Skillbox

Курс «Java разработчик с нуля»

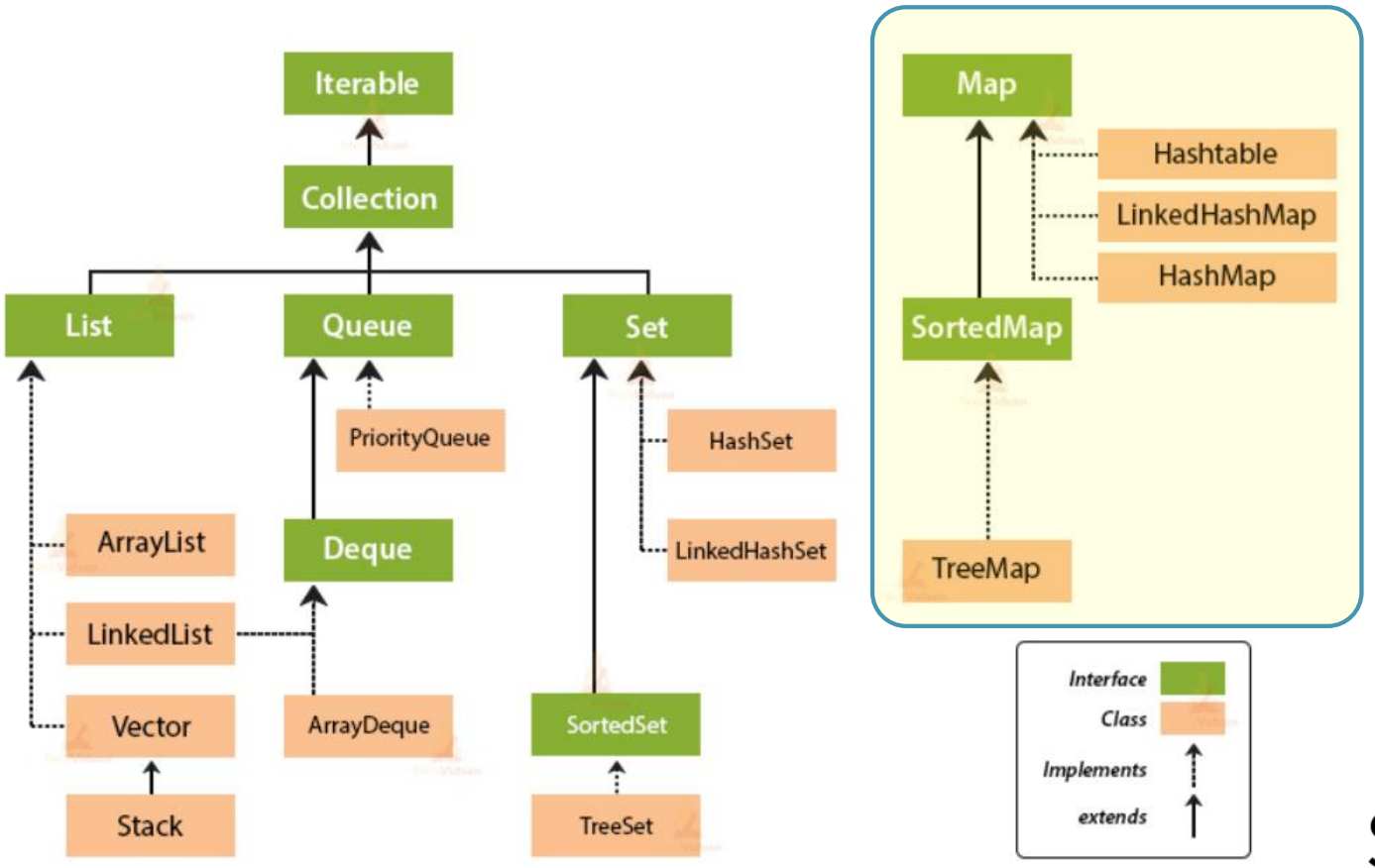
Основные коллекции и
их устройство

java.util.Map

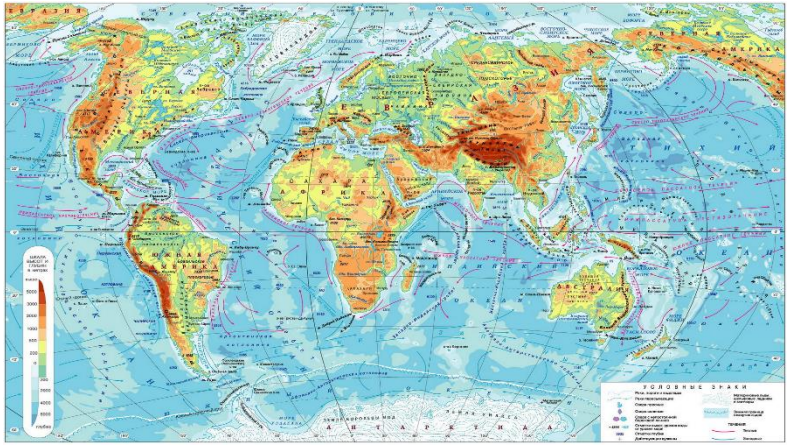
Спикер: Шибков Константин

31.03.2021 19:00 МСК

Иерархия коллекций Java



Map



Ключ (уникальный)	Значение (не уникальное)
SK-223443	Принтер
SK-894937	Диск SSD
SK-1839543	Принтер
SK-037435	Монитор

Когда мы слышим слово мап, то первое, что приходит на ум — это географические карты.

Map как структура данных не имеет ничего с такими картами.

Если посмотреть перевод слова Map, то у него есть несколько других значений
- существительные: карта, план, **таблица**, лицо, личность
- глагол: **сопоставлять**, отображаться

И в нашем случае это структура данных похожая на таблицу, которая сопоставляет уникальный объект (ключ) с содержимым (значением).

Основные представители

Класс	Ключи уникальны	Сортированный порядок ключей	Сохраняется порядок добавления ключей	Потокбезопасная	О-большое доступа к элементу
HashMap	✓	-	-	-	O(1)
TreeMap	✓	✓	-	-	O(log n)
LinkedHashMap	✓	-	✓	-	O(1)
HashTable	✓	-	-	✓	O(1)

Рекомендуется использовать **ConcurrentHashMap** для работы в многопоточное среде, так как HashTable блокируют всю коллекцию, а **ConcurrentHashMap** только нужную часть.

