\*\*\*\*Collections interview programs

Arraylist

1. basic arraylist operation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Description:** | | Here we can see example for basic ArrayList operations like creating object for ArrayList, adding objects into ArrayList, accessing objects based on index, searching an object in ArrayList whether it is listed under this instance or not, adding elements at specific index, checking whether the ArrayList is empty or not, getting object index, and finally size of the ArrayList. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-collections-and-util/arraylist/basic-operations/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | package com.javacoffee.arraylist;    import java.util.ArrayList;    public class MyBasicArrayList {        public static void main(String[] a){            ArrayList<String> al = new ArrayList<String>();          //add elements to the ArrayList          al.add("JAVA");          al.add("C++");          al.add("PERL");          al.add("PHP");          System.out.println(al);          //get elements by index          System.out.println("Element at index 1: "+al.get(1));          System.out.println("Does list contains JAVA? "+al.contains("JAVA"));          //add elements at a specific index          al.add(2,"PLAY");          System.out.println(al);          System.out.println("Is arraylist empty? "+al.isEmpty());          System.out.println("Index of PERL is "+al.indexOf("PERL"));          System.out.println("Size of the arraylist is: "+al.size());      }  } | | |
| |  | | --- | | **Output:** | | [JAVA, C++, PERL, PHP]  Element at index 1: C++  Does list contains JAVA? true  [JAVA, C++, PLAY, PERL, PHP]  Is arraylist empty? false  Index of PERL is 3  Size of the arraylist is: 5 | |

1. how to read all element in arraylist by using iterator.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Description:** | | Here we can see example for reading all elements from ArrayList by using Iterator. Also you can iterate through the ArrayList based on index too. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-collections-and-util/arraylist/iterator/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package com.javacoffee.arraylist;    import java.util.ArrayList;  import java.util.Iterator;    public class ArrayListIterator {        public static void main(String a[]){          ArrayList<String> arrl = new ArrayList<String>();          //adding elements to the end          arrl.add("First");          arrl.add("Second");          arrl.add("Third");          arrl.add("Random");          Iterator<String> itr = arrl.iterator();          while(itr.hasNext()){              System.out.println(itr.next());          }      }  } | | |
| |  | | --- | | **Output:** | | First  Second  Third  Random | |

4.> how to find nth element from end of linked list in java.

**Assumption:**

We do not know size of linkedlist otherwise we can directly iterate and find (length-n)th node

**Algorithm for this problem would be :**

* Use two pointer firstPtr and secondPtr and initialize both to head of linkedlist
* Move firstPtr by n-1 nodes.
* Increment firstPtr and secondPtr until firstPtr.next not equal to null.
* SecondPtr will be at nth from end node.

Java program for this will be :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | package org.arpit.javacoffee;    public class LinkedList{    private Node head;    private static class Node {  private int value;  private Node next;    Node(int value) {  this.value = value;    }  }    public void addToTheLast(Node node) {    if (head == null) {  head = node;  } else {  Node temp = head;  while (temp.next != null)  temp = temp.next;    temp.next = node;  }  }      public void printList() {  Node temp = head;  while (temp != null) {  System.out.format("%d ", temp.value);  temp = temp.next;  }  System.out.println();  }      public Node nthFromLastNode(Node head,int n)  {  Node firstPtr=head;  Node secondPtr=head;    for (int i = 0; i < n; i++) {  firstPtr=firstPtr.next;    }    while(firstPtr!=null)  {  firstPtr=firstPtr.next;  secondPtr=secondPtr.next;  }    return secondPtr;  }    public static void main(String[] args) {  LinkedList list = new LinkedList();  // Creating a linked list  Node head=new Node(5);  list.addToTheLast(head);  list.addToTheLast(new Node(6));  list.addToTheLast(new Node(7));  list.addToTheLast(new Node(1));  list.addToTheLast(new Node(2));    list.printList();  // Finding 3rd node from end  Node nthNodeFromLast= list.nthFromLastNode(head,3);  System.out.println("3th node from end is : "+ nthNodeFromLast.value);    }    } |

Logically our linkedlist look like following:

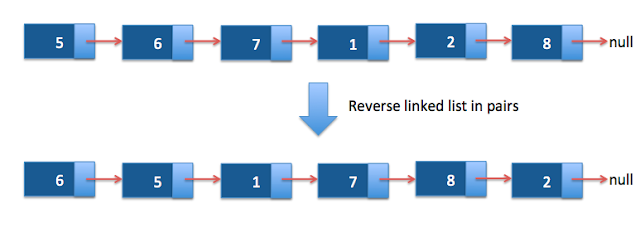
**[https://3.bp.blogspot.com/-foKjwd459OI/VcWoTxo6SUI/AAAAAAAAD2Y/7HhMlNq6u0Q/s640/LinkedListWithMiddleElement.jpg](https://www.java2blog.com/wp-content/uploads/2014/07/LinkedListWithMiddleElement-2.jpg)**

Color node represent 3rd node from last.  
Run above program, you will get following output:

|  |  |
| --- | --- |
| 1  2  3  4 | 5 6 7 1 2  3th node from end is : 7 |

5.> how to reverse linked list in pairs.

