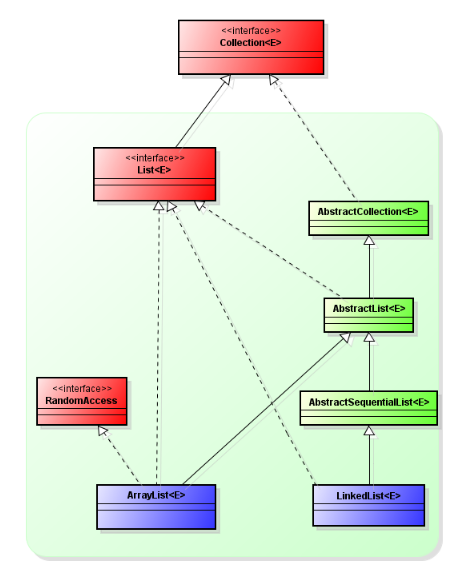
**LIST**

(СПИСОК)



Основные реализации интерфейса List – это классы **ArrayList** и **LinkedList**. Есть не столь популярные Vector, Stack.

List представляет собой упорядоченную коллекцию. Все реализации List могут содержать null элементы и дублекаты.

**ArrayList**

ЭТО УПОРЯДОЧЕННЫЙ СПИСОК, реализованный на основе массива (соответственно все плюсы и минусы массивов). Может хранить **повторяющиеся значения** (при поиске/удалении – действие происходит только с первым элементом). В параметры конструктора ArrayList() можно передать начальный размер (что бы при расширении он каждый раз не пересоздавался) – стандартно создаётся массив на 10 элементов. Согласно java конвенции – необходимо ссылать класс на тип интерфейса, который реализует класс (List <String> list = new ArrayList<>).

Существует 3 способа пройтись по всем элементам списка: через обычный for, через for-each, через лямбда-функции.

**ВСТАВКА (в конец):**

1) Проверка, если места достаточно – просто добавляет.

2) Проверка, если места недостаточно – создаётся новый массив: (oldSize \* 3)/2 + 1. При этом срабатывает метод arraycopy() и переносит все данные в новый массив, добавляя новый элемент. Ссылка на старый отвязывается и сборщик мусора уничтожает массив. Сравнение времени работы операции add ArrayList и LinkedList на 500К элементов:

Время выполнения java.util.ArrayList = 78ms  
Время выполнения java.util.LinkedList = 109ms

**ВСТАВКА (в середину, начало):**

1) Проверка, если места достаточно – сдвигает от нужного индекса все элементы вправо (вызывается метод arraycopy()), затем старое значение под выбранным индексом затирается новым вставляемым значением.

2) Проверка, если места недостаточно – создаётся новый массив, переносятся все значения (1 раз срабатывает arraycopy()), затем от нужного индекса переносятся все значения вправо (2 раз срабатывает метод arraycopy()), и новое значение попадает на своё место. Скорость значительно ниже из-за 2-го вызова метода arraycopy(). Сравнение времени работы операции add (0,значение) ArrayList и LinkedList на 500К элементах:

Время выполнения java.util.ArrayList = 108589ms  
Время выполнения java.util.LinkedList = 187ms

**УДАЛЕНИЕ:**

Удалять элементы можно двумя способами:

— по индексу remove(index);

— по значению remove(value);

При этом все элементы после удалённого – сдвигаются влево на 1 позицию, следовательно операция очень затратная. При этом размер массива не подрезается (нужно использовать trimToSize()).

**по индексу** – вычисляется кол-во элементов для копирования, после чего происходит копирование элементов на место удалённого по индексу, после этого последний элемент забывается(null).

**по значению** - в цикле просматриваются все элементы списка, до тех пор пока не будет найдено соответствие. Удален будет лишь первый найденный элемент, список через метод arraycopy() сдвигается на 1 элемент вслево.

**Если нужно вставить в середину коллекции другую коллекцию:**

В случаях, когда в исходный список необходимо добавить другую коллекцию, да еще и в «середину», стоит использовать метод addAll(index, Collection). И хотя, данный метод скорее всего вызовет System.arraycopy() три раза, в итоге это будет гораздо быстрее поэлементного добавления.

**ИТОГИ:**

— Быстрый доступ к элементам **по индексу** за время **O(1);**

— Доступ к элементам **по значению** за линейное время **O(n);**

— Медленный, когда вставляются и удаляются элементы из «середины» списка;

— Добавление в конец (без увеличения размера коллекции) за **O(1);**

— Удаление последнего за **O(1);**

— Позволяет хранить любые значения в том числе и null;

— **Не синхронизирован**.

Данную реализацию следует применять, если в процессе работы с коллекцией предплагается частое обращение к элементам по индексу. Из-за особенностей реализации поиндексное обращение к элементам выполняется за константное время O(1). Но данную коллекцию рекомендуется избегать, если требуется частое удаление/добавление элементов в середину коллекции.