Объявление Map

```
Map<String, String> orders;
```

```
Map<String, List<Product>> orders;
```

```
Map<OrderId, Map<Product, Price>>;
```

В роли ключа и значения может быть только объекты.

java.util.**HashMap**

Skillbox

HashMap

- Класс реализует интерфейс Map
- Хранит только уникальные значения ключей
- Может хранить null в ключах
- Порядок добавления элементов не сохраняется
- Порядок размещения элементов вычисляется с помощью хэш-кода
- ПотокоНЕбезопасен

HashMap – вызов конструктора по умолчанию

```
Map<String, String> orders = new HashMap<String, String>();
```

Если в объявлении указаны тип ключей и значений, то указывать в new это не требуется:

```
Map<String, String> orders = new HashMap<>();
```

При этом при использовании `var`, в такой Map ключ и значение `Object`:

```
var orders = new HashMap<>();
```

значит необходимо указать тип в конструкторе:

```
var orders = new HashMap<String, String>();
```

Два метода обеспечивающие быструю работу HashMap

`int hashCode()`

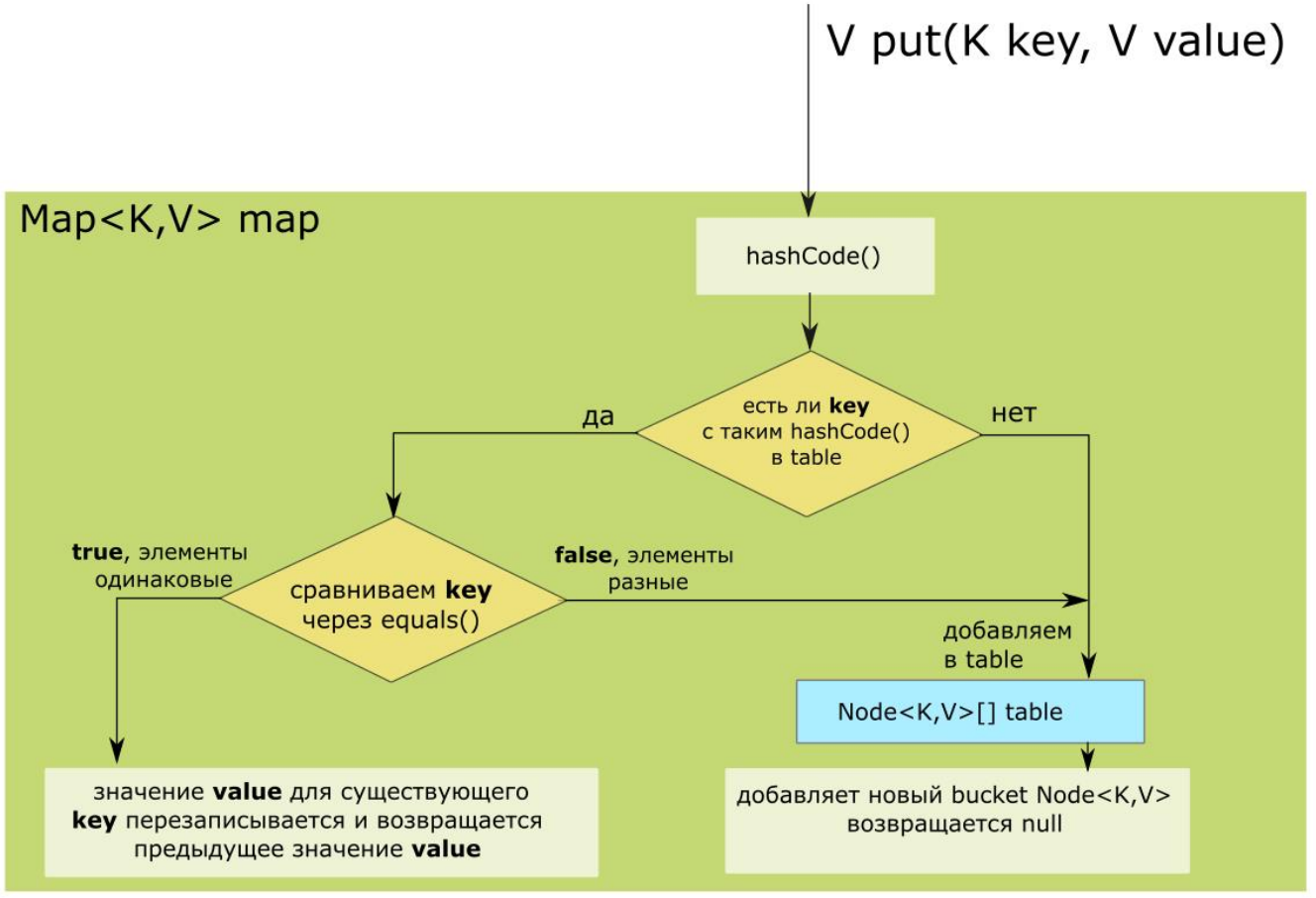
- вычисляется быстро, сравнить два хеша int тоже просто,
- количество хешей ограничено количество значений int
- если хеши двух объектов одинаковые ->
  возможно объекты одинаковые
- если хеши двух объектов разные ->
  объекты точно разные

`boolean equals(Object object)`

- на основе состояние объекта подробно сравнивает объекты, дольше чем сравнение хеша, дает однозначный ответ.

- 🔗 https://habr.com/ru/post/168195/

HashMap – механизм добавления элемента (общий вид)



Skillbox

HashMap – внутреннее строение

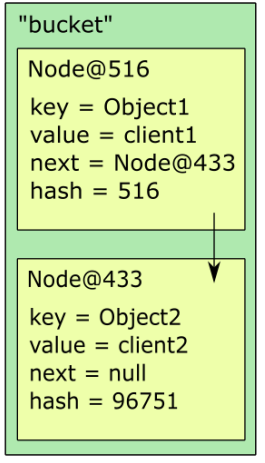
HashMap хранит объекты массиве table, каждый его элемент называется «корзина» bucket, в корзинах хранятся Node<K, V>.

При создании Map, массив пустой, размером 16 элементов (capacity).

`Node<K,V>[]`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null

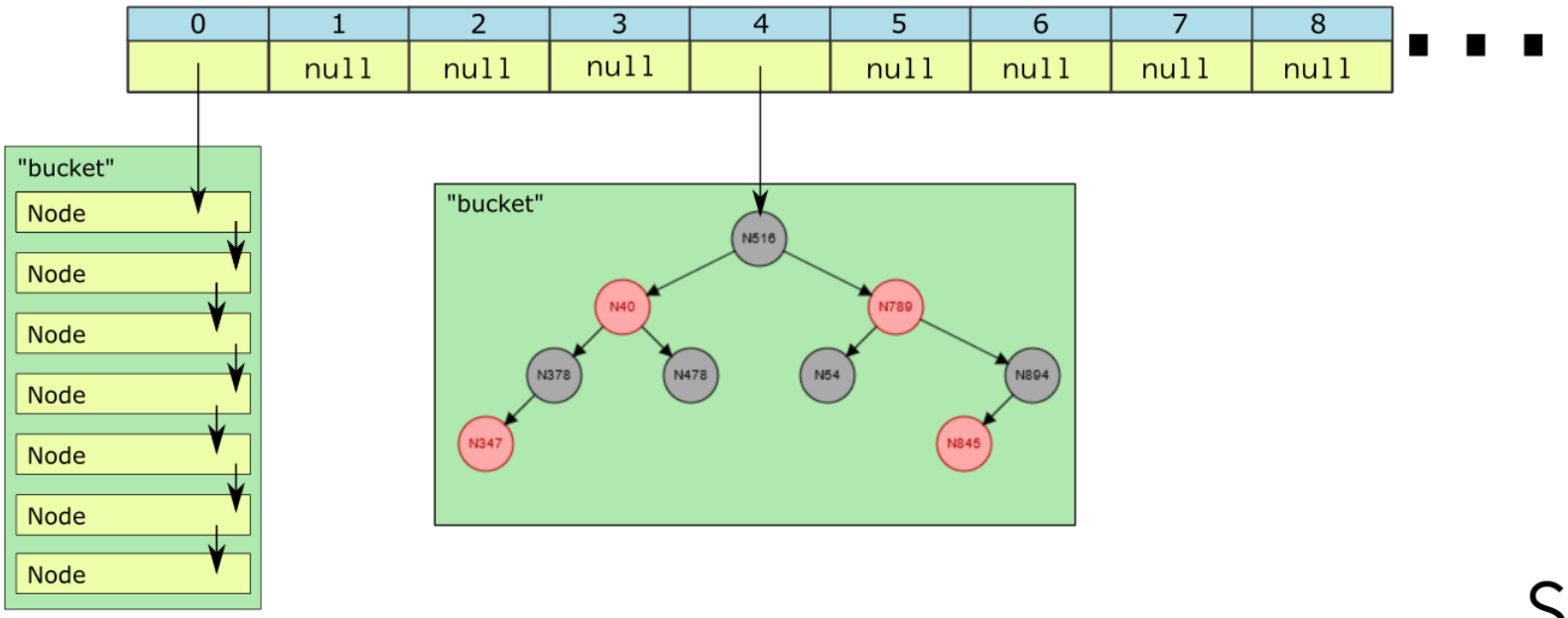
Внутри Node хранится:
- K key – объект, переданный ключ
- V value – объект, переданное значение
- int hash – внутренний хэш ключа
- Node next – ссылка на следующий Node в бакете



