Коллекции, алгоритмы, сложность

Воронцов Александр
Куракин Андрей
5030102/30202

Структуры данных

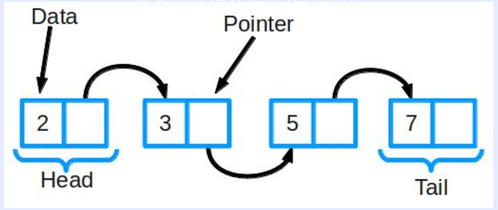
Структуры данных

Примеры структур:

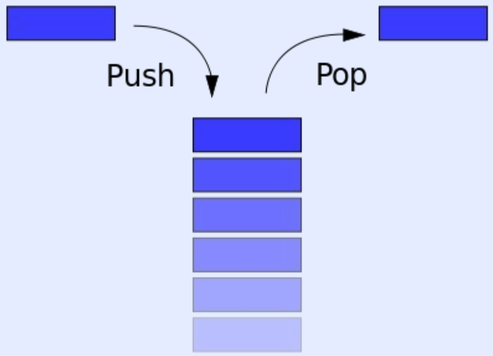
- Массивы
- Связный список
- Стек (Stack)
- Очередь (Queue)
- Множество (Set)
- Таблицы, карты (Map)
- Графы

Структура данных – способ организации информации для более эффективного применения

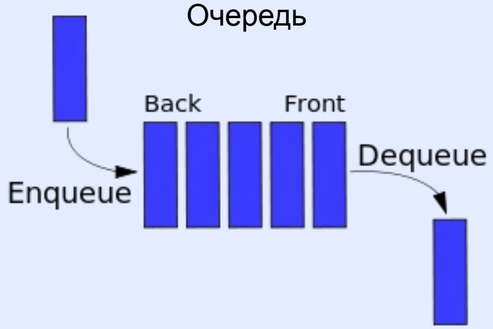
Связный список



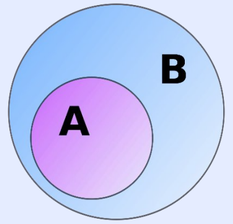
Стек



Очередь



Множество (Set)

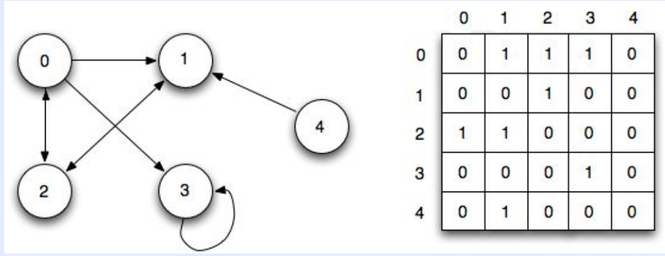


Карта (Map)

KEYS	VALUES
Jan	327.2
Feb	368.2
Mar	197.6
Apr	178.4
May	100.0
Jun	69.9
Jul	32.3
Aug	37.3
Sep	19.0
Oct	37.0
Nov	73.2
Dec	110.9
Annual	1551.0