This is one of popular interview question. In this post, we will see how to reverse linked list in pairs.

**[](https://2.bp.blogspot.com/-8N1scpgMXSQ/Vv7OSOBFbAI/AAAAAAAAEbA/tPo7uYjgUn8rrVFgWQdFcaox-lx78qJww/s1600/ReverseLinkedListInPairs.png)**

There can be two solution for reversing linked list in pairs

* Iterative
* Recursive

**Iterative:**

Logic for this would be:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | public static Node reverseLinkedListInPairItr(Node head) {    Node current=head;  Node temp=null;  Node newHead =null;  while (current != null && current.next != null) {    if (temp != null) {  // This is important step  temp.next.next = current.next;  }  temp=current.next;  current.next=temp.next;  temp.next=current;    if (newHead == null)  newHead = temp;  current=current.next;    }  return newHead;  } |

**Explanation:**

Lets understand with the help of simple example:  
Lets say linkedlist is **5-> 6 -> 7 -> 1**  
If you look closely, below steps are just reversing link to **6->5->7->1** in first iteration (Swapping node 6 and 5) and then advance to next pair (Node 7 and 1)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | // Node 6 is temp node  temp=current.next;  // putting link between 5->7  current.next=temp.next;  // putting link between 6->5  temp.next=current;  // 6 becomes head node here |

You must be wondering why we have put below code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | if (temp != null) {        // This is important step        temp.next.next = current.next;  } |

before swapping two nodes , we need to link nodes properly. When we are swapping 7 and 1 , link between 5->7 will be broken and we need to connect node 5 to node 1.  
As per above code , temp will be null in first iteration(swapping node 6 and 5) but when current node is 7, temp will be 6, here we are linking nodes as below  
**6->5->1->7**

**Java program for this will be :**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | package org.arpit.java2blog;    public class ReverseLinkedListInPair{    private Node head;    private static class Node {  private int value;  private Node next;    Node(int value) {  this.value = value;    }  }    public void addToTheLast(Node node) {    if (head == null) {  head = node;  } else {  Node temp = head;  while (temp.next != null)  temp = temp.next;    temp.next = node;  }  }      public void printList(Node head) {  Node temp = head;  while (temp != null) {  System.out.format("%d ", temp.value);  temp = temp.next;  }  System.out.println();  }    // Reverse linked list in pair  public static Node reverseLinkedListInPairs(Node head) {    Node current=head;  Node temp=null;  Node newHead =null;  while (current != null && current.next != null) {    if (temp != null) {  // This is important step  temp.next.next = current.next;  }  temp=current.next;  current.next=temp.next;  temp.next=current;    if (newHead == null)  newHead = temp;  current=current.next;    }  return newHead;  }      public static void main(String[] args) {  ReverseLinkedListInPair list = new ReverseLinkedListInPair();  // Creating a linked list  Node head=new Node(5);  list.addToTheLast(head);  list.addToTheLast(new Node(6));  list.addToTheLast(new Node(7));  list.addToTheLast(new Node(1));  list.addToTheLast(new Node(2));  list.addToTheLast(new Node(8));    list.printList(head);  //Reversing LinkedList in pairs  Node result=reverseLinkedListInPairs(head);  System.out.println("After reversing in pair");  list.printList(result);  }  } |

Run above program, you will get following output:

|  |  |
| --- | --- |
| 1  2  3  4  5 | 5 6 7 1 2 8  After reversing  6 5 1 7 8 2 |

#### Recursive:

**Base case:** Base case for this would be either node is null or node.next is null

For recursive solution, replace reverseLinkedListInPairs of above program to below function

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public static Node reverseLinkedListInPairs(Node head) {        if (head == null || head.next == null) {            return head;        }    // Lets take example of 5->6->7    // Here head node is 5    // Storing 6 in temp node, it will become our new head       Node temp=head.next;    // Putting link between 5->7       head.next=temp.next;    // putting link between 6->5       temp.next=head;    // recursively calling the function for node 7       head.next=reverseLinkedListInPairRec(head.next);    // returning new head       return temp;    } |

Output of this program will be same as above program.

