IT-Academy

Коллекции

Рассматриваемые вопросы

IT-Academy

- Интерфейс Collection
- Структуры данных
- Интерфейс List
- Класс ArrayList
- Интерфейс Set
- Класс HashSet
- Класс LinkedHashSet
- Интерфейс Comparable

Рассматриваемые вопросы

IT-Academy

- Интерфейс NavigableSet
- Интерфейс Queue и классы
- Интерфейс Iterator
- Интерфейс ListIterator
- Отображение Map
- Класс Collections
- Backed Collections
- Legacy Classes

Коллекции

IT-Academy

Коллекции (Collection Framework) – это хранилища, поддерживающие различные способы накопления и упорядочения объектов с целью обеспечения возможностей эффективного доступа к ним. Были добавлены в версии J2SE 1.2.

Collection framework в языке Java состоит из 3-х частей:

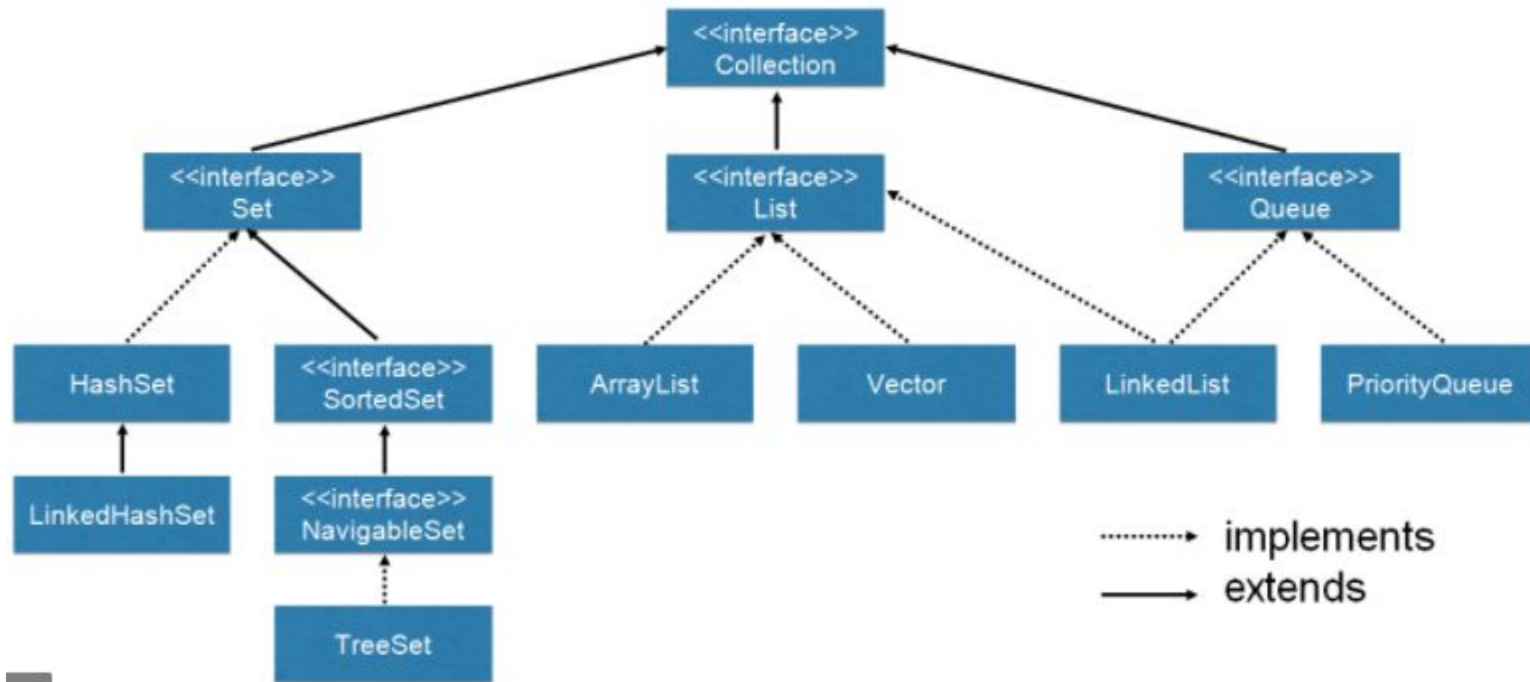
- интерфейсы,
- классы,
- алгоритмы.

В этом уроке рассмотрим основные элементы коллекций.

Интерфейс Collection

IT-Academy

Интерфейс **Collection** - вершина иерархии коллекций, который определяет наименьший набор методов, реализуемых всеми коллекциями.



Интерфейс Collection

Методы интерфейса Collection:

- **boolean add(E obj)** - добавляет obj к вызывающей коллекции. Возвращает true, если obj был добавлен к коллекции.
- **boolean addAll(Collection<? extends E> c)** - добавляет все элементы к вызывающей коллекции. Возвращает true, если операция удалась (то есть все элементы добавлены). В противном случае возвращает false.
- **void clear()** - удаляет все элементы вызывающей коллекции.
- **boolean contains(Object obj)** - возвращает true, если obj является элементом вызывающей коллекции. В противном случае возвращает false.
- **boolean containsAll(Collection<?> c)** - возвращает true, если вызывающая коллекция содержит все элементы c. В противном случае возвращает false.

Интерфейс Collection

IT-Academy

- **boolean equals(Object obj)** - возвращает true, если вызывающая коллекция и obj эквивалентны. В противном случае возвращает false.
- **int hashCode()** - возвращает хешкод вызывающей коллекции.
- **boolean isEmpty()** - возвращает true, если вызывающая коллекция пуста. В противном случае возвращает false.
- **Iterator<E> iterator()** - возвращает итератор для вызывающей коллекции.
- **boolean remove(Object obj)** - удаляет один экземпляр obj из вызывающей коллекции. Возвращает true, если элемент удален. В противном случае возвращает false.
- **boolean removeAll(Collection<?> c)** - удаляет все элементы из вызывающей коллекции. Возвращает true, если в результате коллекция изменяется (то есть элементы удалены). В противном случае возвращает false.

