Q. How to Remove Duplicates from ArrayList in Java?

ArrayList is the most popular implementation of the List interface from Java's Collection framework, but it allows duplicates. Though there is another collection called Set which is primarily designed to store unique elements, there are situations when you receive a List like ArrayList in your code and you need to ensure that it doesn't contain any duplicate before processing. Since with ArrayList you cannot guarantee uniqueness, there is no other choice but to remove repeated elements from ArrayList. The simplest approach to remove repeated objects from ArrayList is to copy them to a Set e.g. HashSet and then copy it back to ArrayList. This will remove all duplicates without writing any more code. One thing to noted is that, if original order of elements in ArrayList is important for you, as List maintains insertion order, you should use LinkedHashSet because HashSet doesn't provide any ordering guarantee.

Java Program to Remove duplicates from ArrayList
Here is our sample program to learn how to remove duplicates from ArrayList. The steps followed in the below example are:

- Copying all the elements of ArrayList to LinkedHashSet. Why we choose LinkedHashSet? Because it removes duplicates and maintains the insertion order.
- Emptying the ArrayList, you can use clear() method to remove all elements of ArrayList and start fresh.
- Copying all the elements of LinkedHashSet (non-duplicate elements) to the ArrayList.

Please find below the complete code:

```
import java.util.ArrayList;

import java.util.LinkedHashSet;

import java.util.List;

import java.util.Set;
```

```
public class ArrayListDuplicateDemo

{

  public static void main(String args[])

  {

      List<Integer> primes = new ArrayList<Integer>();

      primes.add(2);

      primes.add(3);

      primes.add(5);

      primes.add(7);  //duplicate

      primes.add(7);

      primes.add(11);

     System.out.println("list of prime numbers : " + primes);

     Set<Integer> primesWithoutDuplicates = new
LinkedHashSet<Integer>(primes);

     primes.clear();

    primes.addAll(primesWithoutDuplicates);

    System.out.println("list of primes without duplicates : " + primes);

  }

 }
```

```
Output list of prime numbers: [2, 3, 5, 7, 7, 11]

List of primes without duplicates: [2, 3, 5, 7, 11]
```

Q. How to reverse ArrayList in Java with Example ?

You can reverse ArrayList in Java by using the reverse() method of java.util.Collections class. This is one of the many utility methods provided by the Collections class e.g. sort () method for sorting ArrayList. The Collections. reverse () method also accepts a List, so you not only can reverse ArrayList but also any other implementation of List interface e.g. LinkedList or Vector or even a custom implementation. This method has a time complexity of O(n) i.e. it runs on linear time because it uses ListIterator of the given list. It reverses the order of an element in the specified list.

By the way, you cannot reverse an ArrayList using this method if the specified ArrayList or its ListIterator doesn't support set() operation. It switches between two algorithms depending upon the size of List or if List implements RandomAccess interface like ArrayList.

If a number of elements in List are less than REVERSE_THRESHOLD, which is equal to 18 then it uses for loop for swapping elements otherwise it uses list iterator. If you want to learn more about *how the reverse() method of Collections works*, you can see it's code from JDK itself or in the next section

By the way, this is a typesafe generic method and you can use it to reverse Integer, String, Float or any kind of List in Java. You can also see the classic book Core Java Volume 1 - Fundamentals by Cay S. Horstmann to learn more about key classes of the Java Development Kit.

Java Program to reverse ArrayList in Java

You can see that we have added numbers on increasing order but after calling reverse() method, it prints them on decreasing order. Unlike popular misconception, Comparable or Comparator is not used while reversing ArrayList in Java. Though they are used if you want to sort Array in Java.

```
import java.util.ArrayList;

import java.util.Collections;

public class ArrayListReverseDemo

 { public static void main(String args[])

    { ArrayList<String> listOfInts = new ArrayList<>();

      listOfInts.add("1");

      listOfInts.add("2");

      listOfInts.add("3");

      listOfInts.add("4");

      listOfInts.add("5");

     System.out.println("Before Reversing : " + listOfInts);

     Collections.reverse(listOfInts);

      System.out.println("After Reversing : " + listOfInts);

   }

 }
```

How Collections.reverse() method works in Java

Here is the code snippet from java.util.Collections class which you can use to *reverse an ArrayList* or any kind of List in Java. You can see that it uses set() method of List interface for swapping elements and that's why you cannot reverse a read-only ArrayList because it doesn't support set() operation.

