**Table 1. Comparison of Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
|  | List | Queue | Deque |
| Main Applications | Insert, get, and remove elements by INDEX.  Search elements. | Holding elements prior to processing. | Subinterface of Queue. Insert and remove elements at both ends (double ended queue). |
| Mostly Used Implementing Classes | ArrayList, LinkedList, Stack, Vector | ArrayDeque, LinkedList, PriorityQueue | ArrayDeque, LinkedList |
| Mostly Used Methods | add((e), add to end  add(index, e)  get(index)  listIterator()  remove(int index)  remove(Object o)  set(int index, E e)  size()  contains(e) | add(e) , offer(e)  remove(), poll()  element(), peek() | add(e) == addLast(e), addFirst(e)  element()== getFirst(), getLast()  peek() == peekFirst(), peekLast()  poll() == pollFirst(), pollLast()  remove() == removeFirst(), removeLast()  push(e) ~ addFirst(e)  pop() ~ removeFirst()  size()  contains(Object o)  remove(Object o) |
| Note | (1) no remove(); no addFirst(e), addLast(e), removeLast(), removeFirst()  (2) If elements are int, remove(element\_1) does not remove the element\_1, but removes the element at index = element\_1  (3) List<Integer> a = Arrays.asList(1, 2, 3); a would have a fixed size, a.add(4) would throw an exception.  ArrayList<Integer> b = new ArrayList<>(Arrays.asList(1,2,3)); would get an arraylist | (1) Avoid using null as elements.  For empty queue, remove()/element() throws Exception, poll()/peek() returns null.  (2) no index access. No get(index)  (3) no size | (1) Deque can be used as FIFO(add, poll) or LIFO (push, pop).  ArrayDeque should be used in preference to Stack.  (2) no index access  (3) This interface allows null element, but not encouraged. |

Table 2. Comparison of Classes

|  |  |  |  |
| --- | --- | --- | --- |
|  | ArrayList | LinkedList | ArrayDeque |
| Whether can be constructed with a specific size | Yes.  List<Integer> = new ArrayList<>(5) | No | Yes |
| Cache Locality | Good | Bad | Good |
|  | add(e), get(index) , set(index, e) runs in O(1);  remove(index), add(index, e) runs O(n), (? Javadoc “All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the LinkedList implementation.” But remove(index) add(index, e) in ArrayList need shifting the following elements, extra copy than LinkedList.)  Iterating an ArrayList is faster than LinkedList. | add(e) O(1)  get(index), remove(index), add(index, e), O(n) |  |
|  | Iterator.remove(), ListIterator.add(e)  O(n) | Iterator.remove(), ListIterator.add(e)  O(1) |  |

|  |  |  |  |
| --- | --- | --- | --- |
| When to use | (1) Need to access element with index.  (2) does not need to remove or add element through iterator | Remove or add element through iterator.  removeFirst() and addFirst(e) is O(1) | Only need to access the head or the tail.  Faster than Stack when used as a stack, and faster than LinkedList when used as a Queue. |

It's worth noting that **Vector** also implements the List interface and is almost identical toArrayList. The difference is that Vector is synchronized, so it is thread-safe. Because of this, it is also slightly slower than ArrayList.