Интерфейс Collection

IT-Academy

- **boolean retainAll(Collection<?> c)** - удаляет все элементы кроме входящих из вызывающей коллекции. Возвращает true, если в результате коллекция изменяется (то есть элементы удалены). В противном случае возвращает false.
- **int size()** - возвращает количество элементов, содержащихся в коллекции.
- **Object[] toArray()** - возвращает массив, содержащий все элементы вызывающей коллекции. Элементы массива являются копиями элементов коллекции.
- **removeIf(Predicate<? super E> filter)** - удаляет элементы из коллекции, соответствующие заданному условию.

Структуры данных

IT-Academy

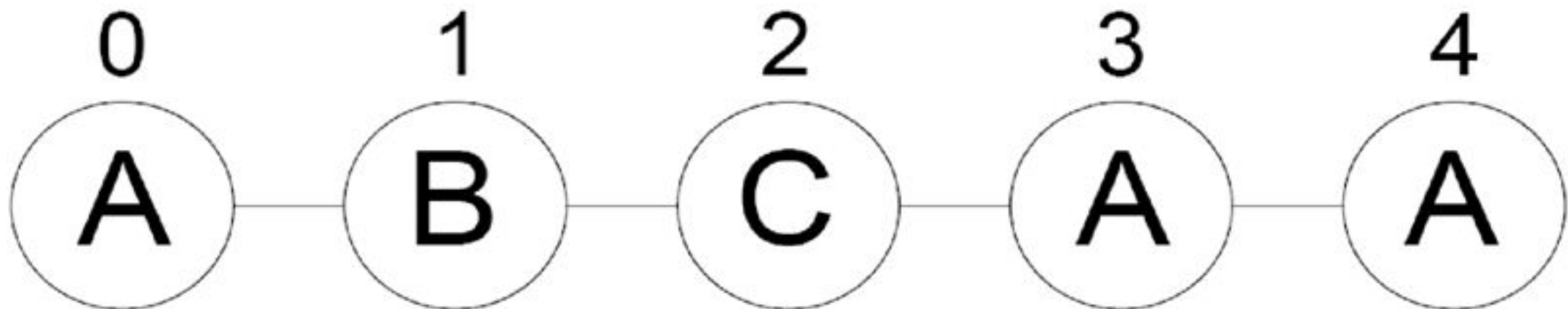
Основные структуры данных условно делятся на:

- **Список**
- **Стек**
- **Очередь**
- **Множество**

Список

IT-Academy

Список — упорядоченный набор элементов, для каждого из которых хранится указатель на следующий (или для двусвязного списка и на следующий и на предыдущий) элементы списка. Иногда называется sequence. Разрешаются повторы.



Стек

IT-Academy

Стек — это коллекция, элементы которой получают по принципу «последний вошел, первый вышел» (Last-In-First-Out или LIFO). Это значит, что мы будем иметь доступ только к последнему добавленному элементу.

Очередь

IT-Academy

Очереди очень похожи на стеки. Они также не дают доступа к произвольному элементу, но, в отличие от стека, элементы помещаются (enqueue) и забираются (dequeue) с разных концов. Такой метод называется «первый вошел, первый вышел» (First-In-First-Out или FIFO). То есть забирать элементы из очереди мы будем в том же порядке, что и помещали. Как реальная очередь или конвейер.

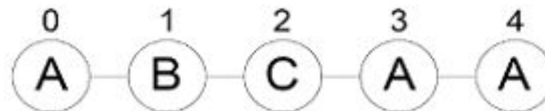
Множество

IT-Academy

Множество — неупорядоченный набор элементов, без повторов.

Интерфейс List

Интерфейс List сохраняет последовательность добавления элементов и позволяет осуществлять доступ к элементу по индексcv.



List добавляет следующие методы:

- **void add(int index, E obj)** - вставляет obj в вызывающий список в позицию, указанную в index. Любые ранее вставленные элементы за указанной позицией вставки смещаются вверх. То есть никакие элементы не перезаписываются.

Интерфейс List

- **boolean addAll (int index,Collection<? extends E> c)** - вставляет все элементы в вызывающий список, начиная с позиции, переданной в index. Все ранее существовавшие элементы за точкой вставки смещаются вверх. То есть никакие элементы не перезаписываются. Возвращает true, если вызывающий список изменяется, и false в противном случае.
- **E get (int index)** - возвращает объект, сохраненный в указанной позиции вызывающего списка.
- **int indexOf(Object obj)** - возвращает индекс первого экземпляра obj в вызывающем списке. Если obj не содержится в списке, возвращается 1.
- **int lastIndexOf(Object obj)** - возвращает индекс последнего экземпляра obj в вызывающем списке. Если obj не содержится в списке, возвращается 1.
- **ListIterator listIterator()** - возвращает итератор, указывающий на начало списка.

Интерфейс List

- **ListIterator listIterator(int index)** - возвращает итератор, указывающий на заданную позицию в списке.
- **E remove(int index)** - удаляет элемент из вызывающего списка в позиции index и возвращает удаленный элемент. Результирующий список уплотняется, то есть элементы, следующие за удаленным, сдвигаются на одну позицию назад.
- **E set (int index, E obj)** - присваивает obj элементу, находящемуся в списке в позиции index.
- **default void sort(Comparator<? super E> c)** - сортирует список, используя заданный компаратор (добавлен в версии JDK 8).
- **List subList (int start, int end)** - возвращает список, включающий элементы от start до end-1 из вызывающего списка. Элементы из возвращаемого списка также сохраняют ссылки в вызывающем списке.

ArrayList

IT-Academy

Класс **ArrayList** поддерживает динамические массивы, которые могут расти по мере необходимости. Элементы **ArrayList** могут быть абсолютно любых типов, в том числе и null. Элементы могут повторяться.

Класс **ArrayList** реализует интерфейс **List**.

Объект класса ArrayList, содержит свойства elementData и size. Хранилище значений elementData есть не что иное, как массив определенного типа (указанного в generic).

У этого класса есть следующие конструкторы:

- **ArrayList ()**
- **ArrayList(Collection <? extends E> c)**
- **ArrayList(int capacity)**

ArrayList. Плюсы и минусы

Достоинства класса ArrayList:

- Быстрый доступ по индексу.
- Быстрая вставка и удаление элементов с конца.