Logic of Collections.reverse() method

Here is the logic and explanation of how the Collections.reverse() method works and how it reverse the given collection.

```
public static void reverse(List<?> list)

 { int size = list.size();

   if (size < REVERSE_THRESHOLD || list instanceof RandomAccess)

   {

     for (int i=0, mid=size>>1, j=size-1; i<mid; i++, j--)

      swap(list, i, j);

   }

   else

   { ListIterator fwd = list.listIterator();

     ListIterator rev = list.listIterator(size);

     for (int i=0, mid=list.size()>>1; i<mid; i++)

    {  Object tmp = fwd.next();

        fwd.set(rev.previous()); rev.set(tmp);

    }

  }

}
```

Q. Difference between Array vs ArrayList in Java ?

Both Array and Array List are used to store elements, which can be either primitive or objects in case of Array and only objects in case of ArrayList in Java. The main *difference between Array vs ArrayList in Java* is the static nature of the Array and the dynamic nature of ArrayList. Once created you can not change the size of Array but ArrayList can re-size itself when needed. Another notable difference between ArrayList and Array is that Array is part of core Java programming and has special syntax and semantics support in Java,
While ArrayList is part of the Collection framework along with other popular classes like Vector, Hashtable, HashMap or LinkedList.
Array vs ArrayList in Java

1) First and Major difference between Array and ArrayList in Java is that Array is a fixed-length data structure while ArrayList is a variable-length Collection class. You can not change the length of Array once created in Java but ArrayList re-size itself when gets full depending upon the capacity and load factor.

Since ArrayList is internally backed by Array in Java, any resize operation in ArrayList will slow down performance as it involves creating a new Array and copying content from the old array to the new array.

2) Another difference between Array and ArrayList in Java is that you can not use Generics along with Array, as Array instance knows about what kind of type it can hold and throws ArrayStoreException if you try to store type which is not convertible into the type of Array. ArrayList allows you to use Generics to ensure type safety.

3) You can also compare Array vs ArrayList on *How to calculate the length of Array or size of ArrayList*. All kinds of Array provides length variable which denotes the length of Array while ArrayList provides size() method to calculate the size of ArrayList in Java.

4) One more major difference between ArrayList and Array is that, you can not store primitives in ArrayList, it can only contain Objects. While Array can contain both primitives and Objects in Java. Though Autoboxing of Java 5 may give you an impression of storing primitives in ArrayList, it actually automatically converts primitives to Object. e.g.

5) Java provides add() method to insert an element into ArrayList and you can simply use the assignment operator to store element into Array e.g. In order to store Object to specified position use

```
Object[] objArray = new Object[10];
objArray[1] = new Object();
```

6) One more difference on Array vs ArrayList is that you can create an instance of ArrayList without specifying size, Java will create Array List with default size but it's mandatory to provide the size of Array while creating either directly or indirectly by initializing Array while creating it. By the way, you can also initialize ArrayList while creating it.

Q. How to Synchronize ArrayList in Java with Example?

ArrayList is a very useful Collection in Java, I guess most used one as well but it is not synchronized. What this mean? It means you cannot share an instance of ArrayList between multiple threads if they are not just reading from it but also

writing or updating elements. So *how can we synchronize ArrayList?* Well, we'll come to that in a second but did you thought why ArrayList is not synchronized in the first place? Since multi-threading is a core strength of Java and almost all Java programs have more than one thread, why Java designer does not make it easy for ArrayList to be used in such an environment?
The answer lies in performance, there is a performance cost associated with synchronization, and making ArrayList synchronized would have made it slower. So, they definitely thought about it and left ArrayList as non-synchronized to keep it fast, but at the same time they have provided easy ways to make it synchronized

The JDK Collections class has several methods to create synchronized List, Set and Map and we will use Collections.synchronizedList() method to make our ArrayList synchronized. This method accepts a List which could be any implementation of the List interface e.g. ArrayList, LinkedList, and returns a synchronized (thread-safe) list backed by the specified list. So you can also use this technique to make LinkedList synchronized and thread-safe in Java.