6.> basic operation of linked list.

## Linked List Operations

The operations on a list are all implemented by the manipulation of the links on each node. However, care must be taken in the order of how the links are handled; if they are done the wrong way, data can be lost.

Why would data be lost if the manipulations were done incorrectly?

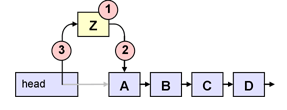
[Answer](javascript:popUp('qn/node-manipulation.html'))

We'll first look at the algorithms, and then write some source code to match the linked list skeleton we looked at earlier. These methods would go inside of the header class.

## Adding to a List

In order to add data to the list, we would need to create a new node (to store the new piece of data), and then manipulate the links so that the chain would also include this new node. Manipulations are done from right-to-left, to avoid nodes being left without a reference to them (and then be summarily destroyed by the garbage collector.)

### Adding to the Head

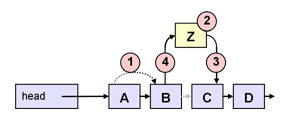


For example, to add to the head of a list, a new data node will need to be 'squeezed' in between the header reference, and the first data node. The order in which this is done to ensure no data loss takes place is:

1. Create the new node, using the data to be inserted
2. Set the new node's next reference, to what the head is pointing to
3. Set the header reference to the new node.

Performing these steps in a different order will result in part of the list not being referable by a variable.

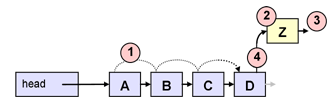
### Adding to the Middle



The basic process is similar for adding to the middle and end of a linked list, however the references that change are different. To add to the middle of a list, the basic process is:

1. Seek through the list, until the node before the desired insertion point is found.
2. Create the new node, using the data to be inserted
3. Set the new node's next reference to the same as the next reference of the node before the insertion point
4. Set the node before the insertion point's next reference to the new node

### Adding to the End



To add to the end of a linked list, the process is:

1. Seek through the list until the final node is reached
2. Create a new node, using the data to be inserted
3. Set the new node's next reference to null
4. Set the last node's next reference to the new node

These three algorithms are very similar (they all perform the same basic manipulation; the only difference is the place where these manipulations occur, and whether header references need to be taken care of.)

"Intelligent" addition algorithms (for example, those that determine where to add data based on certain requirements such as preserving a sort order, or keeping priorities on certain forms of data) may need to apply any one of these three slightly different algorithms under differing circumstances.

We'll be discussing one of these intelligent algorithms - add in sort order - shortly.

### Source Code

This simple addition method will create a node for some integer data that is passed into the method, and place it at the head of the list.

This code fragment is part of [LinkedList.java](http://www.cs.rmit.edu.au/online/blackboard/chapter/05/documents/contribute/chapter/03/documents/LinkedList.java).

public void add(int number)  
{

**ListNode newNode = new ListNode(number);**

create a new node from the number passed in

**newNode.next = head;**

set the new node's next to the existing head

**head = newNode;**

}

set the head to the new node, completing the link

## Traversing a List

It is sometimes necessary to traverse the entire length of the list to perform some function (for example, to count the number of items, or display summary information.)

List traversal forms the basis of many of the list manipulation operations such as [complex] add, retrieve and delete.

list traversal

### Source Code

The code below will traverse the entire list, and print out the data contained in each node.

This code fragment is part of [LinkedList.java](http://www.cs.rmit.edu.au/online/blackboard/chapter/05/documents/contribute/chapter/03/documents/LinkedList.java).

public void traverse()  
{

**ListNode current = head;**

keep a reference to the current list position, starting at the head

**while (current != null)  
 {**

repeat until the end is reached

**System.out.println(current.number);**

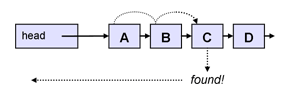
perform an operation at the node

**current = current.next;**  
 **}**  
}

move to the next list item

## Searching Through a List

In order to search through the list (to find a piece of data, or an insertion point for some new data), the only option is to traverse through the data one by one, from the start. This is known as a linear search. More efficient search techniques (such as the binary search) cannot be performed, as the link structure between data forces sequential access.



