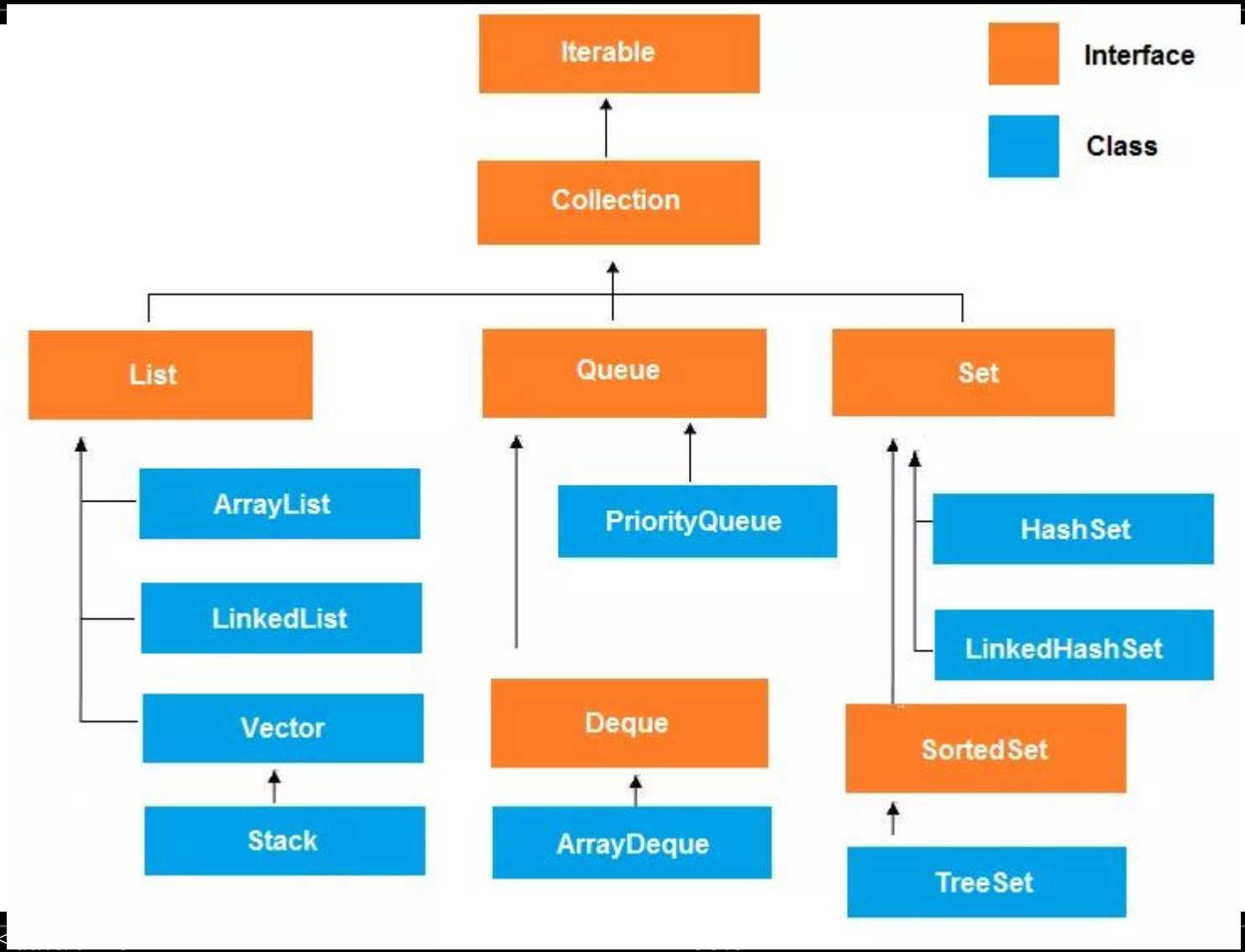
Java Collections

- В библиотеке коллекций Java существует два базовых интерфейса, реализации которых и представляют совокупность всех классов коллекций:
- 1. Collection - коллекция содержит набор объектов (элементов). Здесь определены основные методы для манипуляции с данными, такие как вставка (add, addAll), удаление (remove, removeAll, clear), поиск (contains)
- 2. Map - описывает коллекцию, состоящую из пар "ключ — значение". У каждого ключа только одно значение, что соответствует математическому понятию однозначной функции или отображения (тар). Такую коллекцию часто называют еще словарем (dictionary) или ассоциативным массивом (associative array). Никак НЕ относится к интерфейсу Collection и является самостоятельным.