Недостатки класса ArrayList:

- Медленная вставка и удаление элементов в середину.

Методы класса ArrayList для добавления элементов

- **boolean add(E obj)** - добавляет obj к вызывающей коллекции. Возвращает true, если obj был добавлен к коллекции. (Интерфейс Collection)
- **void add(int index, E obj)** - вставляет obj в вызывающий список в позицию, указанную в index. Любые ранее вставленные элементы за указанной позицией вставки смещаются вверх. То есть никакие элементы не перезаписываются. (Интерфейс List)
- **E set (int index, E obj)** - присваивает obj элементу, находящемуся в списке в позиции index. (Интерфейс List)
- **boolean addAll (Collection<? extends E> c)** - добавляет все элементы к вызывающей коллекции. Возвращает true, если операция удалась (то есть все элементы добавлены). В противном случае возвращает false. (Интерфейс Collection)

ArrayListAddDemo, ArrayListDemo2

Методы класса ArrayList для удаления элементов

- **boolean remove(Object obj)** - удаляет один экземпляр obj из вызывающей коллекции. Возвращает true, если элемент удален. В противном случае возвращает false. (Интерфейс Collection)
- **E remove(int index)** - удаляет элемент из вызывающего списка в позиции index и возвращает удаленный элемент. Результирующий список уплотняется, то есть элементы, следующие за удаленным, сдвигаются на одну позицию назад. (Интерфейс List)
- **boolean removeAll(Collection<?> c)** - удаляет все элементы из вызывающей коллекции. Возвращает true, если в результате коллекция изменяется (то есть элементы удалены). В противном случае возвращает false. (Интерфейс Collection)
- **boolean retainAll(Collection<?> c)** - удаляет все элементы кроме входящих из вызывающей коллекции. Возвращает true, если в результате коллекция изменяется (то есть элементы удалены). В противном случае возвращает false. (Интерфейс Collection)
- **void clear()** - удаляет все элементы вызывающей коллекции. (Интерфейс Collection)

ArrayListRemoveDemo, ArrayListRemoveAllDemo, ArrayListRetainAllDemo

Получение массива из ArrayList

Имеются два варианта метода toArray(), которые объявлены в Collection:

- **Object [] toArray()**
- **<T> T [] toArray(T массив[])**

ArrayListToStringDemo

LinkedList

IT-Academy

Класс **LinkedList** реализует интерфейсы **List**, **Deque**.

LinkedList – это двусвязный список.

Плюсы

- Быстрое добавление и удаление элементов

Минусы

- Медленный доступ по индексу

Конструкторы класса LinkedList:

- **LinkedList()**
- **LinkedList(Collection<? extends E> c)**

Рекомендуется использовать, если необходимо часто добавлять элементы в начало списка или удалять внутренний элемент списка.

LinkedList

Внутри класса LinkedList существует static inner класс Entry, с помощью которого создаются новые элементы:

```
private static class Entry<E> {
        E element;
        Entry<E> next;
        Entry<E> prev;
        Entry(E element, Entry<E> next, Entry<E> prev) {
              this.element = element; this.next = next; this.prev = prev;
        }
}
```

Из **LinkedList** можно организовать стек, очередь, или двойную очередь, со временем доступа O(1). На вставку и удаление из середины списка, получение элемента по индексу или значению потребуется линейное время O(n). Однако, на добавление и удаление из середины списка, используя **ListIterator.add()** и **ListIterator.remove()**, потребуется O(1). Позволяет добавлять любые значения, в том числе и null.

LinkedListDemo

Интерфейс Set

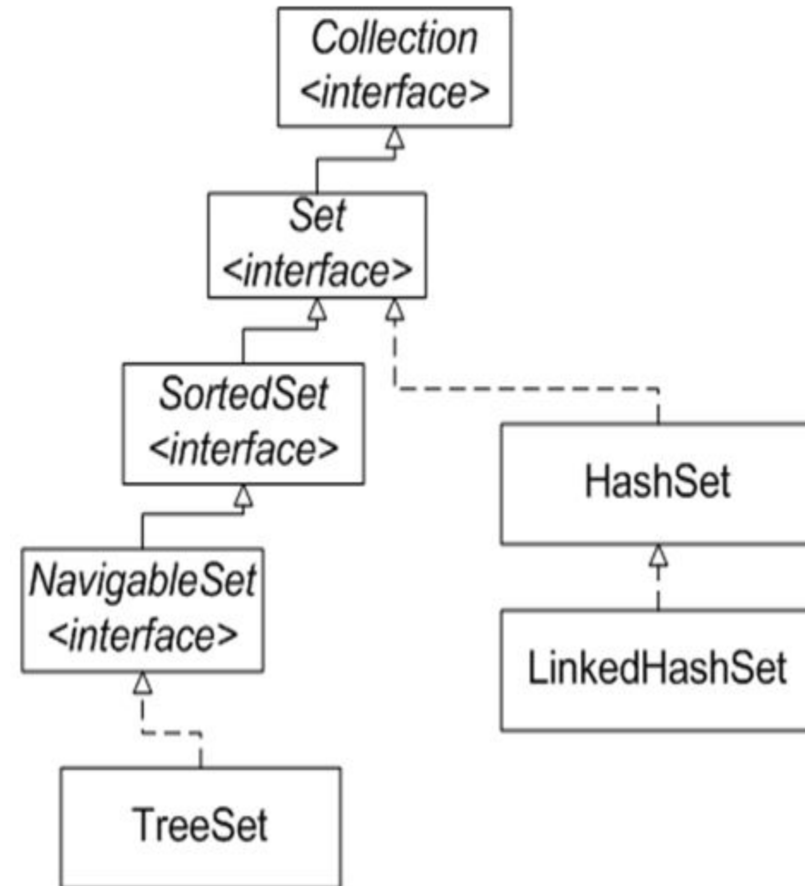
IT-Academy

Интерфейс **Set** определяет множество (набор).

Он расширяет **Collection** и определяет поведение коллекций, не допускающих дублирования элементов. Таким образом, метод **add()** возвращает false, если делается попытка добавить дублированный элемент в набор.