Skillbox

HashMap – внутреннее строение

Если в одной корзине приходится размещать более одной Node, то они организуются в виде LinkedList и каждая первая Node имеет ссылку на следующую. Если в одной корзине более 8 элементов Node, то список перестраивается в красно-черное дерево.



Визуализация структур данных - https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Skillbox

HashMap – механизм добавления элемента

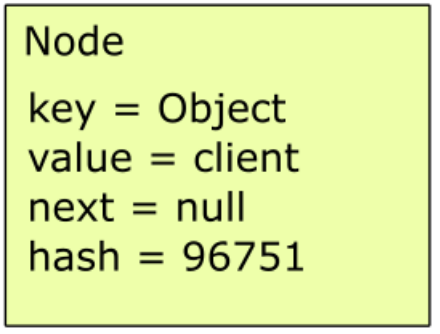
Как происходит проверка наличия объекта по этому хэшу?

- вычисляется индекс ячейки массива где мы хотим расположить объект

- если ячейка пуста (null), то создается Node node и в его значение key вкладываем записываем ключ, а в value записываем переданное значение

- если ячейка не пустая, и объект уникальный – equals() != true, то параметр next у последней в этой корзине node становится равный ссылке на новый node.

-value не участвует в выборе ячейки



Skillbox

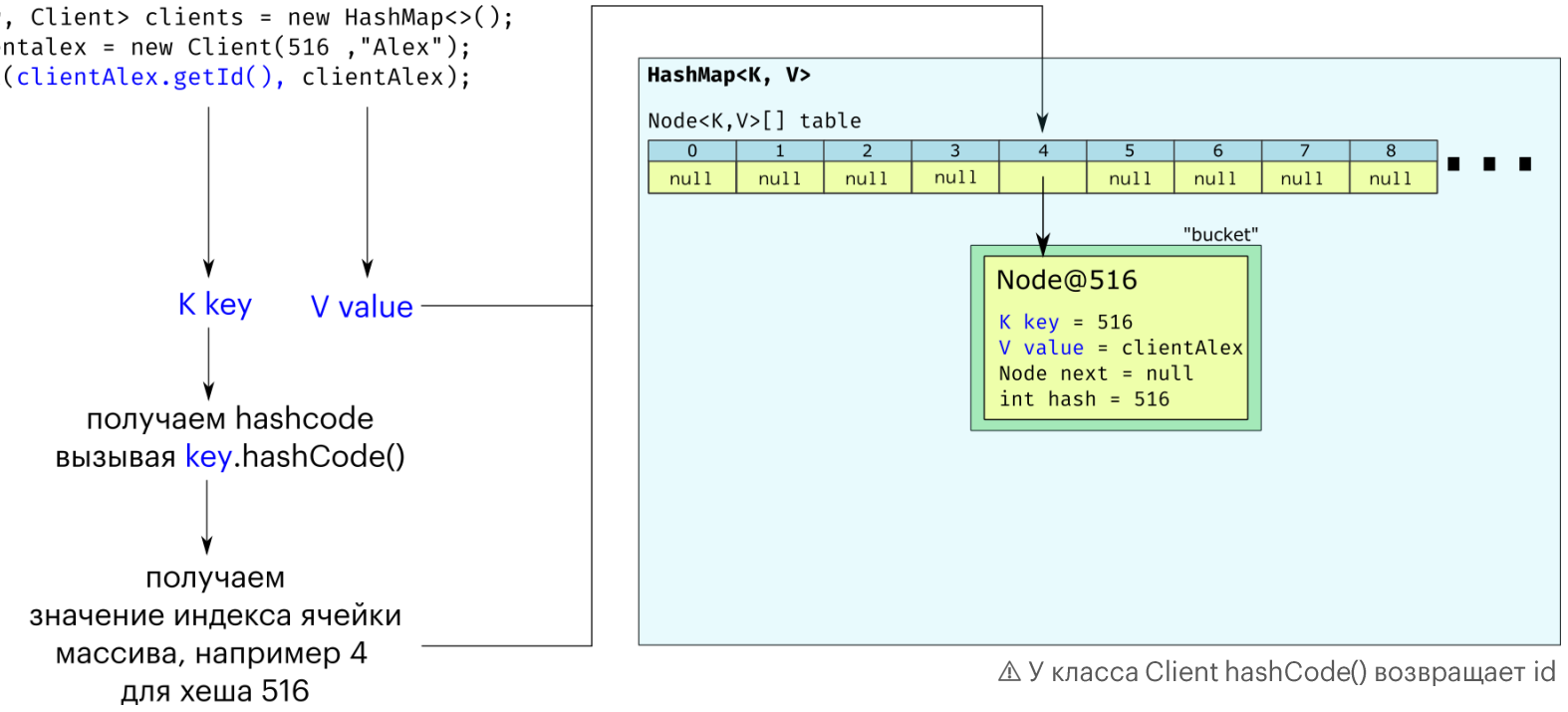
HashMap – механизм добавления элемента

```
class Client{
       private final int id;
       private final String name;

       public Client(int id, String name) {
           this.id = id;
           this.name = name;
       }

       public int getId() {
           return id;
       }

       public String getName() {
           return name;
       }

       @Override
       public boolean equals(Object o) {
           Client client = (Client) o;
           return id == client.id && name.equals(client.name);
       }

       @Override
       public int hashCode() {
           return id;
       }
   }
```

Skillbox

HashMap – механизм добавления элемента

```
Map<Integer, Client> clients = new HashMap<>();
Client clientalex = new Client(516 ,"Alex");
clients.put(clientAlex.getId(), clientAlex);
```



