**Java Data structures**

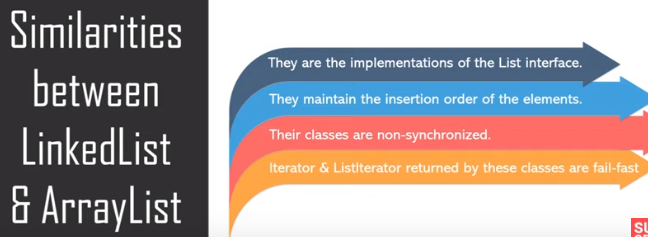
***Iterators***

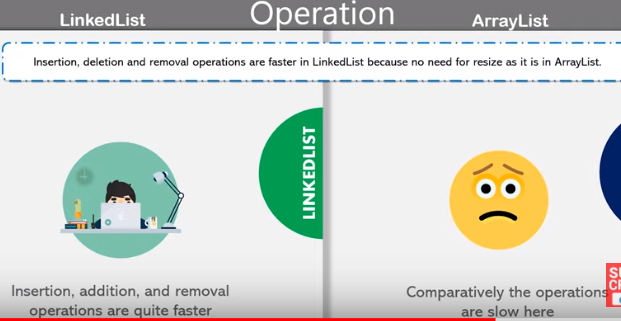
Iterator and Iterable are two interfaces.

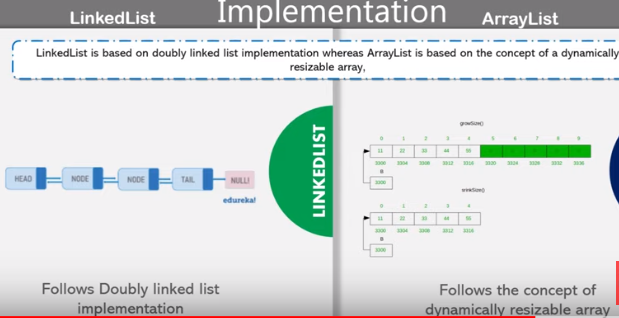
Iterator has hasNext() and next() method and Iterable provides iterator() and next() method.

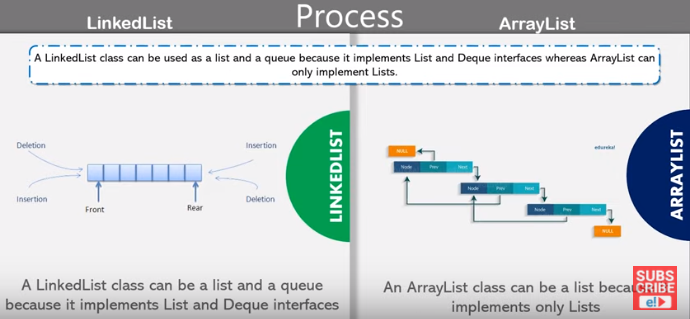


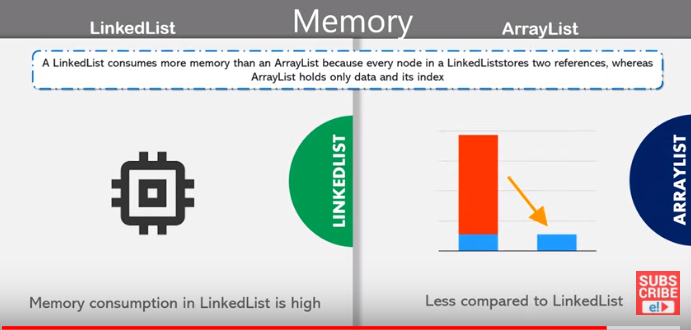
**LinkedList and Arraylist**











|  |  |  |
| --- | --- | --- |
|  | LinkedList | ArrayList |
| **P**rocess | Insert, remove, add are faster since the new node is linked between former and later with the address | Get element is faster because the element is called directly by its index |
| **I**mplementation | Implements List and Deque interfaces | Implements List interface |
| **M**emory | High. Because nodes store value and address of the element | Low. Because it stores only Index and Value of the element. That is why search is also easy |
| **P**erformance | Doubly Linked List | Dynamic Array |

**Note**: in ArrayList and LinkedList, .remove(a) where “a” is Integer removes element at position “a” in the List. Hence to remove an element from list of integers, we use .removeAll(Arrays.asList(a)) or .remove(Integer.valueOf(a))

**Note**: Both are not synchronized (i.e. not thread safe. Two threads can call them and make changes without one waiting for other)

**Vector**

Vectors are similar to ArrayList. The difference is that they instantiate with default size of 10 and if we insert the 11th element into them, then their size automatically becomes 20. Similary, it becomes 30 when we insert 21st element.

Vectors are similar to arrayList as they implement List interface and they also work like Dynamic Array.

Vector is available since JDK 1.0 and collections came up in JDK 1.2. Since Vector is more memory consuming for its size, ArrayList is preferable.

**Stack**

Stacks extend Vector Class. They too are available since JDK 1.0. They follow the approach of Last in First Out (LIFO). Stack behaves exactly like Vector and ArrayList, but it has two special methods:-

*1. Pop()*

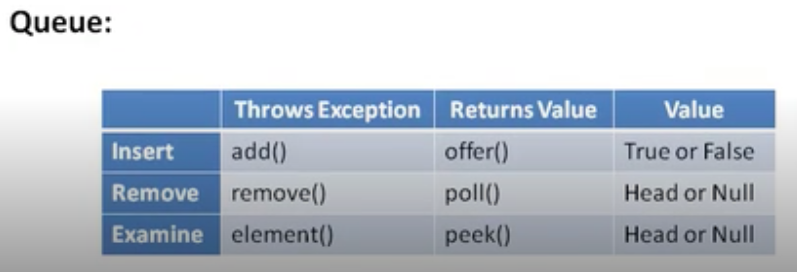
This method is used to remove the lastly added value in the Stack and return the same.

*2.Push(a)*

This method inserts “a” as Latest (uppermost) value in the stack.

**Queue**

Unlike Stack, Queue follows First in First out (or Last in Last Out) approach. Like Stack, it also provides special methods:



**Classes implementing Queue Interface**

[AbstractQueue](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractQueue.html), [ArrayBlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html), [ArrayDeque](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html), [ConcurrentLinkedDeque](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedDeque.html), [ConcurrentLinkedQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html), [DelayQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/DelayQueue.html), [LinkedBlockingDeque](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingDeque.html), [LinkedBlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingQueue.html), [LinkedList](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html), [LinkedTransferQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedTransferQueue.html), [PriorityBlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/PriorityBlockingQueue.html), [PriorityQueue](https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html), [SynchronousQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/SynchronousQueue.html)

**NOTE:**

ArrayBlockingQueue is a class implementing Queue interface where the queue capacity can be fixed while initializing.

**Priority Queue**

Priority queue provides elements by priority. By default, it provides(.poll() or .peek()) string queue in alphabetical order and numbers in increasing order. We can provide this priority definition by passing a comparator object.

<https://www.youtube.com/watch?v=FdObb76AmzM>

<https://www.youtube.com/watch?v=OxhYCLWMdHs>

