Java APIs

Creator: Simon Zhang

website link

java10

```
var companyToEmployees= new HashMap<String, List<String>>();
  for (var entry: companyToEmployees. entrySet()) {
       var employees= entry.getValue();
}
```

Copy

```
System.arraycopy(Object src, int srcPos, Object dest, int destPos, int
length)
Arrays.copyOf(originalArray, newLength) -> O(1)
Arrays.copyOfRange(originalArray, start, end) -> O(n), [start, end)

list.addAll(Collection c); // List<List<String>> res
Collections.addAll(this.list, source);
```

ASCII

```
int[] regular = new int[128];
int[] extend = new int[256];
'a' = 97;
'A' = 65;
'0' = 48;
```

Arrays

```
Arrays.binarySearch(arr, target) -> O(nlogn)
Arrays.sort(array)
Arrays.sort(Object[] array, Collections.reverseOrder());
Arrays.fill(array, number);
Arrays.toString()
Arrays.asList(Integer[] arr) -> O(1) only accept object, 2sum -> 3sum
Arrays.copyOf(originalArray, newLength) -> O(1)
Arrays.copyOfRange(originalArray, start, end) -> O(n), [start, end)
```

```
Arrays.equals(array1, array2) // compare contents
comparable -> compareTo(secondNum)
```

ArrayList

```
List<Integer> list = new ArrayList<>();
list.add(E e);
list.add(int index, E e); // random access
list.addAll(Collection c); // List<List<String>> res
Collections.addAll(this.list, source);


list.get(int index);
list.remove(int index);
list.remove(E e);
list.removeAll(Arrays.asList("acbd")); // remove all "acbd"
list.clear(); // faster than removeAll
list.set(int index, E e);
// list.get(i) += 5 is WRONG, since left hand side is value, not variable,
use
list.set(i, list.get(i) + 5);

list.toArray(); // Object[]
list.toArray(new int[list.size()][]) // list<int[]> -> int[][]

Collections.sort(list, new myComparator());
Comparator -> compare(item1, item2);
Collections.reverse(list);
list.subList(start, end).clear();
list.forEach(k -> sb.append(k));

// array to list
List<Integer> sourceList = Arrays.asList(0, 1, 2, 3, 4, 5); // fixed size
List<Integer> targetList = new ArrayList<Integer>
(Arrays.asList(sourceArray)); // dymatical

// 2D array to list, need Integer
public <T> List<T> twoDArrayToList(T[][] twoDArray) {
    List<T> list = new ArrayList<T>();
    for (T[] array : twoDArray) {
        list.addAll(Arrays.asList(array)); // need Integer, not int
    }
    return list;
}

// int is OK
int[][] 2Darray = ...
List<List<Integer>> lists = new ArrayList<>();
for (int[] ints : 2Darray) {
    List<Integer> list = new ArrayList<>();
    for (int i : ints) {
```

```
            list.add(i);
        }
        lists.add(list);
    }

    // 2D array copy
    int [][] myInt = new int[matrix.length][];
    for(int i = 0; i < matrix.length; i++)
        myInt[i] = matrix[i].clone();


    // list to array
    Integer[] targetArray = sourceList.toArray(new Integer[0]);

    // 2D list to array
    List<int []> myList= new ArrayList<>();
    myList.add(new int[] {1,2});
    return myList.toArray(new int[myList.size()][]);

    String[][] array = new String[list.size()][];
    int i = 0;
    for (List<String> nestedList : list) {
        array[i++] = nestedList.toArray(new String[nestedList.size()]);
    }
```

HashMap

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

// immutable map
ImmutableMap<String, Integer> map = ImmutableMap.of(...);

// ordered map, by insertion
LinkedHashMap<String, Integer> map = new LinkedHashMap<>();

// sorted by key
TreeMap<String, Integer> map = new TreeMap<>();
TreeMap<Integer, List<Character>> frequencyToChar = new TreeMap<>
(Collections.reverseOrder());

treemap.ceilingKey(K key) // Returns the least key greater than or equal
to the given key, or null if there is no such key.
treemap.floorKey(K key) // Returns the greatest key less than or equal to
the given key, or null if there is no such key.
treemap.firstKey() // Returns the first (lowest) key currently in this
map.

map.get(key);
map.getOrDefault(key, defaultValue);

map.put(key, value);
```

```
map.putIfAbsent(key, value); // return null or current value

map.containsKey(key);
map.remove(key);

map.keySet()
map.values()
map.entrySet()

for (Map.Entry<String, Integer> entry : map.entrySet()) {
  String key = entry.getKey();
  Integer value = entry.getValue();
}

map.forEach((k, v) -> System.out.println(k + " " + v));

map.computeIfAbsent(key, k -> V.createFor(k));
prices.computeIfAbsent("Shirt", key -> 280)
prices.computeIfPresent("Shoes", (key, value) -> value + value * 10/100)
graph.computeIfAbsent(connection[0], k -> new ArrayList<>
()).add(connection[1]);
```

HashSet

```
Set<Integer> set = new HashSet<>();

// ordered (by insertion)
LinkedHashSet<Integer> set = new LinkedHashSet<>();