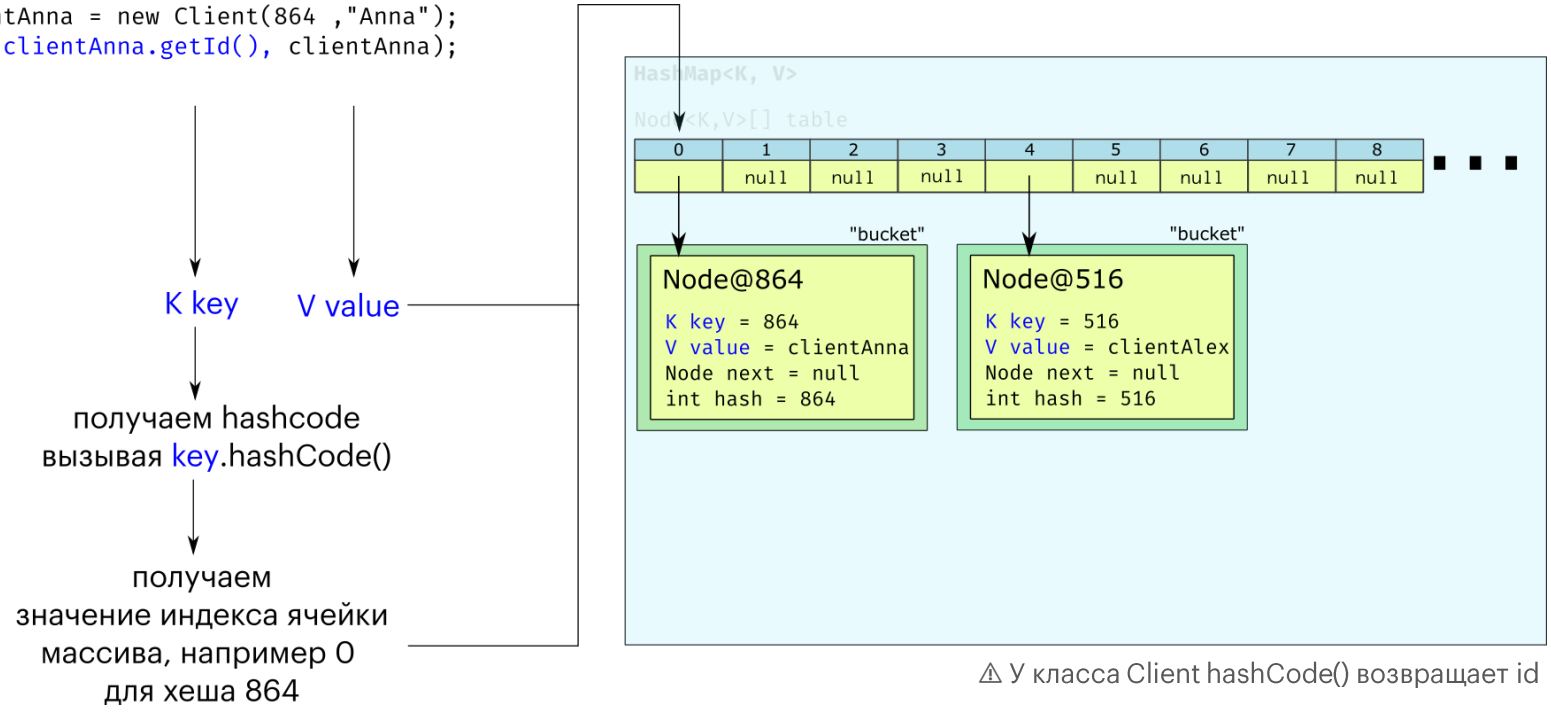
⚠ У класса Client hashCode() возвращает id

```
int hash = key.hashCode() ^ (key.hashCode() >>> 16); //516
int index = (16 - 1) & hash; //4
```

Skillbox

HashMap – механизм добавления элемента

```
Client clientAnna = new Client(864 ,"Anna");
clients.put(clientAnna.getId(), clientAnna);
```



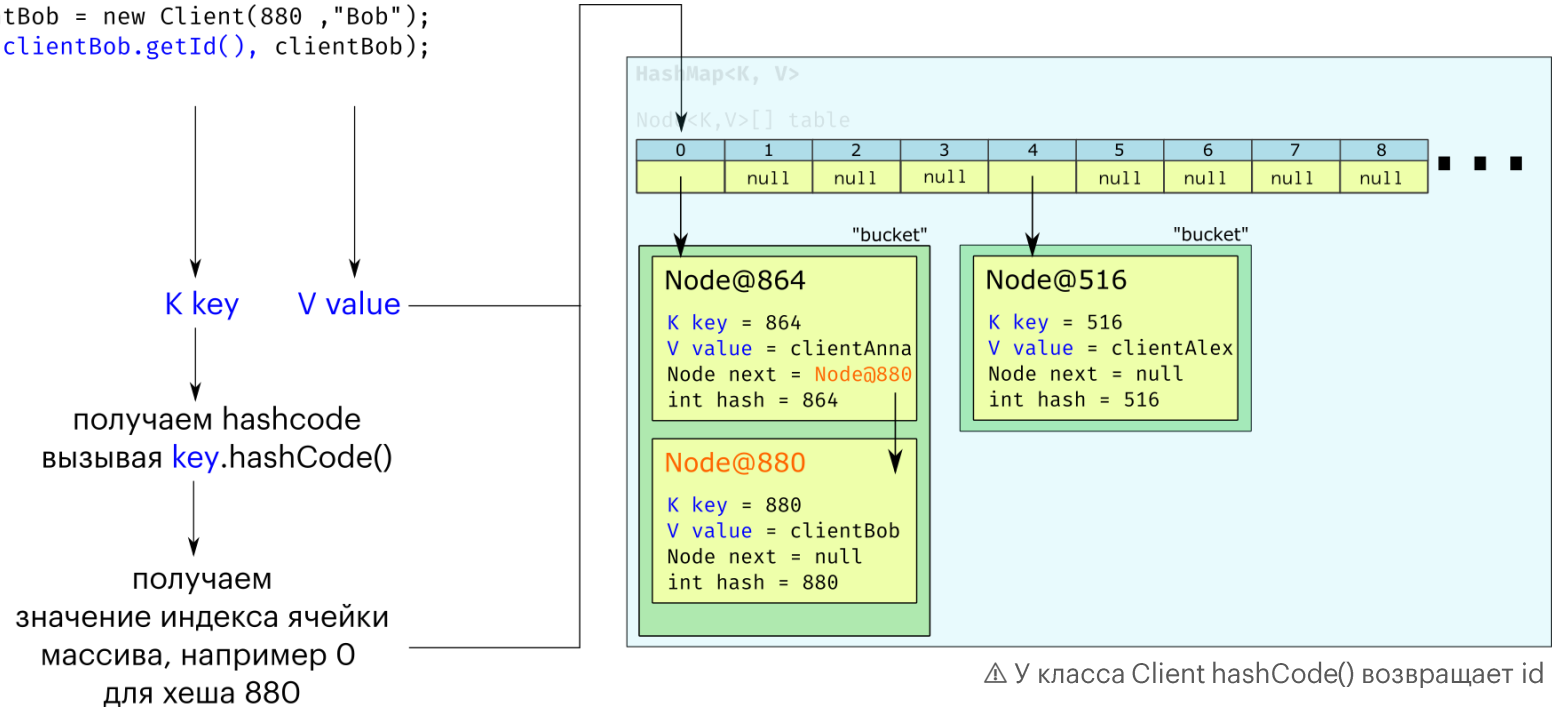
⚠ У класса Client hashCode() возвращает id

```
int hash = key.hashCode() ^ (key.hashCode() >>> 16); //864
int index = (16 - 1) & hash; //0
```

Skillbox

HashMap – механизм добавления элемента

```
Client clientBob = new Client(880 ,"Bob");
clients.put(clientBob.getId(), clientBob);
```



