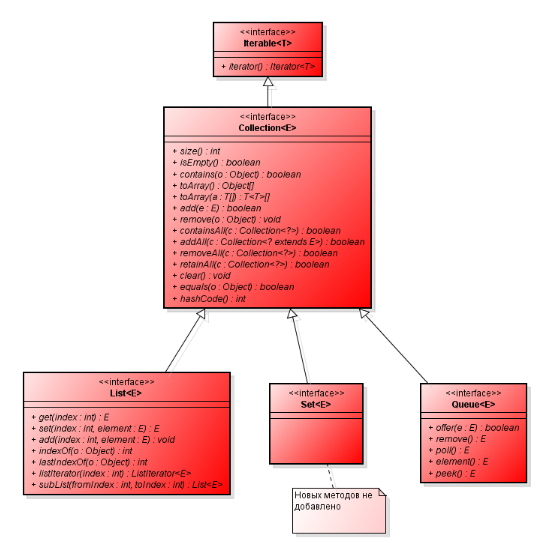
**Иерархия всех коллекций**



**Collection Framework –** это структура данных, которая является частью JDK. На вершине иерархии располагаются основные интерфейсы “Collection” и “Map”. Эти интерфейсы разделяют collection на 2 группы по типу хранения данных: **простые последовательные наборы** **элементов** и **наборы пар “ключ-значение”**.

В версии Java 1.5 – доработаны интерфейсы Collection и Map с приходом дженериков.

В версии Java 8 – добавлены методы для работы с лямбда выражениями.

**Отличие Collection от Map**

Все реализации от **Collection** и находящиеся в них элементы – это совокупность элементов.

Всё что находится в реализациях **Map** – совокупность элементов ключ-значение.

**Что быстрее работает: ArrayList или LinkedList и почему ?**

Преимущества ArrayList: в возможности доступа к произвольному элементу по индексу за постоянное время (так как это массив), минимум накладных расходов при хранении такого списка, вставка в конец списка в среднем производится так же за постоянное время.

В итоге получаем, что при добавлении элемента при необходимости расширения массива, время добавления будет значительно больше, нежели при записи элемента в готовую пустую ячейку. Тем не менее, в среднем время вставки элемента в конец списка является постоянным. Удаление последнего элемента происходит за константное время. Недостатки ArrayList проявляются при вставке/удалении элемента в середине списка — это взывает перезапись всех элементов размещенных «правее» в списке на одну позицию влево, кроме того, при удалении элементов размер массива не уменьшается, до явного вызова метода trimToSize().

LinkedList наоборот, за постоянное время может выполнять вставку/удаление элементов в списке (именно вставку и удаление, поиск позиции вставки и удаления сюда не входит).

Доступ к произвольному элементу осуществляется за линейное время (но доступ к первому и последнему элементу списка всегда осуществляется за константное время — ссылки постоянно хранятся на первый и последний, так что добавление элемента в конец списка вовсе не значит, что придется перебирать весь список в поисках последнего элемента.

**Сложность добавление в начало, середину и конец у ArrayList и LinkedList**

ArrayList – добавление в начало(когда массив полон) вызовет перезапись элементов с добавлением нового, в новый массив. При добавлении в середину то же самое – но метод arraycopy() сработает два раза – до вставляемого элемента скопирует + новый элемент + всё что после нового элемента. В конец вставка похожа с вставкой вперёд ( при полном массиве) и если есть место – произойдёт простое добавление.

LinkedList – при добавлении в начало или конец нам поможет метод addLast() and addFirst() , добавление произойдёт гновенно, за O(1). При вставке в середину – единственное что затармажиает работу это найти то самое место с которого производить действия, если нашли то в нём (узле) поменять,переопределить ссылки на следующий и текущий элемент.

**8. Что такое Collections и его основные методы**

Colections содержит много статических методов для манипуляции с коллекциями.

Методы которые начинаются на checked+название интерфейса – проверяют переданный интерфейс соответствует указанному типу.

Метод addAll добавляет в указанную параметризир коллекцию объекты.

Copy() копирует из одной коллекции в другую.

Reverse() переворачивает список

Sort() сортирует стандартно или через компаратор

**9. Как удалить 10 элементов из середины ArrayList ?**

Есть метод removeAll() который может принимать обьект, в качестве обьекта передаём полученный по методу коллекцию, метод sublist(5,15). Удалим нужное кол-во элементов. Можно ещё извращаться и удалить через for, итератор.

