Коллекции

**Коллекции** – это динамическое хранилище для однородных элементов. Оно бывает разных типов.

Иерархия коллекций:

**Iterable** – это самый интерфейс который наследуют все остальные коллекции кроме map.

Позволяет перебирать коллекцию по принципу for-each

**Iterator** – это интерфейс реализация которого служит для переборки коллекции.

**Методы:**

**next()** – Позволяет получить следующий элемент коллекции.

**hasNext()** – Возвращает true, если следующий элемент коллекции есть.

**remove()** – Удаляет текущий элемент.

**Collection** – это интерфейс который расширяет Iterable. Этот интерфейс содержит основные методы для работы с коллекциями. Все коллекции кроме Map являются эго наследниками и получают его методы.

**Методы:**

**add(E item)** – Добавление элемента в коллекцию. При удачном добавлении возвращает true, при неудачном – false.

**addAll(Collection<?> c)** – Добавляет все элементы указанной коллекции в эту коллекцию.

**clear()** – Удаляет все элементы из коллекции.

**contains(Object item)** – возвращает true, если объект содержится в коллекции, иначе возвращает false.

**containsAll(Collection<?> c)** – возвращает true, если указанная коллекция содержится в этой коллекции, иначе возвращает false.

**retainAll(Collection<?>c)** – удаление из этой коллекции всех элементов переданной коллекции.

**isEmpty()** – возвращает true, если коллекция пуста, иначе возвращает false.

**remove(Object item)** – позволяет удалить элемент из коллекции.

**size()** – возвращает число элементов в коллекции.

**toArray()** – возвращает обычный массив, содержащий все элементы коллекции.

**List** – это интерфейс который расширяет collection. Он формирует список, в котором все элементы упорядочены, а обращается к ним можно по индексу. В таких коллекциях можно хранить одинаковые элементы.

Помимо iterator данный интерфейс может возвратить listerator, который позволяет совершать вставку и замену элементов, а также двунаправленный доступ.

**Методы:**

**add(int index, object obj)** – вставляет элемент obj в позицию index. Старые элементы, начиная с позиции index, сдвигаются, их индексы увеличиваются на единицу.

**addAll(int index, Collection coll)** – вставляет все элементы коллекции coll.

**get(int index)** – возвращает элемент, находящийся на позиции index.

**indexOf(Object obj)** – возвращает индекс первого появления элемента obj в коллекции.

**lastindexOf(object obj)** – возвращает индекс последнего появления элемента obj в коллекции.

**listiterator()** – возвращает итератор коллекции.

**listiterator(int index)** – возвращает итератор коллекции начиная с элемента с позицией index.

**set(int index, object obj)** – заменяет элемент, находящийся на позиции index, элементом obj.

**subList(int from, int to)** – возвращает часть коллекции от позиции from включительно до позиции to исключительно.

**Set** – это интерфейс который расширяет collection. Порожденные от него коллекции могут хранить только не повторяющиеся элементы.

**Методы:**

**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()** – Удаление всех элементов из коллекции.

**contains(Object o)** – Проверка присутствия элемента в наборе. Возвращает true, если элемент найден.

**containsAll(Collection c)** – Проверка присутствия коллекции в наборе. Возвращает true, если все элементы содержатся в наборе.

**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 в коллекции.

**SortedSet** – расширяет set, хранит множество не одинаковых элементов отсортированных в порядке возрастания, или в порядке заданного реализацией интерфейса Comparator.

**Методы:**

**comparator()** – возвращает Компаратор, используемый для упорядочения элементов в этом множестве, или null, если множество использует естественный порядок сортировки.

**first()** - возвращает первый элемент множества.

**headSet(E toElement)** - возвращает часть множества, элементы которого строго меньше переданного элемента.

**last()** - возвращает последний элемент множества.

**spliterator()** - создает Spliterator (разделитель) над элементами в этом отсортированном множестве.

**subSet(E fromElement, E toElement)** - возвращает часть множества, элементы которой варьируются от fromElement включительно, до toElement исключительно.