The step-by-step algorithm to search is, starting at the first data node, and comparing the search key with the corresponding data in the node:

1. if the data matches, the search is complete.
2. if there is no match, move to the next node and repeat;
3. If the next reference is null, the end of the list has been reached; therefore, the data does not exist in the list. The algorithm can now terminate.

### Source Code

The method below is the full implementation of searching for data, however instead of retrieving the data, it merely reports that it exists. (In this particular linked list, returning the data is of little value - the search key was the data.)

It is a minor matter to modify this to return the data; all that is needed is a change to what it is that gets returned from the method.

This code fragment is part of [LinkedList.java](http://www.cs.rmit.edu.au/online/blackboard/chapter/05/documents/contribute/chapter/03/documents/LinkedList.java).

public boolean exists(int target)  
{

**ListNode current = head;**

keep a variable to store where we are in the list

**while (current != null)  
 {**

while we haven't reached the end of the list...

**if (current.number == target)  
 return true;**

if we have found the data, return true

**current = current.next;**

**}**

otherwise, move to the next item and try again

**return false;**  
}

if we get here, nothing was ever found, so return false

## Adding in Sort Order

Adding data in sort order is somewhat more complex. This is because we do not know beforehand which insertion strategy is needed, and must be determined on-the-fly. In order to find where we need to add, we'll need to use traversal and searching, as well as any one of the possible addition algorithms.

The algorithm is as follows:

1. Traverse the list and find the place where the data should be stored. Both ‘current’ and ‘previous’ references will need to be maintained.
2. Create a new node to store the given data.
3. Set the new node’s next reference to that of the insertion point node.
4. Reset the insertion point’s next reference to point to the newly created node.

Also note that steps 3 and 4 will need to be implemented slightly differently if the insertion point is the head.

### Source Code

This complication is reflected in the code; the traversal needs to know when to stop iterating, and must also maintain all variables necessary for either insertion strategy.

This code fragment is part of [LinkedList.java](http://www.cs.rmit.edu.au/online/blackboard/chapter/05/documents/contribute/chapter/03/documents/LinkedList.java).

public void addInOrder(int data)  
{

**ListNode current = head;  
 ListNode previous = null;  
 ListNode newNode = new ListNode(data);**

ListNode references are defined to store the new node, and the current and previous position for traversal

**while ( (current != null) &&  
 (current.number < data) )  
 {  
 previous = current;  
 current = current.next;  
 }**

The while loop will iterate through the list until the correct insertion point is found

**if (current == head)  
 {  
 newNode.next = head;  
 head = newNode;  
 }**

If the insertion point is the head, the data must be added with respect to the head reference

**else if (current == null)  
 {  
 previous.next = newNode;  
 }**

If it is to insert at the end, append the new node to the end of the list

**else  
 {  
 previous.next = newNode;  
 newNode.next = current;  
 }**  
}

If adding into the middle of the list, it must add the data with respect to the two nodes between the insertion point

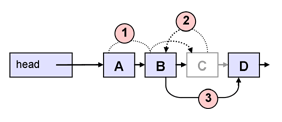
## Deleting From a List

Searching and deletion operations are related by the fact that in order to delete an item from the list, it must first be found. Deletion of nodes relies on manipulating the links between data items so one is 'left out' of the chain - so it is then deleted by the garbage collector.

Here, we are taking advantage of the garbage collector's behaviour of destroying objects that don't have any references to them. (In languages which don't have an automatic garbage collector, such as C++, extra steps need to be taken to "free up" the memory before removing the object.)

The extent to which the destruction occurs is kept in check by ensuring that only one data node ends up without a reference.

Deletion operations are similar for items that are at the middle or end of a linked list; however deleting the first item in the list needs to be handled slightly differently, as explained below.



1. Search the data item to delete.
2. If found, go to the node immediately preceding the target. This cannot be done by simply following links; the search will have to hold the 'previous' node, as well as the 'currently testing' node during the search operation.
3. Set the previous node's next reference to point to the same node as the target node's next reference. The target node is then no longer referred to by any variables, and is deleted by the garbage collector

In the case where the item to delete is the first item in the list structure, there is no 'previous' node to operate with. In this case, the header reference takes the place of the previous node.

### Source Code

This method accepts an integer to be removed, and will traverse the list and delete every occurrence of that number.

This code fragment is part of [LinkedList.java](http://www.cs.rmit.edu.au/online/blackboard/chapter/05/documents/contribute/chapter/03/documents/LinkedList.java).

public void delete(int target)  
{

**ListNode current = head;  
 ListNode previous = null;**

make variables to store where we are in the list, and the node before

**while (current != null)  
 {**

search through the list until the end is reached

**if (current.number == target)  
 {**

if the target has been found at the current position...

