collection) is known as Concurrent Modification

**Difference between Fail Fast iterator and Fail Safe iterator**  
  
**Fail fast Iterator**  
  
Fail fast iterator while iterating through the collection , instantly throws Concurrent Modification Exception if there is structural modification  of the collection . Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.   
  
Fail-fast iterator can throw ConcurrentModificationException in two scenarios :

**Single Threaded Environment**   
After the creation of the iterator , structure is modified at any time by any method other than iterator's own remove method.   
   
**Multiple Threaded Environment**  
  
 If one thread is modifying the structure of the collection while other thread is iterating over it .

According to  [Oracle docs](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html" \t "_blank) , **the fail-fast behavior of an iterator cannot be guaranteed** as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness:**the fail-fast behavior of iterators should be used only to detect bugs.**

**Interviewer : How  Fail  Fast Iterator  come to know that the internal structure is modified ?**  
Iterator read internal data structure (object array) directly . The internal data structure(i.e object array) should not be modified while iterating through the collection. To ensure this it maintains an internal  flag *"mods" .*Iterator checks the *"mods" flag*whenever it gets the next value (using hasNext() method and next() method). Value of*mods* flag changes whenever there is an structural modification. Thus indicating iterator to throw ConcurrentModificationException.  
  
  
**Fail Safe Iterator :**  
  
Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure.Any structural modification done to the iterator affects the copied data structure.  So , original data structure remains  structurally unchanged .Hence , no ConcurrentModificationException throws by the fail safe iterator.  
  
Two  issues associated with Fail Safe Iterator are :  
  
1. Overhead of maintaining the copied data structure i.e memory.  
  
2.  Fail safe iterator does not guarantee that the data being read is the data currently in the original data structure.   
  
According to [Oracle docs](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) , fail safe iterator is ordinarily too costly, but may be more efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don’t want to synchronize traversals, yet need to preclude interference among concurrent threads. The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This **array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException**.The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove(), set(), and add()) are not supported. These methods throw UnsupportedOperationException.

**Fail fast Iterator Example :**

**import** **java.util.HashMap**;

**import** **java.util.Iterator**;

**import** **java.util.Map**;

**public** **class** **FailFastExample**

{

**public** **static** **void** **main**(String[] args)

{

Map<String,String> premiumPhone = **new** HashMap<String,String>();

premiumPhone.put("Apple", "iPhone");

premiumPhone.put("HTC", "HTC one");

premiumPhone.put("Samsung","S5");

Iterator iterator = premiumPhone.keySet().iterator();

**while** (iterator.hasNext())

{

System.out.println(premiumPhone.get(iterator.next()));

premiumPhone.put("Sony", "Xperia Z");

}

}

}

**Output :**

iPhone

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.HashMap$HashIterator.nextEntry(Unknown Source)

at java.util.HashMap$KeyIterator.next(Unknown Source)

at FailFastExample.main(FailFastExample.java:**20**)

**Fail Safe Iterator Example :**

**import** **java.util.concurrent.ConcurrentHashMap**;

**import** **java.util.Iterator**;

**public** **class** **FailSafeExample**

{

**public** **static** **void** **main**(String[] args)

{

ConcurrentHashMap<String,String> premiumPhone =

**new** ConcurrentHashMap<String,String>();

premiumPhone.put("Apple", "iPhone");

premiumPhone.put("HTC", "HTC one");

premiumPhone.put("Samsung","S5");

Iterator iterator = premiumPhone.keySet().iterator();

**while** (iterator.hasNext())

{

System.out.println(premiumPhone.get(iterator.next()));

premiumPhone.put("Sony", "Xperia Z");

}

}

}

**Output :**

S5

HTC one

iPhone

# Difference between Comparable and Comparator

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc. | Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc. |
| 2) Comparable **affects the original class** i.e. actual class is modified. | Comparator **doesn't affect the original class** i.e. actual class is not  modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is found in **java.lang** package. | Comparator is found in **java.util** package. |
| 5) We can sort the list elements of Comparable type by**Collections.sort(List)** method. | We can sort the list elements of Comparator type by**Collections.sort(List,Comparator)** method. |