⚠ У класса Client hashCode() возвращает id

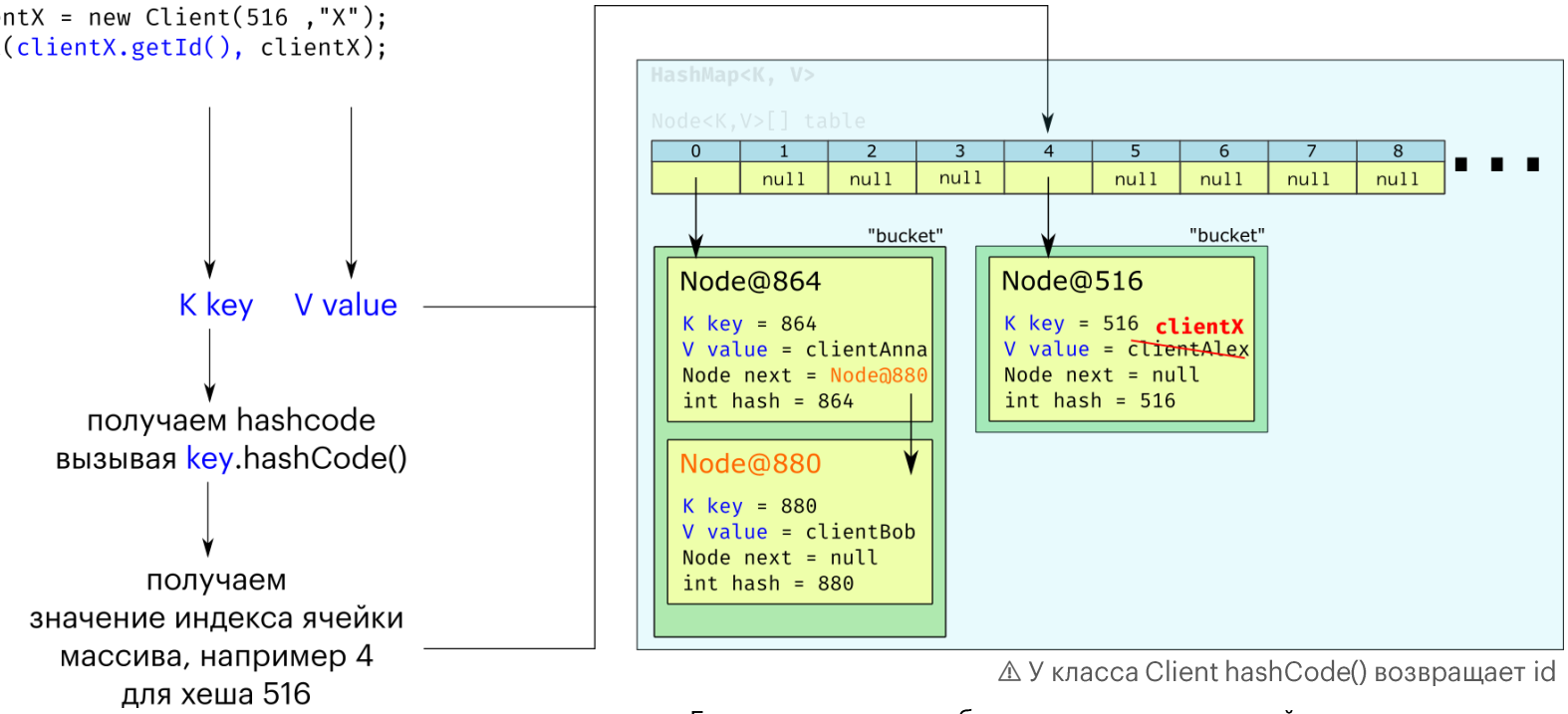
Так как index ячейки массива table у элементов одинаковый,
а сами объекты ключей equals() == false,
то они складываются в одну корзину

```
int hash = key.hashCode() ^ (key.hashCode() >>> 16); //880
int index = (16 - 1) & hash; //0
```

Skillbox

HashMap – механизм добавления элемента

```
Client clientX = new Client(516 ,"X");
clients.put(clientX.getId(), clientX);
```



⚠ У класса Client hashCode() возвращает id

Если элемент при добавлении в непустую ячейку массива, при сравнении с ключам объектами в корзине equals() == true, то тогда происходит замена значения value в этой Node

```
int hash = key.hashCode() ^ (key.hashCode() >>> 16); //516
int index = (16 - 1) & hash; //4
```

Skillbox

HashMap – создание

Конструкторы:

По умолчанию (initialCapacity=16, loadFactor=0,75):

```
HashMap()
```

Указывается начальный размер и коэффициент загрузки:

```
HashMap(int initialCapacity, float loadFactor)
```

Указывается начальный размер (loadFactor=0,75):

```
HashMap(int initialCapacity)
```

Создание Map на основе другой Map:

```
HashMap(Map<? extends K, ? extends V> m)
```

HashMap – создание

Статические методы Map:

Пары ключ-значение передаются в параметрах, K (key) – ключ, V (value) - значение:

```
Map<K, V> of(K k1, V v1)
```

```
Map<K, V> of(K k1, V v1, K k2, V v2)
```

Перегруженные методы of() существуют до 10 пар ключ-значение:

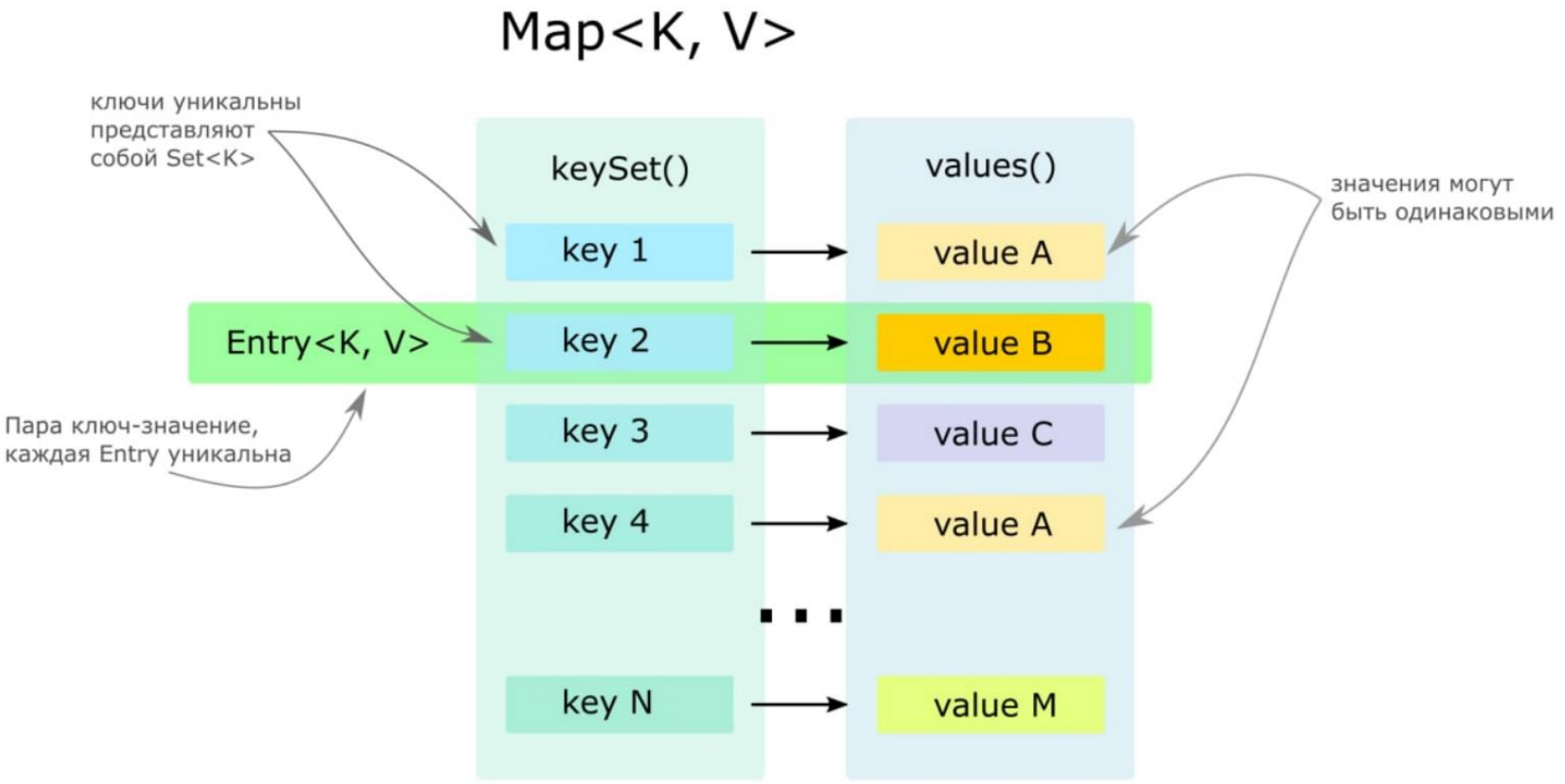
```
Map<K, V> of(K k1, V v1, K k2, V v2, K k3, V v3,
             K k4, V v4, K k5, V v5, K k6, V v6,
             K k7, V v7, K k8, V v8, K k9, V v9, K k10, V v10)
```