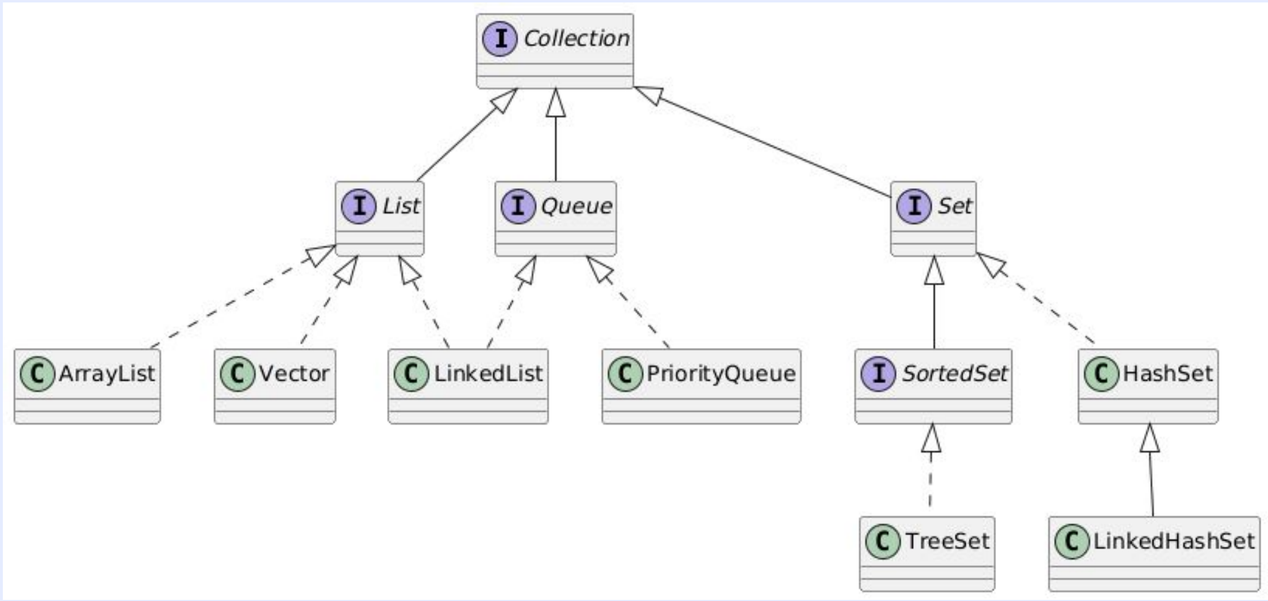
Aug → Aug → 37.3

Граф и матрица смежности



	0	1	2	3	4
0	0	1	1	1	0
1	0	0	1	0	0
2	1	1	0	0	0
3	0	0	0	1	0
4	0	1	0	0	0

Иерархия структур в Java



Java Collection Framework – иерархия интерфейсов и их реализаций, которая является частью JDK

Списки Java

List в Java Collection Framework

```
List<?> arrayList = new ArrayList<Integer>();
List<?> linkedList = new LinkedList<Integer>();
List<?> vectorList = new Vector<Integer>();
```

```
List<Integer> arrayList = new ArrayList<>();
List<Integer> vectorList = new Vector<>();

final int arraySize = 300_000_000;
long startTime = System.currentTimeMillis();

for (int i = 0; i < arraySize; i++) {
    arrayList.add(1);
}
System.out.println("ArrayList: timeMs - " + (System.currentTimeMillis() - startTime) );
arrayList.clear();

startTime = System.currentTimeMillis();
for (int i = 0; i < arraySize; i++) {
    vectorList.add(1);
}
System.out.println("Vector: timeMs - " + (System.currentTimeMillis() - startTime) );
vectorList.clear();

// N = 100_000_000
// ArrayList: timeMs - 3151
// Vector: timeMs - 4572

// N = 300_000_000
// ArrayList: timeMs - 10015
// Vector: timeMs - 10367
// Reversed
// Vector: timeMs - 10140
// ArrayList: timeMs - 7425
```

List – интерфейс, предоставляющий возможность поддерживать упорядоченную структуру

Упорядоченная структура (Списки)

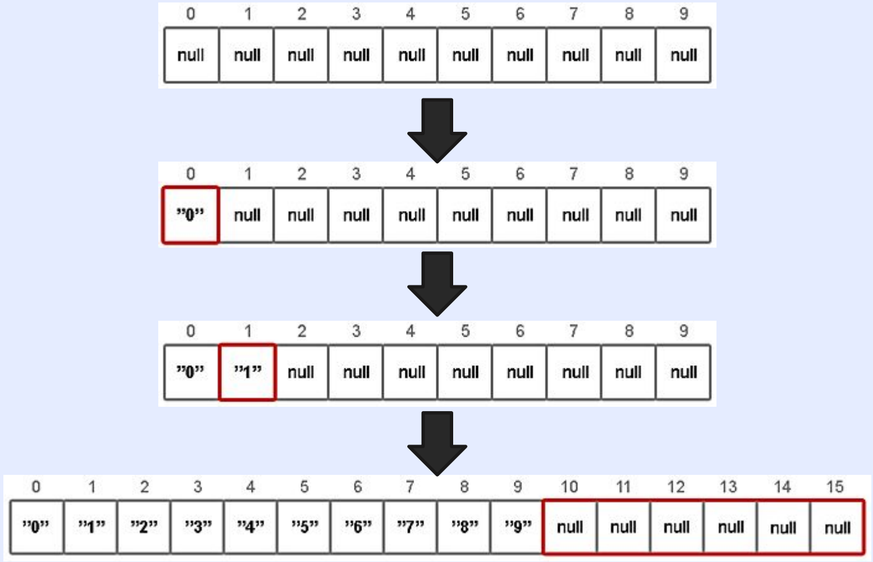
ArrayList - представляет собой динамический массив

LinkedList – представляет собой двухсвязный список

Vector – представляет собой динамический массив

ArrayList vs Vector?

