# Прекрасный мир Java

Часть 8.

# Как реализовать стек (модель LIFO)?

```
public class StackException {
public int STACK_OVERFLOW = 0;
public int STACK_EMPTY = 1;
private int errorCode;

public StackException(int
code) {
   errorCode = code;
}
```

```
public String getMessage() {
   String strResult = "Error:
      Unknown error!";
   switch (errorCode) {
   case STACK_OVERFLOW: {
      strResult = "Error:
         Stack is full!";
   }
   case STACK_EMPTY: {
      strResult = "Error:
         Stack is empty!";
   }
   }
   return strResult;
}
```

```
public class StackException
extends Exception {

public final static int
STACK_OVERFLOW = 0;
public final static int
STACK_EMPTY = 1;
private int errorCode;

public StackException(int
code) {
   errorCode = code;
}
```

```
public String getMessage() {
   switch (errorCode) {
   case STACK_OVERFLOW: {
      return "Error:
      Stack is full!";
   }
   case STACK_EMPTY: {
      return "Error:
      Stack is empty!";
   }
   default: {
      return "Unknow error!";
   }
   }
}
```

Контейнер — объект, используемый для хранения других объектов.

# Интерфейсы стандартной библиотеки контейнеров Java

- Iterable
  - Collection
    - Set
      - SortedSet
    - Queue
      - Deque
    - List
- Map
  - SortedMap

```
Stack<Integer> stack =
  new Stack<Integer>();
stack.push(new Integer(15));
stack.push(new Integer(7));
stack.push(new Integer(23));
Iterator iter = stack.iterator();
while (iter.hasNext()) {
  Integer i = iter.next();
}
```

```
Stack<Integer> stack =
  new Stack<Integer>();
stack.push(new Integer(15));
stack.push(new Integer(7));
stack.push(new Integer(23));
for (Integer i : stack) {
  ...
}
```

Интерфейс Collection (коллекция) декларирует базовую функциональность любого контейнера, содержащего некоторый набор элементов. Сюда относятся методы для добавления/удаления элементов или групп элементов, методы для получения размера коллекции.

Интерфейс Set (набор) переопределяет контракт интерфейса Collection так, чтобы обеспечить отсутствие двух одинаковых элементов в контейнере.

Интерфейс Queue (очередь, «кью») модифицирует общий контракт интерфейса Collection так, чтобы обеспечить функционирование буферной очереди (то есть контейнера, способного накапливать элементы перед их обработкой).

Интерфейс List (список) модифицирует общий контракт интерфейса Collection так, чтобы обеспечить индексированный доступ к элементам контейнера. Соответственно, индексы элементов определяют порядок на множестве хранимых элементов.

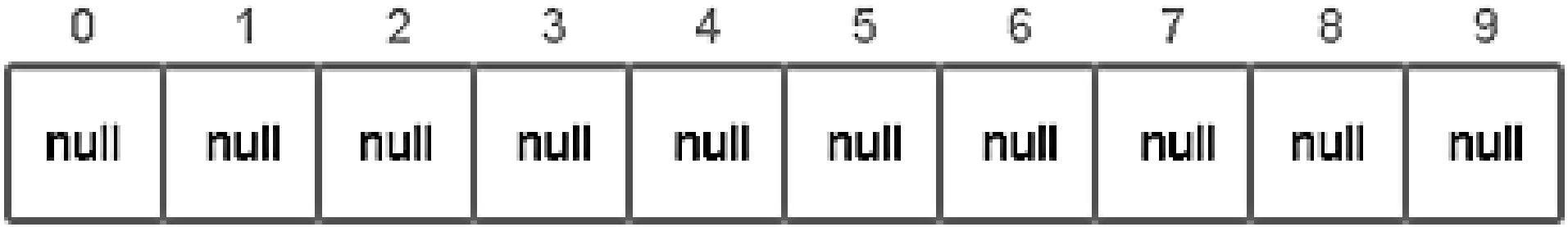
Интерфейс Map (карта) декларирует функциональность для реализации инъективных отображений объектов (ассоциативный массив). То есть элементы такого контейнера «индексируются» не целыми числами, а другими объектами.

```
Map<String, String> map =
   new HashMap<String, String>(10);
map.put("day", "14");
map.put("month", "November");
map.put("year", "2013");

for (String key : map.keySet())
{
   ...
}
```