**if (previous == null)  
 head = current.next;**

if the previous reference is null, we're at the start of the list; get the header reference to jump past the target

**else  
 previous.next = current.next;**  
 **}**

otherwise, get the previous node to jump past the target

**previous = current;  
 current = current.next;**  
 }  
}

move along to the next node

7.> how to remove element from linked list.

To remove an item in the middle of the linked list, set the previous item's "link" pointer to the "link" pointer of the object you want to remove. For instance, you could add something like this to your LinkedList class:

public void removeNode(Node previousNode, Node nodeToRemove) {

if (previousNode != null) {

previousNode.setLink(nodeToRemove.getLink());

}

}

To think about this better, draw a picture.

N1 -> N2 -> N3 -> N4

N1's "link" is N2, etc. If you want to remove N2, just set N1's "link" to N3.

N1 -> N3 -> N4

To remove an item in the middle of the linked list, set the previous item's "link" pointer to the "link" pointer of the object you want to remove. For instance, you could add something like this to your LinkedList class:

public void removeNode(Node previousNode, Node nodeToRemove) {

if (previousNode != null) {

previousNode.setLink(nodeToRemove.getLink());

}

}

To think about this better, draw a picture.

N1 -> N2 -> N3 -> N4

N1's "link" is N2, etc. If you want to remove N2, just set N1's "link" to N3.

N1 -> N3 -> N4

8.> how to add element in last position in linked list.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Description:** | | Below example shows how to add element at last position in LinkedList. LinkedList provides few methods to add element at last position, those methods are:  **addLast():**Appends the specified element to the end of this list. **offerLast():**Inserts the specified element at the end of this list. **offer():**Adds the specified element as the tail (last element) of this list. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-collections-and-util/linkedlist/add-last/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package com.javacoffee.linkedlist;    import java.util.LinkedList;    public class MyAddLast {        public static void main(String a[]){            LinkedList<String> arrl = new LinkedList<String>();          arrl.add("First");          arrl.add("Second");          arrl.add("Third");          arrl.add("Random");          System.out.println(arrl);          System.out.println("Adding element at last position...");          arrl.addLast("I am last");          System.out.println(arrl);          System.out.println("Adding element at last position...");          arrl.offerLast("I am last - 1");          System.out.println(arrl);          System.out.println("Adding element at last position...");          arrl.offer("I am last - 2");          System.out.println(arrl);      }  } | | |
| |  | | --- | | **Output:** | | [First, Second, Third, Random]  Adding element at last position...  [First, Second, Third, Random, I am last]  Adding element at last position...  [First, Second, Third, Random, I am last, I am last - 1]  Adding element at last position...  [First, Second, Third, Random, I am last, I am last - 1, I am last - 2] | |

9.> how to add element in first position in linked list.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Description:** | | Below example shows how to add element at first position in LinkedList. LinkedList provides few methods to add element at first position, those methods are:  **addFirst():**Inserts the specified element at the beginning of this list. **offerFirst():**Inserts the specified element at the front of this list. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-collections-and-util/linkedlist/add-first/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package com.javacoffee.linkedlist;    import java.util.LinkedList;    public class MyAddFirst {        public static void main(String a[]){            LinkedList<String> arrl = new LinkedList<String>();          arrl.add("First");          arrl.add("Second");          arrl.add("Third");          arrl.add("Random");          System.out.println(arrl);          System.out.println("Adding element at first position...");          arrl.addFirst("I am first");          System.out.println(arrl);          System.out.println("Adding element at first position...");          arrl.offerFirst("I am first - 2");          System.out.println(arrl);      }  } | | |
| |  | | --- | | **Output:** | | [First, Second, Third, Random]  Adding element at first position...  [I am first, First, Second, Third, Random]  Adding element at first position...  [I am first - 2, I am first, First, Second, Third, Random] | |

how to remove duplicate element in arraylist in java.

ArrayList is the most popular implementation of 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 e.g. 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. There are multiple ways to do this, you can follow the approach we used for [removing duplicates from array in Java](http://java67.blogspot.sg/2012/12/how-to-remove-element-from-array-in-java-example.html), where we loop through array and inserting each element in a Set, which ensures that we discard duplicate because Set doesn't allow them to insert, or you can also use remove method of ArrayList to get rid of them, once you found that those are duplicates.  
  