**tailSet(E fromElement)** - возвращает часть множества, элементы которого больше или равны fromElement.

**NavigableSet** – это интерфейс который расширяет SortedSet и добавляет более удобные методы для поиска по коллекции.

**Методы:**

**ceiling(E e)** - возвращает наименьший элемент множества, который будет больше или равен указанному элементу, или null, если такого элемента нет.

**descendingIterator()** - возвращает итератор по элементам в этом множестве в порядке убывания.

**descendingSet() -** возвращает элементы множества в обратном порядке. Результирующий набор поддерживается вызывающим набором (это называется backed-collection).

**floor(E e)** - возвращает наибольший элемент в этом наборе, который будет меньше или равен указанному элементу, или null, если такого элемента нет.

**headSet(E toElement)** - возвращает часть множества, элементы которого строго меньше, чем toElement.

**headSet(E toElement, boolean inclusive)** - возвращает NavigableSet, включающий все элементы вызывающего набора, меньшие toElement. Результирующий набор поддерживается вызывающим набором (это называется backed-collection).

**higher(E e)** - возвращает наименьший элемент в этом множестве, строго больший, чем данный элемент, или null, если такого элемента нет.

**iterator()**- возвращает итератор по элементам в этом множестве в порядке возрастания.

**lower(E e)** - возвращает наибольший элемент в этом множестве строго меньше, чем данный элемент, или null, если такого элемента нет.

**pollFirst()**- получает и удаляет первый элемент или возвращает null, если это множество пустое.

**pollLast()**- получает и удаляет последний элемент или возвращает null, если это множество пустое.

**subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)** - возвращает NavigableSet, включающий все элементы вызывающего набора, которые больше fromElement и меньше toElement. Если fromInclusive равно true, то элемент, равный fromElement, включается. Если toInclusive равно true, также включается элемент, равный toInclusive.

**subSet(E fromElement, E toElement)** - возвращает представление части этого множества, элементы которого варьируются от fromElement включительно, до toElement исключительно.

**tailSet(E fromElement)** - возвращает представление части этого множества, элементы которого больше или равны fromElement.

**tailSet(E fromElement, boolean inclusive)** - возвращает представление части этого множества, элементы которого больше fromElement (или равны, если inclusive равно true). Результирующий набор поддерживается вызывающим набором (это называется backed-collection).

**Queue** – это интерфейс который расширяет collection. Интерфейс является однонаправленной очередью. Работает по принципу FIFO, то есть чем раньше элемент добавлен, тем раньше он будет удален.

**Методы:**

**element()** — возвращает, но не удаляет, элемент из начала очереди. Если очередь пуста, генерирует исключение NoSuchElementException.

**offer(E obj)** — добавляет элемент obj в конец очереди. Если элемент удачно добавлен, возвращает true, иначе – false.

**peek()** — возвращает без удаления элемент из начала очереди. Если очередь пуста, возвращает значение null.

**poll()** — возвращает с удалением элемент из начала очереди. Если очередь пуста, возвращает значение null.

**remove()** — возвращает с удалением элемент из начала очереди. Если очередь пуста, генерирует исключение NoSuchElementException.

**Deque** – это интерфейс который расширяет queue. Интерфейс является двунаправленной очередью. Его можно использовать и как LIFO и как FOFI. То есть элементы можно добавлять и удалять и в конец, и в начало.

**Методы:**

**addFirst(E obj)** — добавляет элемент в начало очереди

**addLast(E obj)** — добавляет элемент obj в конец очереди

**getFirst()** — возвращает без удаления элемент из головы очереди. Если очередь пуста, генерирует исключение NoSuchElementException

**getLast()** — возвращает без удаления последний элемент очереди. Если очередь пуста, генерирует исключение NoSuchElementException

**offerFirst(E obj)** — добавляет элемент obj в самое начало очереди. Если элемент удачно добавлен, возвращает true, иначе - false

**offerLast(E obj)** — добавляет элемент obj в конец очереди. Если элемент удачно добавлен, возвращает true, иначе - false

