**Array vs ArrayList vs LinkedList vs Vector with very good overview and examples.**

An **ArrayList** is ordered collection (also known as a sequence). It means that the user of controls where each element is inserted.

An **ArrayList** is better than **Array** to use when you have no knowledge in advance about elements number. **ArrayList** are slower than **arrays**. So, if you need efficiency try to use **Arrays** if possible.

Short Overview on main features of **ArrayList**:

- - The add operation runs **O(n)** time.
- - The **isEmpty()**, **size()**, **iterator()**, **set()**, **get()** and **listIterator()** operations require the same amount of time, independently of element you access.
- - Only Objects can be added to an **ArrayList**
- - Implements all methods from the **List** interface
- - Permits null elements

An **ArrayList** could be a good choice if you need very **flexible** **Array** type of collection with limited functionality.

One small remark for those who needs faster **ArrayLists**. **ArrayLists** are internally implemented as an **Array**. So when run an "add" command a new **Array** will be created with **n+1** dimension. All "older" elements will be copied to first **n** elements and last **n+1** one will filled with the value which you provide with add() method.

If you have higher requirements on number of accessible methods for your **Array** like collection instead of **ArrayList** you have better choice - **LinkedList**.

**LinkedList** is much more **flexible** and lets you insert, add and remove elements from both sides of your collection - it can be used as queue and even double-ended queue!

Internally a **LinkedList** does not use **arrays**, it is much more modern! **LinkedList** is a sequence of nodes, which are double linked. Each node contains header, where actually objects are stored, and two links or pointers to next or previous node. A **LinkedList** looks like a chain, consisting of people who hold each other's hand. You can insert people or node into that chain or remove. Linked lists permit node insert/remove operation at any point in the list in constant time.

**LinkedList** is actually luxury extension of an **ArrayList**: it gives you many methods which you usually implement yourself around of different type of collections.

For example if you need to add new element to the end of an **ArrayList** you have to look at the size of the collection and then add that new element at **n+1** place. In **LinkedList** you do it directly by **addLast(E e)** method.

Another example. Depending on your expectations you can chose either **getFirst**/**getLast** or **peekFirst**/**peekLast** methods. Last "peek" methods are completelly new sort of methods and introduced in Java 1.6.

They slightly differ from **set**/**get** methods - they do not throw any kind of exceptions if this list is empty, return just null.

New "poll" methods - **pollFirst** and **pollLast** combine two methods: **get** and **remove** an element from your collection. For example **pollFirst**() method gets and removes the first element from this list. This method does throw any kind of exception like **IndexOutOfBoundsException** in case of **ArrayList**, instead it just returns null (if this list is empty).

Because the **LinkedList** implements queue interface you can pop and push new elements from/into your collection.

If you created capacity restricted collection you can "examine" the result of an operation by using **offerFirst**/**offerLast** methods. They are also new (since Java 1.6) and return boolean result on the operations instead of throwing **IllegalStateException** as **addFirst**/**addLast** methods do. In case if possible to insert an element at the beginning/end of collection you get "true" result.

From http://www.javafaq.nu/java-article1111.html