Коллекции:



**Collection<T>**: List,SortedSet,Deque,Queue

общие методы: size(), isEmpty(),clear(), contains(E), containsAll(E/Coll), remove(E), remove(Coll), add(E), addAll(Collection), toArray()

queue1.forEach(e -> System.out.print(e + " ")); - содержимое через пробелы

**List:**

Упорядоченный список indexOf(e),lastIndexOf(e) – индекс первого/последнего вхождения элемента,

set(i,e)-изменить значение,

get(i)-получить значение.

add(i,E)-добавить элемент

в позицию

addAll(i,Coll) – добавить коллекцию в позицию

remove(index)-удалить

элемент по индексу,

toArray() – конвертация в массив, subList(1,3) – получение сечения списка

toArray() – преобразование в массив, Collections.sort(List list) – сортировка

List<Integer> list=new ImpList(Arrays.asList(1,5,8 or Integer[])) – преобразование массива или значений через запятую в ImpList

ArrayList: список переменной длинны на основе обычного массива конструкторы: ArrayList()-емкость 10, ArrayList(n)-емкость n,

ArrayList(Collection<T> c)-сразу создается список из коллекции

LinkedList: список на основе двусвязанного списка, проигрывает ArrayList конструкторы: LinkedList()-пустой список, LinkedList(Collection<T> c)

Vector:(устарел) – аналог ArrayList, работает медленнее, но потокобезопасен. Впрочем, в ArrayList можно включить потокобезопасность.

Stack: ArrayList в качестве стека работает быстрее, ArrayDeque - еще быстрее

доп.методы: push(value) – поместить в стек, pop() -извлечь из стека

**Queue:**

add(value) вставка элемента в конец очереди немедленно, при уже заполненой очереди будет исключение IllegalStateException

offer(value) тоже вставка в пределах емкости.

Type remove()/poll() извлекает и удаляет элемент из очереди, возвращает NoSuchElementException/null если очередь пуста.

Type element()/peek() читает элемент из очереди, не удаляя его, возвращает NoSuchElementException/null если очередь пуста.

PrioritQueue: приоритетная очередь.

**Deque:** двунаправленная очередь.

! есть конструкторы, принимающий список Arrays.asList(массив)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **First Element(Head)** | | **Last Element (Tail)** | |
|  | Throws exeption | Special value | Throws exeption | Special value |
| **insert** | addFirst(E) | offerFirst(E) | addLast(E) | offerLast(E) |
| **remove** | removeFirst() | pollFirst() | removeLast() | pollLast() |
| **examine** | getFirst() | peekFirst() | getLast() | peekLast() |

ArrayDeque: очередь на основе массива, нет доступа по индексу.

**Set:** набор уникальных элементов, нет доступа по индексу.

Не имеет дополнительных интерфейсов.



! Сортировать множество нельзя, однако можно преобразовать его в список:

List<String> list=new ArrayList(str);//и отсортировать уже его

[HashSet](http://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html): базируется на HashMap. Внутри используется объект HashMap для хранения данных. В качестве ключа используется добавляемый элемент, а в качестве значения — объект-пустышка (new Object()). Порядок элементов не гарантируется при добавлении и может меняться во времени.

[LinkedHashSet](http://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashSet.html): отличается от HashSet только тем, что работает чуть медленнее, а в основе лежит LinkedHashMap (связанный список). Порядок элементов при обходе коллекции является идентичным порядку добавления элементов и не меняется в дальнейшем.

[TreeSet](http://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html): аналогично другим классам-реализациям интерфейса Set содержит в себе объект NavigableMap, что и обуславливает его поведение. Предоставляет возможность управлять порядком элементов в коллекции при помощи объекта Comparator, либо сохраняет элементы с использованием "natural ordering". HashSet работает намного быстрее, чем TreeSet, но последний имеет дополнительные методы:

Comparator<? super E> **comparator**() возвращает компаратор, либо null, если используется естественная сортировка.

E **first**(), E **last**() вернёт first/last (наименьший/наибольший) элемент

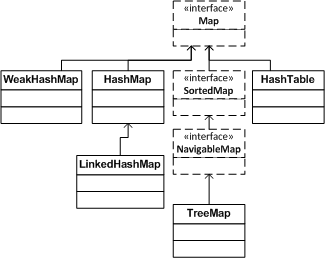
SortedSet<E> **headSet**(E toElement) множеcтво элементов, меньших E

SortedSet<E> **tailSet**(E fromElement) больших или равных E

SortedSet<E> **subSet**(E1 fromElement, E2 toElement) больших или равных E, но строго меньших чем E2.

EnumSet: Специализированная реализация для использования с перечислимыми типами.

**Map<K,V>:**

! Множество, содержащее пары ключ-значение, как массив. Значение ключей должны быть уникальными. K-Тип ключа, V-тип содержимого.

общие методы: size(), isEmpty(), containsKey(key), containsValue(value), clear()

get(key) – значение либо null по ключу, remove(key) – удалить пару K-V

V **put**(K key, V value) — добавить пару ключ-значение, если такой ключ есть, измениться значение.

V **putIfAbsent**(K key, V value) — если ключа нет, либо он связан с null, метод создает пару K-V и возвращает null, в противном случае возвращает V.

V **getOrDefault**(Object key, V defaultValue) — возвращает значение, либо defaultValue если ключ ни с чем не связан.

Set<K> **key**Set() – список ключей словаря

Collection<V> **value**() – список значений словаря в виде коллекции

Set<Map.Entry<K,V>> **entrySet**() – Map в виде множества пар ключ-значение.

[**Hashtable**](http://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html) — реализация такой структуры данных, как хэш-таблица, является синхронизированной. Нельзя использовать null в качестве ключа или значения. Низкая производительность, не рекомендуется к иcпользованию. Не упорядочена.

[**HashMap**](http://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html) — коллекция является альтернативой Hashtable. Не синхронизирована, допускается исп. null в качестве ключа или значения. Не упорядочена. HashMap можно синхронизировать с помощью вызова такого метода: Map myMap = Collections.synchonizedMap(hashMap);, но в многопоточной среде лучше использовать ConcurrentHashMap.

[**LinkedHashMap**](http://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html) — это упорядоченная реализация хэш-таблицы. Порядок итерирования равен порядку добавления элементов. Упорядочена.

[**TreeMap**](http://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html) — реализация Map основанная на красно-чёрных деревьях. Упорядочена. Автосортируется натуральным порядком или порядком, указанным с помощью компаратора при создании объекта. Допметоды:

Comparator<? super K> **comparator**() возвращает компаратор, либо null, если используется естественная сортировка.

E **firstKey**()/E **lastKey**() вернёт first/last (наименьший/наибольший) элемент

SortedMap<K, V> **headMap**(K toKey) - submap K-V, где ключи < K

SortedMap<K, V> **tailMap**(K fromKey) - submap K-V, где ключи >= K

SortedMap<K, V> **subMap**(K fromKey, E toKey) — submap K-V, Kkey>=...<Ekey

[**WeakHashMap**](http://docs.oracle.com/javase/8/docs/api/java/util/WeakHashMap.html) — реализация хэш-таблицы, которая организована с использованием слабых ссылок. Сборщик мусора автоматически удалит элемент из коллекции при следующей сборке мусора, если на ключ этого элемента нет жёстких или мягких ссылок.