Synchronized List and Iteration

One of the main challenges of sharing ArrayList between multiple threads is how to deal with a situation where one thread is trying to access the element which is removed by other. If you are using methods
like get(index) or remove(index) method to retrieve or remove elements then it's also possible that other thread may also be removing other elements.

This means you cannot call get(index) or remove(index) reliably without checking the size of the list first and then you also needs to provide extra synchronization between your call to size() and remove(int index).

In order to guarantee serial access, it is critical that all access to the backing list is accomplished through the returned list and it is imperative that the user manually synchronizes on the returned list when iterating over it as shown in the following sample code :

```
List list = Collections.synchronizedList(new ArrayList());

    ...
  synchronized(list) {
     Iterator i = list.iterator(); // Must be in synchronized block
     while (i.hasNext())
        foo(i.next());
 }
```

Java Program to Synchronize ArrayList

Here is the complete example of synchronizing an ArrayList in Java. If you look, we have created a List of String and added a couple of elements to it. Then we passed this ArrayList to Collections.synchronizedList() method, which returned a thread-safe, synchronized version of the backed list.
You can now safely share this list among multiple threads, but you need to be a little bit careful while retrieving or removing elements from ArrayList. In order to have safe access, you should use Iterator for getting and removing elements from the list and that too in a synchronized manner as shown in this example.

```
import java.util.ArrayList;
import java.util.Collections;

import java.util.Iterator;

import java.util.List;

public class SynchronizedArrayListDemo

{ public static void main(String args[])

   { // An ArrayList which is not synchronize

      List<String> listOfSymbols = new ArrayList<String>();
```

```
       listOfSymbols.add("RELIANCE");
       listOfSymbols.add ("TATA");
       listOfSymbols.add ("TECHMAH");
       listOfSymbols.add ("HDFC");
       listOfSymbols.add ("ICICI"); // Synchronizing ArrayList in Java
      listOfSymbols = Collections.synchronizedList (listOfSymbols);
       synchronized (listOfSymbols)
        { Iterator<String> myIterator = listOfSymbols.iterator ();
          while(myIterator.hasNext())
          {  System.out.println(myIterator.next());
          }
        }
      }
  }
```

Q. When to use ArrayList vs LinkedList in Java?

ArrayList and LinkedList are two popular concrete implementations of the List interface from Java's popular Collection framework. Being List implementation both ArrayList and LinkedList are ordered, the index-based and allows duplicate. Despite being from the same type of hierarchy there are a lot of differences between these two classes which makes them popular among Java interviewers. The main difference between ArrayList vs LinkedList is that the former is backed by an array while the latter is based upon the linked list data structure, which makes the performance of add(), remove(), contains(), and iterator() different for both ArrayList and LinkedList. The difference between ArrayList and LinkedList is also an important Java collection interview question, as much popular as Vector

vs ArrayList or HashMap vs HashSet in Java. Sometimes this is also asked as for when to use LinkedList and when to use ArrayList in Java.
In this Java collection tutorial, we will compare LinkedList vs ArrayList on various parameters which will help us to decide when to use ArrayList over LinkedList in Java. We will not focus on the array and linked list data structure much, which is subject to data structure and algorithm, we'll only focus on the Java implementations of these data structures which are ArrayList and LinkedList.

When to use ArrayList vs LinkedList in Java

Before comparing differences between ArrayList and LinkedList, let's see What is common between ArrayList and LinkedList in Java:
1) Both ArrayList and LinkedList are an implementation of the List interface, which means you can pass either ArrayList or LinkedList if a method accepts the java.util.List interface.

2) Both ArrayList and LinkedList are not synchronized, which means you cannot share them between multiple threads without external synchronization. See here to know.

3) ArrayList and LinkedList are ordered collection e.g. they maintain insertion order of elements i.e. the first element will be added to the first position.

4) ArrayList and LinkedList also allow duplicates and null, unlike any other List implementation e.g. Vector.

5) An iterator of both LinkedList and ArrayList are fail-fast which means they will throw ConcurrentModificationException if a collection is modified structurally once the Iterator is created. They are different than CopyOnWriteArrayList whose Iterator is fail-safe.

Difference between LinkedList and ArrayList in Java
Now let's see some differences between ArrayList and LinkedList and when to use ArrayList and LinkedList in Java.
1. Underlying Data Structure

The first difference between ArrayList and LinkedList comes with the fact that ArrayList is backed by Array while LinkedList is backed by LinkedList. This will lead to further differences in performance.

2. LinkedList implements Deque

Another difference between ArrayList and LinkedList is that apart from the List interface, LinkedList also implements the Deque interface, which provides first in first out operations for add() and poll() and several other Deque functions.

3. Adding elements in ArrayList

Adding an element in ArrayList is O(1) operation if it doesn't trigger re-size of Array, in which case it becomes O(log(n)), On the other hand, appending an element in LinkedList is O(1) operation, as it doesn't require any navigation.

4. Removing an element from a position

In order to remove an element from a particular index e.g. by calling remove(index), ArrayList performs a copy operation which makes it close to O(n) while LinkedList needs to traverse to that point which also makes it O(n/2), as it can traverse from either direction based upon proximity.

5. Iterating over ArrayList or LinkedList

Iteration is the O (n) operation for both LinkedList and ArrayList where n is a number of an element.

6. Retrieving element from a position

the get (index) operation is O (1) in ArrayList while its O (n/2) in LinkedList, as it needs to traverse till that entry. Though, in Big O notation O (n/2) is just O (n) because we ignore constants there.

7. Memory

LinkedList uses a wrapper object, Entry, which is a static nested class for storing data and two nodes next and previous while ArrayList just stores data in Array. So memory requirement seems less in the case of ArrayList than LinkedList except for the case where Array performs the re-size operation when it copies content from one Array to another.

If Array is large enough it may take a lot of memory at that point and trigger Garbage collection, which can slow response time.

From all the above differences between ArrayList vs LinkedList, It looks like ArrayList is the better choice than LinkedList in almost all cases, except when you

do a frequent add () operation than remove (), or get ().
It's easier to modify a linked list than ArrayList, especially if you are adding or removing elements from start or end because the linked list internally keeps references of those positions and they are accessible in O(1) time.
In other words, you don't need to traverse through the linked list to reach the position where you want to add elements, in that case, addition becomes an O(n) operation. For example, inserting or deleting an element in the middle of a linked list

Q. Difference between ArrayList and HashSet in Java?

1. First and most important difference between ArrayList and HashSet is that ArrayList implements List interface while HashSet implements Set interface in Java.
2. Another difference between ArrayList and HashSet is that ArrayListallow duplicates while HashSet doesn't allow duplicates. This is the side effect of first difference and property of implementing List and Set interface.

3. The differences between ArrayList and HashSet is that ArrayList is an ordered collection and maintains insertion order of elements while HashSet is an unordered collection and doesn't maintain any order.

4. The difference between ArrayList and HashSet is that ArrayList is backed by an Array while HashSet is backed by a HashMap instance

5. Fifth difference between HashSet and ArrayList is that it's index-based you can retrieve objects by calling get(index) or remove objects by calling remove(index) while HashSet is completely object-based. HashSet also doesn't provide the get() method.

Similarities ArrayList and HashSet

1) Both ArrayList and HashSet are non synchronized collection classes and not meant to be used in multi-threading and concurrent environments. You can make ArrayList and HashSet synchronized by

Using Collections.synchroinzedCollection () just like we
Make ArrayList and HashSet read-only other days

2) Both ArrayList and HashSet can be traversed using Iterator. This is in fact a preferred way if you want to perform operations on all elements.
3) Iterator of ArrayList and HashSet both are fail-fast, i.e. they will throw ConcurrentModificationException if ArrayList or HashSet is modified structurally once Iterator has been created.

Q. Difference between Vector and ArrayList in Java?

ArrayList and Vector are the two most widely used Collection classes in Java and are used to store objects in an ordered fashion. Every Java programmer which is introduced to Java Collection Framework either started with Vector or ArrayList