Также есть метод в который можно передать массив или набор Entry:

```
Map<K, V> ofEntries(Entry<? extends K, ? extends V>... entries)
```

```
Map<String, Integer> map = Map.ofEntries(Map.entry("1", 1), Map.entry("2", 6));
```

HashMap



Skillbox

HashMap – основные методы

```
Map<Integer, Client> clients = new HashMap<>();
```

Добавление элементов:

```
clients.put(516, new Client(516, "Alex"));
```

```
clients.putAll(otherObjectImplementsMap); // добавить все Entry другой Map
```

Получить значение по ключу, если такого ключа нет - вернется null:

```
Client client = clients.get(516);
```

При удалении, возвращается value на который указывает переданный ключ, если такого ключа нет – вернется null:

```
Client removedClient = clients.remove(516);
```

Получить размер, количество элементов:

```
int clientsSize= clients.size();
```

HashMap – перебор элементов

```
Map<Integer, Client> clients = new HashMap<>();
```

Получить Set состоящий из ключей и работать с ним:

```
for (Integer clientId : clients.keySet()){
    System.out.println(clientId);
}
```

Получить Collection состоящую из значений всех ключей:

```
for (Client client : clients.values()){
    System.out.println(client.getName());
}
```

Получить Set из Entry (пара ключ-значение), и у Entry можно получить ключ и значение:

```
for (Map.Entry<Integer, Client> entry : clients.entrySet()){
    System.out.printf("%s - %s%n",
        entry.getKey(), entry.getValue().getName());
}
```

Skillbox

HashMap – большое (O)

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

- https://www.bigocheatsheet.com/

Skillbox

java.util.TreeMap

Skillbox

TreeMap

- Класс реализует интерфейс Map
- Хранит только уникальные значения ключей
- Элементы отсортированы, по используемому компаратору.
- Данные хранятся в виде красно-чёрного дерева
- ПотокоНЕбезопасен

TreeMap – вызов конструктора по умолчанию

⚠ Важное требование при создании TreeMap - ключи должны быть сравниваемы (implements Comparable):

```
Map<String, String> orders = new TreeMap<String, String>();
```

Если в объявлении указаны тип ключей и значений, то указывать в new это не требуется:

```
Map<String, String> orders = new TreeMap<>();
```

При этом при использовании var, в такой Map значение Object,а ключи должны быть сравниваемы между собой:

```
var orders = new TreeMap<>();
```

```
var products = new TreeMap<>();
products.put(new Product(), "");
products.put(new AutoProduct(), "");
```

```
class Product implements Comparable<Product> {
    @Override
    public int compareTo(Product o) {
        return Integer.compare(o.hashCode(), hashCode());
    }
}

class AutoProduct extends Product {}
```

Skillbox

TreeMap – вызов конструктора по умолчанию

Для того чтобы ключи использовать в TreeMap они должны:

- или имплементировать Comparable,

```
class Animal implements Comparable<Animal> {
```

и реализовать метод

```
int compareTo(Animal o)
```

- или передать в TreeMap компаратор для этого типа объектов

```
class AnimalComparator implements Comparator<Animal> {
```

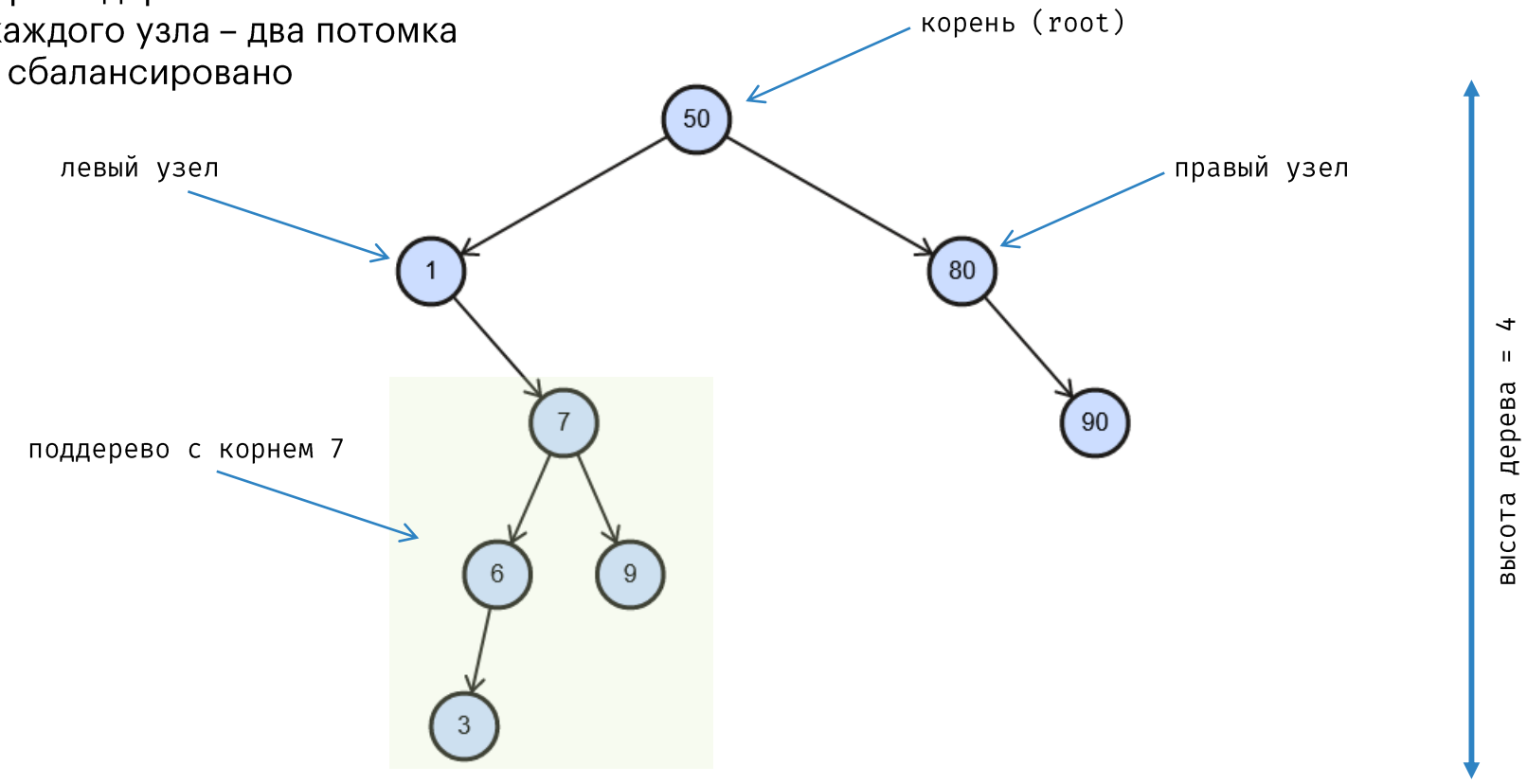
и реализовать метод

```
int compare(Animal o1, Animal o2)
```

```
Map<Animal, List<Animal> ancestors = new TreeMap(new AnimalComparator());
```

