



JAVA™

# Программирование на Java

Аксенов Сергей Алексеевич



# Классы-коллекции

Неудобства при использовании классических массивов:

- Длина массива задается заранее.
- Удаление или перемещение элемента не является тривиальной операцией.
- ...



# Классы-коллекции

## Структуры данных:

- стек,
- очередь,
- список,
- множество,
- Карта (Ассоциативный массив),
- ...





# Классы-коллекции

**Интерфейс Collection** (содержит набор общих методов, которые используются в большинстве коллекций)

- **add(Object item)** — добавляет в коллекцию новый элемент. Метод возвращает true, если добавление прошло успешно и false — если нет\*. Если элементы коллекции каким-то образом упорядочены, новый элемент добавляется в конец коллекции.
- **clear()** — удаляет все элементы коллекции.
- **contains(Object obj)** — возвращает true, если объект obj содержится в коллекции и false, если нет.
- **isEmpty()** — проверяет, пуста ли коллекция.
- о методах интерфейса Collection
- **remove(Object obj)** — удаляет из коллекции элемент obj. Возвращает false, если такого элемента в коллекции не нашлось.
- **size()** — возвращает количество элементов коллекции.



# Классы-коллекции

**Интерфейс List** (описывает упорядоченный список. Элементы списка пронумерованы, начиная с нуля и к конкретному элементу можно обратиться по целочисленному индексу) Расширяет Collection следующими методами:

- **add(int index, Object item)** — вставляет элемент item в позицию index, при этом список раздвигается (все элементы, начиная с позиции index, увеличивают свой индекс на 1);
- **get(int index)** — возвращает объект, находящийся в позиции index;
- **indexOf(Object obj)** — возвращает индекс первого появления элемента obj в списке;
- **lastIndexOf(Object obj)** — возвращает индекс последнего появления элемента obj в списке;
- **add(int index, Object item)** — заменяет элемент, находящийся в позиции index объектом item;
- **subList(int from, int to)** — возвращает новый список, представляющий собой часть данного (начиная с позиции from до позиции to-1 включительно).



# Классы-коллекции

**Интерфейс Set** (описывает множество. Элементы множества не упорядочены, множество не может содержать двух одинаковых элементов. Интерфейс Set унаследован от интерфейса Collection, но никаких новых методов не добавляет. Изменяется только смысл метода `add(Object item)`)

- **`add(Object item)`** — добавляет в множество новый элемент. Метод возвращает `true`, если добавление прошло удачно и `false` — если нет\*. Метод не станет добавлять объект `item`, если он уже присутствует в множестве.



# Классы-коллекции

**Интерфейс SortedSet** (описывает упорядоченное множество, отсортированное по естественному порядку возрастания его элементов или по порядку, заданному реализацией интерфейса `comparator`. Элементы не нумеруются, но есть понятие первого, последнего, большего и меньшего элемента)

Дополнительные методы интерфейса:

- **`comparator comparator ()`** — возвращает способ упорядочения коллекции;
- **`object first ()`**— возвращает первый, меньший элемент коллекции;
- **`SortedSet headset (Object toElement)`** — возвращает начальные, меньшие элементы до элемента `toElement` исключительно;
- **`object last ()`** — возвращает последний, больший элемент коллекции;
- **`SortedSet subset (Object fromElement, Object toElement)`** — Возвращает подмножество коллекции от элемента `fromElement` включительно до элемента `toElement` исключительно;
- **`SortedSet tailSet (Object fromElement)`** — возвращает последние, большие элементы коллекции от элемента `fromElement` включительно.



# Классы-коллекции

**Интерфейс Queue** (Описывает очередь. Элементы могут добавляться в очередь только с одного конца, а извлекаться с другого (аналогично очереди в магазине))

Специфические для очереди методы:

- **poll()** — возвращает первый элемент и удаляет его из очереди.
- **peek()** — возвращает первый элемент очереди, не удаляя его.
- **offer(Object obj)** — добавляет в конец очереди новый элемент и возвращает true, если вставка удалась.



# Классы-коллекции

**Интерфейс Map** (описывает коллекцию, состоящую из пар "ключ — значение". У каждого ключа только одно значение. Такую коллекцию часто называют еще *словарем* (dictionary) или *ассоциативным массивом* (associative array).

- Интерфейс Map содержит методы, работающие с ключами и значениями:
- **boolean containsKey (Object key)** — проверяет наличие ключа key;
- **boolean containsValue (Object value)** — проверяет наличие значения value;
- **Set entryset ()** — представляет коллекцию в виде множества, каждый элемент которого — пара из данного отображения, с которой можно работать методами вложенного интерфейса Map. Entry;
- **object get (object key)** — возвращает значение, отвечающее ключу key;
- **set keyset ()** — представляет ключи коллекции в виде множества;
- **Object put(Object key, Object value)** — добавляет пару "key— value", если такой пары не было, и заменяет значение ключа key, если такой ключ уже есть в коллекции;
- **void putAll (Map m)** — добавляет к коллекции все пары из отображения m;
- **collection values ()** — представляет все значения в виде коллекции.

В интерфейс Map вложен

Интерфейс **Map.Entry**, содержащий методы работы с отдельной парой.

- методы **getKey()** и **getValue()** позволяют получить ключ и значение пары;
- метод **setValue (object value)** меняет значение в данной паре.



