Оглавление

[Коллекция List в Java. 3](#_Toc125056435)

[Реализации. 4](#_Toc125056436)

[Поиск элементов. 4](#_Toc125056437)

[Удаление элементов. 5](#_Toc125056438)

[Подсписок. 6](#_Toc125056439)

[Проверка содержится ли элемент. 7](#_Toc125056440)

[Удаление элементов. 7](#_Toc125056441)

[Очередь (Queue). 8](#_Toc125056442)

[Deque. 8](#_Toc125056443)

[Set. 9](#_Toc125056444)

[HashSet. 9](#_Toc125056445)

[SortedSet. 9](#_Toc125056446)

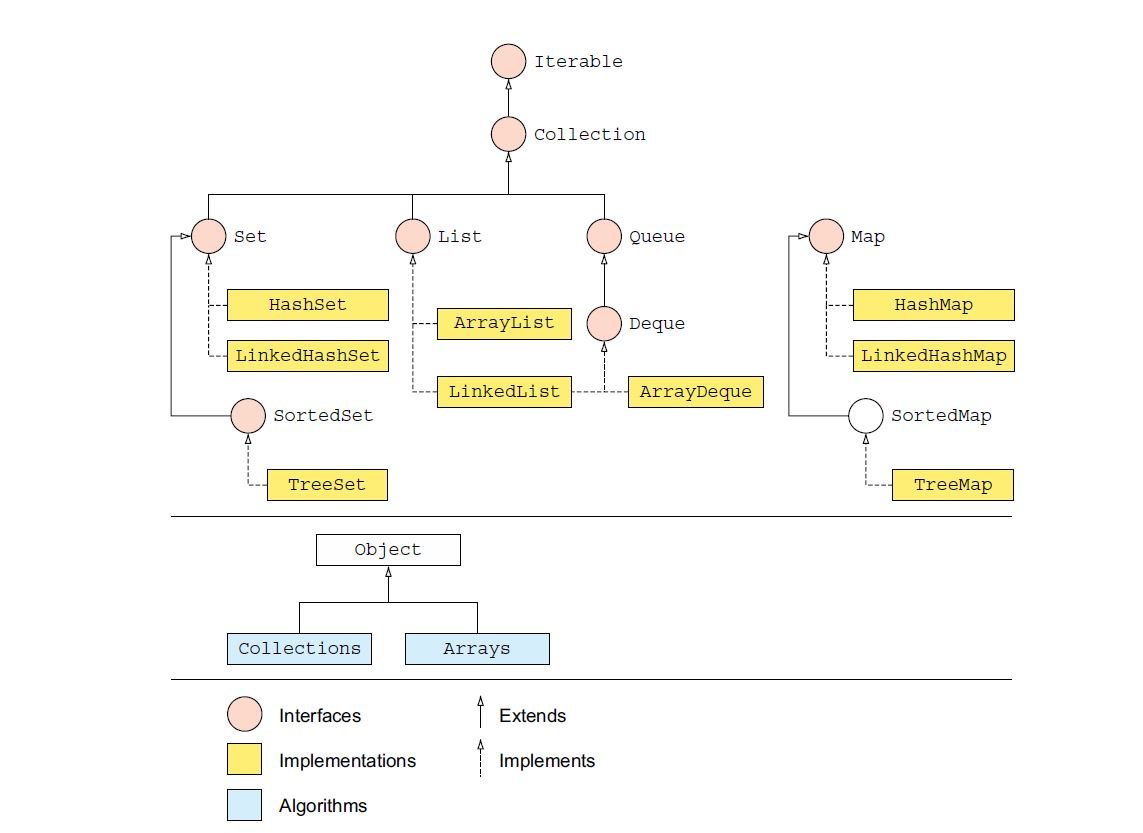
[NavigableSet. 9](#_Toc125056447)

[Карты (Map). 10](#_Toc125056448)

Collection.

Коллекции — это наборы однородных элементов. Например, страницы в книге, яблоки в корзине или люди в очереди.

Инструменты для работы с такими структурами в Java содержатся в **Java Collections Framework**. Фреймворк состоит из интерфейсов, их реализаций и утилитарных классов для работы со списками: сортировки, поиска, преобразования.