TreeMap – немного про деревья

Бинарное дерево:
- у каждого узла – два потомка
- не сбалансировано



- 🔗 http://www.btv.melezinek.cz/binary-search-tree.html

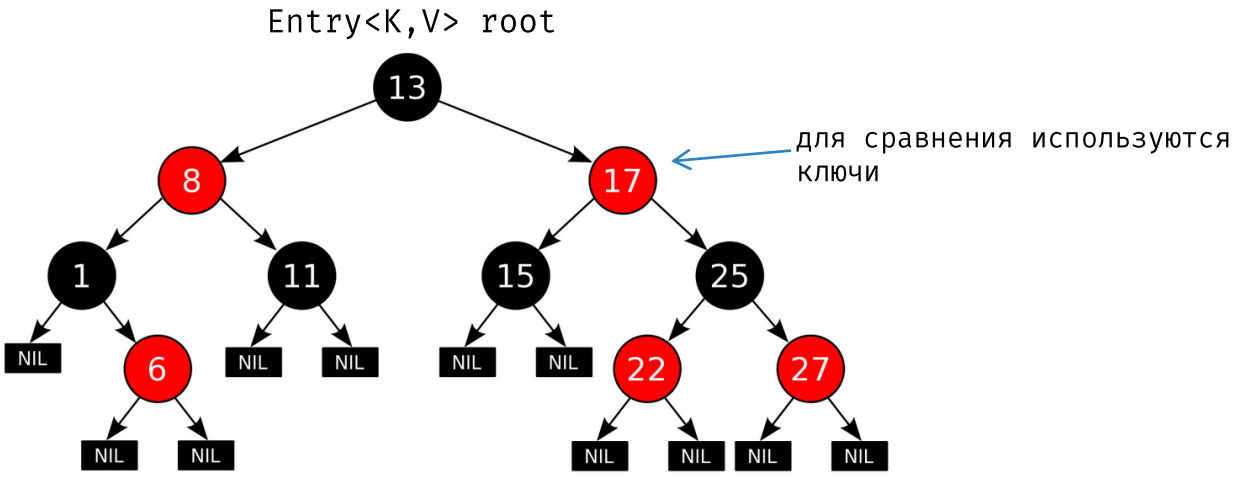
Skillbox

TreeMap – немного про деревья

В TreeMap используются красно-черные деревья:
- Каждый узел промаркирован красным или чёрным цветом
- Корень и конечные узлы (листья) дерева — чёрные
- У красного узла родительский узел — чёрный
- Все простые пути из любого узла x до листьев содержат одинаковое количество чёрных узлов (или разница в высоте между простыми путями не более 1)
- Чёрный узел может иметь чёрного родителя

TreeMap.Entry<K,V>

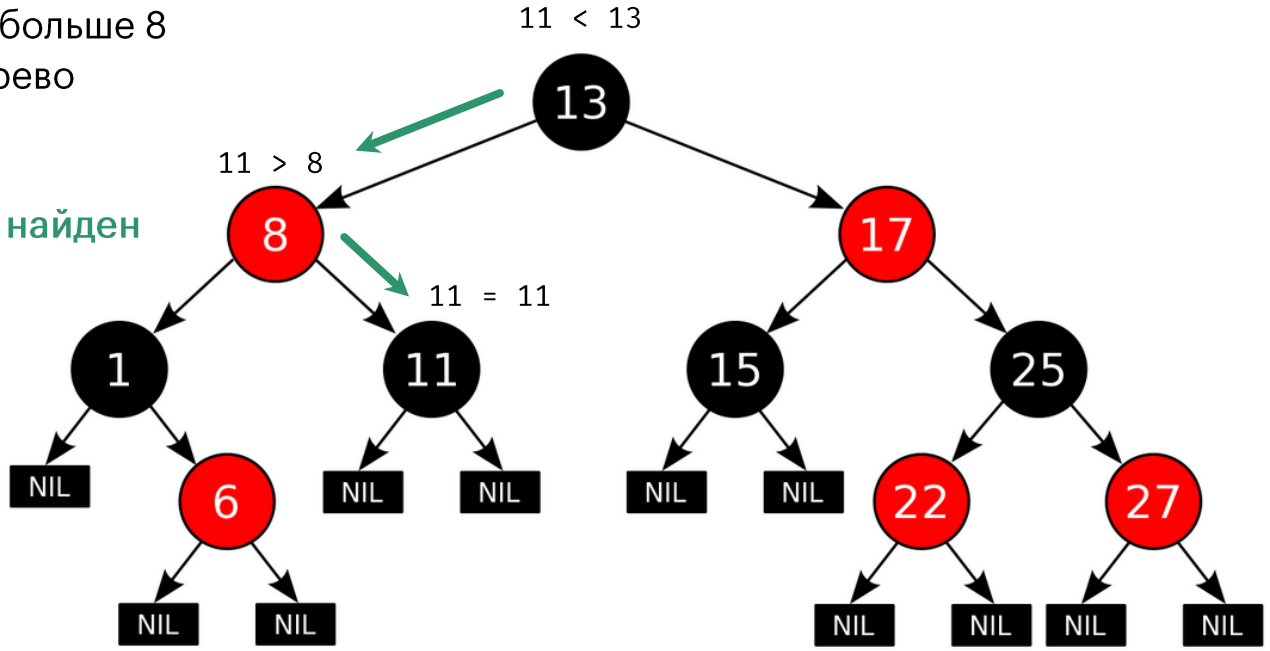
```
K key;
V value;
Entry<K,V> left;
Entry<K,V> right;
Entry<K,V> parent;
boolean color = BLACK;
```



Skillbox

TreeMap – поиск

Ищем 11:
- сравниваем с корнем, 11 меньше 13
  идем в левое поддерево
- корень поддерева 8
- сравниваем с корнем, 11 больше 8
  идем в правое поддерево
- корень поддерева 11
- сравниваем с корнем,
  11 равно 11 – элемент найден