ArrayList<Object>




```java
ArrayList<Integer> list = new ArrayList<>();

list.add(1);  // [1]
list.add(2);  // [1, 2]
list.addLast( element: 5);  // [1, 2, 5]
list.addFirst( element: 3);  // [3, 1, 2, 5]

System.out.println(list.get(0));  // 3
System.out.println(list.contains(5)); // true

list.remove( index: 3); // [3, 1, 2]

System.out.println(list.contains(5)); // false
```

ArrayList – реализует интерфейс List, также может менять свой размер во время исполнения

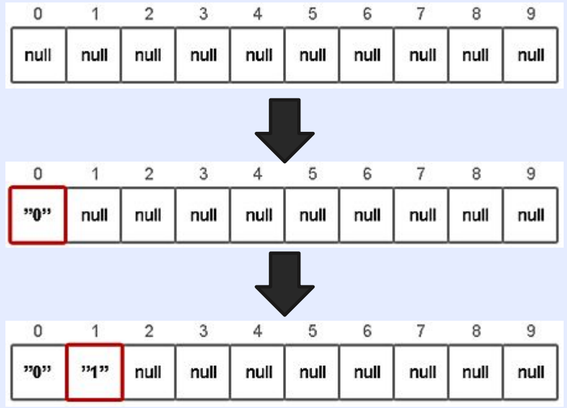
ArrayList (Список массива)

Основные методы ArrayList –
add – добавить элемент O(1)
get – получить O(1)
remove – удалить O(n)
addFirst– добавить в начало O(n)
addLast – добавить в конец O(1)

new ArrayList() – создаст объект с пустым массивом внутри.
add(new Object) – добавит объект и проинициализирует массив с размером new Object[10] {}

Если добавляется 10-ый элемент, то стартует копирование массива и создание нового **Object[10] -> new Object[N] {}.**
N = oldCapacity >> 1

Vector<Object>




```
Vector<Integer> list = new Vector<>();

list.add(1);  // [1]
list.add(2);  // [1, 2]
list.addLast( e: 5);  // [1, 2, 5]
list.addFirst( e: 3);  // [3, 1, 2, 5]

System.out.println(list.get(0));  // 3
System.out.println(list.contains(5)); // true

list.remove( index: 3); // [3, 1, 2]

System.out.println(list.contains(5)); // false
```

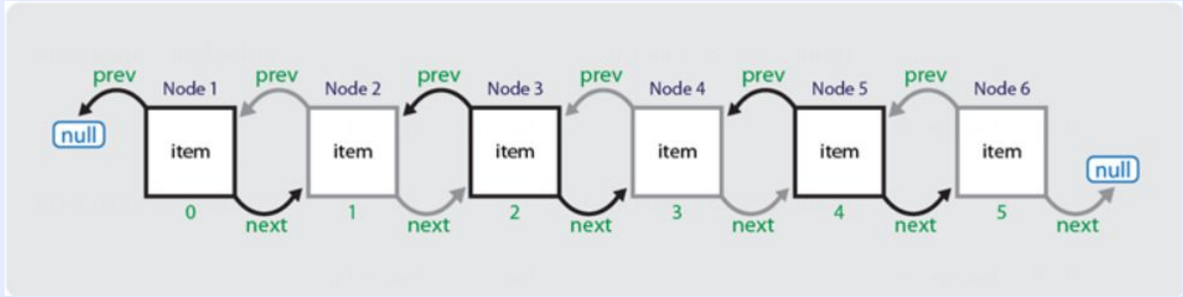
Vector – реализует интерфейс List, также может менять свой размер во время исполнения, схож на ArrayList

Vector (Синхронизированный список массива)

Основные методы Vector –
add – добавить элемент O(1)
get – получить O(1)
remove – удалить O(n)
new Vector() – создаст объект с массивом внутри размером 10 элементов.
add(new Object) – добавит объект в массив

Если добавляется 10-ый элемент, то стартует копирование массива и создание нового **Object[10] -> new Object[N] {}.**
N = increment > 0 ? Increment : oldCapacity

LinkedList<Object>



LinkedList – реализует интерфейс List, Deque, является двухсвязным списком объектов.

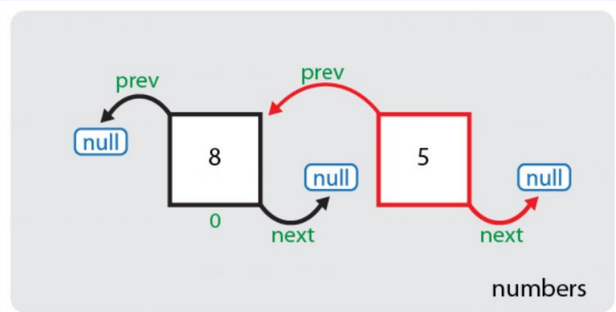
```
LinkedList<Integer> list = new LinkedList<>();
list.add(1);  // 1
list.add(2);  // 1 -> 2
list.add( index: 1,  element: 3);  // 1 -> 3 -> 2
list.remove( index: 2);  // 1 -> 3
System.out.println(list);
```