Он не определяет никаких собственных дополнительных методов.

Интерфейс Set заботится об уникальности хранимых объектов, уникальность определятся реализацией метода **equals**().



Класс HashSet

Класс **HashSet** реализует интерфейс **Set** и создает коллекцию, которая используется для хранения хеш-таблиц.

Элементы хеш-таблицы хранятся в виде пар ключ-значение. Ключ определяет ячейку для хранения значения. Содержимое ключа служит для определения однозначного значения, называемого хеш-кодом.

Мы можем считать, что хеш-код это ID объекта, хотя он не должен быть уникальным. Этот хеш-код служит далее в качестве индекса, по которому сохраняются данные, свя

Рассмотрим пример вычислени

я хеш-кодов:

Ключ	Алгоритм хеш-кода	Хеш-код
Alex	A(1) + L(12) + E(5) + X(24)	42
Bob	B(2) + O(15) + B(2)	19
Dirk	D(4) + I(9) + R(18) + K(11)	42
Fred	F(6) + R(18) + E(5) + D(4)	33

Правила написания методов hashCode() и equals()

1. Для одного и того же объекта, хеш-код всегда будет одинаковым.
2. Если объекты одинаковые, то и хеш-коды одинаковые (но не наоборот).
3. Если хеш-коды равны, то входные объекты не всегда равны.
4. Если хеш-коды разные, то и объекты гарантированно будут разные.

Выгода от хеширования состоит в том, что оно обеспечивает постоянство время выполнения операций **add**(), **contains**(), **remove**() и **size**(), даже для больших наборов.

Класс HashSet не гарантирует упорядоченности элементов, поскольку процесс хеширования сам по себе обычно не приводит к созданию отсортированных множеств. HashSetDemo

Класс LinkedHashSet

Класс LinkedHashSet языка Java расширяет HashSet, не добавляя никаких новых методов.

LinkedHashSet поддерживает связный список элементов набора в том порядке, в котором они вставлялись. Это позволяет организовать упорядоченную итерацию вставки в набор.

Работает дольше чем класс HashSet.

`LinkedHashSetDemo`

Интерфейс SortedSet

Интерфейс SortedSet языка Java, расширяющий интерфейс Set, описывает упорядоченное множество, отсортированное в возрастающем порядке или по порядку, заданному реализацией интерфейса Comparator.

Методы интерфейса SortedSet:

- **Comparator<? super E> comparator()** - возвращает компаратор сортированного множества. Если для множества применяется естественный порядок сортировки, возвращается null.
- **E first()** - возвращает первый элемент вызывающего сортированного множества.
- **E last()** - возвращает последний элемент вызывающего сортированного множества.

Методы SortedSet

- **SortedSet headSet(E toElement)** - возвращает SortedSet, содержащий элементы из вызывающего множества, которые предшествуют end.
- **SortedSet subSet(E fromElement, E toElement)** - возвращает SortedSet, содержащий элементы из вызывающего множества, находящиеся между start и end-1.
- **SortedSet tailSet(E fromElement)** - возвращает SortedSet, содержащий элементы из вызывающего множества, которые следуют за end.

TreeSetDemo2

Класс TreeSet

Класс **TreeSet** реализует интерфейс **NavigableSet**, который поддерживает элементы в отсортированном по возрастанию порядке. Объекты сохраняются в отсортированном порядке по возрастанию.

Обработка операций удаления и вставки объектов происходит медленнее, чем в хэш-множествах, но быстрее, чем в списках.

Конструкторы класса **TreeSet**:

- **TreeSet()**
- **TreeSet(Collection<? extends E> collection)**
- **TreeSet(Comparator<? super E> comparator)**
- **TreeSet(SortedSet<E> sortedSet)**

TreeSetDemo1

Интерфейс Comparable

IT-Academy

Существует два способа сравнения объектов:

- С помощью интерфейса Comparable.
- С помощью интерфейса Comparator.

Для того чтобы объекты можно было сравнить и сортировать, они должны реализовать параметризованный интерфейс **Comparable**. Интерфейс **Comparable** содержит один единственный метод **int compareTo(T item),** который сравнивает текущий объект с объектом, переданным в качестве параметра.

Если этот метод возвращает отрицательное число, то текущий объект будет располагаться перед тем, который передается через параметр. Если метод вернет положительное число, то, наоборот, после второго объекта. Если метод возвращает ноль, значит, оба объекта равны.

Person, ComparePersonDemo

Интерфейс Comparator

Если класс по какой-то причине не может реализовать интерфейс **Comparable**, или же просто нужен другой вариант сравнения, используется интерфейс **Comparator**.

Интерфейс содержит метод **int compare(T o1, T o2),** который должен быть реализован классом, реализующим компаратор.

Метод **compare()** возвращает числовое значение - если оно отрицательное, то объект o1 предшествует объекту o2, иначе - наоборот. А если метод возвращает ноль, то объекты равны.

Для применения интерфейса нам вначале надо создать класс компаратора, который реализует этот параметризованный интерфейс.

PersonComparator, PersonComparatorDemo, PersonComparingDemo

Интерфейс NavigableSet

IT-Academy

Интерфейс **NavigableSet** появился в Java SE 6. Он расширяет **SortedSet** и добавляет методы для более удобного поиска по коллекции:

- **E ceiling(E obj)** - ищет в наборе наименьший элемент e, для которого истинно e >= obj. Если такой элемент найден, он возвращается. В противном случае возвращается null.
- **E floor(E obj)** - ищет в наборе наибольший элемент e, для которого истинно e <= obj. Если такой элемент найден, он возвращается. В противном случае возвращается null.
- **E higher(E obj)** - ищет в наборе наибольший элемент e, для которого истинно e > obj. Если такой элемент найден, он возвращается. В противном случае возвращается null.
- **E lower(E obj)** - ищет в наборе наименьший элемент e, для которого истинно e < obj. Если такой элемент найден, он возвращается. В противном случае возвращается null.

