**List, ArrayList, LinkedList**

**Применение List или ArrayList.**

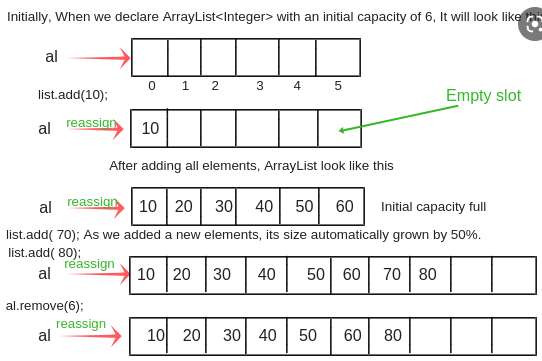
(1) **List**<?> myList = new ArrayList<?>();

(2) **ArrayList**<?> myList = new ArrayList<?>();

Почти всегда первый предпочтительнее второго. Первое имеет то преимущество, что реализация **List** может измениться (например, на **[LinkedList](http://java.sun.com/javase/7/docs/api/java/util/LinkedList.html)**), не затрагивая остальную часть код. Это будет трудной задачей для **ArrayList** не только потому, что вам нужно будет изменить **ArrayList** на **LinkedList** всюду, но также и потому, что вы могли использовать специальные методы **ArrayList**.

**Отличия и применение ArrayList и LinkedList.**

**ArrayList** - это список на основе массива.



\* - не совсем верно нарисовано, т.к. изначально элементов в массиве не 6 а 10.

**LinkedList** - связанный список на основе элементов и связи между ними. В качестве LinkedList лучше всего подходит представление вагонов поезда сцепленных последовательно.

**ArrayList** следует использовать, когда в приоритете доступ по индексу, так как эти операции выполняются за константное время. Добавление в конец списка в среднем тоже выполняется за константное время. Кроме того в ArrayList нет дополнительных расходов на хранение связки между элементами. Минусы в скорости вставки/удаления элементов находящихся не в конце списка, так как при этой операции все элементы правее добавляемого/удаляемого сдвигаются.

**LinkedList** удобен когда важнее быстродействие операций вставки/удаления, которые в LinkedList выполняются за константное время. Операции доступа по индексу производятся перебором с начала или конца (смотря что ближе) до нужного элемента. Дополнительные затраты на хранение связки между элементами.

ArrayList более предсказуем для процессора, с точки зрения расположения данных. Это массив, а там элементы расположены последовательно, занимая непрырывную область памяти. Это хорошо, так как позволяет подгружать данные в кэши процессора без cache miss'ов. Процессор не простаивает, ожидая данные из оперативной памяти. С LinkedListтакого нет, т.к. элементы располагаются в разных участках памяти, и предугадать расположение следующего элемента процессору не под силам.

Одним словом - если часто вставляете/удаляете - выбирайте в пользу **LinkedList**, в противном случае **ArrayList**