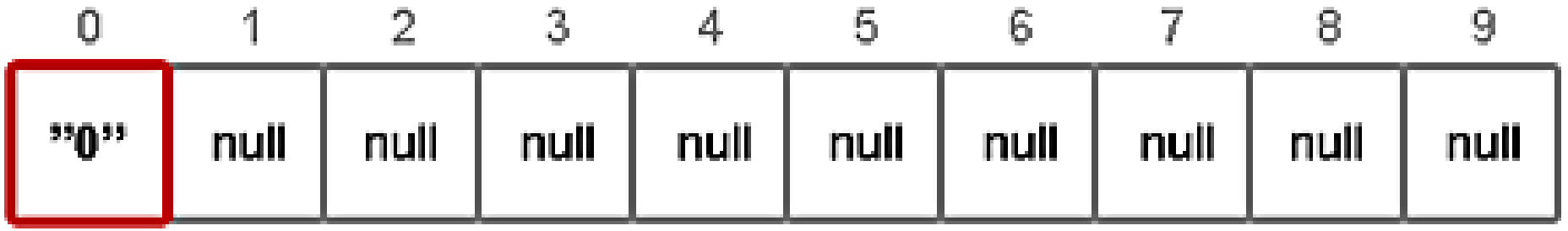
Интерфейсы SortedSet и SortedMap добавляют в соответствующие контракты обязанность поддержки порядка на множестве элементов (соответственно, итераторы возвращают элементы в указанном порядке).
Для указания правила сравнения элементов используются интерфейсы Comparable и Comparator.

**ArrayList** — реализует интерфейс List.
В Java массивы имеют фиксированную длину, и после того как массив создан, он не может расти или уменьшаться. ArrayList может менять свой размер во время исполнения программы, при этом не обязательно указывать размерность при создании объекта.
Элементы ArrayList могут быть абсолютно любых типов в том числе и null.

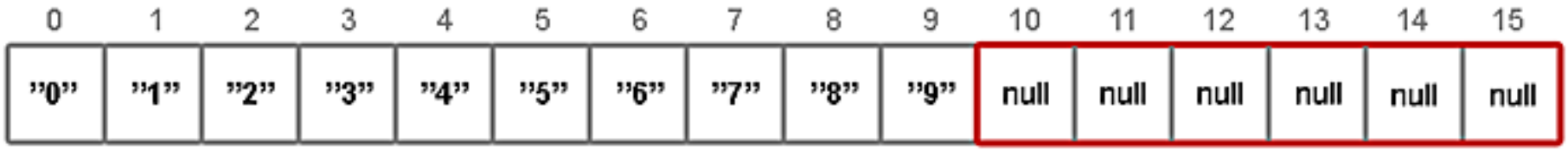


`add(value)`

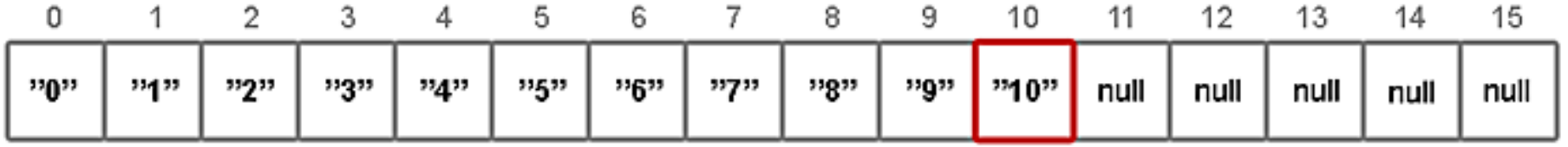
1) проверяется, достаточно ли места в массиве для вставки нового элемента
2) добавляется элемент в конец (согласно значению **size**) массива