Интерфейс NavigableSet

IT-Academy

- **NavigableSet headSet(E upperBound, boolean incl)** - возвращает NavigableSet, включающий все элементы вызывающего набора, меньшие upperBound. Результирующий набор поддерживается вызывающим набором.
- **NavigableSet subSet(E lowerBound, boolean lowIncl, E upperBound, boolean highIncl)** - возвращает NavigableSet, включающий все элементы вызывающего набора, которые больше lowerBound и меньше upperBound. Если lowIncl равно true, то элемент, равный lowerBound, включается. Если highIncl равно true, также включается элемент, равный upperBound.
- **E pollLast()** - возвращает последний элемент, удаляя его в процессе. Поскольку набор сортирован, это будет элемент с наибольшим значением. Возвращает null в случае пустого набора.

Интерфейс NavigableSet

IT-Academy

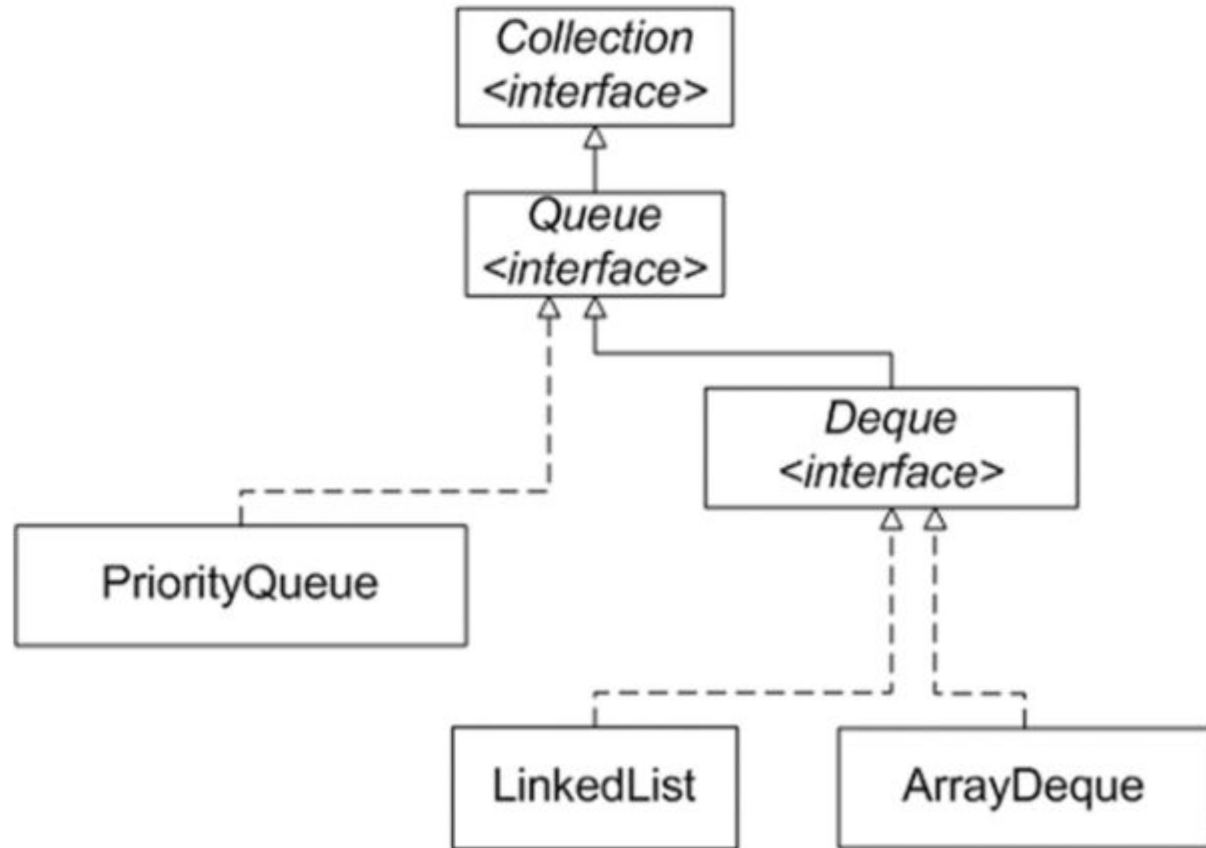
- **E pollFirst()** - возвращает первый элемент, удаляя его в процессе. Поскольку набор сортирован, это будет элемент с наименьшим значением. Возвращает null в случае пустого набора.
- **Iterator descendingIterator()** - возвращает итератор, перемещающийся от большего к меньшему, другими словами, обратный итератор.
- **NavigableSet descendingSet()** - возвращает NavigableSet, представляющий собой обратную версию вызывающего набора. Результирующий набор поддерживается вызывающим набором.

Класс, реализующие интерфейс **NavigableSet** - это **TreeSet**.

Ferry

Интерфейс Queue

Интерфейс **Queue** расширяет **Collection** и объявляет поведение очередей, которые представляют собой список с дисциплиной "первый вошел, первый вышел" (**FIFO**). Существуют разные типы очередей, в которых порядок основан на некотором критерии. Очереди не могут хранить значения **null**.



Методы интерфейса Queue

- **E element()** - возвращает элемент из головы очереди. Элемент не удаляется. Если очередь пуста, инициируется исключение NoSuchElementException.
- **E remove()** - удаляет элемент из головы очереди, возвращая его. Инициирует исключение NoSuchElementException, если очередь пуста.
- **E peek()** - возвращает элемент из головы очереди. Возвращает null, если очередь пуста. Элемент не удаляется.
- **E poll()** - возвращает элемент из головы очереди и удаляет его. Возвращает null, если очередь пуста.
- **boolean offer(E obj)** - пытается добавить obj в очередь. Возвращает true, если obj добавлен, и false в противном случае.

QueueExample

Интерфейс Deque

IT-Academy

Интерфейс **Deque** появился в Java 6. Он расширяет **Queue** и описывает поведение двунаправленной очереди. Двунаправленная очередь может функционировать как стандартная очередь **FIFO** либо как стек **LIFO**.

Методы интерфейса Deque:

- **void addFirst(E obj)** - добавляет obj в голову двунаправленной очереди. Возбуждает исключение IllegalStateException, если в очереди ограниченной емкости нет места.
- **void addLast(E obj)** - добавляет obj в хвост двунаправленной очереди. Возбуждает исключение IllegalStateException, если в очереди ограниченной емкости нет места.
- **E getFirst()** - возвращает первый элемент двунаправленной очереди. Объект из очереди не удаляется. В случае пустой двунаправленной очереди возбуждает исключение NoSuchElementException.