Btw, 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](http://java67.blogspot.com/2014/01/when-to-use-linkedhashset-vs-treeset-vs-hashset-java.html" \t "_blank) because [HashSet](http://java67.blogspot.com/2015/09/how-to-sort-hashset-in-java-example.html" \t "_blank) doesn't provide any ordering guarantee.  
  
If you are using deleting duplicates while iterating, make sure you use [Iterator's remove() method](http://javarevisited.blogspot.sg/2014/01/ow-to-remove-objects-from-collection-arraylist-java-iterator-traversing.html) and not the ArrayList one to avoid ConcurrentModificationException.  In this tutorial we will see this approach to remove duplicates.

## Java Program to removed 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](http://javarevisited.blogspot.sg/2012/01/convert-arraylist-to-set-java-example.html" \t "_blank). 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.
* 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;

/\*\*

\* Java Program to remove repeated elements from ArrayList in Java.

\*

\* @author WINDOWS 8

\*/

public class ArrayListDuplicateDemo{

public static void main(String args[]){

// creating ArrayList with duplicate elements

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);

// let's print arraylist with duplicate

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

// Now let's remove duplicate element without affecting order

// LinkedHashSet will guaranteed the order and since it's set

// it will not allow us to insert duplicates.

// repeated elements will automatically filtered.

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

// now let's clear the ArrayList so that we can copy all elements from LinkedHashSet

primes.clear();

// copying elements but without any duplicates

primes.addAll(primesWithoutDuplicates);

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

}

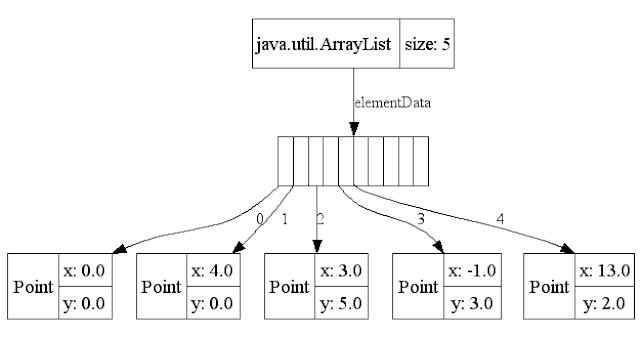
}

Output

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

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

In this example, you can see we have created an ArrayList and added numbers into it, all prime numbers. We added '7' twice, so that it become duplicate. Now we print the ArrayList and you can see that it contains number 7 twice.

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fjava-fundamentals-collections)

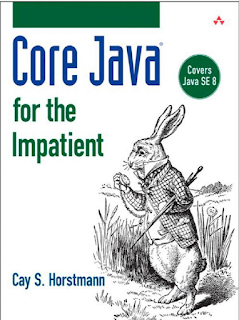
After that we created a LinkedHashSet from our ArrayList, clear our original ArrayList and then added all elements from set to the list. This time we should not have any duplicates because Set doesn't allow them and they should have filtered when elements copied from ArrayList to HashSet by Java. This is proved by printing the ArrayList again, now it doesn't contain 7 twice, but only once.  
  
That's all about **how to remove duplicates from ArrayList in Java**. Though there are multiple ways to do this, I think using LinkedHashSet is the simplest one because its simple and also preserve the order of elements.

4.> how to delete all element in array list.

There are two ways to remove all elements of an ArrayList in Java, either by using clear() or  by using the removeAll() method. Both methods are defined in the java.util.List and java.util.Collection interface, hence they are available not just to ArrayList but also to [Vector](http://www.java67.com/2012/09/arraylist-vs-vector-in-java-interview.html) or [LinkedList](http://www.java67.com/2012/12/difference-between-arraylist-vs-LinkedList-java.html" \t "_blank) etc. Both elements removes all objects from ArrayList but there is a subtle difference in how they do. The clear() method is straightforward, it [traverse through the ArrayList](http://www.java67.com/2012/08/how-to-traverse-iterate-or-loop-ArrayList-in-java-example-tutorial.html) and sets all indices to null, which means the ArrayList becomes **empty** and all elements become eligible to Garbage collection, provided there is no more references to them. The time taken by clear() method is in O(n), which means the bigger the arraylist the longer it will take to empty it.  
  
On the other hand removeAll(Collection c) accepts a Collection and then iterate over List. At each iteration it checks if current element in the ArrayList is present in the Collection c using contains() method, which takes its O(n) time to confirm because it also uses [Iterator](http://www.java67.com/2013/02/java-iterator-example.html) instead of random access.  
  
