

**Методы интерфейса java.util.Collection<E>**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| *Iterator*<E> **iterator**() | Возвращает итератор для обращения к элементам набора данных. |
| *int* **size**() | Возвращает количество элементов в наборе данных. |
| *boolean* **isEmpty**() | Возвращает значение true, если набор пустой. |
| *boolean* **contains** (Object obj) | Возвращает true, если набор содержит объект, эквивалентный obj. |
| *boolean* **containsAll** (Collection<?> other) | Возвращает true, если текущий набор содержит все объекты набора данных other. |
| *boolean* **add** (Object element) | Добавляет элемент в набор. Возвращает true, если в результате вызова метода набор данных изменился. |
| *boolean* **addAll** (Collection<? extends E> other) | Добавляет все элементы в набор. Возвращает true, если в результате вызова метода набор данных изменился. |
| *boolean* **remove** (Object obj) | Удаляет объект obj. Возвращает true, если в результате вызова метода набор данных изменился. |
| *boolean* **removeAll** (Collection<?> other) | Удаляет из текущего набора данных все элементы, содержащиеся в наборе other. Возвращает true, если в результате вызова метода набор данных изменился. |
| *void* **clear** () | Удаляет из текущего набора данных все элементы. |
| *boolean* **retainAll** (Collection<?> other) | Удаляет из набора данных элементы, не совпадающие с теми, которые содержатся в наборе other. Возвращает true, если в результате вызова метода набор данных изменился. |
| **Object[] toArray ()** | Возвращает массив с объектами из набора данных. |

**Методы итератора java.util.Iterator<E>**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| *boolean* **hasNext**() | Возвращает значение true, если в коллекции имеется следующий элемент, иначе возвращает false |
| *Object* **next**() | Возвращает следующий элемент. Если достигнут конец набора,то генерируется исключение NoSuchElementException |
| *void* **remove**() | Удаляет последний прочитанный элемент. Этот метод должен быть вызван сразу же после обращения к элементу. Если после чтения элемента набор данных изменился, данный метод генерирует исключение IllegalStateException |

**Методы интерфейса java.util.Set<E>**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **add**(Object o) | Добавление элемента в коллекцию, если он отсутствует. Возвращает true, если элемент добавлен. |
| **addAll**(Collection c) | Добавление элементов коллекции, если они отсутствуют. |
| **clear**() | Очистка коллекции. |
| **contains**(Object o) | Проверка присутствия элемента в наборе. Возвращает true, если элемент найден. |
| **containsAll**(Collection c) | Проверка присутсвия коллекции в наборе. Возвращает true, если все элементы содержатся в наборе. |
| **equals**(Object o) | Проверка на равенство. |
| **hashCode**() | Получение hashCode набора. |
| **isEmpty**() | Проверка наличия элементов. Возвращает true если в коллекции нет ни одного элемента. |
| **iterator**() | Функция получения итератора коллекции. |
| **remove**(Object o) | Удаление элемента из набора. |
| **removeAll**(Collection c) | Удаление из набора всех элементов переданной коллекции. |
| **retainAll**(Collection c) | Удаление элементов, не принадлежащих переданной коллекции. |
| **size**() | Количество элементов коллекции |
| **toArray**() | Преобразование набора в массив элементов. |
| **toArray**(T[] a) | Преобразование набора в массив элементов. В отличии от предыдущего метода, который возвращает массив объектов типа Object, данный метод возвращает массив объектов типа, переданного в параметре. |

**List:**

* **void add**(int index, E obj): добавляет в список по индексу index объект obj
* **boolean addAll**(int index, Collection<? extends E> col): добавляет в список по индексу index все элементы коллекции col. Если в результате добавления список был изменен, то возвращается true, иначе возвращается false
* **E get**(int index): возвращает объект из списка по индексу index
* **int indexOf**(Object obj): возвращает индекс первого вхождения объекта obj в список. Если объект не найден, то возвращается -1
* **int lastIndexOf**(Object obj): возвращает индекс последнего вхождения объекта obj в список. Если объект не найден, то возвращается -1
* **ListIterator<E> listIterator ()**: возвращает объект ListIterator для обхода элементов списка
* **static <E> List<E> of(элементы)**: создает из набора элементов объект List
* **E remove(int index)**: удаляет объект из списка по индексу index, возвращая при этом удаленный объект
* **E set**(int index, E obj): присваивает значение объекта obj элементу, который находится по индексу index
* **void sort**(Comparator<? super E> comp): сортирует список с помощью компаратора comp
* **List<E> subList**(int start, int end): получает набор элементов, которые находятся в списке между индексами start и end

Обобщите различия между **List.of(создает immutable коллекцию)** и **Arrays.asList(создает обычную коллекцию)**

* List.of лучше всего использовать, когда набор данных меньше и неизменен, а Arrays.asList может использоваться наилучшим образом в случае большого и динамического набора данных.
* List.of занимает очень мало места накладных расходов, потому что он имеет реализацию на основе полей и потребляет меньше места кучи, как с точки зрения фиксированных накладных расходов, так и на основе каждого элемента. в то время как Arrays.asList занимает больше служебного пространства, потому что при инициализации он создает больше объектов в куче.
* Сбор, возвращенный List.of, является неизменным и, следовательно, поточно-безопасным, тогда как Collection, возвращаемый Arrays.asList, является изменяемым и не является потокобезопасным. (Неизменяемые экземпляры коллекции обычно потребляют гораздо меньше памяти, чем их изменяемые копии.)
* List.of не разрешает нулевые элементы, а Arrays.asList допускает нулевые элементы.

**Queue:**

**offer()** вставляет элемент в очередь, если это не удалось - возвращает false. Этот метод отличается от метода **add()** интерфейса Collection тем, что метод add() может неудачно добавить элемент только с использованием unchecked исключения.

**remove()** и **poll()** удаляют верхушку очереди и возвращают ее. Какой элемент будет удален (первый или последний) зависит от реализации очереди. Методы remove() и poll() отличаются лишь поведением, когда очередь пустая: метод remove() генерирует исключение, а метод poll() возвращает null.

**element()** и **peek()** возвращают (но не удаляют) верхушку очереди.

То есть, вышеприведенные методы существуют в двух формах: одни генерируют исключение если операция не удалась, другие - возвращают специальное значение. Это можно продемонстрировать следующей таблицей:

| **Тип операции** | **Генерирует исключение** | **Возвращает специальное значение** |
| --- | --- | --- |
| Добавление | add(e) | offer(e) |
| Удаление | remove() | poll() |
| Получение верхушки | element() | peek() |

Обычно, очередь не позволяет добавлять null элементы из-за того, что null используется как служебный объект метода poll() и peek(). Однако в некоторых реализация интерфейса Queue есть возможность добавление null элементов (например, LinkedList). Также не разрешается добавлять элементы, которые нельзя сравнить с помощью класса Comparator.