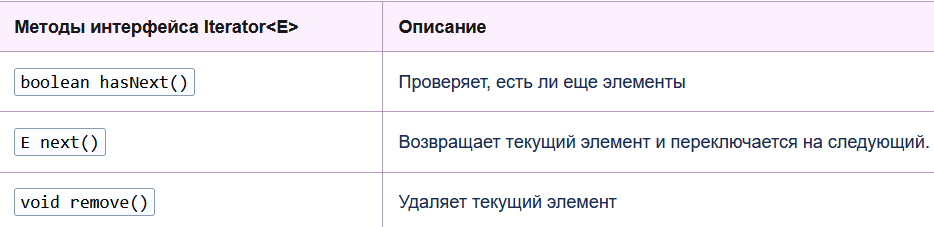


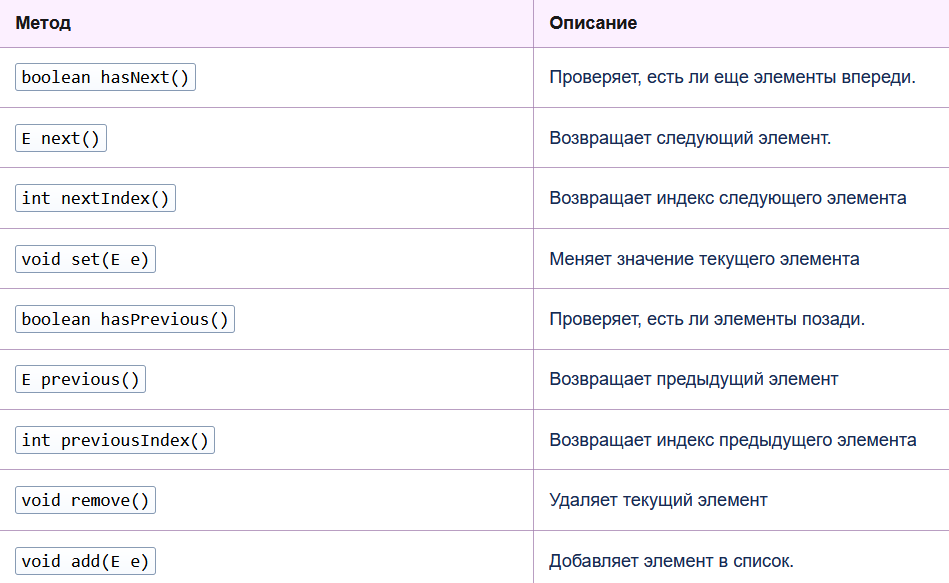
Интерфейс **Iterable<T>** означает перебираемый. Если некий класс реализует Iterable<T>, значит, он содержит внутри себя элементы T, которые можно перебирать с помощью цикла for-each. У этого интерфейса есть единственный метод:



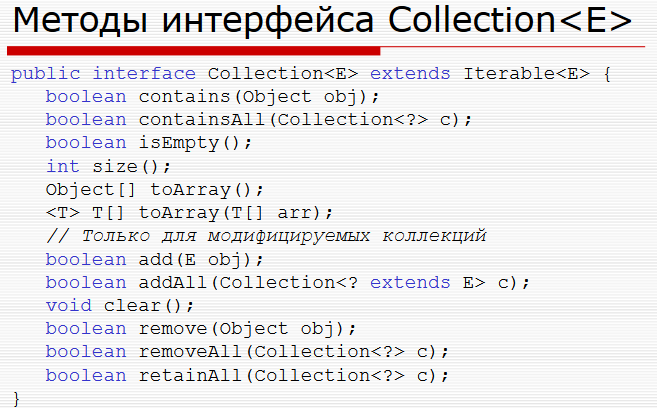
Итератор – объект, позволяющий перебирать элементы коллекции, знающий ее внутреннюю структуру. Для этого класс должен реализовывать интерфейс Iterator<T>. Он имеет следующие методы:



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



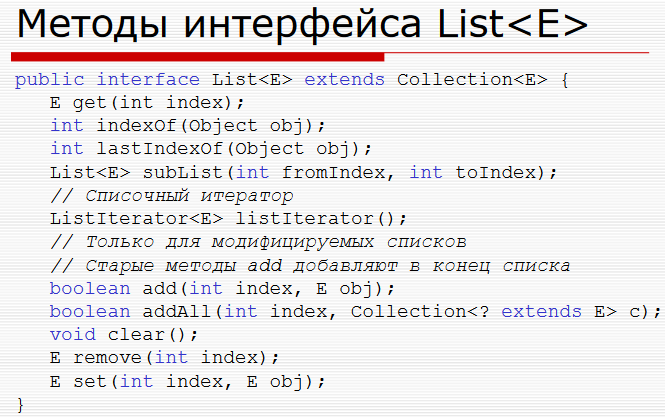
Интерфейс Collection<E> расширяет Iterable - корневой интерфейс иерархии коллекций. Коллекция состоит из элементов, которые мо умолчанию могут дублироваться и неупорядочены.



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.

**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.

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.

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

У **HashSet** под капотом HashMap, а в качестве ключа используется сам элемент.

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

**MAP**

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

**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.