So overall time taken to remove all elements using removeAll() method is in order of O(n^2) which means time will increase in quadratic of number of elements in [array](http://www.java67.com/2012/12/difference-between-array-vs-arraylist-java.html). This difference is not much if your ArrayList is small and just contains 10 to 100 elements but for a big ArrayList e.g. of 1 million objects, this time could be significant.   
  
This is also one of the frequently asked [ArrayList question](http://java67.blogspot.com/2015/06/20-java-arraylist-interview-questions.html" \t "_blank) from Java Interviews, so knowing the key difference will help you there as well.  So choice is yours, I suggest to use clear() if you want to remove all elements from the ArrayList and use removeAll() if you want to remove selected elements given to you in a Collection. Let's see the example of both of them in Java.

## How to empty an ArrayList in Java

Here is a complete Java program to remove all elements and make an ArrayList empty in Java. This program demonstrate how you can remove all elements from a given ArrayList by using both clear() and removeAll() method. If you want to remove just single element then you can use the remove() method as discussed [here](http://www.java67.com/2014/03/2-ways-to-remove-elementsobjects-from-ArrayList-java.html).  
  
The program prints all objects of ArrayList before and after calling the clear() and removeAll() method to show that method is actually working and the ArrayList is empty afterwards.  
  
You can reuse the ArrayList by clearing it but make sure you don't do that in multi-threading environment e.g. one thread is calling the clear() method while other thread is calling the add() method to insert elements.  The [ArrayList class is not thread-safe](http://javarevisited.blogspot.com/2016/01/9-difference-between-array-vs-arraylist-in-java.html" \t "_blank) and sharing the same ArrayList between multiple thread will crate thread-safety related problem and erroneous result.   
  
See [Core Java for the Impatient](http://www.amazon.com/Core-Java-Impatient-Cay-Horstmann/dp/0321996321?tag=javamysqlanta-20) to learn more about problems of using ArrayList in muti-threading application.

[](http://www.amazon.com/Core-Java-Impatient-Cay-Horstmann/dp/0321996321?tag=javamysqlanta-20)

**Java Program to make an ArrayList empty**

**import** **java.util.ArrayList**;

/\*

\* Java Program to remove all elements of ArrayList.

\* This is also known as emptying an AraryList

\*/

**public** **class** **Main** {

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

System.out.println("Welcome to Java Program to empty an ArrayList");

ArrayList<String> listOfInsurance = **new** ArrayList<>();

listOfInsurance.add("Car Insurnace");

listOfInsurance.add("Health Insurnace");

listOfInsurance.add("Life Insurance");

listOfInsurance.add("Home Furniture Insurance");

listOfInsurance.add("Home loan Insurance");

System.out.println("ArrayList before emptying: ");

System.out.println(listOfInsurance);

// Emptying an ArrayList in Java

listOfInsurance.clear();

System.out.println("ArrayList after emptying: ");

System.out.println(listOfInsurance);

ArrayList<String> listOfLoans = **new** ArrayList<>();

listOfLoans.add("Car loan");

listOfLoans.add("Persona loan");

listOfLoans.add("Balance transfer");

listOfLoans.add("Home loan");

System.out.println("ArrayList before removing all elements: ");

System.out.println(listOfLoans);

// Emptying an ArrayList in Java

listOfLoans.removeAll(listOfLoans);

System.out.println("ArrayList after removing all elements: ");

System.out.println(listOfLoans);

}

}

Output

Welcome to Java Program to empty an ArrayList

ArrayList before *emptying:*

[Car Insurnace, Health Insurnace, Life Insurance, Home Furniture Insurance, Home loan Insurance]

ArrayList after *emptying:*

[]

ArrayList before removing all *elements:*

[Car loan, Persona loan, Balance transfer, Home loan]

ArrayList after removing all *elements:*

[]

You can see that both the ArrayLists are empty after calling the clear() and removeAll() methods. So its working!!  
  
That's all about how to remove all elements from an ArrayList in Java. As I said, clear() takes less time than removeAll() to remove all objects, hence you should always use clear() to *make an ArrayList empty*. But, if you are not removing all elements and list of elements to be removed are provided to you in a Collection or List then use the removeAll() method.

5.> how to sort arraylist using comparator.

* Create a new [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html" \t "_blank).
* Populate the arrayList with elements, using add(E e) API method of ArrayList.
* Invoke reverseOrder() API method of [Collections](http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html) to get a [Comparator](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html) that imposes the reverse of the natural ordering on the list’s elements.
* Invoke sort(List list, Comparator c) API method of Collections in order to sort the arrayList elements using the comparator. The arrayList’s elements will be sorted according to the comparator.