Интерфейс Deque

- **E getLast()** - возвращает последний элемент двунаправленной очереди. Объект из очереди не удаляется. В случае пустой двунаправленной очереди возбуждает исключения NoSuchElementException.
- **boolean offerFirst(E obj)** - пытается добавить obj в голову двунаправленной очереди. Возвращает true, если obj добавлен, и false в противном случае. Таким образом, этот метод возвращает false при попытке добавить obj в полную двунаправленную очередь ограниченной емкости.
- **boolean offerLast(E obj)** - пытается добавить obj в хвост двунаправленной очереди. Возвращает true, если obj добавлен, и false в против ном случае.
- **E pop()** - возвращает элемент, находящийся в голове двунаправленной очереди, одновременно удаляя его из очереди. Возбуждает исключение NoSuchElementException, если очередь пуста.

Интерфейс Deque

IT-Academy

- **void push(E obj)** - добавляет элемент в голову двунаправленной очереди. Если в очереди фиксированного объема нет места, возбуждает исключение IllegalStateException.
- **E peekFirst()** - возвращает элемент, находящийся в голове двунаправленной очереди. Возвращает null, если очередь пуста. Объект из очереди не удаляется.
- **E peekLast()** - возвращает элемент, находящийся в хвосте двунаправленной очереди. Возвращает null, если очередь пуста. Объект из очереди не удаляется.
- **E pollFirst()** - возвращает элемент, находящийся в голове двунаправленной очереди, одновременно удаляя его из очереди. Возвращает null, если очередь пуста.
- **E pollLast()** - возвращает элемент, находящийся в хвосте двунаправленной очереди, одновременно удаляя его из очереди. Возвращает null, если очередь пуста.

Интерфейс Deque

IT-Academy

- **E removeLast()** - возвращает элемент, находящийся в конце двунаправленной очереди, удаляя его в процессе. Возбуждает исключение NoSuchElementException, если очередь пуста.
- **E removeFirst()** - возвращает элемент, находящийся в голове двунаправленной очереди, одновременно удаляя его из очереди. Возбуждает исключение NoSuchElementException, если очередь пуста.
- **boolean removeLastOccurrence(Object obj)** - удаляет последнее вхождение obj из двунаправленной очереди. Возвращает true в случае успеха и false если очередь не содержала obj.
- **boolean removeFirstOccurrence(Object obj)** - удаляет первое вхождение obj из двунаправленной очереди. Возвращает true в случае успеха и false, если очередь не содержала obj.

Класс ArrayDeque

ArrayDeque создает двунаправленную очередь.

Конструкторы класса ArrayDeque:

- **ArrayDeque()** - создает пустую двунаправленную очередь с вместимостью 16 элементов.
- **ArrayDeque(Collection<? extends E> c)** - создает двунаправленную очередь из элементов коллекции c в том порядке, в котором они возвращаются итератором коллекции c.
- **ArrayDeque(int numElements)** - создает пустую двунаправленную очередь с вместимостью numElements.

ArrayDequeExample

Класс PriorityQueue

PriorityQueue – это класс очереди с приоритетами. По умолчанию очередь с приоритетами размещает элементы согласно естественному порядку сортировки используя **Comparable**. Элементу с наименьшим значением присваивается наибольший приоритет. Если несколько элементов имеют одинаковый наивысший элемент – связь определяется произвольно. Также можно указать специальный порядок размещения, используя **Comparator**. Конструкторы класса PriorityQueue:

- **PriorityQueue()** - создает очередь с приоритетами начальной емкостью 11, размещающую элементы согласно естественному порядку сортировки (Comparable);
- **PriorityQueue(Collection<? extends E> c);**
- **PriorityQueue(int initialCapacity);**
- **PriorityQueue(int initialCapacity, Comparator<? super E> comparator);**
- **PriorityQueue(PriorityQueue<? extends E> c);**
- **PriorityQueue(SortedSet<? extends E> c).** PriorityQueueExample

Интерфейс Iterator

Перебор содержимого коллекции может быть осуществлен двумя способами: с помощью цикла for each и с помощью итератора.

Итератор позволяет осуществлять обход коллекции и при желании удалять избранные элементы. Используется интерфейс **Iterator**.

Чтобы получить объект итератора, вызовите метод **Iterator<E> iterator().**

Методы интерфейса **Iterator**:

- **boolean hasNext()** - возвращает true, если есть еще элементы. В противном случае возвращает false.
- **E next()** - возвращает следующий элемент. Если следующий элемент коллекции отсутствует, то метод next() генерирует исключение NoSuchElementException.
- **void remove()** - удаляет текущий элемент. Выкидываем исключение IllegalStateException, если предпринимается попытка вызвать remove(), которой не предшествовал вызов next(). IteratorDemo

Интерфейс ListIterator

IT-Academy

Интерфейс **ListIterator** расширяет интерфейс **Iterator** и используется для двустороннего обхода списка и видоизменения его элементов.

ListIterator можно получить вызывая метод **listIterator**() для коллекций, реализующих **List**.

Методы интерфейса ListIterator:

- **void add(E obj)** - вставляет obj перед элементом, который должен быть возвращен следующим вызовом next().
- **boolean hasNext()** - возвращает true, если есть следующий элемент. В противном случае возвращает false.
- **boolean hasPrevious()** - возвращает true, если есть предыдущий элемент. В противном случае возвращает false.
- **E next()** - возвращает следующий элемент. Если следующего нет, инициируется исключение NoSuchElementException.
- **int nextIndex()** - возвращает индекс следующего элемента. Если следующего нет, возвращается размер списка.

Что такое Map

Map представляет собой объект, сохраняющий связи между ключами и значениями в виде пар "ключ-значение". По заданному ключу можно найти его значение.