**10. Что чаще используется: ArrayList или LinkedList и почему ?**

В 90% случае ArrayList будет быстрее и экономичнее LinkedList, так что обычно используют ArrayList, но тем не менее всегда есть 10% случаев для LinkedList. Если необходимо вставлять (или удалять) в середину коллекции много элементов, то лучше использовать LinkedList. Во всех остальных случаях – ArrayList.

**11. Что быстрее: вставить 100 элементов в середину огромного ArrayList или огромного LinkedList ? Почему ?**

Вставка в ArrayList черевата тем – что смешаться массив будет 100 раз, но можно обойти использовав addAll. У LinkedList только поменять 2 ссылки на элементы – очень быстрая вставка.

По хорошему я бы сказал, что сложность для каждого варианта одинакова:  
ArrayList: O(1) + O(N) - поиск нужного элемента O(1), тк мы быстро находим элемент + вставка O(N), тк надо сдвигать все остальные элементы  
LinkedList: O(N) + O(1) - поиск элемента O(N), тк надо перебирать все элементы + вставка O(1), тк нам достаточно только сменить ссылки на паре элементов для вставки  
  
Итого исходя из выше указанного можно сказать, что сложность одинакова.

А далее тк мы знаем, что в эррейлисте реально используется нативные метод для сдвига, а не поэлементный сдвиг, то на практике он работает на порядок быстрее. Как результат - лучше использовать ArrayList

PS: кроме того, если изучить сложность для вставки в начало и в конец, получается, что при вставке в начало списка достаточно большое преимущество у LinkedList (не тратится время на поиск), а при вставке в конец эти списки работают примерно одинаково

**Заметки:**

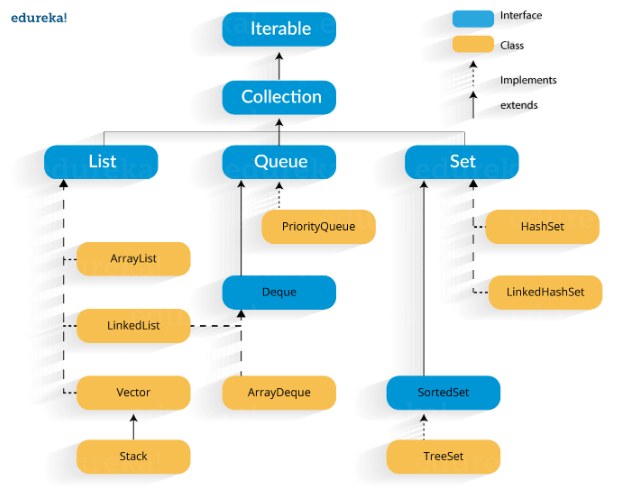
* В ArrayList можно вставлять даже null значения.
* По умолчанию размер ArrayList() = 10
* **ArrayList использовать:** при .get(), при .set(начало и середина), при .add(), и при remove(кроме начала списка).
* **LinkedList использовать**: добавление много данных в начало списка, удаление с начала(index = 0) списка, и .set в конце списка.

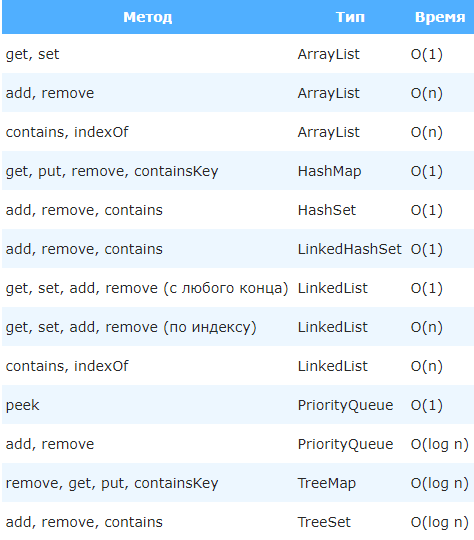
**Размер новой коллекции ArrayList()**

Когда места в коллекции не достаточно – затрачивается дополнительная память, на создание коллекции большего размера (старое size() \* 1,5) и копирование всех элементов.









**Big – O: ArrayList:**

add() за время – O(1)

add(index,element) – в среднем за O(n) время

get() – всегда за О(1)

remove() и contains() – работает за линейное время О(n)

**Big – O: LinkedList:**

add() поддерживает O (1) постоянную вставку в любой позиции

get() – за О(n)

remove() - за O(1)

contains() – работает за линейное время О(n)