We can get the arrayList’s elements before and after sorting to check how they are sorted. Before sorting the arrayList elements are sorted by insertion order, and after sorting the elements are in reverse than the natural ordering.

Let’s take a look at the code snippet that follows:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package com.javacoffee.snippets.core;    import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;    public class SortArrayListComparator {      public static void main(String[] args) {        // Create an ArrayList and populate it with elements      ArrayList arrayList = new ArrayList();      arrayList.add("element\_1");      arrayList.add("element\_3");      arrayList.add("element\_5");      arrayList.add("element\_2");      arrayList.add("element\_4");        // ArrayList implementation maintains the insertion order for its elements      System.out.println("Elements in ArrayList prior sorting :");      for(int i=0; i < arrayList.size(); i++)    System.out.println(arrayList.get(i));        /\*    To get a comparator that imposes reverse order on a Collection's elements    we can use static Comparator reverseOrder() operation of Collections class      \*/      Comparator comparator = Collections.reverseOrder();        // Collection.sort(List list, Comparator c) static operation sorts ArrayList elements using a Comparator      Collections.sort(arrayList,comparator);        System.out.println("Elements in ArrayList after sorting :");      for(int i=0; i < arrayList.size(); i++)    System.out.println(arrayList.get(i));      }  } |

**Output:**

Elements in ArrayList prior sorting :

element\_1

element\_3

element\_5

element\_2

element\_4

Elements in ArrayList after sorting :

element\_5

element\_4

element\_3

element\_2

element\_1

6.> how to convert array to arrylist and arraylist to array in java.

**convert ArrayList to array** using toArray() method with example. toArray() method returns an array containing all of the elements in the list in proper sequence (from first to last element).

## 1. ArrayList toArray() syntax

toArray() is [overloaded](https://howtodoinjava.com/oops/what-is-polymorphism-in-java/) method and comes in two forms:

|  |
| --- |
| toArray() method syntax |
| public Object[] toArray();  public <T> T[] toArray(T[] a); |

1. The **first method** does not accept any argument and returns the array of object type. We must iterate the objects array to find the desired element and typecast to desired class type.
2. In **second method**, the runtime type of the returned array is that of the specified array. If the list fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list.

After filling all array elements, it there is more space left in array then 'null' is populated in all those spare positions.

## 2. ArrayList toArray() example to convert ArrayList to Array

#### 2.1. ArrayList toArray() – convert to object array

Java program to convert an arraylist to object array and iterate through array content.

|  |
| --- |
| contains() example |
| import java.util.ArrayList;  import java.util.Arrays;    public class ArrayListExample  {      public static void main(String[] args)      {          ArrayList<String> list = new ArrayList<>(2);            list.add("A");          list.add("B");          list.add("C");          list.add("D");            //Convert to object array          Object[] array = list.toArray();            System.out.println( Arrays.toString(array) );            //Iterate and convert to desired type          for(Object o : array) {              String s = (String) o;                System.out.println(s);          }      }  } |

Program output.

|  |
| --- |
| Console |
| [A, B, C, D]    A  B  C  D |

#### 2.2. ArrayList toArray(T[] a) – convert to string array

Java program to convert an arraylist to string array.

|  |
| --- |
| a) example"]  import java.util.ArrayList;  import java.util.Arrays;    public class ArrayListExample  {      public static void main(String[] args)      {          ArrayList<String> list = new ArrayList<>(2);            list.add("A");          list.add("B");          list.add("C");          list.add("D");            //Convert to string array          String[] array = list.toArray(new String[list.size()]);            System.out.println(Arrays.toString(array));      }  } |

Program output.

|  |
| --- |
| Console |
| [A, B, C, D] |

7.> write a java program to sort an arraylist.

To sort the contents of an ArrayList in descending order

1. Create an ArrayList.
2. Sort the contents of the ArrayList using the **sort()** method of the Collections class.
3. Then, reverse array list using the **reverse()** method of the Collections class.

**Example:**

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListSample {

   public static void main(String[] args) {

      ArrayList<String> list = new ArrayList<String>();

      list.add("JavaFx");

      list.add("Java");

      list.add("WebGL");

      list.add("OpenCV");

      Collections.sort(list);

      System.out.println(list);

      Collections.reverse(list);

      System.out.println(list);

   }

}

**Output:**

[Java, JavaFx, OpenCV, WebGL]

[WebGL, OpenCV, JavaFx, Java]