**Collections**

**http://javabypatel.blogspot.in/2015/10/what-is-load-factor-and-rehashing-in-hashmap.html**

* **What is the collection hierarchy and advantage of collection framework?**
* **Why Map interface doesn’t implement collection interface?**
* **What is the difference and similarities b/w vector, Array List and Linked List?**
* **Different method in ArrayList, vector and LinkedList?**
* **What will be the output if two iterators iterate a list?**
* **Difference b/w Contains () and binary Search () method?**
* **Array List to Array and Array to array List conversion?**
* **Difference b/w List and Set?**
* **What is the difference between ArrayList clear () and remove All () methods?**
* **If an ArrayList contains duplicate objects and remove () method is invoked on the same object, will it remove the duplicates?**
* **How to remove object from ArrayList by using remove method?**
* **Similarity b/w LinkedList and ArrayList?**
* **How to clone collection in java and add All use which cloning?**
* **How to synchronize ArrayList?**
* **Difference b/w list, set and map?**
* **Is Collection.synchronizedList thread-safe?**
* **Explain set hierarchy?**
* **What is the initial capacity of has-set and what is load factor?**
* **Difference between Hash Set, Linked Hash Set and Tree Set?**
* **What copy technique internally used by HashSet clone () method?**
* **How HashSet and LinkedHashSet works internally?**
* **How TreeSet works internally?**
* **Difference b/w Iterator and Enumerator?**
* **Difference b/w Con-Current Hash Map and Hash table?**
* **Why collection interface not implementing clone and sterilization interface.**
* **Difference between Concurrent Hash Map and Collections.synchronizedMap?**
* **Difference b/w Hash map and Hash table?**
* **What is the fail-safe and fail-fast iterator in java?**
* Difference/similarity b/w array List Hash Map?
* Difference and similarity b/w Hash Map, Linked HashMap and Tree-Map?
* **Difference b/w Keyset and enter Set?**
* **Difference b/w tree set and tree map?**
* **How to make collection read-only?**
* **What are Identity HashMap and Weak HashMap?**
* **Difference between Hash Map and Weak HashMap in Java?**
* **What is Enum Set in Java?**
* **How hash map works in java?**
* **How will you retrieve Value object if two Keys will have same hash code?**
* **What happens On HashMap in Java if the size of the Hash Map exceeds a given threshold defined by load factor?**
* **Difference between Iterator and List Iterator?**

**----------------------------------------------------------------------------------------------------------------------**

### What is the collection hierarchy?

### Collection is the root interface of collection framework. It is implemented by Set and List interface.

### MAP is not implementing the Collection interface. It is member of collection framework.

### Advantage:

* It provides high-performance implementations of useful data structures and algorithms that increases the performance.
* Reduces effort to learn and to use new APIs.
* Reduces effort to design new APIs.
* Reduced development effort by using core collection classes rather than implementing our own collection classes.
* Code quality is enhanced with the use of well-tested collections framework classes.
* Reduced effort for code maintenance by using collection classes shipped with JDK.
* Reusability and Interoperability

### 

### Why Map interface doesn’t implement collection interface?

* Some methods declared in Collections do not fit a Map interface. Map views as key value pair.

### Array List to Array and Array to array List conversion?

### To convert array List to array we use toArray () method.

### Object [] objects = list.toArray ()

### To convert array to as we sue asList ();

### List al = Arrays.asList (array Name);

### The list generated by this method fixed size, we cannot change the size.

### If we modify any element in list, it will be modified in array also.

### What is the difference and similarities b/w vector, Array List and Linked List?

### Similarities:

### Both are index based.

### Both maintain the insertion order.

### Both allow null and duplicate value.

### Both iterator and List Iterator are fail fast.

### Both implement list interface.

### Both implemented using array data structure.

### Difference:

### Vector is synchronized while array List is not.

### Vector is thread safe while array List is not.

### Vector is slow than array List.

### Whenever vector cross the threshold specified it increases twice itself by value specified in capacity Increment. While we can increase the size of array by calling ensure Capacity method ();

### LinkedList has more overhead than ArrayList and vector because in ArraList and vector each index store only data but in LinkedList each index store data and address of next node.

### Insertion and removal is faster in LinkedList because each node store the address of next node so for insert or delete just we need to change the address of next and previous node but in case of ArrayList and vector we need to iterate and adjust the index for all element.