Iterable
Interface
Class
Collection
List
Queue
Set
ArrayList
LinkedList
Vector
Stack
PriorityQueue
Deque
ArrayDeque
HashSet
LinkedHashSet
SortedSet
TreeSet

List

- ArrayList - пожалуй самая часто используемая коллекция. ArrayList инкапсулирует в себе обычный массив, длина которого автоматически увеличивается при добавлении новых элементов.

- Так как ArrayList использует массив, то время доступа к элементу по индексу минимально (В отличии от LinkedList). При удалении произвольного элемента из списка, все элементы находящиеся «правее» смещаются на одну ячейку влево, при этом реальный размер массива (его емкость, capacity) не изменяется. Если при добавлении элемента, оказывается, что массив полностью заполнен, будет создан новый массив размером (n * 3) / 2 + 1, в него будут помещены все элементы из старого массива + новый, добавляемый элемент.

LinkedList

- LinkedList - Двусвязный список. Это структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и две ссылки («связки») на следующий и предыдущий узел списка. Доступ к произвольному элементу осуществляется за линейное время (но доступ к первому и последнему элементу списка всегда осуществляется за константное время — ссылки постоянно хранятся на первый и последний, так что добавление элемента в конец списка вовсе не значит, что придется перебирать весь список в поисках последнего элемента). В целом же, LinkedList в абсолютных величинах проигрывает ArrayList и по потребляемой памяти и по скорости выполнения операций.

Set

- HashSet - коллекция, не позволяющая хранить одинаковые объекты(как и любой Set). HashSet инкапсулирует в себе объект HashMap (то-есть использует для хранения хэш-таблицу).Выгода от хеширования состоит в том, что оно обеспечивает константное время выполнения методов add(), contains(), remove() и size() , даже для больших наборов.

- Если Вы хотите использовать HashSet для хранения объектов СВОИХ классов, то вы ДОЛЖНЫ переопределить методы hashCode() и equals(), иначе два логически-одинаковых объекта будут считаться разными, так как при добавлении элемента в коллекцию будет вызываться метод hashCode() класса Object

- Класс HashSet не гарантирует упорядоченности элементов, поскольку процесс хеширования сам по себе обычно не порождает сортированных наборов. Если вам нужны сортированные наборы, то лучшим выбором может быть другой тип коллекций, такой как класс TreeSet.

- LinkedHashSet - поддерживает связный список элементов набора в том порядке, в котором они вставлялись. Это позволяет организовать упорядоченную итерацию вставки в набор. То есть, когда идет перебор объекта класса LinkedHashSet с применением итератора, элементы извлекаются в том порядке, в каком они были добавлены.

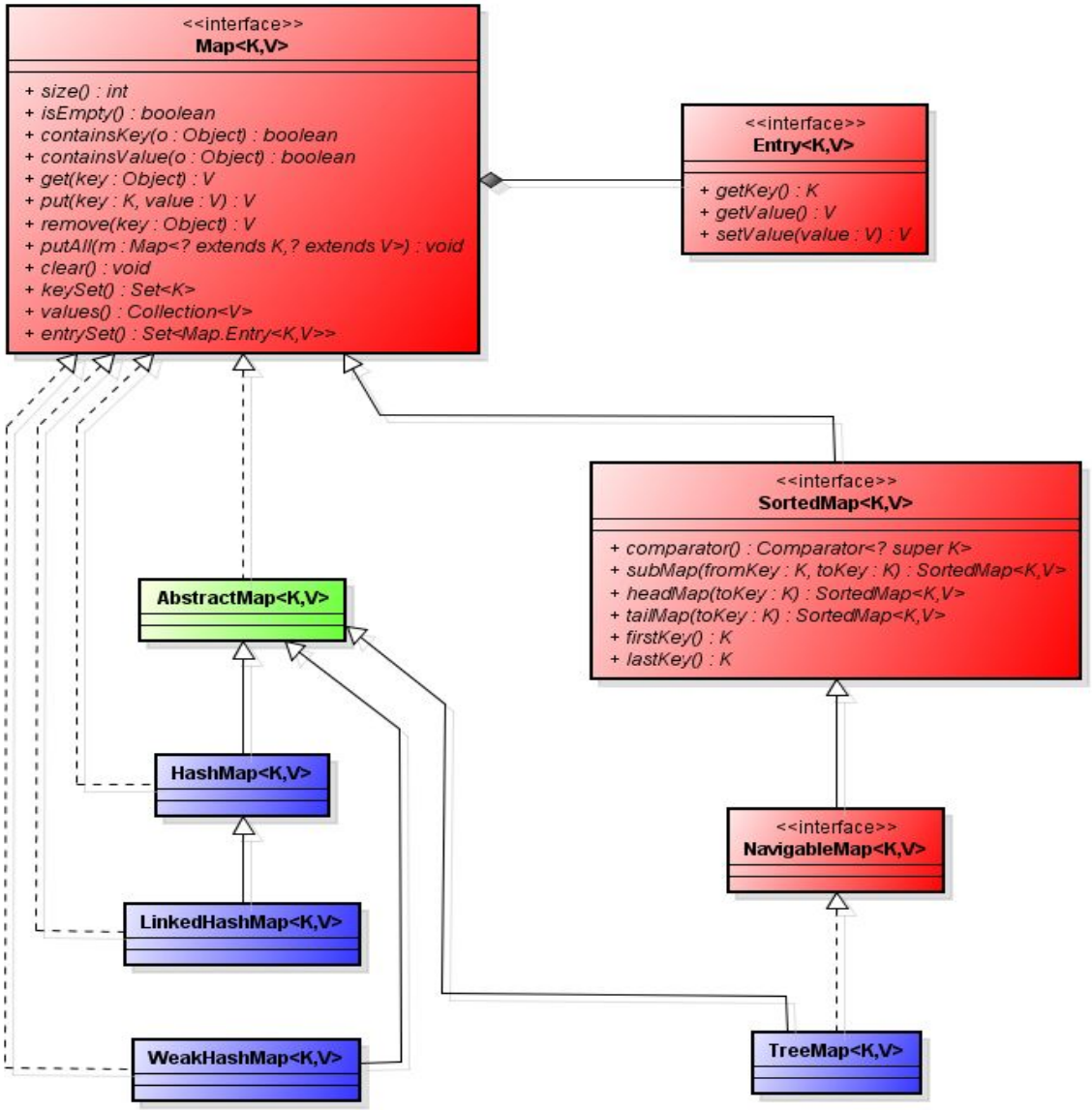
- TreeSet - коллекция, которая хранит свои элементы в виде упорядоченного по значениям дерева. TreeSet инкапсулирует в себе TreeMap, который в свою очередь использует сбалансированное бинарное красно-черное дерево для хранения элементов. TreeSet хорош тем, что для операций add, remove и contains потребуется гарантированное время log(n).