# Коллекция List в Java.

Интерфейс Java List, java.util.List, представляет упорядоченную последовательность объектов. Элементы, содержащиеся в списке Java, могут быть вставлены, повторены и удалены в соответствии с порядком их появления в нем. Порядок элементов – вот почему эта структура данных называется списком.

Каждый элемент имеет индекс: первый элемент – 0, второй – 1 и т. д. Индекс означает «сколько элементов находится в начале». Таким образом, первый элемент находится на расстоянии 0 элементов от начала списка, потому что он находится в начале.

Вы можете добавить любой объект в список. Если он не типизирован с использованием Java Generics, то вы можете даже смешивать объекты разных типов (классов) в одном и том же списке. Однако на практике это происходит не часто.

Интерфейс List является стандартным интерфейсом и подтипом интерфейса Collection, что означает, что List наследуется от Collection.

## Реализации.

Будучи подтипом Collection, все методы в интерфейсе Collection также доступны в интерфейсе List.

Поскольку List – это интерфейс, вам нужно создать конкретную реализацию интерфейса, чтобы использовать его, либо выбрать между следующими в API коллекций:

* java.util.ArrayList;
* java.util.LinkedList;
* java.util.Vector;
* java.util.Stack.

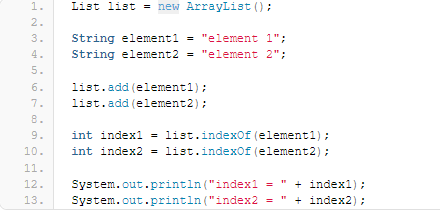
Также есть параллельные реализации в пакете java.util.concurrent.

## Поиск элементов.

Используется один из этих двух методов:

* indexOf();
* LastIndexOf().

Метод indexOf() находит индекс первого вхождения в списке заданного элемента:



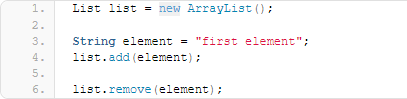
## Удаление элементов.

Вы можете удалить элементы с помощью этих двух методов:

remove(Object element);

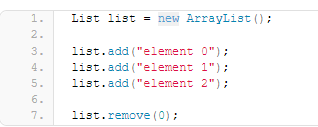
remove(int index).

remove(Object element) удаляет этот элемент в списке, если он присутствует. Все последующие элементы затем перемещаются вверх по списку. Таким образом, их индекс уменьшается на 1. Вот пример:



Этот пример сначала добавляет элемент в список, а затем снова удаляет его.

Метод remove(int index) удаляет элемент по указанному индексу. Все последующие элементы в списке затем перемещаются вверх. Таким образом, их индекс уменьшается на 1. Вот пример:



После запуска этого примера кода список будет содержать элементы 1 и 2 элемента Java String с индексами 0 и 1. Первый элемент (элемент 0) был удален из списка.

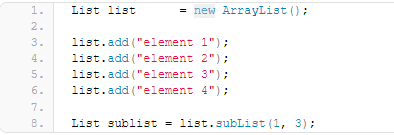
## Подсписок.

Интерфейс List имеет метод subList(), который может создавать новый List с подмножеством элементов из исходного List.

Метод subList() принимает 2 параметра:

1. Начальный индекс – это индекс первого элемента из исходного списка для включения в подсписок.
2. Конечный индекс является последним индексом подсписка, но элемент в последнем индексе не включается в подсписок.

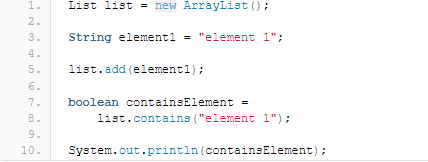
Это похоже на то, как работает метод подстроки Java String. Вот пример:



После выполнения инструкции list.subList(1,3) подсписок будет содержать элементы с индексами 1 и 2. Помните, что исходный список содержит 4 элемента с индексами от 0 до 3. Вызов list.subList(1,3) будет включить индекс 1, но исключить индекс 3, тем самым сохраняя элементы с индексами 1 и 2.

## Проверка содержится ли элемент.

Используется метод List contains():



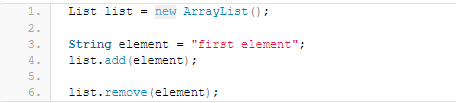
Чтобы определить, содержит ли List этот элемент, List будет внутренне выполнять итерации своих элементов и сравнивать каждый с объектом, переданным в качестве параметра. Для сравнения используется метод равенства.

