**List**:

|  |  |
| --- | --- |
| List |  |
| difference between ArrayList and Vector? |  |
| difference between ArrayList and LinkedList? |  |
| difference between Iterator and ListIterator? |  |
| difference between Iterator and Enumeration? |  |
| How to make ArrayList **read only** in Java | ArrayList readableList = new ArrayList();  List unmodifiableList = Collections.unmodifiableList(readableList); |
| **length()** of array and **size()** of ArrayList | length() = capacity  size() = number of elements currently stored |
| **CopyOnWriteArrayList vs ArrayList** | CopyOnWriteArrayList is thread-safety by creating a separate copy of List for each writes operation.  CopyOnWriteArrayList is a thread-safe collection while ArrayList is not thread-safe .  Iterator of ArrayList is fail-fast and throw ConcurrentModificationException once detect any modification in List once iteration begins but Iterator of CopyOnWriteArrayList is fail-safe and doesn't throw ConcurrentModificationException.  The third difference between CopyOnWriteArrayList vs ArrayList is that Iterator of former doesn't support remove operation while Iterator of later supports remove() operation.  In short, use CopyOnWriteArrayList if you mostly require to Iterate over list without modifying. |
| remove objects from ArrayList in Java? | Iterator<Integer> itr = numbers.iterator();  itr.remove();    List<Integer> list = new ArrayList<Integer>();  list.remove(1); |
| **CopyOnWriteArrayList in Java?** | 1. concurrent collection class which is introduced as an alternative of synchronized List in Java. 2. This class take advantage of advanced thread-safety technique instead of locking. 3. Its very efficient if ArrayList is mostly used for reading purpose, because it allows multiple threads to read data without locking, which was not possible with synchronized ArrayList. |
| **sublist from ArrayList in Java?** | arrayList.subList(1,3); |
| **HashMap and ArrayList in Java?** | HashMap access object using key while ArrayList access elements using index.  Though both provide O(1) search performance,  ArrayList's performance is guaranteed but HashMap can vary depending upon collision level. |
| **sort ArrayList in Java?** | Collections.sort() |
| initialize ArrayList in one line | List<String> coolStringList = Arrays.asList("Java", "Scala", "Groovy"); |
| Vector and ArrayList | Vector is synchronized while ArrayList is not synchronized |
| loop over ArrayList in Java |  |
| ArrayList and HashSet in Java? | ArrayList allowed duplicates.  HashSet doesn't allow duplicates, no ordering guarantee. |
| **When to use ArrayList and LinkedList in Java?** | **Similarity**:   1. Both ArrayList and LinkedList are an implementation of List interface 2. Both ArrayList and LinkedList are not synchronized 3. ArrayList and LinkedList are ordered collection e.g. they maintain insertion order 4. ArrayList and LinkedList also allow duplicates and null, 5. Iterator of both LinkedList and ArrayList are fail-fast which means they will throw ConcurrentModificationException 6. CopyOnWriteArrayList whose Iterator is fail-safe.   **Difference**:   1. ArrayList is backed by Array while LinkedList is backed by LinkedList. 2. Retrieving element from a position: O(1) in ArrayList while its O(n) in LinkedList 3. Adding elements: ArrayList is O(1), O(log(n)) in case of resize, LinkedList O(1)   especially if you are adding or removing elements from **start or end** because linked list internally keeps references of those positions and they are accessible in O(1) time. |
| **How to synchronize ArrayList in Java?** | Collections.synchronizedList() |
| **Difference between an array and ArrayList in Java?** | main difference between array and ArrayList is that former is static and later is dynamic. You cannot change the size of array once created, but ArrayList can grow and increase its size automatically. |
| **Reverse List** | Collections.reverse(list) method |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |