**Интерфейсы Iterator и Iterable**

В Java есть специальный интерфейс Iterator, позволяющий поочередно получить все элементы коллекции. Вот какие у него методы:

|  |  |
| --- | --- |
| **Методы интерфейса Iterator<E>** | **Описание** |
| boolean **hasNext()** | Проверяет, есть ли еще элементы |
| E **next()** | Возвращает текущий элемент и переключается на следующий. |
| void **remove()** | Удаляет текущий элемент |

Кроме итератора есть еще промежуточный (*между интерфейсом Iterator и коллекцией*) интерфейс Iterable – его должны реализовывать все коллекции, которые поддерживают итератор. У него есть единственный метод, с помощью которого у любой коллекции можно получить объект итератор для обхода ее элементов:

|  |  |
| --- | --- |
| **Методы interface Iterable<T>** | **Описание** |
| Iterator<T> **iterator()** | Возвращает объект-итератор |

Давай обойдем все элементы дерева в коллекции **TreeSet**:

**Пример**

TreeSet<String> col = **new** TreeSet<String>();

Iterator<String> iter = col.**iterator**(); //получаем объект итератора

для обхода коллекции “col”

**while** (iter.**hasNext**()) //к полученному объекту итератора “ iter ” применяем

проверочный метод “.hasNext()”

{

String item = iter.**next**();//получаем текущий элемент коллекции и переходим

System.out.println(item); //печатаем

}

Такое использование итератора не очень удобно – слишком много лишнего и очевидного кода.

Ситуация упростилась, когда в Java появился цикл по итератору – **for-each**, который уже имеет в себе встроенный(*автоматически генерируемый*):

- код по получению объекта итератора методом “*.iterator()*”,

- код по получению текущего элемента коллекции методом “*.next()*”.

Теперь такой код гораздо компактнее и читабельнее:

|  |  |
| --- | --- |
| **Было** | **Стало** |
| TreeSet**<**String**>** col = **new** TreeSet**<**String**>**();  Iterator<String> iter = col.iterator();  **while** (iter.**hasNext**())  {  String item = iter.**next**();  System.out.println(item);  } | TreeSet<String> col = **new** TreeSet<String>();  **for**(String item : col)  {  System.out.println(item);  } |

Это один и тот же код! Итератор используется и там, и там.

Цикл **for-each** можно использовать для любых объектов, которые поддерживают итератор. Т.е. можно написать свой класс, добавить ему метод **iterator**() и использовать его объекты в правой части конструкции**for-each.**

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

Списки, независимо от реализации, обладают порядком элементов, что в свою очередь позволяет работать с ними через итератор чуть более удобно.

Вот какие методы есть у интерфейса **ListIterator**<E>:

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| boolean **hasNext()** | Проверяет, есть ли еще элементы впереди. |
| E **next()** | Возвращает следующий элемент |
| int **nextIndex()** | Возвращает индекс следующего элемента |
| void **set(E e)** | Меняет значение текущего элемента |
| boolean **hasPrevious()** | Проверяет, есть ли элементы позади |
| E **previous()** | Возвращает предыдущий элемент |
| int **previousIndex()** | Возвращает индекс предыдущего элемента |
| void **remove()** | Удаляет текущий элемент |
| void **add(E e)** | Добавляет элемент в список |

Т.е. тут мы можем ходить не только вперед, но и назад. И еще пара фич по мелочи.

Например, необходимо двигаться туда-обратно по связному списку.

При этом операция get будет довольно медленной, а операция next() очень быстрой.