Vector vs ArrayList in Java

1. Synchronization

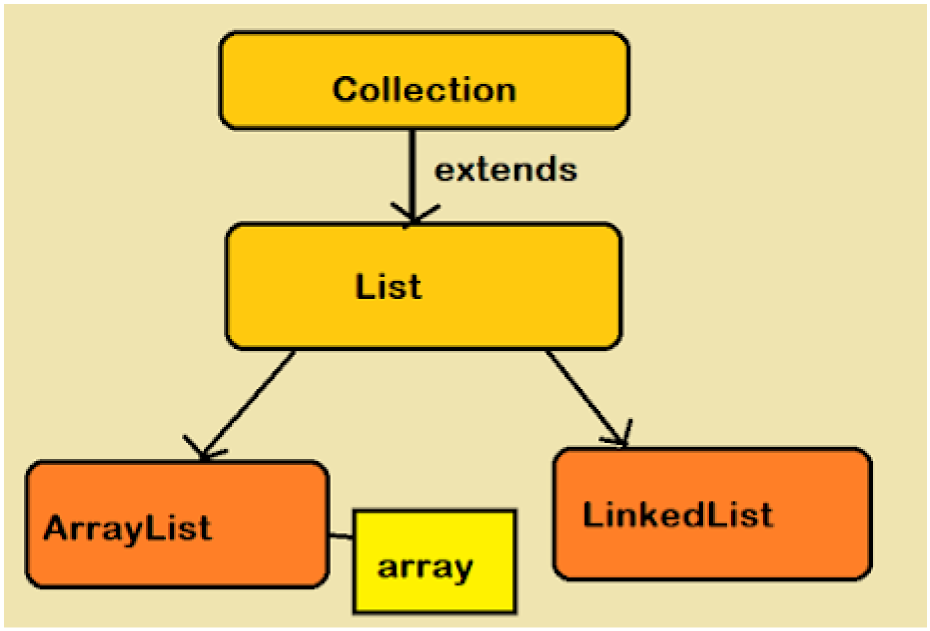
The first and most common *difference between Vector vs ArrayList* is that Vector is synchronized and thread-safe while ArrayList is neither Synchronized nor thread-safe. Now, What does that mean? It means if multiple thread try to access Vector same time they can do that without compromising Vector's internal state. Same is not true in case of ArrayList as methods
like add(), remove() or get() is not synchronized.

2. Speed

The second major difference on Vector vs ArrayList is Speed, which is directly related to previous difference. Since Vector is synchronized, its slow and ArrayList is not synchronized its faster than Vector.

3. Legacy Class

The third difference on Vector vs ArrayList is that Vector is a legacy class and initially it was not part of the Java Collection Framework. From Java 1.4 Vector was retrofitted to implement List interface and become part of Collection Framework.



Q. How to sort ArrayList in Java?

Sorting ArrayList in Ascending Order in Java

First, let's see how to sort an array in ascending order in Java using the Collections.sort() method. This method is overloaded, which means you can sort the ArrayList in natural order by leveraging the default comparator, which sorts the list in the natural order, and you can use the Collections.sort(list, Comparator) to sort the ArrayList in custom order defined by Comparator. Since we are sorting ArrayList of String which implements a Comparable method, we can use the first method to sort the ArrayList string into the natural order or ascending order. You can also use this method to sort an ArrayList of Integer into increasing order because that's the default order for an Integer object.

```
import java.util.*;

class Main

{  public static void main(String[] args)

   { List<Integer> listofYears = new ArrayList<Integer>();

      listofYears.add(2021);

      listofYears.add(2019);

      listofYears.add(2018);

    listofYears.add(2020); // print the ArrayList before sorting

   System.out.println("ArrayList before sorting");

   System.out.println(listofYears); // sorting an ArrayList of Integer in ascending
order

   Collections.sort(listofYears); // print the ArrayList after sorting

    System.out.println("ArrayList after sorting");

      System.out.println(listofYears);

  }

 }
```

Q. Difference between ArrayList and HashMap in Java?

1. The first difference between ArrayList and HashMap is
that ArrayList implements a List interface
while HashMap implements Map interface in Java.

2. The second difference between ArrayList and HashMap is that ArrayList only stores one object while HashMap stores two objects key and value.

3. The third difference between HashMap and ArrayList is that keys of HashMap must implement equals and hashCode method correctly, ArrayList doesn't have that requirement but its good to have that because contains() method of ArrayList will use the equals() method to see if that object already exists or not.

4. The fourth difference between HashMap and ArrayList is that ArrayList maintains the order of objects, in which they are inserted while HashMap doesn't provide any ordering guarantee.

5. Another difference between ArrayList and HashMap is that ArrayList allows duplicates but HashMap doesn't allow duplicates key though it allows duplicate values.

6. ArrayList get(index) method always gives an O(1) performance but HashMap get(key) can be O(1) in the best case and O(n) in the worst case.