При добавлении 11-го элемента, проверка показывает, что места в массиве нет. Соответственно создается новый массив и вызывается **System.arrayCopy()**




После этого добавление элементов продолжается

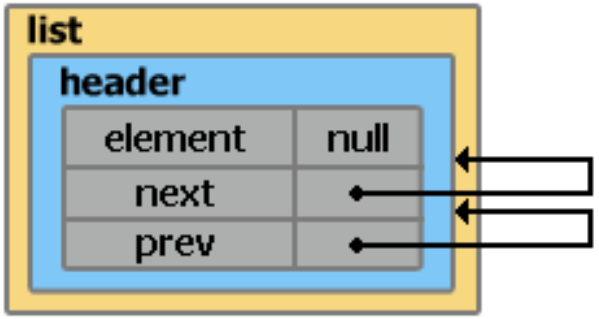


remove(index)
remove(value)

Сначала определяется какое количество элементов надо скопировать, затем копируем элементы используя **System.arraycopy()** уменьшаем размер массива и забываем про последний элемент.

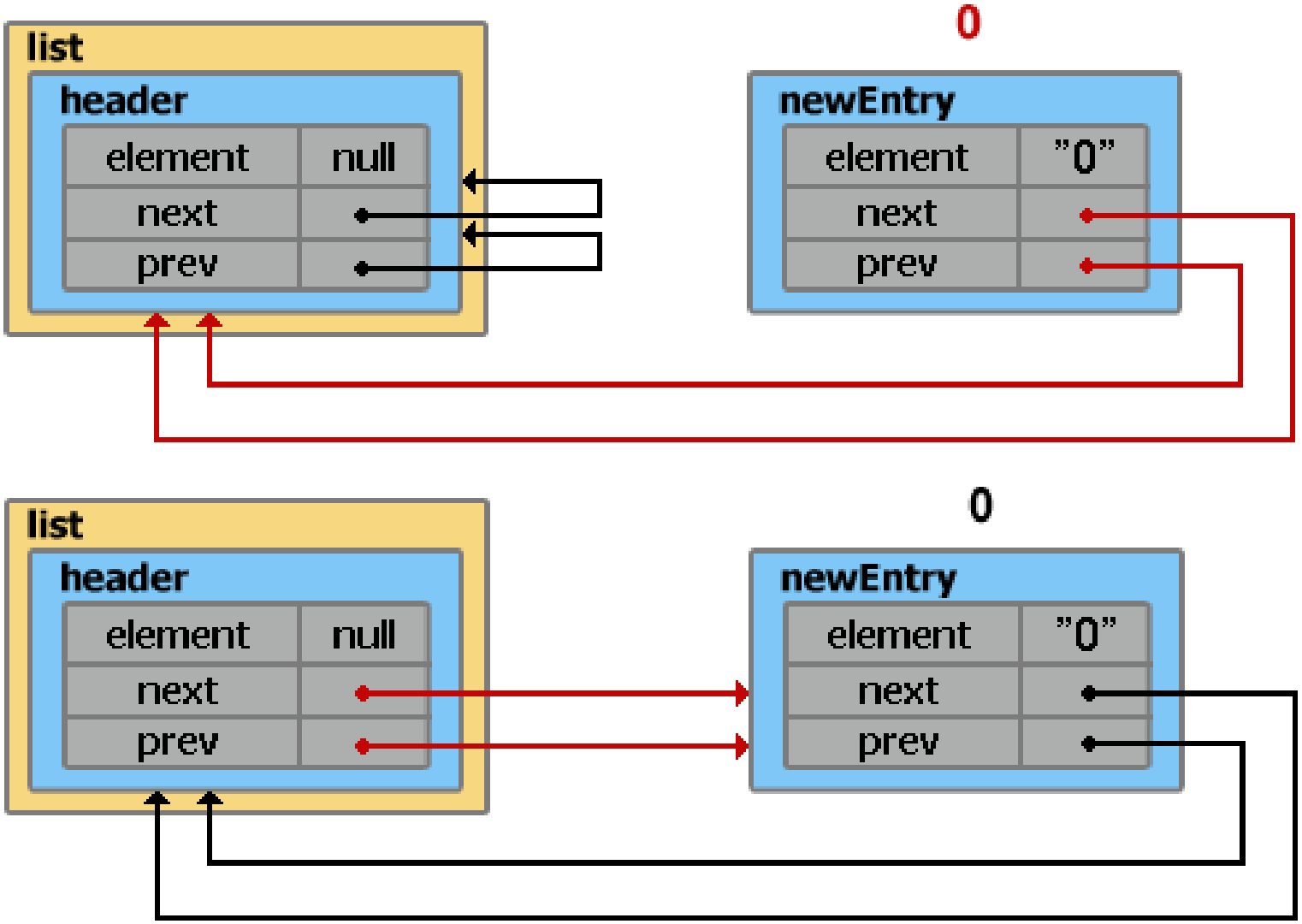
**LinkedList** — реализует интерфейс List. Является представителем двунаправленного списка, где каждый элемент структуры содержит указатели на предыдущий и следующий элементы. Итератор поддерживает обход в обе стороны. Реализует методы получения, удаления и вставки в начало, середину и конец списка.

