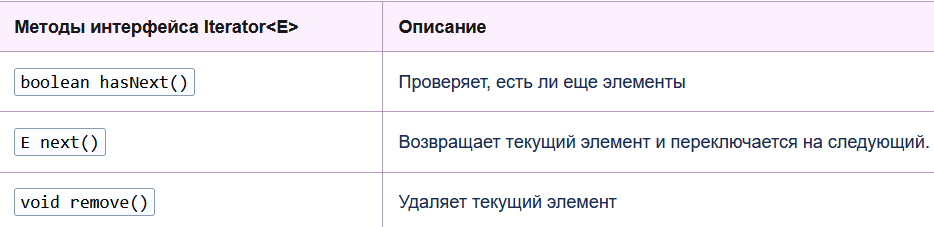


Коллекциz — это набор однородных элементов.

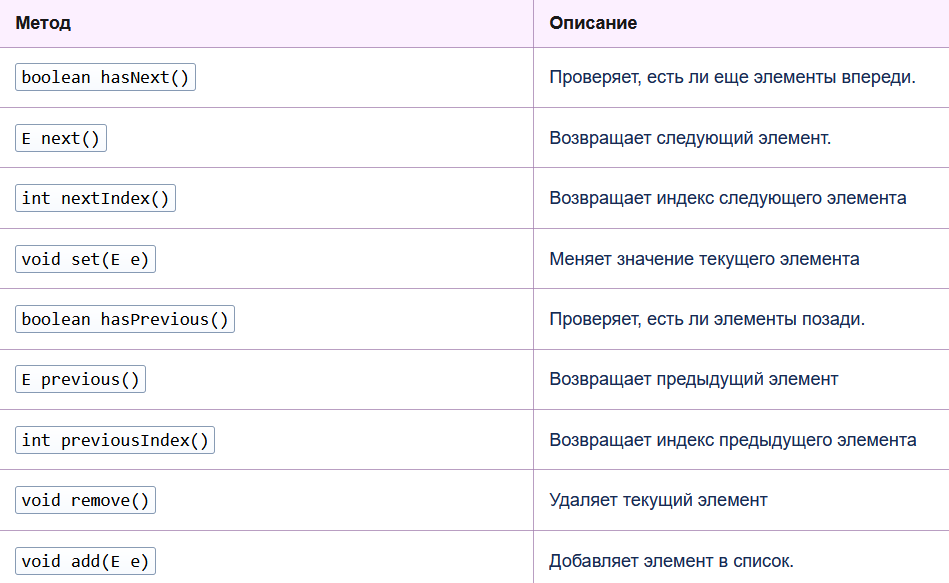
Интерфейс **Iterable<T>** означает перебираемый. Если некий класс реализует Iterable<T>, значит, он содержит внутри себя элементы T, которые можно перебирать с помощью цикла for-each. У этого интерфейса есть единственный метод:



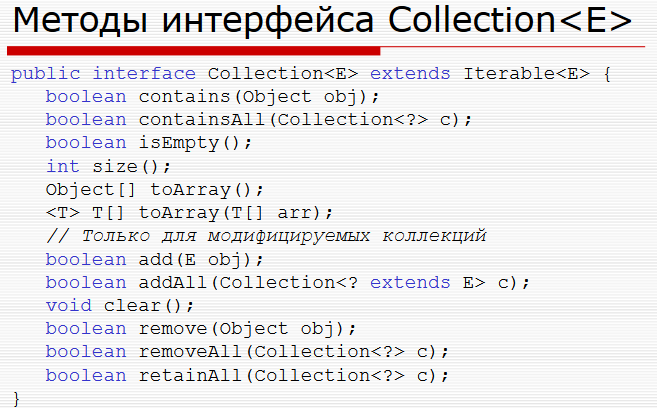
Итератор – объект, позволяющий перебирать элементы коллекции, знающий ее внутреннюю структуру. Для этого класс должен реализовывать интерфейс Iterator<T>. Он имеет следующие методы:



Есть еще одна популярная разновидность итераторов, для которой даже придумали свой интерфейс. Речь идет об итераторе для списков – **ListIterator**.



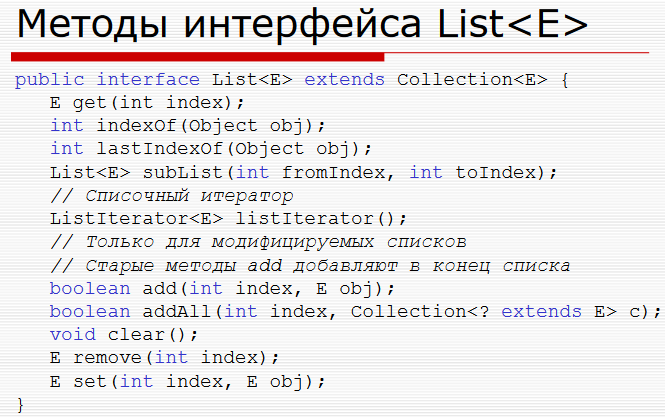
Интерфейс Collection<E> расширяет Iterable - корневой интерфейс иерархии коллекций. Коллекция состоит из элементов, которые мо умолчанию могут дублироваться и неупорядочены.



Интерфейс Collection напрямую не реализуется классами из JDK.

Реализуются его расширения: List, Queue, Set.

Интерфейс *List<E>* - список, упорядоченная коллекция, в которой элементы могут дублироваться.



Основные реализации **ArrayList** и **LinkedList.**

**ArrayList** реализован с помощью массива. Изначально на 10 элементов, но в последствии расширяется. новая емкость рассчитывается по формуле **(oldCapacity \* 3) / 2 + 1.** При расширении создается новый массив, куда затем копируется старый, используя **native**(написан не на java) метод **System.arraycopy().**

**LinkedList** реализован в виде связного списка: набора отдельных элементов, каждый из которых хранит ссылку на следующий и предыдущий элементы. Содержит ссылки на первый элемент и на последний.

**Vector** реализован как и ArrayList, но является синхронизированным и работает медленнее. Устарел.

**SET**

Реализация интерфейса **Set** представляет собой неупорядоченную коллекцию, которая не может содержать дублирующие данные.

* public Iterator iterator()
* public int size()
* public boolean isEmpty()
* public boolean contains(Object o)
* public boolean add(Object o)
* public boolean addAll(Collection c)
* public Object[] toArray()
* public boolean remove(Object o)
* public boolean removeAll(Collection c)
* public boolean retainAll(Collection c) - (retain — сохранить). Выполняет операцию "пересечение множеств".
* public void clear()
* public Object clone()

Интерфейс SortedSet предназначен для создания коллекций, который хранят элементы в отсортированном виде.

Интерфейс NavigableSet расширяет интерфейс SortedSet и позволяет извлекать элементы на основании их значений. А также возможность получения элементов отображения относительно других элементов. Реализацией является TreeSet.

* **NavigableSet<E> descendingSet()**: возвращает объект NavigableSet, который содержит все элементы первичного набора NavigableSet в обратном порядке
* **NavigableSet<E> headSet(E upperBound, boolean incl)**: возвращает объект NavigableSet, который содержит все элементы первичного набора NavigableSet до upperBound. Параметр incl при значении true, позволяет включить в выходной набор элемент upperBound
* **NavigableSet<E> tailSet(E lowerBound, boolean incl)**: возвращает объект NavigableSet, который содержит все элементы первичного набора NavigableSet, начиная с lowerBound. Параметр incl при значении true, позволяет включить в выходной набор элемент lowerBound
* **NavigableSet<E> subSet(E lowerBound, boolean lowerIncl, E upperBound, boolean highIncl)**: возвращает объект NavigableSet, который содержит все элементы первичного набора NavigableSet от lowerBound до upperBound.

У **TreeSet** под капотом красно-черное дерево(логарифмическая сложность).

У **HashSet** под капотом HashMap, в качестве ключа используется сам элемент, а в качестве значения заглушка – константа PRESENT, являющаяся экземпляром класса Object.