// sorted (object need comparator)
SortedSet<Integer> set = new TreeSet<>();

set.add(key);
set.contains(key);
set.remove(key); // return true / false
set.toArray();
Collections.addAll(Set<String> set, String[] element); // array to set, 去
重, de-duplicate
```

Integer

```
int i = Integer.parseInt(String);
String str = Integer.toString(number)
Math.round(double) // to nearest integer
Math.ceil(4.3) // return 5.0, double
```

PriorityQueue / Heap

```
// min heap
Queue<X> minHeap = new PriorityQueue<>(); // default min heap

// max heap
Queue<X> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
```

Deque

```
Deque<X> dq = new ArrayDeque<>();
dp.offerFirst(E e); // offerLast(E e)
dp.pollFirst(); // pollLast()
dp.peekFirst(); // peekLast()
```

Queue

```
Queue<X> queue = new ArrayDeque<>();
Queue<X> queue = new LinkedList<>();

// return null
queue.offer(E e);
queue.poll();
queue.peek();

// throw exception
queue.add(E e);
queue.remove();
queue.elements();
```

Stack

~~Stack stack = new Stack<>();~~

```
Deque<X> stack = new ArrayDeque<>();

stack.push(E e); // stack.addFirst(E, e); throw exception
stack.pop(); // stack.removeFirst();
stack.peek(); // stack.getFirst();

// return null
stack.offerFirst(E e);
stack.pollFirst();
stack.peekFirst();
```

LinkedList

- getFirst() / peekFirst()
- getLast() / peekLast()
- addFirst(e) / offerFirst(e)
- removeFirst() / pollFirst()
- List<> list = new LinkedList<>();
- pop = removeFirst
- add = addLast

String:

```
s.length()
s.isEmpty()
s.indexOf(char) // return -1 if not found
s.indexOf(String str, int start) // first position of str >= start
s.lastIndexOf(char)
s.contains("word")

str1.compareTo(str2)
s.equals(s2) // compare Unicode
str2.equalsIgnoreCase(str1);
s1.regionMatches(ignoreCase = true, start = 0, other = "Hello", otherStart
= 5, len = 5);
s.startsWith(Stirng prefix, int start = 0)
s.endsWith(str)

// return new String
s.trim()
s.replace(" ", "-"); // repalce all old char or String or char sequence
x.replaceAll("\\s+", ""); // remove white space, regular expression
s.replaceAll("[^A-Za-z\\d]+", "").toLowerCase(); // ^ = !, `+` = [1, inf]
s.toUpperCase()
s.isUpperCase()

// convert
String.valueOf(num); // int i=10; Now it will return "10", int / char to
string
new String(char[] charArray)
Integer.toString(number) // int to string
String.join(",", char[]); "a,b,c"

String s = new String(char[]) // char array to string
String s = new String(char[], int start, int count)

s.split("/") // String[]

s.substring(start)
s.substring(start, end)
s.toCharArray()

word.matches("[A-Z]*|.[a-z]*") // regex, valid captical: USA, leetcode,
Hello
```

StringBuilder

```
StringBuilder sb = new StringBuilder() // contructor: (int capacity/
CharSequence seq / String str)
sb.append('a' / char[] / int / boolean)
sb.delete(int start, int end)
sb.deleteCharAt(sb.length() - 1);
sb.insert(int index, char... ch) // O(n)

sb.setLength(int newLength) // clear
sb.setCharAt(int index, char ch)
sb.subSequence(int start, int end) // return CharSequence
StringBuilder(sb).reverse().toString();
sb.replace(start, end, str)
```

Character

```
Character.toLowerCase(c)
Character.isLetterOrDigit(c)
```

char

```
string.charAt(index) - 'a' // char -> a-z / ASCII 128 char
string.charAt(index) - '0' // char to int
Character.toString(char c) // char -> String
String.valueOf(char c) // char -> String
(char)(10 + '0') // int to char
```

Collection

```
Collections.binarySearch(collection, target)
Collections.sort(list, new myComparator())
Collections.reverse(arrayList)
Collections.max(map.values())
Collections.min(int[] nums)
Collections.addAll(to, from)
Collections.shuffle(List<?> list)
```

Class Creation

Comparator

```
PriorityQueue<Integer> queue = new PriorityQueue<>((a, b) -> b - a); //
biggest pop first, 30 20 10

PriorityQueue<Map.Entry<String, Integer>> pq = new PriorityQueue<>(
  (a, b) -> a.getValue().equals(b.getValue()) ?
  a.getKey().compareTo(b.getKey()) :
  a.getValue() - b.getValue()
); // Max heap that contains pairs - if values for pairs are the same,
// then they will be sorted ascending (a-z) according to key

// 少用，有些平台不支持 Comparator
private class myComparator implements Comparator<Integer> {
  @Override
  public int compare(Integer i1, Integer i2) {
    if (i1.equals(i2)) {
      return 0;
    }
    return i1 > i2 ? -1 : 1;
  }
}
```

```
Pair<Integer, String> pair = new Pair<>(1, "One");
  Integer key = pair.getKey();
  String value = pair.getValue();

private Object[] getPair() {
    // ...
    return new Object[] {key, value};
}
```