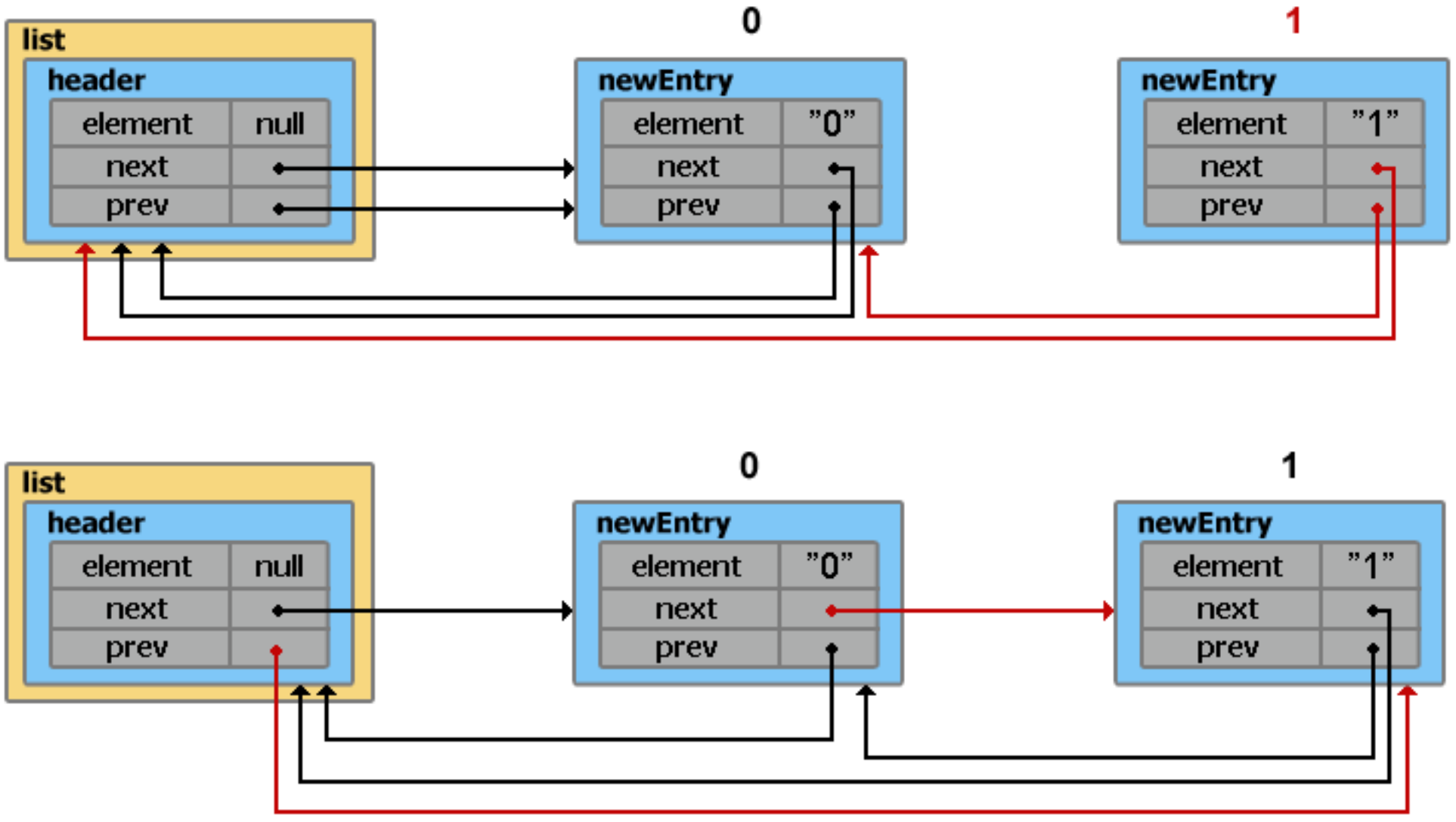
Только что созданный объект list, содержит свойства header и size.
Header — псевдо-элемент списка. Его значение всегда равно null, а свойства next и prev всегда указывают на первый и последний элемент списка соответственно. Так как на данный момент список еще пуст, свойства next и prev указывают сами на себя (т.е. на элемент header).
Размер списка size равен 0.