LinkedList

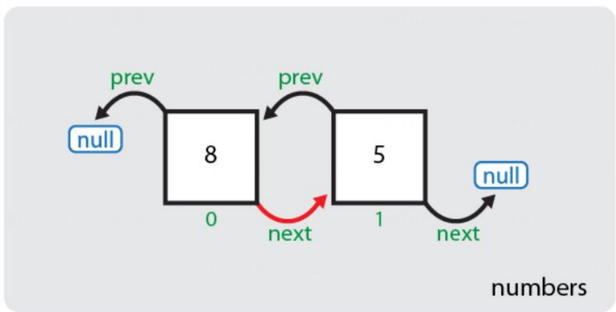
Содержит всего 3 поля –
size – размер
Node<E> first – первый элемент
Node<E> last – последний элемент
add() – добавить в конец. O(1)
remove()– удалить элемент O(n)

LinkedList – создается пустым и при добавлении нового элемента создается новый объект класса Node<E> и он связывается с предыдущем и следующим

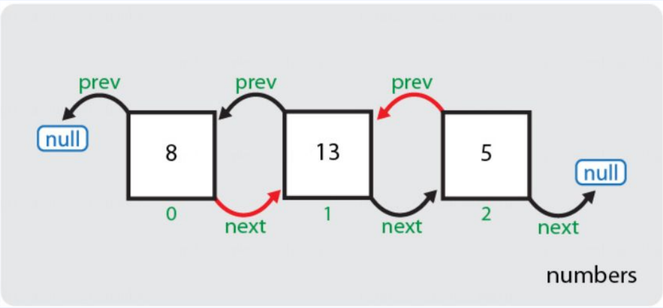
Класс Node<E>



size	1
first	null 8 null
last	8 5 null



size	2
first	null 8 5
last	8 5 null



size	3
first	null 8 13
last	13 5 null

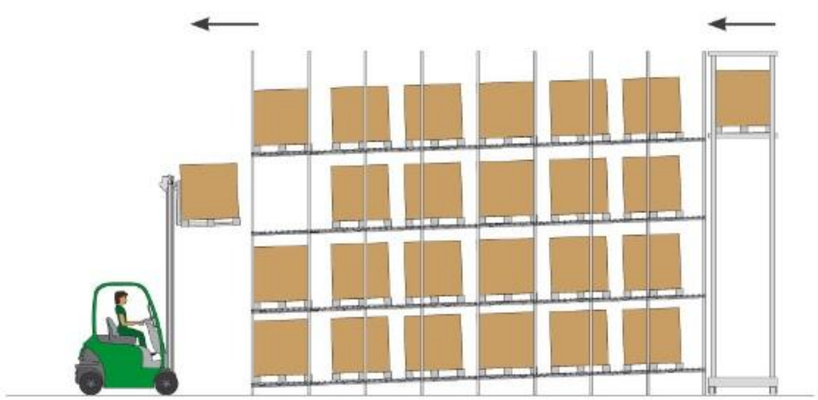
```
private static class Node<E> {
    35 usages
    E item;
    28 usages
    Node<E> next;
    19 usages
    Node<E> prev;

    4 usages
    Node(Node<E> prev, E element, Node<E> next) {
        this.item = element;
        this.next = next;
        this.prev = prev;
    }
}
```

Node<E> – вспомогательный закрытый класс для реализации двойной связи объектов.

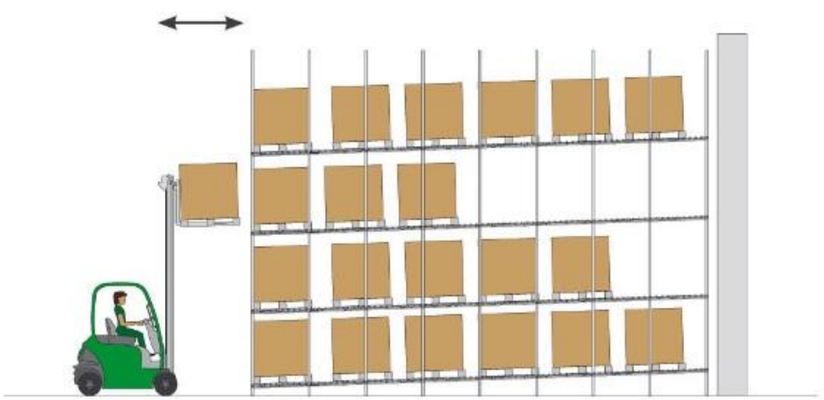
Очередь и стек Java

LIFO and FIFO. Вспомним

FIFO - стойки с непрерывным потоком

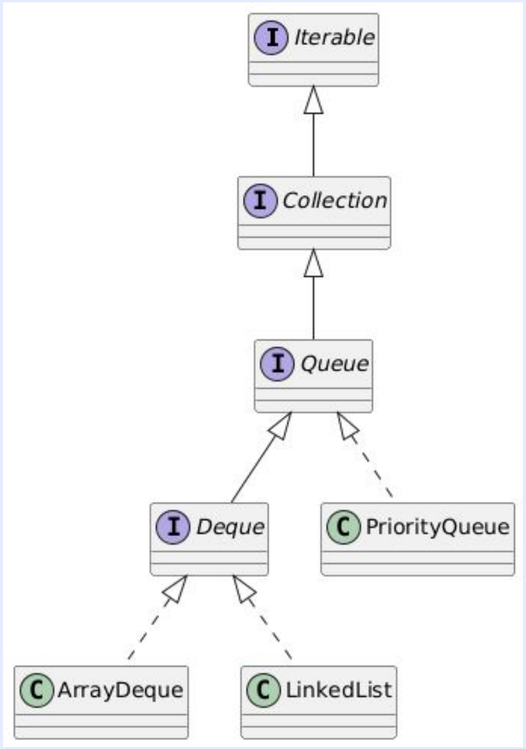


LIFO - откидные стойки



Queue в Java Collection Framework

```
Queue<?> arrayQueue = new ArrayDeque<>();
Queue<?> linkedQueue = new LinkedList<>();
Queue<?> syncQueue = new SynchronousQueue<>();
```



Queue – интерфейс, предоставляющий возможность поддерживать упорядоченную структуру, в виде очереди

Queue

Queue – используется для создания структуры данных очереди, в формате LIFO, FIFO

Основная особенность – это организация добавления и удаления элементов.

Методы –

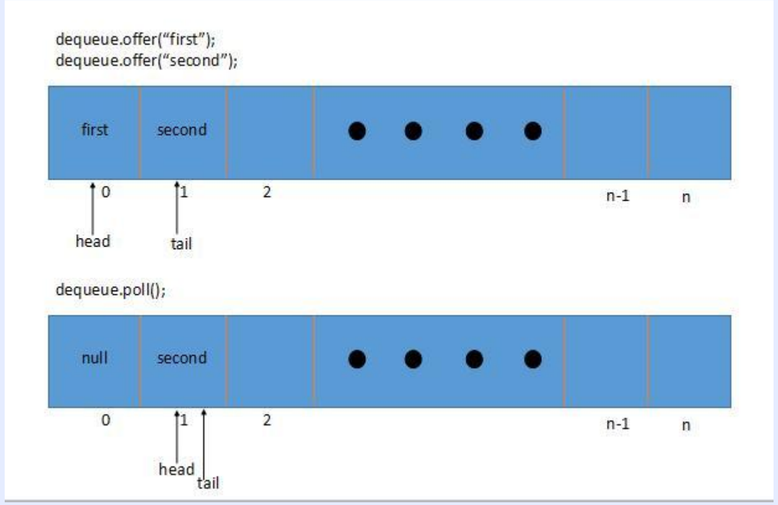
peek() – возвращать элемент без удаления из очереди

poll() – возвращает элемент и удаляет его

offer() – добавляет элемент в конец очереди

Deque – определяет поведение двунаправленной очереди по принципу LIFO

ArrayDeque<Object>




```
ArrayDeque<Integer> queue = new ArrayDeque<>();

queue.push( e: 1);  // [1]
queue.add(2);  // [1, 2]
queue.offer( e: 3);  // [1, 2, 3]

queue.remove();  // [2, 3]
queue.pop();  // [3]
queue.poll(); // []

System.out.println(queue);
```

ArrayDeque – двунаправленная очередь, реализованная на массиве

ArrayDeque

Основные методы –

add() – добавить в конец очереди O(1)

push() – добавить в начало очереди O(1)

offer() – добавить в конец очереди O(1)

remove() – удалить первый элемент из очереди O(1)

pop() – удалить первый элемент из очереди O(1)

poll() – удалить первый элемент из очереди O(1)

new ArrayDeque() – создаст объект с массивом внутри размер 17.

Рост вычисляется как -> вычисляется размер «прыжка», далее если размер прыжка больше чем необходимое количество поднять, то поднимается на уровень прыжка, если уровень прыжка меньше, то на необходимое кол-во

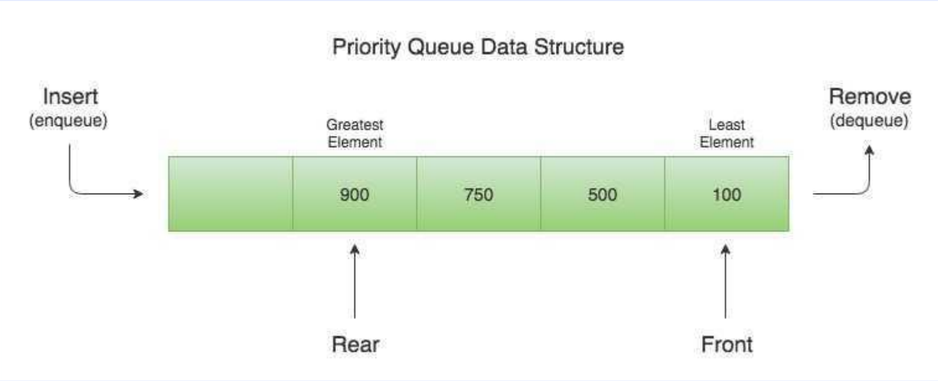
Рост объектов в ArrayDeque<Object>

```
private void grow(int needed) {
    // overflow-conscious code
    final int oldCapacity = elements.length;
    int newCapacity;
    // Double capacity if small; else grow by 50%
    int jump = (oldCapacity < 64) ? (oldCapacity + 2) : (oldCapacity >> 1);
    if (jump < needed
        || (newCapacity = (oldCapacity + jump)) - MAX_ARRAY_SIZE > 0)
        newCapacity = newCapacity(needed, jump);
    final Object[] es = elements = Arrays.copyOf(elements, newCapacity);
    // Exceptionally, here tail == head needs to be disambiguated
    if (tail < head || (tail == head && es[head] != null)) {
        // wrap around; slide first leg forward to end of array
        int newSpace = newCapacity - oldCapacity;
        System.arraycopy(es, head,
                         es, destPos: head + newSpace,
                         length: oldCapacity - head);
        for (int i = head, to = (head += newSpace); i < to; i++)
            es[i] = null;
    }
}
```

```
private int newCapacity(int needed, int jump) {
    final int oldCapacity = elements.length, minCapacity;
    if ((minCapacity = oldCapacity + needed) - MAX_ARRAY_SIZE > 0) {
        if (minCapacity < 0)
            throw new IllegalStateException("Sorry, deque too big");
        return Integer.MAX_VALUE;
    }
    if (needed > jump)
        return minCapacity;
    return (oldCapacity + jump - MAX_ARRAY_SIZE < 0)
        ? oldCapacity + jump
        : MAX_ARRAY_SIZE;
}
```

ArrayDeque – двунаправленная очередь, реализованная на массиве

PriorityQueue<Object>



PriorityQueue

По умолчанию создается простая очередь без приоритета с размером массива 11 элементов

add() – добавить в конец очереди O(log n)

offer() – добавить в конец очереди O(log n)

remove() – удалить первый элемент из очереди O(1)

poll() – удалить первый элемент из очереди O(1)

Priority Queue – очередь с приоритетом, реализующий интерфейс Queue

```
PriorityQueue<Integer> queue = new PriorityQueue<>();

queue.add(5);  // [5]
queue.add(2);  // [2, 5]
queue.add(3);  // [2, 5, 3]

System.out.println(queue.poll());  // 2
System.out.println(queue.poll());  // 3
System.out.println(queue.poll());  // 5
```

Приоритет устанавливается с помощью интерфейсов для работы с структурами данных.

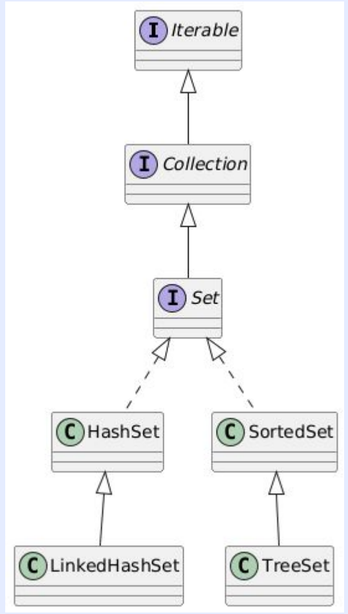
Множества в Java

Set в Java Collection Framework

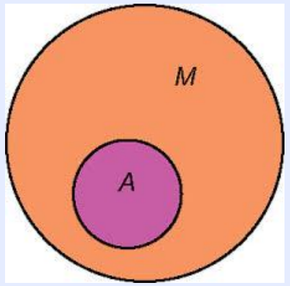
```
Set<?> hashSet = new HashSet<>();
Set<?> treeSet = new TreeSet<>();
Set<?> linkedHashSet = new LinkedHashSet<>();
```

Set (множества)

Множество – главная особенность множество, это невозможность добавить один и тот же элемент
{ 1, 2, 3, 4, 5} – добавить { 1 } -> { 1, 2, 3, 4, 5}



Методы –
add() – добавить элемент если существует, игнорировать
remove() – удалить элемент