**peekFirst()** — возвращает без удаления элемент из начала очереди. Если очередь пуста, возвращает значение null

**peekLast()** — возвращает без удаления последний элемент очереди. Если очередь пуста, возвращает значение null

**pollFirst()** — возвращает с удалением элемент из начала очереди. Если очередь пуста, возвращает значение null

**pollLast()** — возвращает с удалением последний элемент очереди. Если очередь пуста, возвращает значение null

**pop()** — возвращает с удалением элемент из начала очереди. Если очередь пуста, генерирует исключение NoSuchElementException

**push(E element)** — добавляет элемент в самое начало очереди

**removeFirst()** — возвращает с удалением элемент из начала очереди. Если очередь пуста, генерирует исключение NoSuchElementException

**removeLast()** — возвращает с удалением элемент из конца очереди. Если очередь пуста, генерирует исключение NoSuchElementException

**removeFirstOccurrence(Object obj)** — удаляет первый встреченный элемент obj из очереди. Если удаление произошло, то возвращает true, иначе возвращает false.

**removeLastOccurrence(Object obj)** — удаляет последний встреченный элемент obj из очереди. Если удаление произошло, то возвращает true, иначе возвращает false.

**Map** – интерфейс который ключами каждый элемент коллекции. Ключи должны быть уникальны.

**Методы:**

**containsKey(Object k**) – возвращает true, если коллекция содержит ключ k.

**containsValue(Object v)** – возвращает true, если коллекция содержит значение v.

**entrySet()** – возвращает набор элементов коллекции. Все элементы представляют объект Map.Entry.

**get(Object k)** – возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение null.

**put(K k, V v)** – помещает в коллекцию новый объект с ключом k и значением v. Если в коллекции уже есть объект с подобным ключом, то он перезаписывается. После добавления возвращает предыдущее значение для ключа k, если он уже был в коллекции. Если же ключа еще не было в коллекции, то возвращается значение null.

**putIfAbsent(K k, V v)**: помещает в коллекцию новый объект с ключом k и значением v, если в коллекции еще нет элемента с подобным ключом.

**keySet()** – возвращает набор всех ключей отображения.

**values()** – возвращает набор всех значений отображения.

**putAll(Map<? extends K, ? extends V> map)** – добавляет в коллекцию все объекты из отображения map.

**remove(Object k)** – удаляет объект с ключом k.

**size()** – возвращает количество элементов коллекции.

**Map.Entry** – это интерфейс который является элементом коллекции Map.

**Методы:**

**getValue()** – возвращает значение объекта отображения.

**getKey()** – возвращает ключ объекта отображения.

**setValue(V v)** – устанавливает для текущего ключа значение v.

**SortedMap** – это интерфейс который расширяет интерфейс Map. Элементы размещаются возрастающем порядке значении ключа или в порядке заданного реализацией интерфейса Comparator.

**Методы:**

**comparator()** - Возвращает компаратор, используемый для упорядочения ключей этой мапы, или null, если мапа использует естественный порядок своих ключей.

**entrySet()** - Возвращает набор сопоставлений, содержащихся на этой мапе.

**firstKey()** - Возвращает первый ключ этой мапы.

**headMap(K toKey)** - Возвращает часть элементов этой мапы, ключи которой строго меньше, чем toKey.

**keySet()** - Возвращает Set ключей этой мапы.

**lastKey()** - Возвращает последний ключ этой мапы.

**subMap(K fromKey, K toKey)** - Возвращает часть этой мапы, ключи которой варьируются от fromKey включая, до toKey не включая .

**tailMap(K fromKey)** - Возвращает часть этой мапы, ключи которой больше или равны fromKey.

**values()** - Возвращает Collection значений, содержащихся в этой маке.

**NavigableMap** – это интерфейс который расширяет SortedMap. Добавляет ряд методов для более удобной навигации.

**Методы:**

**lowerEntry(K), floorEntry(K), ceilingEntry(K) и higherEntry(K)** **-** возвращают объекты Map.Entry, связанные с ключами соответственно меньше, меньше или равны, больше или равно и больше, чем данный ключ, возвращая null, если такого ключа нет.

**lowerKey(K), floorKey(K), ceilingKey(K) и higherKey(K)** - возвращают ключ соответственно меньше, меньше или равно, больше или равно и больше, чем данный ключ, возвращая null, если такого ключа нет.

**descendingMap()** - возвращает NavigableMap в обратном порядке.

**subMap(K, boolean, K, boolean), headMap(K, boolean) и tailMap(K, boolean) -** отличаются от аналогичных методов SortedMap принятием дополнительных аргументов, описывающих, являются ли нижняя и верхняя границы инклюзивными по сравнению с эксклюзивными.

**firstEntry(), pollFirstEntry(), lastEntry() и pollLastEntry()** - возвращают и/или удаляют наименьшие и наибольшие отображения, если таковые имеются, в противном случае возвращая null.

**subMap(K, K), headMap(K) и tailMap(K)** - предназначены для возврата части NavigableMap.

Реализации коллекций

**ArrayList –** это класс который реализует интерфейса list.

**LinkedList** – это класс который реализует интерфейсы list, dequeue, queue он соединяет функциональностью списка и очереди. Позволяет очень эффективно добавлять и удалить элементы с произвольным индексом, но итерироваться по нему менее эффективно. Так-что в большинстве случаев он не используется.

**Vector** – это класс который реализация list, в отличии от arraylist он синхронизирован.

**Stack** – это подкласс vector, который работает по принципу LIFO и расширяет vector на 5 методов:

**empty()** - проверяет, пуст ли этот стек.

**peek()** - смотрит на объект в верхней части этого стека, не удаляя его из стека.

**pop()** - удаляет объект в верхней части этого стека и возвращает этот объект.

**push(E item)** - Кладет элемент в стек сверху.

**search(Object o)** - Ищет элемент в стеке. Если найден, возвращается его смещение от вершины стека. В противном случае возвращается 1.

**PriorityQueue** – это класс который реализация интерфейса Queue.

**ArrayDeque** – это класс который реализация интерфейса Deque.

**HashSet** – это класс который реализация интерфейса Set. Основана на хэш таблице. Элементы не упорядочены, порядок элементов может минется.

**LinkedHashSet** – это класс который реализация интерфейса Set. Которая поддерживает связный список элементов в том порядке, в котором они были добавлены. Что позволяет упорядоченную итерацию по множеству.

**TreeSet** – это класс который реализация интерфейса NavigableSet, который поддерживает элементы в отсортированном виде.

**HashMap** – это класс который реализует интерфейс Map.Основан на хэш таблице. Отсортировано относительно значений хэш кодов.

**LinkedHashMap** – это класс который расширяет HashMap. Поддерживает связный список в том порядке, в котором добавлены элементы. Позволяет организовать итерацию по карте в порядке возрастания.

**HashTable** – это класс который реализует Map. Является синхронизированным аналогом HashMap.

**WeekHashMap** – это класс который реализует Map со слабыми ключами. Записи автоматически удаляются Garbage Collector(Сборщиком мусора) сразу как ссылка выйдет за область видимости приложения.

**TreeMap**  - это класс реализующий интерфейс NavigableMap. То файлы в нем отсортированы.

Сортировка коллекций

Comparable— это интерфейс реализация которого сортирует коллекции благодаря методу compareTo который должен выдавать отрицательное, ноль или положительное число на основе которого идет сортировка. Списки и массива реализующие данный интерфейс могут быть отсортированный методом Collections.sort() и Arrays.sort(), а объекты, реализующие данный интерфейс, могут быть отсортированы в SortedSet и SortedMap без указания Компаратора.

Comparator– это интерфейс используется, когда в класс не реализован, либо реализован с неподходящей стороны Comparable. Данный интерфейс можно реализовать в класс с нужной нам сортировкой и передать его коллекции, которая умеет сортировать как аргумент.

Все о структурах данных - https://habr.com/ru/articles/696184/