Добавление элемента в конец списка с помощью методом add(value), addLast(value) и добавление в начало списка с помощью addFirst(value) выполняется за время O(1).
Внутри класса LinkedList существует static inner класс Entry, с помощью которого создаются новые элементы. Каждый раз при добавлении нового элемента, по сути выполняется два шага:

1) создается новый новый экземпляр класса Entry
2) переопределяются указатели на предыдущий и следующий элемент


list
header
element
null
next
prev
0
newEntry
element
"0"
next
prev
list
header
element
null
next
prev
0
newEntry
element
"0"
next
prev


list
header
element
null
next
prev
0
newEntry
element
"0"
next
prev
1
newEntry
element
"1"
next
prev
list
header
element
null
next
prev
0
newEntry
element
"0"
next
prev
1
newEntry
element
"1"
next
prev

Удаление элементов:

- из начала или конца списка с помощью removeFirst(), removeLast() за время O(1);
- по индексу remove(index) и по значению remove(value) за время O(n).

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. Разрешение коллизий осуществляется с помощью метода цепочек.

Новоявленный объект HashMap, содержит ряд свойств:

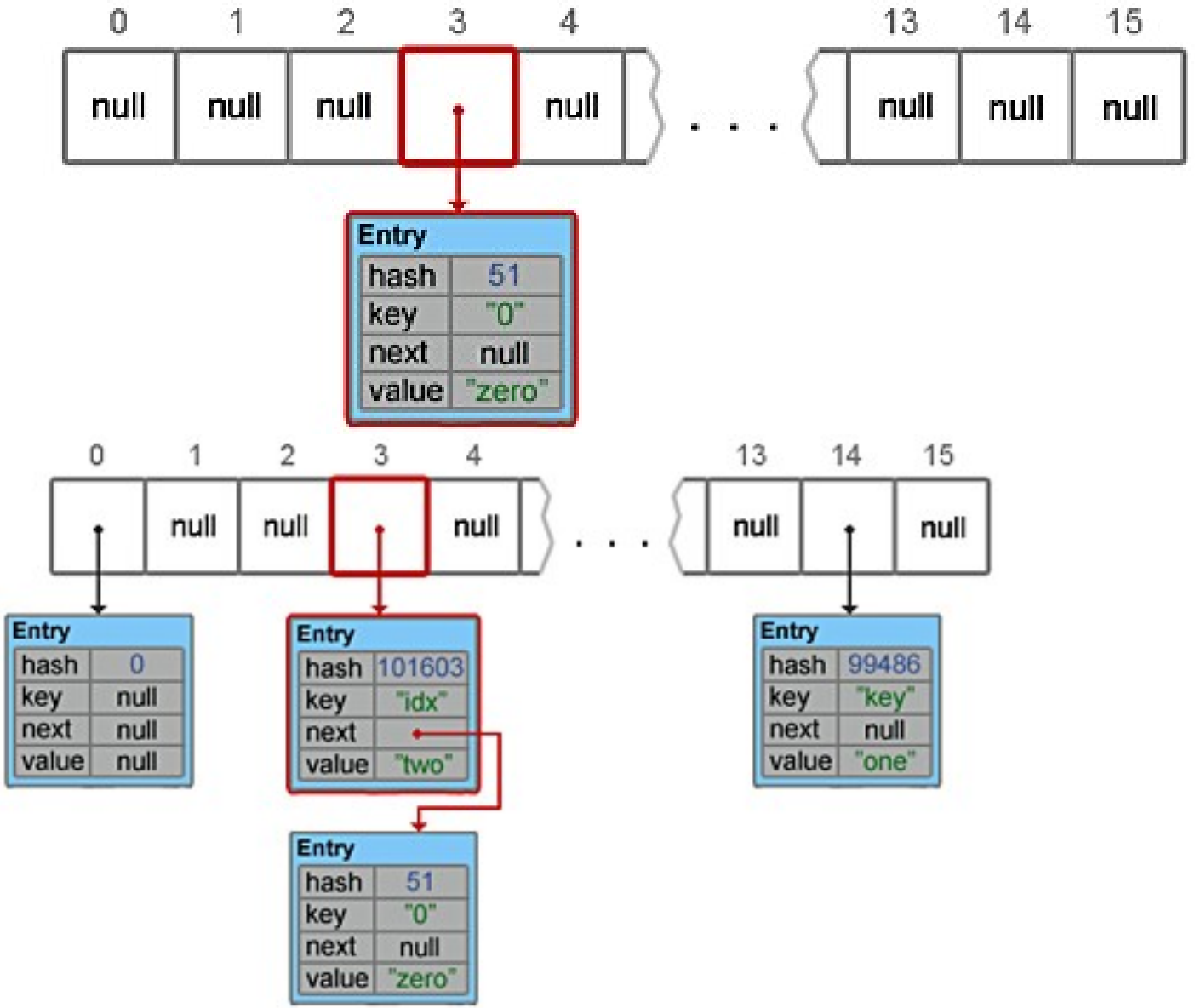
- table — Массив типа Entry[], который является хранилищем ссылок на списки (цепочки) значений;