# Классы-коллекции

**Интерфейс SortedMap** (описывает упорядоченную по ключам коллекцию Map. Сортировка производится либо в естественном порядке возрастания ключей, либо, в порядке, описываемом в интерфейсе Comparator. Элементы не нумеруются, но есть понятия большего и меньшего из двух элементов)

Эти понятия описываются следующими методами:

- **comparator comparator ()** — возвращает способ упорядочения коллекции;
- **object firstKey()** — возвращает первый, меньший элемент коллекции;
- **SortedMap headMap(Object toKey)** — возвращает начало коллекции до элемента с ключом toKey исключительно;
- **object lastKey()** — возвращает последний, больший ключ коллекции;
- **SortedMap subMap (Object fromKey, Object toKey)** — возвращает часть коллекции от элемента с ключом fromKey включительно до элемента с ключом toKey исключительно;
- **SortedMap tailMap (object fromKey)** — возвращает остаток коллекции от элемента fromKey включительно.



# Классы-коллекции

Вы можете создать свои коллекции, реализовав рассмотренные интерфейсы. Чтобы облегчить эту задачу, в Java API введены частичные реализации интерфейсов — **абстрактные классы-коллекции**:

- **AbstractCollection** реализует интерфейс Collection, но оставляет нереализованными методы iterator (), size ().
- **AbstractList** реализует интерфейс List, но оставляет нереализованным метод get(mt) и унаследованный метод size() Этот класс позволяет реализовать коллекцию прямым доступом к элементам, подобно массиву
- **AbstractSequentialList** реализует интерфейс List, но оставляет нереализованным метод listIterator(int index) и унаследованный метод size (). Данный класс позволяет реализовать коллекции с последовательным доступом к элементам с помощью итератора ListIterator
- **AbstractSet** реализует интерфейс Set, но оставляет нереализованными методы, унаследованные от AbstractCollection
- **AbstractMap** реализует интерфейс Map, но оставляет нереализованным метод entrySet (),



# Классы-коллекции

Вы можете создать свои коллекции, реализовав рассмотренные интерфейсы. Чтобы облегчить эту задачу, в Java API введены частичные реализации интерфейсов — **абстрактные классы-коллекции**:

- **AbstractCollection** реализует интерфейс Collection, но оставляет нереализованными методы iterator (), size ().
- **AbstractList** реализует интерфейс List, но оставляет нереализованным метод get(mt) и унаследованный метод size() Этот класс позволяет реализовать коллекцию прямым доступом к элементам, подобно массиву
- **AbstractSequentialList** реализует интерфейс List, но оставляет нереализованным метод listIterator(int index) и унаследованный метод size (). Данный класс позволяет реализовать коллекции с последовательным доступом к элементам с помощью итератора ListIterator
- **AbstractSet** реализует интерфейс Set, но оставляет нереализованными методы, унаследованные от AbstractCollection
- **AbstractMap** реализует интерфейс Map, но оставляет нереализованным метод entrySet (),



# Классы-коллекции

Полностью реализованные  
классы-коллекции:

- **Vector,**
- **Stack,**
- **Hashtable,**
- **Properties,**
- **ArrayList,**
- **LinkedList,**
- **HashSet,**
- **TreeSet,**
- **HashMap,**
- **TreeMap,**
- **WeakHashMap .**

Для работы с этими классами  
разработаны интерфейсы:



# Классы-коллекции

**Интерфейс Comparator** описывает два метода сравнения:

- **int compare (Object obj1, Object obj2)** — возвращает отрицательное число, если obj1 в каком-то смысле меньше obj2; нуль, если они считаются равными; положительное число, если obj1 больше obj2. метод сравнения обладает свойствами тождества, антисимметричности и транзитивности;
- **boolean equals (Object obj)** — сравнивает данный объект с объектом obj, возвращая true, если объекты совпадают в каком-либо смысле, заданном этим методом.



# Классы-коллекции

**Интерфейс Iterator** (описывает способ обхода всех элементов коллекции)

В каждой коллекции есть метод `iterator ()`, возвращающий реализацию интерфейса `iterator` для указанной коллекции.

В интерфейсе `iterator` описаны всего три метода:

- **`bool hasNext ()`** возвращает `true`, если обход еще не завершен;
- **`Object Next()`** делает текущим следующий элемент коллекции и возвращает его в виде объекта класса `Object`;
- **`Remove()`** удаляет текущий элемент коллекции.

Метод `remove ()` уже не относится к задаче обхода коллекции, но позволяет при просмотре коллекции удалять из нее ненужные элементы.



# Классы-коллекции

**Класс Vector** — набор упорядоченных элементов, к каждому из которых можно обратиться по индексу. По сути эта коллекция представляет собой обычный список. Особенность вектора в том, что помимо текущего размера, определяемого количеством элементов в нем и возвращаемого методом `size()`, он имеет емкость, возвращаемую методом `capacity()` — размер выделенной под элементы памяти, которая может быть больше текущего размера. Ее можно увеличить методом `ensureCapacity(int minCapacity)` или сравнить с размером вектора методом `trimToSize()`. Каждый раз, когда нужно добавить в полностью «заполненный» вектор новый элемент, емкость увеличивается с запасом.

Конструкторы:

- **Vector()** — создает пустой вектор с емкостью в 10 элементов;
- **Vector(int capacity)** — создает пустой вектор указанной емкости;
- **Vector(int capacity, int increment)** — создает пустой вектор указанной емкости, а также задает число, на которое будет увеличиваться емкость при необходимости;
- **Vector(Collection c)** — создает вектор и заполняет его элементами другой коллекции.