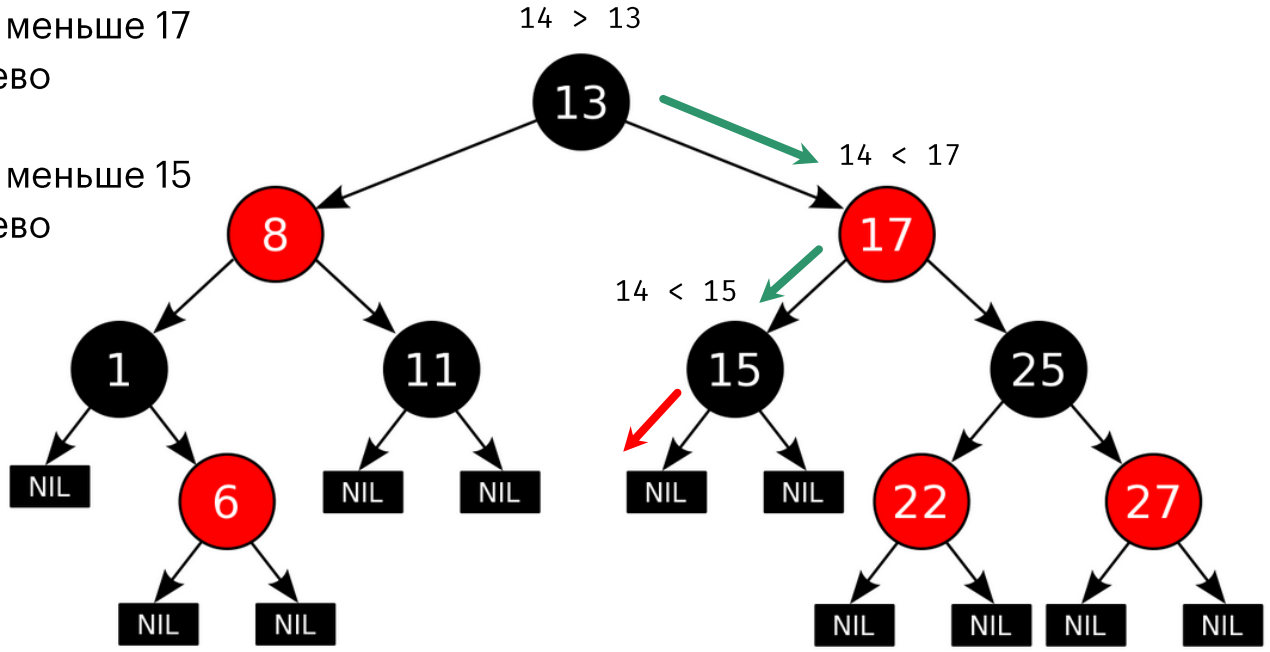


Skillbox

TreeMap – поиск

Ищем 14:

- сравниваем с корнем, 14 больше 13
  идем в правое поддерево
- корень поддерева 17
- сравниваем с корнем, 14 меньше 17
  идем в левое поддерево
- корень поддерева 15
- сравниваем с корнем, 14 меньше 15
  идем в левое поддерево
- левого поддерева нет –
  элемент не найден

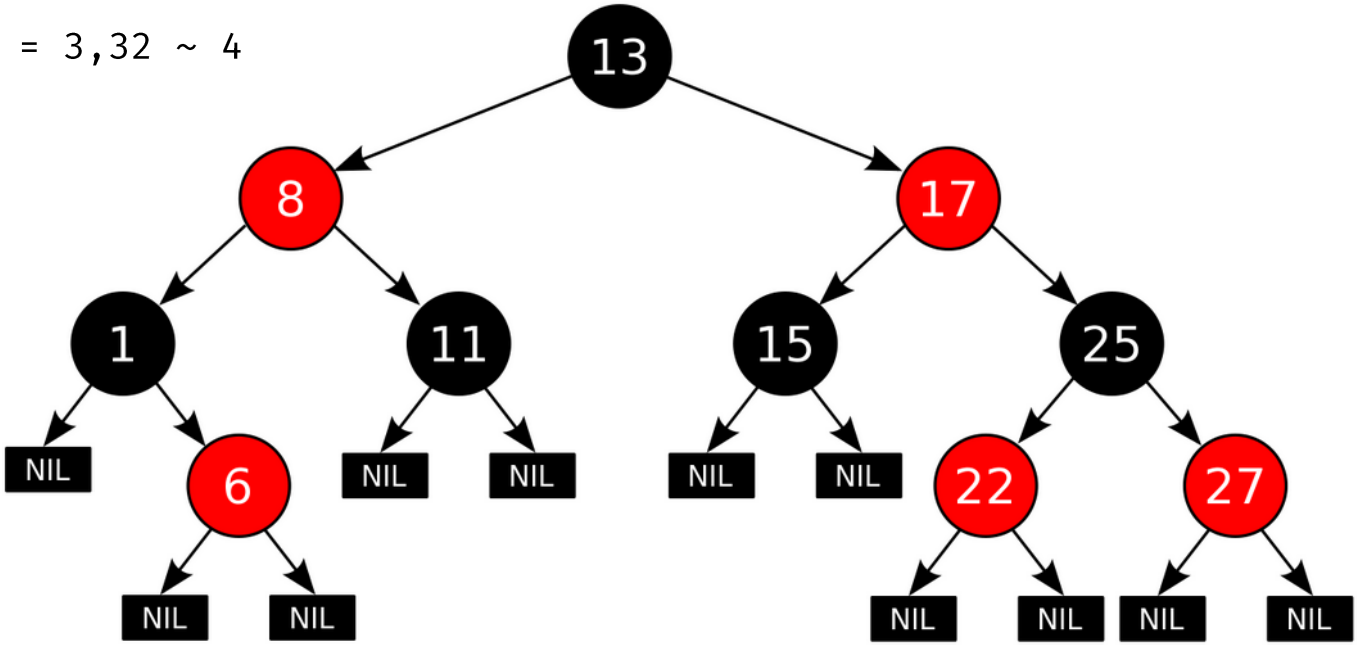


Skillbox

TreeMap – поиск

Исходя из особенностей дерева, количество сравнений для поиска, не превышает высоты дерева. А данном случае 4. Для расчета количества операций сравнения при поиске используется log2(n).

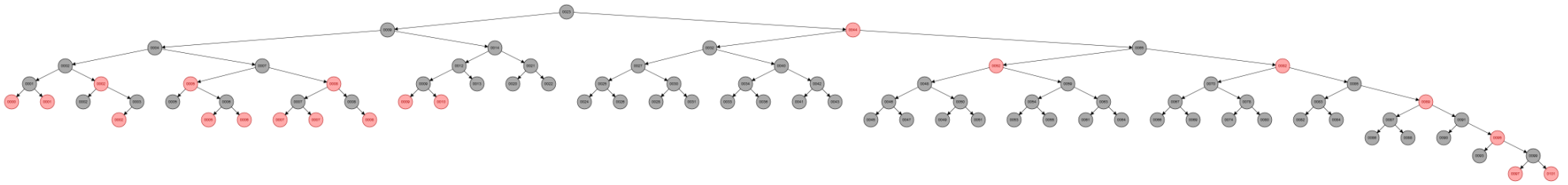
```
log2(10) = 3,32 ~ 4
```



Skillbox

TreeMap – поиск, сравнение с ArrayList

Количество элементов в коллекции	ArrayList количество операций сравнения при поиске (худшее)	TreeMap количество операций сравнения при поиске (худшее)
1	1	1
10	10	4
100	100	7
1 000	1 000	10
100 000	100 000	17



TreeMap – большое O

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

- https://www.bigocheatsheet.com/

Skillbox

Спасибо за внимание!

Skillbox