- loadFactor — Коэффициент загрузки. Значение по умолчанию 0.75 является хорошим компромиссом между временем доступа и объемом хранимых данных;
- threshold — Предельное количество элементов, при достижении которого, размер хэш-таблицы увеличивается вдвое. Рассчитывается по формуле (capacity * loadFactor);
- size — Количество элементов HashMap-а;

При добавлении элемента, последовательность шагов следующая:

1. Сначала ключ проверяется на равенство null. Если это проверка вернула true, будет вызван метод putForNullKey(value).
2. Далее генерируется хэш на основе ключа. Для генерации используется метод hash(hashCode), в который передается key.hashCode().
3. С помощью метода indexFor(hash, tableLength), определяется позиция в массиве, куда будет помещен элемент.

Хэш и ключ нового элемента поочередно сравниваются с хэшами и ключами элементов из списка и, при совпадении этих параметров, значение элемента перезаписывается.

Если же предыдущий шаг не выявил совпадений, будет вызван метод addEntry(hash, key, value, index) для добавления нового элемента.


0
1
2
3
4
13
14
15
null
null
null
null
null
null
null
null
Entry
hash 51
key "0"
next null
value "zero"
0
1
2
3
4
13
14
15
null
null
null
null
null
Entry
hash 0
key null
next null
value null
Entry
hash 101603
key "idx"
next
value "two"
Entry
hash 99486
key "key"
next null
value "one"
Entry
hash 51
key "0"
next null
value "zero"