### Searching is easy in ArrayList and vector but slow in LinkedList.

### LinkedList initial capacity is zero and arrayList is 10.

### ArrayList implements Random Access interface but LinkedList does not.

### Different method in ArrayList, vector and LinkedList?

### What will be the output if two iterator iterate a list?

### IllegalStateException will be thrown.

### Difference b/w Contains () and binary Search () method?

### Contains () method is faster then binarySearch () method if the collections implement Random-access interface.

* If list is not sorted, the result can be undefined in case of binarySearch ().
* If list has multiple elements equal to specified object, there is no guarantee, which will be returned.

### Difference b/w List and Set?

* List allows duplicates while Set does not allow.
* List is ordered collection while set is unordered collection

### What is the difference between ArrayList clear () and removeAll () methods?

* Clear () method removes all the elements from an ArrayList.
* RemoveAll () method takes a collection as parameter. It removes all of the ArrayList elements that are part of the collection.

### If an ArrayList contains duplicate objects and remove () method is invoked on the same object, will it remove the duplicates?

* No, it removed first occurrence.

### How to remove object from ArrayList by using remove method?

* We need to implement equal method to remove object from ArrayList.

### Similarity b/w LinkedList and ArrayList?

|  |  |  |
| --- | --- | --- |
| 1 | synchronization | ***ArrayList*** *and* **LinkedList both** are **not synchronized** (because 2 threads on same ArrayList/LinkedList object can access it at same time) in java. |
| 2 | Iterator and listIterator are Fail-fast | Iterator and listIterator returned by ***ArrayList*** *and* **LinkedList both** are [**Fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) in java. |
| 3 | Enumeration is fail-fast | **Enumeration** of ***ArrayList*** *and* **LinkedList both** is **fail-fast**; means any modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException in java.   |  | | --- | | Enumeration<String> listEnum= Collections.*enumeration*(arrayList);  **while**(listEnum.hasMoreElements()){  //adding element will throw   ConcurrentModificationException        System.*out*.println(listEnum.nextElement());  } | |
| 4 | Insertion order | ***ArrayList*** *and* **LinkedList both maintains insertion order** in java. |
| 5 | Allows null | ***ArrayList*** *and* **LinkedList** both **allows to store null** in java. |
| 6 | Implements java.util.List | ***ArrayList*** *and* **LinkedList** both are implementation of the java.util.**List** interface. |
| 7 | Introduced in which java version | ***ArrayList*** *and* **LinkedList** both were introduced in second version of java (1.2) i.e. **JDK 2.0** |

### How to clone collection in java and addAll use which cloning?

### To clone the collection we should use deep cloning. AddAll use shallow copy

### How to synchronize ArrayList?

### ArrayList return by collection.sy

### Difference b/w list, set and map?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Property | ***java.util.List*** | ***java.util.Set*** | ***java.util.Map*** |
| 1 | Duplicate elements | List **allows to store duplicate elements** in java. | *Set does* ***not allow to store duplicate elements*** in java*.* | *Map stores data in form of* ***key-value pair*** *it does not allow to store duplicate keys but allows duplicate values* in java*.* |
| 2 | Insertion order | Java.util.List is ordered collection it **maintain insertion order** in java. | *Most of the java.util.Set implementation* does not **maintain insertion order**.  Hash Set does not maintains insertion order in java.  Thought LinkedHashSet maintains insertion order in java.    TreeSet is sorted by natural order in java. | *Most of the java.util.Map implementation* does not **maintain insertion order**.  Hash Map does not maintains insertion order in java.  Thought LinkedHashMap maintains insertion order of keys in java.    Tree Map is sorted by natural order of keys in java. |
|  |  |  |  |  |
| 3 | Null keys | List allows to store **many null keys** in java. | Most of the Set implementations allow to add only **one null** in java**.**  [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) and [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html) does not allow to add null in java. | Lets look at Map implementations -  [HashMap](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) allows one null key and many null values.  [LinkedHashMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) allows one null key and many null values.  [TreeMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) doesn't allow null key but allow many null values.  [Hashtable](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html) doesn't allow null key or null values.  [ConcurrentHashMap](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html) doesn't allow null key or null values.  [ConcurrentSkipListMap](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) doesn't allow null key or null values. |
| 4 | Getting element on specific **index** | List implementations provide get method to get element on specific index in java.  ArrayList, Vector, copyOnWriteArrayList and LinkedList provides -  *get(int index)*  Method returns element on specified *index*.  **Get method directly gets element on specified index. Hence, offering O(1) complexity.** | Set implementations does not provide any such get method to get element on specified index in java. | Map implementations does not provide any such get method to get element on specified index in java. |
| 5 | Implementing classes | [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***,*** [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes implements [**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface in java. | [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) classes implements [**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface in java. | [HashMap, Hashtable, ConcurrentHashMap,  LinkedHashMap,  TreeMap,  ConcurrentSkipListMap,  IdentityHashMap,WeakHashMap,  EnumMap classes](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) implements [Map](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) interface in java. |
| 6 | listIterator | **listIterator** method returns listIterator to iterate over elements in List in java.  **listIterator provides** additional methods as compared to iterator like  **hasPrevious(), previous(), nextIndex(), previousIndex(), add(E element), set(E element)** | Set does not provide anything like listIterator. It simply return Iterator in java. | Map provides three type of iterators -  *map. Keyset().iterator()* method returns iterator to iterate over keys in HashMap  *map. Values ().iterator ()* method returns iterator to iterate over keys in HashMap in java.  *map.entrySet ().iterator ()* method returns iterator to iterate over keys in HashMap. |
| 7 | Structure and resizable | **List** are Resizable-array implementation of the java.util.**List** interface in java. | Set uses [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html)for their implementation.  Hence, structure is map based and resizing depends on Map implementation.  *Example >* [***Hash Set***](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) *internally uses* [*Hash Map*](http://javamadesoeasy.com/2015/02/hashmap-custom-implementation.html)*.* | [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) **uses hashing technique for storing** key-value pairs. |
| 8 | Index based structure /Random-access | As **ArrayList** uses array for implementation it is index based structure, hence provides random access to elements.  But [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) is not indexed based structure in java. | Set is not index-based structure at all in java. | Map is not index-based structure at all in java. |
| 9 | unsynchronized implementations | [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) | [**Hash Set**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) | [HashMap, LinkedHashMap, TreeMap, IdentityHashMap, WeakHashMap, EnumMap](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) |
| 10 | synchronized implementations | [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) | [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html) | [Hashtable](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html), [ConcurrentHashMap](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html), [ConcurrentSkipListMap](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html), |