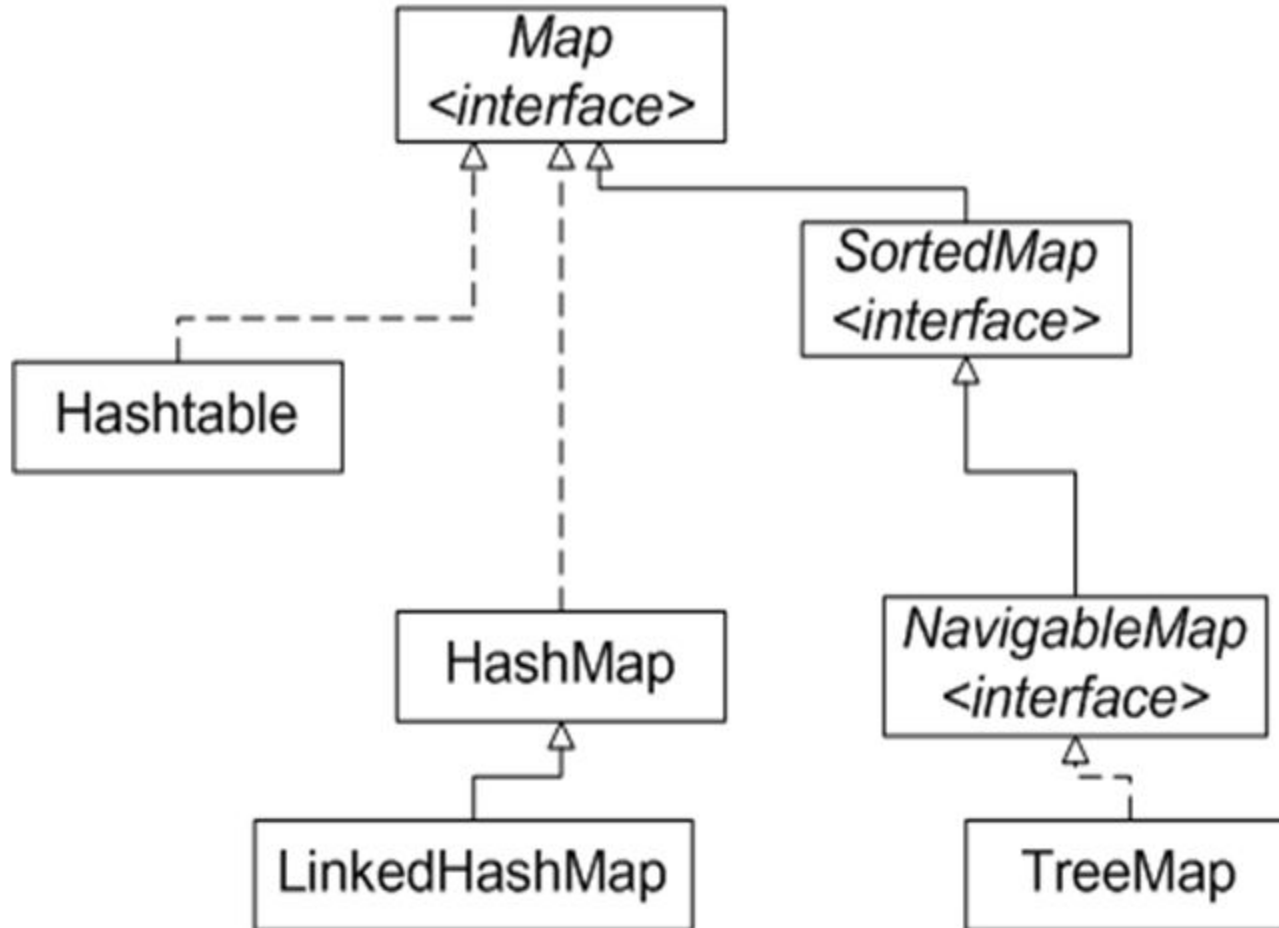
Ключи и значения являются объектами. Ключи должны быть уникальными, а значения могут быть дублированными.

В одних отображениях допускаются null ключи и null значения, а в других - они не допускаются.

Уникальность ключей определяет реализация метода **equals**().

Для корректной работы с картами необходимо переопределить методы **equals**() и **hashCode**(). Допускается добавление объектов без переопределения этих методов, но найти эти объекты в Map вы не сможете.

Иерархия Map



Интерфейс Map

Интерфейс **Map** отображает уникальные ключи на значения.

Ключ это объект, который вы используете для последующего извлечения данных. Задавая ключ и значение, вы можете помещать значения в объект **Map**. После того как это значение сохранено, вы можете получить его по ключу.

Map обобщенный интерфейс

```
interface Map<K, V>
```

Здесь К указывает тип ключей, а V тип хранимых значений.

Методы интерфейса Map

IT-Academy

- **void clear()** - удаляет все пары "ключ-значение" из вызывающей карты.
- **boolean containsKey(Object k)** - возвращает true, если вызывающая карта содержит ключ k. В противном случае возвращает false.
- **boolean containsValue(Object v)** - возвращает true, если вызывающая карта содержит значение v. В противном случае возвращает false.
- **Set<Map. Entry<K, V> entrySet()** - возвращает Set, содержащий все значения карты. Набор содержит объекты типа Map.Entry. То есть этот метод представляет карту в виде набора.
- **V get(Object K)** - возвращает значение, ассоциированное с ключом k. Возвращает null, если ключ не найден.
- **boolean isEmpty()** - возвращает true, если вызывающая карта пуста. В противном случае возвращает false.
- **Set<K> keySet()** - возвращает Set, который содержит ключи вызывающей карты. Этот метод представляет ключи вызывающей карты в виде набора.

Методы интерфейса Map

IT-Academy

- **V put(К k, V v)** - помещает элемент в вызывающую карту, перезаписывая любое предшествующее значение, ассоциированное с ключом. Ключ и значение это k и v соответственно. Возвращает null, если ключ ранее не существовал. В противном случае возвращается предыдущее значение, связанное с ключом.
- **void putAll(Map<? extends К, ? extends V> m)** - помещает все значения из m в карту.
- **V remove(Object k)** - удаляет элемент, чей ключ равен k.
- **int size()** - возвращает число пар "ключ-значение" в карте.
- **Collection<V> values()** - возвращает коллекцию, содержащую значения карты. Этот метод представляет значения, содержащихся в карте, в виде коллекции.