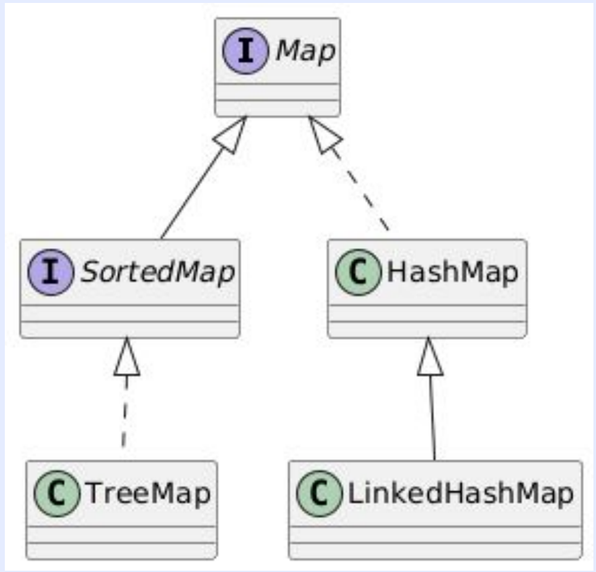


Set – интерфейс, предоставляющий возможность поддерживать структуру множеств

Map в Java Collection Framework

```
Map<?, ?> hashMap = new HashMap();
Map<?, ?> treeMap = new TreeMap();
Map<?, ?> linkedHashMap = new LinkedHashMap();
```

Map – интерфейс, предоставляющий структуру данных ключ<->значение



Map

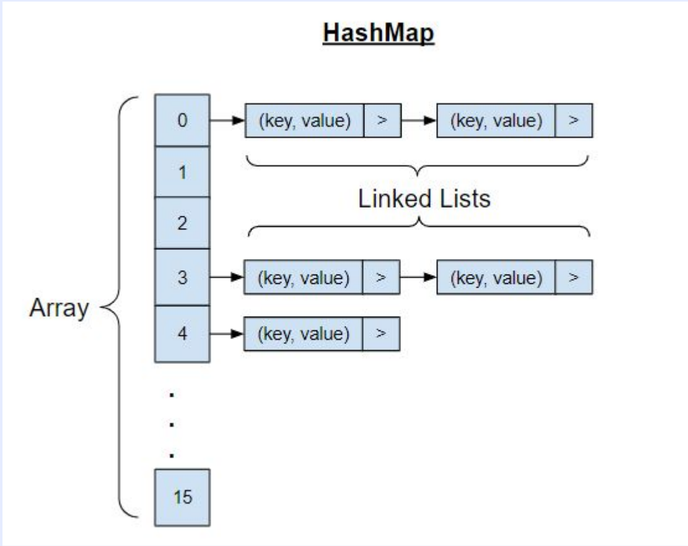
Множество – главная особенность множество, это невозможность добавить один и тот же элемент
{ 1, 2, 3, 4, 5} – добавить { 1 } -> { 1, 2, 3, 4, 5}

Методы –

V get(key) – получить элемент значение O(1)

put(key, value) – добавить элемент O(1)

HashMap<Object>



HashMap – хэш-таблица, реализующая интерфейс структуры ключ-значения (карта)

```
HashMap<String, String> map = new HashMap<>();

map.put("1", "apple");  // {1: apple}
map.put("2", "cherry");  // {1: apple, 2: cherry}

map.put("1", "banana");  // {1: banana, 2: cherry}

System.out.println(map.get("1"));  // banana
System.out.println(map.get("42")); // null
```

HashSet

Основные методы –

put(key, value) – добавить элемент O(1)

V get(key) – получить элемент O(1)

new HashMap() – создаст объект с размером по умолчанию 16 элементов.

Рост вычисляется как -> инициализируется дополнительный параметр loadFactor и по-умолчанию = 0.75 и дальше работает сложная функция

```java
public V put(K key, V value) {
    return putVal(hash(key), key, value, onlyIfAbsent: false, evict: true);
}
```

