Difference between ArrayList<E> and LinkedList<E>

Previously we [differentiated in ArrayList<E> and array](http://data-structure-learning.blogspot.com/2015/05/difference-between-arraylist-and-arrays.html). We had also discussed ways to [traverse over ArrayList<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-6iterating-over.html) and [HashMap<K, V>](http://data-structure-learning.blogspot.com/2015/05/different-ways-to-traverse-map.html).

This is a basic interview question. We are expected to compare 2 different implementations of List<E> interface of Java Collections Framework.

1. First let us understand what are differences in terms of **implementation?** 
   1. ArrayList<E> as we know **is backed by an array** with some initial capacity. If that array is filled then contents are moved to new array who’s size is roughly 1.5 times of previous array.
   2. LinkedList<E> is an implementation of [Linked List data structure](http://data-structure-learning.blogspot.com/2015/05/let-us-start-understanding-linked-list.html). private static class Node defines the structure of node using the constructor taking 3 parameters as previous pointer, element and next pointer.
2. Now comes **performance**. The performance of both of the class depends on the type of operations that we perform.
   1. **add(E e)** 
      1. ArrayList<E> adds element at the end of the list. This can be done in O(1) time. But let say that size of backing array is to be incremented then old elements is to be copied to new array and then new element is appended. This takes O(n) time.
      2. In LinkedList<E> this operation takes O(1) time. The reason is LinkedList<E> maintains the tail pointer and hence element is just appended by setting the links.
   2. **remove(int index) and remove(Object o)**
      1. ArrayList<E> removes element once it is able to find it. If element is found then all other elements in backing array are shifted one place. This requires O(n) time.
      2. LinkedList<E> has several implementations of remove such as
         * 1. removeFirst() – removes first element. O(1) time.
           2. removeLast()– removes last element. O(1) time.
           3. remove(int index) – removes element at index. O(n) time.
           4. remove(Object o) – removes element by searching it. O(n) time.
           5. remove() – removes first element. O(1) time. This method calls removeFirst.
           6. removeFirstOccurrence(Object o) – removes first occurrence of Object.
           7. removeLastOccurrence(Object o) – removes last occurrence of Object.
   3. **get(int index)**
      1. ArrayList<E> gets the element at index in O(1).
      2. LinkedList has 3 different method that returns the element.
         * 1. get(int index) –returns the element at index O(n) time because link list does not provide the indexed retrieval.
           2. getFirst() – returns the first element from the list.
           3. getLast() – returns the last element from the list.
   4. **Iteration**
      1. ArrayList<E> provides 2 iterators. [Iterator<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-8-iterator.html) and [ListIterator<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-9-listiterator.html). Study about their [differences here](http://data-structure-learning.blogspot.com/2015/05/5-difference-between-iterator-and.html). Traverse or [iterate ArrayList in 6 different ways](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-6iterating-over.html).
      2. LinkedList<E> provides 3 iterators. Iterator<E>, ListIterator<E> and decendingIterator. decendingIterator means it has 3 method of Iterator that works in decening manner i.e. from back to front.
   5. **Capacity**
      1. ArrayList<E> has initial capacity of 10 elements.
      2. LinkedList<E> does not define any capacity. It just creates new nodes and appends them.
   6. **Memory Overhead**
      1. In ArrayList<E> backing array is used to store element at index so memory overhead is 0(zero).
      2. In LinkedList<E> previous and next links are to be maintained which can be troublesome. So it has memory head.

We will also see Similarities between ArrayList<E> and LinkedList<E> and when to use ArrayList<E> or LinkedList<E>