Queue

- PriorityQueue - единственная прямая реализация интерфейса Queue (не считая LinkedList, который больше является списком, чем очередью).
- Эта очередь упорядочивает элементы либо по их натуральному порядку (используя интерфейс Comparable), либо с помощью интерфейса Comparator, полученному в конструкторе.

Map

- Интерфейс Map соотносит уникальные ключи со значениями. Ключ — это объект, который вы используете для последующего извлечения данных. Задавая ключ и значение, вы можете помещать значения в объект карты. После того как это значение сохранено, вы можете получить его по ключу. Интерфейс Map — это обобщенный интерфейс, объявленный так: interface Map<K, V>.


<<interface>>
Map<K,V>
+ size() : int
+ isEmpty() : boolean
+ containsKey(o : Object) : boolean
+ containsValue(o : Object) : boolean
+ get(key : Object) : V
+ put(key : K, value : V) : V
+ remove(key : Object) : V
+ putAll(m : Map<? extends K,? extends V>) : void
+ clear() : void
+ keySet() : Set<K>
+ values() : Collection<V>
+ entrySet() : Set<Map.Entry<K,V>>
<<interface>>
Entry<K,V>
+ getKey() : K
+ getValue() : V
+ setValue(value : V) : V
<<interface>>
SortedMap<K,V>
+ comparator() : Comparator<? super K>
+ subMap(fromKey : K, toKey : K) : SortedMap<K,V>
+ headMap(toKey : K) : SortedMap<K,V>
+ tailMap(toKey : K) : SortedMap<K,V>
+ firstKey() : K
+ lastKey() : K
AbstractMap<K,V>
HashMap<K,V>
LinkedHashMap<K,V>
WeakHashMap<K,V>
<<interface>>
NavigableMap<K,V>
TreeMap<K,V>

HashMap

- HashMap — основан на хэш-таблицах, реализует интерфейс Map (что подразумевает хранение данных в виде пар ключ/значение). Ключи и значения могут быть любых типов, в том числе и null. Данная реализация не дает гарантий относительно порядка элементов с течением времени.

- LinkedHashMap - расширяет класс HashMap. Он создает связный список элементов в карте, расположенных в том порядке, в котором они вставлялись. Это позволяет организовать перебор карты в порядке вставки. То есть, когда происходит итерация по коллекционному представлению объекта класса LinkedHashMap, элементы будут возвращаться в том порядке, в котором они вставлялись. Вы также можете создать объект класса LinkedHashMap, возвращающий свои элементы в том порядке, в котором к ним в последний раз осуществлялся доступ.