Отличие метода spliteterator() от метода iterator():

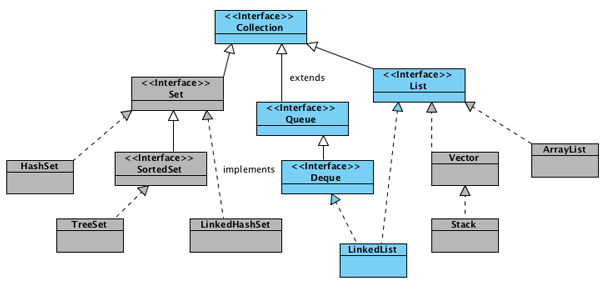
- умение разделяться

- нет метода remove (никогда не модифицирует источник)

Статья: https://habr.com/ru/post/128269/

**LinkedList**

(ДВУНАПРАВЛЕННЫЙ СПИСОК)



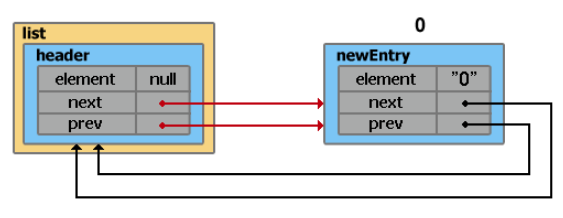
Особенностью реализации данной коллекции является то, что в её основе лежит двунаправленный связный список (каждый элемент имеет ссылку на предыдущий и следующий).

Внутри LinkedList присутствует внутренний класс **Node** – с помощью которого создаются новые элементы (которые будут содержать поля element, next, prev).

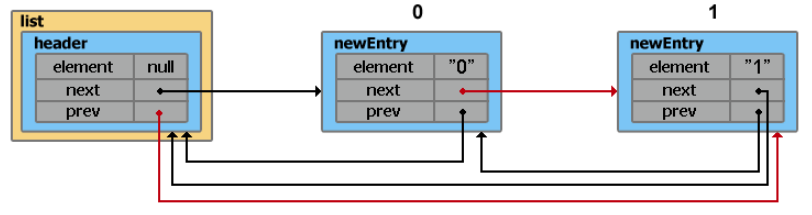
Внутри данной коллекции при её создании всегда имеется 1(стартовый) псевдо-элемент **HEADER** (значение которого = **null**, ссылка на следующий (**next**) элемент -> сама на себя или на следующий элемент (если коллекция не пуста), ссылка на предыдущий (**prev**) элемент -> сама на себя или на последний элемент (если коллекция не пуста)).

Так же внутри есть параметр **size** – который содержит данные о колличестве элементов в коллекции.

При добавлении нового элемента ссылки в конкретном месте переопределяются.



При добавлении ещё одного элемента, происходит опять переопределение ссылок **next** и **prev**:



**ИТОГИ:**

- Добавление add(elem) оно же и добавление в конец addLast(elem) или в начало addFirst(elem) – за **О(1).**

- Добавление в середину add(index, elem) – за **О(n);** Под капотом вызывается вспомогательный метод, который пробегает по всему списку (но по факту он проходит только половину).

-Однако, на добавление и удаление из середины списка, используя ListIterator.add() и ListIterator.remove(), потребуется O(1);

- Удаление элементов с конца и с начала (методы removeFirst(elem), removeLast(elem)) – за **О(1).**

- Удаление по индексу (remove (index)) и по значению (remove(value)) – за **О(n).**

**-** Из LinkedList можно организовать стэк, очередь, или двойную очередь, со временем доступа O(1);

- Позволяет хранить любые значения, включая null.

- Не синхронизирован.

**Итераторы**

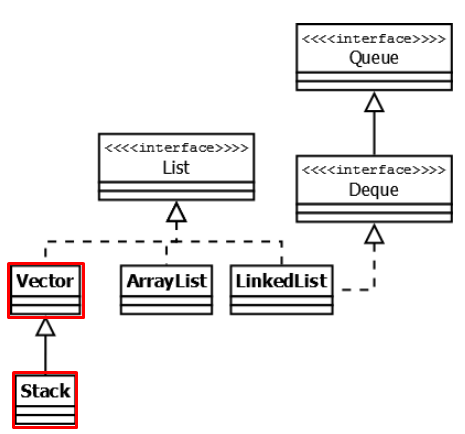
С помощью его можно двигаться по элементам связанного списка, узнавать есть ли следующий элемент, какой его индекс и др. Для этого необходимо на объекте LinkedList вызвать метод **listIterator().** Если начал работать с ListIterator – то нельзя одновременно использовать родные методы list.



Указатель итератора при этом оказывается перед первым элементом, можно в конструктор метода получения итератора предать позицию, в которую установить указатель. Можно поставить в конец списка указатель.

Стоит помнить, что ListIterator свалится с ConcurrentModificationException, если после создания итератора, список был изменен не через собственные методы итератора.

**ДРУГИЕ РЕАЛИЗАЦИИ LIST**



**Vector** - позволяет хранить любые данные, включая null в качестве элемента. Все операции потокобезопасные, в отличие от других реализаций List.

**Stack** — данная коллекция является расширением коллекции Vector. Реализация стека LIFO. Является частично **синхронизированной** коллекцией (кроме метода добавления push()). После добавления в Java 1.6 интерфейса Deque, рекомендуется использовать именно реализации этого интерфейса, например ArrayDeque.

**ФИШКИ**

Можно создать объект и так, но нельзя туда ничего добавлять и удалять, во время компиляции вылезит исключение UnsupportedOperationException:

