

# Java APIs

---

Creator: [Simon Zhang](#)

[website link](#)

## 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(int[] arr) -> O(1) only accept object
Arrays.copyOf(originalArray, newLength) -> O(1)
Arrays.copyOfRange(originalArray, start, end) -> O(n)
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.get(int index);
list.remove(int index);
list.remove(E e);
list.clear(); // faster than removeAll
list.set(int index, E e);
list.toArray(); // Object[]

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

## HashMap

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

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

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

// sorted map
SortedMap<String, Integer> map = new TreeMap<>();

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));
```

## 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();
```

## Integer

```
int i = Integer.parseInt(String);  
String str = Integer.toString(number)
```

## PriorityQueue / Heap

```
// min heap  
Queue<X> minHeap = new PriorityQueue<>();  
  
// 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<>();  
  
// throw exception  
queue.add(E e);  
queue.remove();  
queue.elements();  
  
// return null  
queue.offer(E e);  
queue.poll();  
queue.peek();
```

## 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<>();

## String:

- s.length()
- s.isEmpty()
- s.equals(s2)
- s.contains("word") -> true/false
- s.indexOf(char)
- s.lastIndexOf(char)
- s.startsWith(str)
- s.endsWith(str)
- s.substring(start)
- s.substring(start, end)
- s.toCharArray()
- String s = new String(char[])
- s.split("/") -> String[]
- s.split(",");
- s.trim()
- s.replaceAll("[^A-Za-z\d]+", "").toLowerCase();
- String.valueOf(num); //int i=10; Now it will return "10"
- Integer.toString(number)
- String.join(",", char[]); "a,b,c"

## StringBuilder

[StringBuilder] **StringBuilder sb = new StringBuilder()**

- contructor: (int capacity/ CharSequence seq / String str)
- sb.append('a' / char[] / int / boolean)
- sb.deleteCharAt(sb.length() - 1);
- sb.size()
- sb.insert(int index, char ch) -> O(n)
- sb.setLength(int newLength)

- sb.setCharAt(int index, char ch)
- CharSequence sb.subSequence(int start, int end)
- StringBuilder(sb).reverse().toString();

## Character

- Character.toLowerCase(c)
- Character.isLetterOrDigit(c)

## char

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

## Collection

- Collections.binarySearch(collection, target)
- Collections.sort(list, new myComparator())
- Collections.reverse(arrayList)

## Class Creation

---

### Comparator

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

```
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};  
}
```