TreeMap

- TreeMap - расширяет класс AbstractMap и реализует интерфейс NavigatebleMap. Он создает коллекцию, которая для хранения элементов применяет дерево. Объекты сохраняются в отсортированном порядке по возрастанию. Время доступа и извлечения элементов достаточно мало, что делает класс TreeMap блестящим выбором для хранения больших объемов отсортированной информации, которая должна быть быстро найдена.

WeakHashMap

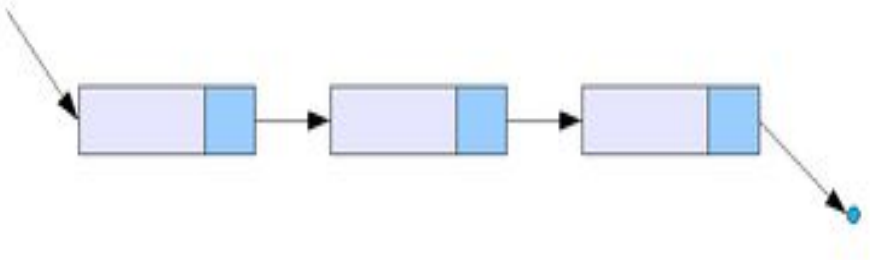
- WeakHashMap - коллекция, использующая слабые ссылки для ключей (а не значений). Слабая ссылка (англ. weak reference) — специфический вид ссылок на динамически создаваемые объекты в системах со сборкой мусора. Отличается от обычных ссылок тем, что не учитывается сборщиком мусора при выявлении объектов, подлежащих удалению. Ссылки, не являющиеся слабыми, также иногда именуют «сильными».

Класс ArrayList и интерфейс List

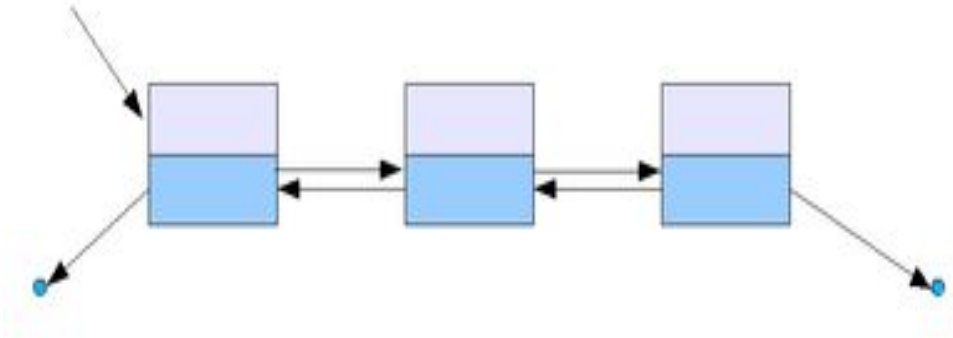
- ArrayList имеет следующие конструкторы:
- ArrayList(): создает пустой список
- ArrayList(Collection <? extends E> col): создает список, в который добавляются все элементы коллекции col.
- ArrayList (int capacity): создает список, который имеет начальную емкость capacity

LinkedList

Односвязный список



Двусвязный список



Описание	Операция	ArrayList	LinkedList
Взятие элемента	get	Быстро	Медленно
Присваивание элемента	set	Быстро	Медленно
Добавление элемента	add	Быстро	Быстро
Вставка элемента	add(i, value)	Медленно	Быстро
Удаление элемента	remove	Медленно	Быстро

- List<String> linkedList = new LinkedList<>();
- linkedList.add("Barsik");
- linkedList.add("Murzik");
- linkedList.add("Ryzhik");
- Iterator<String> iterator = linkedList.iterator();
- String firstCat = iterator.next(); // обратиться к первому элементу
- Log.i("Test", firstCat);
- String secondCat = iterator.next(); // обратиться ко второму элементу
- Log.i("Test", secondCat);

- Используемый в примере итератор имеет ограниченные возможности, например, его метод add() добавляет новый элемент в конец списка. Чтобы иметь возможность вставлять в середину, используйте интерфейс ListIterator. Кроме того, в ListIterator имеются методы previous() и hasPrevious() для прохода по списку в обратном направлении.

- Метод add() итератора вводит новый элемент до текущей позиции итератора. Например, в следующем примере пропускается первый элемент связного списка и вводится элемент "Пушок" перед вторым элементом.

- List<String> linkedList = new LinkedList<>();
- linkedList.add("Barsik");
- linkedList.add("Murzik");
- linkedList.add("Ryzhik");
- ListIterator<String> iterator = linkedList.listIterator();
- iterator.next(); // пропускаем первый элемент списка
- iterator.add("Пушок");
- Log.i("Test", linkedList.toString());