## Удаление элементов.

Вы можете удалить элементы с помощью этих двух методов:

* remove(Object element);
* remove(int index).

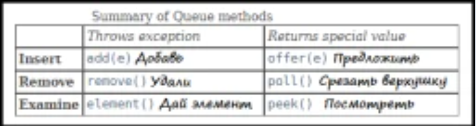
remove(Object element) удаляет этот элемент в списке, если он присутствует. Все последующие элементы затем перемещаются вверх по списку. Таким образом, их индекс уменьшается на 1. Вот пример:



# Очередь (Queue).

Очередь (Queue) — это структура, знакомая нам из жизни. Очереди в магазины, к врачам. Кто первее пришёл (First In), тот первее и выйдет из очереди (First Out). В Java очередь представлена интерфейсом [java.util.Queue](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html).

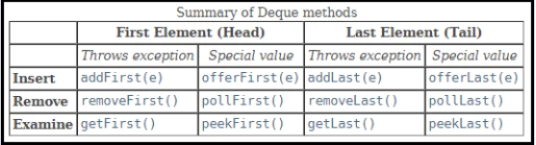
Согласно Javadoc очереди, очередь добавляет следующие методы:



Есть методы-приказы (их невыполнение чревато исключением) и есть методы-просьбы (невозможность их выполнить не приводит к ошибкам). Кроме того, можно получить элемент без удаления (peek или элемент).

## [Deque](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html).

У интерфейса очереди есть так же полезный наследник — [Deque](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html). Это так называемая "двусторонняя очередь". То есть такая очередь позволяет использовать эту структуру как с начала, так и с конца. В документации сказано, что "Deques can also be used as LIFO (Last-In-First-Out) stacks. This interface should be used in preference to the legacy Stack class.", то есть вместо Stack рекомендуется использовать реализации Deque. В Javadoc показано, какие методы описывает интерфейс Deque:



# Set.

## **HashSet.**

Класс **HashSet** использует для хранения данных в хеш-таблице. Это значит, что при манипуляциях с элементами используется хеш-функция — **hashCode()** в Java.

Хеш-таблица — структура данных, в которой все элементы помещаются в бакеты (buckets), соответствующие результату вычисления хеш-функции.

**Set** — это неупорядоченное множество уникальных элементов.

Например, мешочек с бочонками для игры в лото: каждый номер от 1 до 90 встречается в нём ровно один раз, и заранее неизвестно, в каком порядке бочонки вынут при игре.

Как сказано в документации, Set — это "collection that contains no duplicate elements". Интересно, что сам интерфейс Set не добавляет новых методов к интерфейсу Collection, а лишь уточняет требования (про то, что не должно содержать дубликатов). Кроме того, из прошлого описания следует, что просто так из Set нельзя получить элемент. Для получения элементов используется Iterator.

## SortedSet.

Set имеет ещё несколько связанных с собой интерфейсов. Первый это SortedSet. Как и следует из названия, SortedSet указывает на то, что такой набор отсортирован, а следовательно элементы реализуют интерфейс Comparable или указан Comparator.

## NavigableSet.

У SortedSet есть наследник — NavigableSet. Цель этого интерфейса — описать методы навигации, которые нужны для более точного определения подходящих элементов. Из интересного — NavigableSet добавляет к привычному iterator (который идёт от меньшего к большему) итератор для обратного порядка — descendingIterator. Кроме того, NavigableSet позволяет при помощи метода descendingSet получить вид на себя (View), в котором элементы идут в обратном порядке. Это называется View, потому что через полученный элемент можно изменять элементы изначального Set. То есть, по сути, это представление изначальных данных другим способом, а не их копия.

# Карты (Map).

Класс**HashMap** хранит данные в виде хеш-таблицы, как и HashSet. Более того, HashSet внутри использует HashMap. При этом ключом выступает сам элемент.

**Map** состоит из пар «ключ-значение». Ключи уникальны, а значения могут повторяться. Порядок элементов не гарантирован. Map позволяет искать объекты (значения) по ключу.

Пример: стопка карточек с иностранными словами и их значениями. Для каждого слова (ключ) на обороте карточки есть вариант перевода (значение), а вытаскивать карточки можно в любом порядке.

**Map** не наследуется от интерфейса Collection, но входит в состав фреймворка Collections.