**LinkedHashSet** запоминает порядок добавления элементов. Под капотом LinkedHashMap.

**MAP**

**<Map>** – коллекция которая состоит с пар “ключ”-“значение”.

Среди методов интерфейса Map можно выделить следующие:

* void clear(): очищает коллекцию
* boolean containsKey(Object k): возвращает true, если коллекция содержит ключ k
* boolean containsValue(Object v): возвращает true, если коллекция содержит значение v
* Set<Map.Entry<K, V>> entrySet(): возвращает набор элементов коллекции. Все элементы представляют объект Map.Entry (Ключ значение)
* boolean equals(Object obj): возвращает true, если коллекция идентична коллекции, передаваемой через параметр obj
* boolean isEmpty: возвращает true, если коллекция пуста
* V get(Object k): возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение null
* V getOrDefault(Object k, V defaultValue): возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение defaultVlue
* V put(K k, V v): помещает в коллекцию новый объект с ключом k и значением v. Если в коллекции уже есть объект с подобным ключом, то он перезаписывается. После добавления возвращает предыдущее значение для ключа k, если он уже был в коллекции. Если же ключа еще не было в коллекции, то возвращается значение null
* V putIfAbsent(K k, V v): помещает в коллекцию новый объект с ключом k и значением v, если в коллекции еще нет элемента с подобным ключом.
* Set<K> keySet(): возвращает набор всех ключей отображения
* Collection<V> values(): возвращает набор всех значений отображения
* void putAll(Map<? extends K, ? extends V> map): добавляет в коллекцию все объекты из отображения map
* V remove(Object k): удаляет объект с ключом k
* int size(): возвращает количество элементов коллекции

Интерфейс SortedMap создает Map, в котором все элементы отсортированы в порядке возрастания их ключей.

Интерфейс **NavigableMap** расширяет интерфейс SortedMap и обеспечивает возможность получения элементов отображения относительно других элементов. Реализацией является TreeMap.

Методы

* headMap() – все элементы, меньше данного.
* tailMap() – все элементы, равные или больше данного.
* subMap() – вырезать часть map, в соответсвии с границами (не включительно).

**HashMap –** внутри состоит с так званых корзин и списка элементов, на которые ссылаются корзины.

**Корзины –** массив.  
**Элементы –** связной список.

При добавлении высчитывается хеш-код, который определяет в какую ячейку отправится элемент. Если ячейка занята, то они выстраиваются в связный список. Если элемент уже находится в списке(проверяется с помощью equals) он заменяется.

**LinkedHashMap** использует HashMap, а также запоминает порядок добавления элементов с помощью связного списка. Каждый раз при добавлении перестраиваются ссылки в двухсвязном списке.

**TreeMap** использует под капотом красно-черное дерево. Элементы, хранящиеся в нем, должны реализовывать интерфейс Comporable, для сравнения путем «естественной сортировки». Можно также передавать в конструктор компаратор.

**HashTable** реализован как и HashMap только синхронизирован. Устарел.

**EnumSet и EnumMap** используются там, где Enum используются в качестве ключа.

**ДРУГИЕ КОЛЛЕКЦИИ**

Интерфейс ***Queue*** расширяет *Collection* и объявляет поведение очередей, которые представляют собой список с дисциплиной "первый вошел, первый вышел" (*FIFO*).

***Deque****-* это подинтерфейс of ***Queue***, предоставляющий методы для вставки элемента в начало или конец, а также методы для доступа или удаления его первого или последнего элемента.

***Deque*** также включает в себя методы, которые позволяют ему действовать как **stack**

*ArrayDeque* - **это особый вид растущего массива, который позволяет нам добавлять или удалять элемент с обеих сторон.**

**Stack** наследуется от вектора , реализует LIFO.

Чтобы сравнивать объекты мы должны реализовать интерфейс Comparable и реализовывают его единственный метод — compareTo(). Он является типизированным:

class A implements Comporable<A>{

@Override

public int compareTo(A a){

//return 1, 0, -1

return this.field — a.field

}

Реализованный в Comparable метод сравнения называют «natural ordering» — естественной сортировкой. Это потому, что в методе compareTo() ты описываешь наиболее распростаненныую сортировку.

Natural Ordering уже присутствует в Java для строк, чисел и т.д.

В TreeSet можно передать компаратор, у которого есть метод Compare.

Если же вдруг нам нужна другая логика, нам приходит на помощь другой интерфейс — Comparator. Этот компаратор затем можно передать в метод sort.

Comparator — это отдельный класс-«сравниватель».

public class CustomComparator implements Comparator<A> {

@Override

public int compare(A a1, A a2) {

return a1.getField() - a2.getField();

}

}

Т. о. В TreeSet для сравнения используются 2 метода: compare и compareTo. То есть в TreeSet можно положить только объекты, реализующие интерфейс Comporable.

В HashSet можно положить любой элемент, если в угловых скобках не указать тип: HashSet<>.