* **Is Collection.synchronizedList thread-safe?**

### NO

* **Explain set hierarchy?**

### 

* **What is the initial capacity of hasset and what is loadfactor?**
* Initial capacity is 10 and Default load factor is 0.75
* That means when set will be 75% filled, it’s capacity will be doubled
* **Difference between Hash Set, Linked Hash Set and Tree Set?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Property | *java.util.HashSet* | *java.util.LinkedHashSet* | *java.util.TreeSet* |
| 1 | Insertion order | Java.util.HashSet does not maintains insertion order in java. | Java.util.LinkedHashSet maintains insertion order in java. | Java.util.TreeSet is sorted by natural order in java. |
| 2 | Null elements | HashSet allows storing **one null** in java**.** | LinkedHashSet allows storing **one null** in java. | TreeSet does **not** allows storing **any null** in java.  Any attempt to add null throws runtime Exception (NullPointerException). |
| 3 | Data structure internally used for storing data | For storing elements, HashSet internally uses HashMap. | For storing elements, LinkedHashSet internally uses LinkedHashMap. | For storing elements, TreeSet internally uses TreeMap. |
| 4 | Introduced  in which java version | java.util.HashSet was introduced in second version of java (1.2) i.e. **JDK 2.0** | java.util.LinkedHashSet was introduced in second version of java (1.4) i.e. **JDK 4.0** | java.util.TreeSet was introduced in second version of java (1.2) i.e. **JDK 2.0** |
| 5 | Implements which interface | HashSet implements **java.util.**[**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html)interface. | LinkedHashSet implements **java.util.Set** interface. | TreeSet implements **java.util.Set**  **java.util.SortedSet**  **Java.util.NavigableSet** interface. |
| 6 | Performance | HashSet comes first in performance. | LinkedHashSet comes in 2nd in performance. | TreeSet is comes in last in performance. |
|  | Comparison | HashSet use equal to compare element. | LinkedHashSet use equal to compare element. | TreeSet uses compare |

* **What copy technique internally used by HashSet clone () method?**
* Shallow copy
* **How set works internally and LinkedHashSet?**
* **HashSet** uses HashMap internally to store its objects. Whenever you create a HashSet object, one **HashMap** object associated with it is also created. This HashMap object is used to store the elements you enter in the HashSet. The elements you add into HashSet are stored as **keys** of this HashMap object. The value associated with those keys will be a **constant**.
* <https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html>
* LinkedHashSet works similar to HashSet. It uses the LinkedHashMap to store the element added to LinkedHashSet. LinkedHashSet does not have any method it inherit all the method from super class HashSet.
* **How TreeSet works internally?**