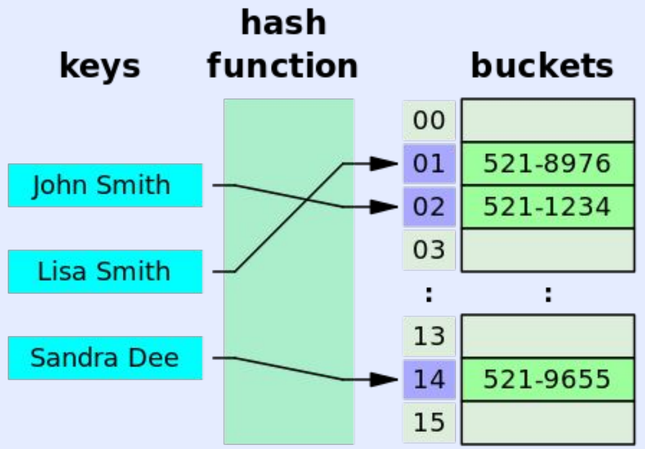
```java
final V putVal(int hash, K key, V value, boolean onlyIfAbsent,
               boolean evict) {
    Node<K, V>[] tab; Node<K, V> p; int n, i;
    if ((tab = table) == null || (n = tab.length) == 0)
        n = (tab = resize()).length;
    if ((p = tab[i = (n - 1) & hash]) == null)
        tab[i] = newNode(hash, key, value, next: null);
    else {
        Node<K, V> e; K k;
        if (p.hash == hash &&
            ((k = p.key) == key || (key != null && key.equals(k))))
            e = p;
        else if (p instanceof TreeNode)
            e = ((TreeNode<K, V>) p).putTreeVal( map: this, tab, hash, key, value);
        else {
            for (int binCount = 0; ; ++binCount) {
                if ((e = p.next) == null) {
                    p.next = newNode(hash, key, value, next: null);
                    if (binCount >= TREEIFY_THRESHOLD - 1) // -1 for 1st
                        treeifyBin(tab, hash);
                    break;
                }
                if (e.hash == hash &&
                    ((k = e.key) == key || (key != null && key.equals(k))))
                    break;
                p = e;
            }
        }
        if (e != null) { // existing mapping for key
            V oldValue = e.value;
            if (!onlyIfAbsent || oldValue == null)
                e.value = value;
            afterNodeAccess(e);
            return oldValue;
        }
    }
    ++modCount;
```

```java
public V get(Object key) {
    Node<K, V> e;
    return (e = getNode(key)) == null ? null : e.value;
}
```

```java
final Node<K, V> getNode(Object key) {
    Node<K, V>[] tab; Node<K, V> first, e; int n, hash; K k;
    if ((tab = table) != null && (n = tab.length) > 0 &&
            (first = tab[(n - 1) & (hash = hash(key))]) != null) {
        if (first.hash == hash && // always check first node
                ((k = first.key) == key || (key != null && key.equals(k))))
            return first;
        if ((e = first.next) != null) {
            if (first instanceof TreeNode)
                return ((TreeNode<K, V>) first).getTreeNode(hash, key);
            do {
                if (e.hash == hash &&
                        ((k = e.key) == key || (key != null && key.equals(k))))
                    return e;
            } while ((e = e.next) != null);
        }
    }
    return null;
}
```

HashSet<Object>

```java
public class HashSet<E>
        extends AbstractSet<E>
        implements Set<E>, Cloneable, java.io.Serializable
{
    @java.io.Serial
    static final long serialVersionUID = -5024744406713321676L;

    transient HashMap<E, Object> map;

    // Dummy value to associate with an Object in the backing Map
    static final Object PRESENT = new Object();
```



HashSet

Основные методы –
add() – добавить элемент O(1)
remove() – удалить элемент O(1)

new HashSet() – создаст объект с размером по умолчанию 16 элементов.

Под капотом используется HashMap, самый чистый пример агрегации в JCF

HashSet – хэш-таблица, реализующая интерфейс множеств

```java
HashSet<String> set = new HashSet<>();

set.add("apple");  // {apple}
set.add("banana");  // {apple, banana}
set.add("cherry");  // {apple, banana, cherry}

System.out.println(set.contains("apple"));  // true

set.add("apple");  // {apple, banana, cherry}

System.out.println(set.contains("Jojo"));  // false
```

```
public boolean contains(Object o) { return map.containsKey(o); }
```

```
public boolean containsKey(Object key) {
    return getNode(key) != null;
}
```

```
public boolean add(E e) { return map.put(e, PRESENT) == null; }
```

```
final V putVal(int hash, K key, V value, boolean onlyIfAbsent,
               boolean evict) {
```

Сравнение базовых операций
интерфейс Collection

	add(E e)	**remove(Object o)**	**contains(Object o)**
ArrayList	O(1)*	O(N)	O(N)
Vector	O(1)*	O(N)	O(N)
LinkedList	O(1)	O(N)	O(N)
ArrayDeque	O(1)*	O(N)	O(N)
PriorityQueue	O(logN)	O(logN)	O(N)
HashMap	O(1)±	O(1)±	O(1)±
HashSet	O(1)±	O(1)±	O(1)±

O(1)± - В среднем случае O(1), в худшем - O(logN)
O(1)* - амортизированная сложность, при расширении O(N)

Интерфейс List

	add(int index, E e)	set(int index, E e)	remove(int index)	get(int index)
ArrayList	O(N)	O(1)	O(N)	O(1)
Vector	O(N)	O(1)	O(N)	O(1)
LinkedList	O(N)	O(N)	O(N)	O(N)

Интерфейс Queue

	offer(E e)	peek()	poll()
ArrayDeque	O(1)*	O(1)	O(1)
PriorityQueue	O(logN)	O(1)	O(logN)
LinkedList	O(1)	O(1)	O(1)

Интерфейс Map

	put(K key, V value)	get(Object key)	remove(Object key)	containsKey(Object o)	containsValue(Object o)
HashMap	O(1)*	O(1)	O(1)	O(1)	O(N)
LinkedHashMap	O(1)	O(1)	O(1)	O(1)	O(N)
TreeMap	O(logN)	O(logN)	O(logN)	O(N)	O(N)