Q. Java program to get SubList from ArrayList – Example?

Sometimes we need *subList from ArrayList in Java*. For example, we have an ArrayList of 10 objects and we only need 5 objects or we need an object from index 2 to 6, these are called subList in Java. Java collection API provides a method to *get SubList from ArrayList*. In this Java tutorial, we will see an example of getting SubList from ArrayList in Java. In this program, we have an ArrayList which contains 4 String objects. Later we call ArrayList.subList () method to get part of that List

Example with Source code

```
import java.util.ArrayList;

import java.util.List;

public class GetSubListExample {
       public static void main(String[] args) {

         ArrayList arrayList = new ArrayList();

           //Add elements to Arraylist

           arrayList.add("Java");

           arrayList.add("C++");

           arrayList.add("PHP");

           arrayList.add("Scala");

           lst = arrayList.subList(1,3);

         System.out.println("Sub list contains : ");

           for(int i=0; i&lt; lst.size() ; i++)

             System.out.println(lst.get(i));

           //remove one element from sub list

           Object obj = lst.remove(0);

           System.out.println(obj + " is removed from sub list");

           //print original ArrayList

   System.out.println("After removing " + obj + " from sub list, original ArrayList
contains : ");

           for(int i=0; i&lt; arrayList.size() ; i++)

             System.out.println(arrayList.get(i));

         }

    }
```

Q. Array length vs ArrayList Size in Java?

One of the confusing parts in learning Java for a beginner to understand how to find the length of array and ArrayList in Java? The main reason for the confusion is an inconsistent way of calculating the length between two. Calling size() method on arrays and length, or even length() on ArrayList is a common programming error made by beginners. The main reason for the confusion is the special handling of an array in Java. Java native arrays have built-in length attribute but no size() method while the Java library containers, known as Collection classes like ArrayList<>, Vector<>, etc, all have a size() method.
There is one more thing which adds to this confusion, that is capacity, at any point capacity of any collection class is the maximum number of elements collection can hold. the size of collection must be less than or equal to its capacity.

Though in reality, collection resize themselves even before it reaches its capacity, controlled by laod factor. I have mentioned this before on my post difference between ArrayList and Array in Java, if you not read it already, you may find some useful detail there as well.

So, use length attribute to get number of elements in a array, also known as length, and for same thing in Collection classes e.g. ArrayList, Vector, use size() method. To give you more context, consider following lines of code, can you spot the error, which is bothering our beginner friend:

```
import java.util.*;

public class ArrayListTest

{   public static void main(String[] args)

  {    ArrayList<String> arrList = new ArrayList<String>();

      String[] items = { "One", "Two", "Three", "Four", "Five" };

     for(String str: items){

          arrList.add(str);
```

```
        }
  int size = items.size();

        System.out.println (size);

    }

}
Arrays.asList("First", "Second", "Third", "Fourth", "Fifth");
```

It's handy way to declare and initialize ArrayList in same place, similar to array in Java. So next time don't confuse between length and size(), array are special object in Java and has attribute called length, which specifies number of buckets in array, while ArrayList is a Collection class, which inherit size() method, which returns number of elements inside Collection. Remember, size is different than capacity of collection. At any point, size <= capacity of collection.

Q. What is CopyOnWriteArrayList in Java ?

CopyOnWriteArrayList is a concurrent Collection class introduced in Java 5 Concurrency API along with its popular cousin ConcurrentHashMap in Java. CopyOnWriteArrayList implements List interface like ArrayList, Vector, and LinkedList but its a thread-safe collection and it achieves its thread-safety in a slightly different way than Vector or other thread-safe collection class. As the name suggests CopyOnWriteArrayList creates a copy of underlying ArrayList with every mutation operation e.g. add, remove, or when you set values. That's why it is only suitable for a small list of values which are read frequently but modified rarely e.g. a list of configurations.
Normally CopyOnWriteArrayList is very expensive because it involves costly Array copy with every writes operation but it's very efficient if you have a List where Iteration outnumbers mutation e.g. you mostly need to iterate the ArrayList and don't modify it too often.
Iterator of CopyOnWriteArrayList is fail-safe and doesn't throw ConcurrentModificationException even if underlying CopyOnWriteArrayList is modified once Iteration begins because

Iterator is operating on a separate copy of ArrayList. Consequently, all the updates made on CopyOnWriteArrayList is not available to Iterator

Difference between CopyOnWriteArrayList and ArrayList in Java.