Интерфейс SortedMap

Интерфейс **SortedMap** расширяет **Map**. Он гарантирует, что элементы размещаются в возрастающем порядке значений ключей.

Методы интерфейса **SortedMap**:

- Comparator<? super К> comparator() - возвращает компаратор вызывающей сортированной Map. Если в Map используется естественный порядок, возвращается null.
- К firstKey() - возвращает первый ключ вызывающей Map.
- К lastKey() - возвращает последний ключ вызывающей Map.
- SortedMap<K, V> headMap(К end) - Возвращает сортированную Map, содержащую те элементы вызывающей Map, ключ которых меньше end.
- SortedMap<K, V> subMap(К start, К end) - возвращает Map, содержащую элементы вызывающей Map, чей ключ больше или равен start и меньше end.
- SortedMap<K, V> tailMap (К start) - возвращает сортированный Map, содержащую те элементы вызывающей Map, ключ которых больше start.

Интерфейс NavigableMap

Интерфейс **NavigableMap** был добавлен в Java 6. Она расширяет **SortedMap** и определяет поведение карты, поддерживающей извлечение элементов на основе ближайшего соответствия заданному ключу или ключам.

Методы интерфейса NavigableMap:

Методы позволяют получить соответственно меньший, меньше или равный, больший, больше или равную пару “ключ-значение” по отношению к заданному:

- **Map.Entry<K,V> lowerEntry(K key)**
- **Map.Entry<K,V> floorEntry(K key)**
- **Map.Entry<K,V> higherEntry(K key)**
- **Map.Entry<K,V> ceilingEntry(K key)**

Возвращает Map, отсортированную в обратном порядке:

- **NavigableMap<K,V> descendingMap()**

Интерфейс NavigableMap

Методы позволяют получить соответственно меньший, меньше или равный, больший, больше или равный ключ по отношению к заданному.

- **K lowerKey(K key)**
- **K floorKey(K key)**
- **K higherKey(K key)**
- **K ceilingKey(K key)**

Методы **pollFirstEntry** и **pollLastEntry** возвращают соответственно первый и последний элементы карты, удаляя их из коллекции. Методы **firstEntry** и **lastEntry** также возвращают соответствующие элементы, но без удаления:

- **Map.Entry<K,V> pollFirstEntry()**
- **Map.Entry<K,V> pollLastEntry()**
- **Map.Entry<K,V> firstEntry()**
- **Map.Entry<K,V> lastEntry()**

Интерфейс NavigableMap

Методы, позволяющие получить набор ключей, отсортированных в прямом и обратном порядке соответственно:

- **NavigableSet<K> navigableKeySet()**
- **NavigableSet<K> descendingKeySet()**

Методы, позволяющие извлечь из Map подмножество. Указываем в параметрах начальный и конечный элементы массива ключей, а также необходимость включения в выходной набор граничных элементов. Если не указывать флаги, то будет использован интервал ключевых значений со включённым начальным элементом, но с исключённым конечным элементом:

- **NavigableMap<K,V> subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)**
- **NavigableMap<K,V> headMap(K toKey, boolean inclusive)**
- **NavigableMap<K,V> tailMap(K fromKey, boolean inclusive)**
- **SortedMap<K,V> subMap(K fromKey, K toKey)**
- **SortedMap<K,V> headMap(K toKey)**
- **SortedMap<K,V> tailMap(K fromKey)**

Интерфейс Map.Entry

Интерфейс **Map.Entry** позволяет работать с элементом карты. Метод **entrySet**(), объявленный в интерфейсе Map, возвращает Set, содержащий элементы карты. Каждый из элементов этого набора представляет собой объект типа **Map.Entry**.

Класс HashMap

Класс **HashMap** реализует интерфейс **Map**. Он использует хеш-таблицу для хранения карты. Это позволяет обеспечить константное время выполнения методов **get**() и **put**() даже при больших наборах.

Ключи и значения могут быть любых типов, в том числе и null.

HashMap обобщенный класс со следующим объявлением:

```
class HashMap<K, V>
```

HashMapDemo

Класс TreeMap

TreeMap – хранит элементы в порядке сортировки. **TreeMap** сортирует элементы по возрастанию от первого к последнему. Порядок сортировки может задаваться реализацией интерфейсов Comparator и Comparable. Реализация Comparator передается в конструктор TreeMap, Comparable используется при добавлении элемента в Map.

Конструкторы класса TreeMap:

- **TreeMap()**
- **TreeMap(Comparator<? super K> comp)**
- **TreeMap(Map<? extends K, ? extends V> m)**
- **TreeMap(SortedMap<K, ? extends V> sm)**

TreeMapDemo

Класс LinkedHashMap

Класс **LinkedHashMap** расширяет **HashMap**. Он создает связный список элементов в карте, расположенных в том порядке, в котором они вставлялись. Это позволяет организовать итерацию по карте в порядке вставки.

ProductKeyDemo

Класс TreeMap

IT-Academy

TreeMap – хранит элементы в порядке сортировки. **TreeMap** сортирует элементы по возрастанию от первого к последнему. Порядок сортировки может задаваться реализацией интерфейсов Comparator и Comparable. Реализация Comparator передается в конструктор TreeMap, Comparable используется при добавлении элемента в Map.

Конструкторы класса TreeMap:

- **TreeMap()**
- **TreeMap(Comparator<? super K> comp)**
- **TreeMap(Map<? extends K, ? extends V> m)**
- **TreeMap(SortedMap<K, ? extends V> sm)**

TreeMapDemo

Класс TreeMap

TreeMap – хранит элементы в порядке сортировки. **TreeMap** сортирует элементы по возрастанию от первого к последнему. Порядок сортировки может задаваться реализацией интерфейсов Comparator и Comparable. Реализация Comparator передается в конструктор TreeMap, Comparable используется при добавлении элемента в Map.

Конструкторы класса TreeMap:

- **TreeMap()**
- **TreeMap(Comparator<? super K> сотр)**
- **TreeMap(Map<? extends K, ? extends V> т)**
- **TreeMap(SortedMap<K, ? extends V> sm)**

TreeMapDemo

Вопросы

IT-Academy

IT-Academy

Спасибо за внимание