1) First and foremost difference between CopyOnWriteArrayList and ArrayList in Java is that CopyOnWriteArrayList is a thread-safe collection while ArrayList is not thread-safe and cannot be used in the multi-threaded environment.

2) The second difference between ArrayList and CopyOnWriteArrayList is that 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.

3) The third difference between CopyOnWriteArrayList vs ArrayList is that Iterator of former doesn't support remove operation while Iterator of later supports remove() operation. If you want to learn more about collections, I suggest you go through Complete Java MasterClass, one of the best Java course on Udemy.

```
import java.util.Iterator;

import java.util.concurrent.CopyOnWriteArrayList;

public class CopyOnWriteArrayListExample

{

    public static void main(String args[])

    {

     CopyOnWriteArrayList<String> threadSafeList = new
CopyOnWriteArrayList<String>();
```

```
        threadSafeList.add("Java");

        threadSafeList.add("J2EE");

        threadSafeList.add("Collection");


    Iterator<String> failSafeIterator = threadSafeList.iterator();

        while(failSafeIterator.hasNext()){

            System.out.printf("Read from CopyOnWriteArrayList : %s %n",
failSafeIterator.next());

            failSafeIterator.remove(); //not supported in CopyOnWriteArrayList in Java

        }

    }

}
```

Q. Ways to Remove Elements/Objects From ArrayList in Java ?

There are *two ways to remove objects from ArrayList in Java*, first, by using the remove() method, and second by using Iterator. ArrayList provides overloaded remove () method, one accepts the index of the object to be removed i.e. remove(int index), and the other accept objects to be removed, i.e. remove (Object obj). The rule of thumb is, If you know the index of the object, then use the first method, otherwise use the second method. By the way, you must remember to use ArrayList remove methods, only when you are not iterating over ArrayList if you are iterating then use Iterator. Remove() method, failing to do so may result in ConcurrentModificationException in Java
Another gotcha can have occurred due to autoboxing. If you look closely that two remove methods, remove (int index) and remove (Object obj) are indistinguishable if you are trying to remove from an ArrayList of Integers.

Code Example To Remove Elements from ArrayList

Let's test the above theory with a simple code example of ArrayList with Integers. The following program has an ArrayList of Integers containing 1, 2, and 3 i.e. [1, 2, 3], which corresponds exactly to the index.

Suppose you have three objects in ArrayList i.e. [1,2,3] and you want to remove the second object, which is 2. You may call remove(2), which is actually a call to remove(Object) if consider autoboxing, but will be interpreted as a call to remove 3rd element, by interpreting as remove(index).

Q. How to Create Read Only, Unmodifiable ArrayList in Java?

You can create read-only Collection by using Collections.unmodifiableCollection () utility method. it returns an unmodifiable or read-only view of Collection in which you cannot perform any operation which will change the collection like add(), remove() and set() either directly or while iterating using Iterator or ListIterator. It will throw UnsupportedOperationException whenever you try to modify the List.
One of the common misconceptions around read-only ArrayList is that you can create read-only ArrayList by using Arrays.asList(String{[]), which is apparently not true as this method only return a fixed-size list.
You cannot perform add() and remove() but the set() method is still allowed which can change the contents of ArrayList. Collections class also provides a different method to make List and Set read-only. In this Java tutorial, we will learn *How to make any collection read only* and How to create a fixed size List as well.

Example

```
import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collection;

import java.util.Collections;

import java.util.List;
```

```
public class ReadOnlyCollection {


  public static void main(String args[])

    {

     Collection readOnlyCollection = Collections.unmodifiableCollection(new
ArrayList<String>());

               ArrayList readableList = new ArrayList();

               readableList.add("Jeffrey Archer");

               readableList.add("Khalid Hussain");

               List unmodifiableList = Collections.unmodifiableList(readableList);

                unmodifiableList.add("R.K. Narayan");

                unmodifiableList.remove(0);

                unmodifiableList.set(0, "Anurag Kashyap");

                fixedLengthList.set(0, "J.K. Rowling");

                 System.out.println(fixedLengthList.get(0));

     }

}
```