public class Arrays

This class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists.

The methods in this class all throw a `NullPointerException`, if the specified array reference is null, except where noted.

The documentation for the methods contained in this class includes briefs description of the *implementations*. Such descriptions should be regarded as *implementation notes*, rather than parts of the *specification*. Implementors should feel free to substitute other algorithms, so long as the specification itself is adhered to. (For example, the algorithm used by `sort(Object[])` does not have to be a MergeSort, but it does have to be *stable*.)

This class is a member of the Java Collections Framework.

Since: 1.2

Java.utill.Array

asList()	public static <T> List<T> asList (T… a)
binarySearch()	public static int binarySearch (int[] a, int key)
copyOf()	public static int[] copyOf (int[] original, int newLength)
copyOfRange()	public static int[] copyOfRange (int[] original, int from, int to)
Sort()	public static void sort (int[] a)
toString()	public static String toString (int[] a)
Stream()	public static IntStream stream (int[] array)
Fill()	public static void fill (int[] a, int val)
Equals()	public static boolean equals (int[] a, int[] a2)
hashCode()	public static int hashCode (int[] a)
deepEquals()	public static boolean deepEquals (Object[] a1, Object[] a2)
deepToString()	public static String deepToString (Object[] a)
deepHashCode()	public static int deepHashCode (Object[] a)
parallelSort()	public static void parallelSort (int[] a)
ParallelPrefix()	public static void parallelPrefix (int[] array, IntBinaryOperator op)
parallelSetAll()	public static void parallelSetAll (int[] array, IntUnaryOperator generator)
setAll()	public static void setAll (int[] array, IntUnaryOperator generator)
Spliterator()	public static OfInt spliterator (int[] array)

@SafeVarargs @SuppressWarnings({"varargs"})

1. public static <T> List<T> asList (T… a)

Returns a fixed-size list backed by the specified array. (Changes to the returned list "write through" to the array.) This method acts as bridge between array-based and collection-based APIs, in combination with `Collection.toArray`. The returned list is serializable and implements `RandomAccess`.

This method also provides a convenient way to create a fixed-size list initialized to contain several elements:

```
List<String> stooges = Arrays.asList("Larry", "Moe", "Curly");
```

Parameters:

`<T>`	the class of the objects in the array
`a`	the array by which the list will be backed

Returns: a list view of the specified array

2. public static int binarySearch (int[] a, int key)

Searches the specified array of shorts for the specified value using the binary search algorithm. The array must be sorted (as by the `sort(short[])` method) prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

Parameters:

`a`	the array to be searched
`key`	the value to be searched for

Returns: index of the search key, if it is contained in the array; otherwise, `(-(`*insertion point*`) - 1)`. The *insertion point* is defined as the point at which the key would be inserted into the array: the index of the first element greater than the key, or `a.length` if all elements in the array are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

This method has overloaded for more info go to: https://www.falkhausen.de/Java-8/java.util/Arrays.html

3. public static int[] copyOf (int[] original, int newLength)

Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. For all indices that are valid in both the original array and the copy, the two arrays will contain identical values. For any indices that are valid in the copy but not the original, the copy will contain `0`. Such indices will exist if and only if the specified length is greater than that of the original array.

Parameters:

`original`	the array to be copied
`newLength`	the length of the copy to be returned

Returns: a copy of the original array, truncated or padded with zeros to obtain the specified length

Exceptions:

`NegativeArraySizeException`	if `newLength` is negative
`NullPointerException`	if `original` is null

Since: 1.6 . This method has overloaded for more info go to: https://www.falkhausen.de/Java-8/java.util/Arrays.html

4. public static void sort (int[] a)

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers O(n log(n)) performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Parameters:

a the array to be sorted

This method has overloaded for more info go to: https://www.falkhausen.de/Java-8/java.util/Arrays.html

Ex:

```
public static void sort (Object[] a, int fromIndex, int toIndex)

public static <T> void sort (T[] a, Comparator<? super T> c)  and many more.
```

5. public static int[] copyOfRange (int[] original, int from, int to)

Copies the specified range of the specified array into a new array. The initial index of the range (`from`) must lie between zero and `original.length`, inclusive. The value at `original[from]` is placed into the initial element of the copy (unless `from == original.length` or `from == to`). Values from subsequent elements in the original array are placed into subsequent elements in the copy. The final index of the range (`to`), which must be greater than or equal to `from`, may be greater than `original.length`, in which case `0` is placed in all elements of the copy whose index is greater than or equal to `original.length - from`. The length of the returned array will be `to - from`.

Parameters:

`original`	the array from which a range is to be copied
`from`	the initial index of the range to be copied, inclusive
`to`	the final index of the range to be copied, exclusive. (This index may lie outside the array.)

Returns: a new array containing the specified range from the original array, truncated or padded with zeros to obtain the required length

Exceptions:

`ArrayIndexOutOfBoundsException`	if `from < 0` or `from > original.length`
`IllegalArgumentException`	if `from > to`
`NullPointerException`	if `original` is null

Since: 1.6 This method has overloaded for more info go to:
https://www.falkhausen.de/Java-8/java.util/Arrays.html

6. public static String toString (int[] a)

Returns a string representation of the contents of the specified array. The string representation consists of a list of the array's elements, enclosed in square brackets (`"[]"`). Adjacent elements are separated by the characters `", "` (a comma followed by a space). Elements are converted to strings as by `String.valueOf(int)`. Returns `"null"` if `a` is `null`.

Parameters:

`a`	the array whose string representation to return

returns: a string representation of `a (over loaded method)`

7 . public static IntStream stream (int[] array)

Returns a sequential `LongStream` with the specified array as its source.

Parameters:

`array` the array, assumed to be unmodified during use

Returns: a `LongStream` for the array

Since: 1.8 https://www.falkhausen.de/Java-8/java.util/Arrays.html

8. public static void fill (int[] a, int val)

Assigns the specified int value to each element of the specified array of ints.

Parameters:

`a` the array to be filled
`val` the value to be stored in all elements of the array

Overloaded method

9. public static boolean equals (int[] a, int[] a2)

Returns `true` if the two specified arrays of ints are *equal* to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are `null`.

Parameters:

`a` one array to be tested for equality
`a2` the other array to be tested for equality

Returns: `true` if the two arrays are equal over-loaded

10. public static int hashCode (int[] a)

Returns a hash code based on the contents of the specified array. For any two non-null `int` arrays `a` and `b` such that `Arrays.equals(a, b)`, it is also the case that `Arrays.hashCode(a) == Arrays.hashCode(b)`.

The value returned by this method is the same value that would be obtained by invoking the `hashCode` method on a `List` containing a sequence of `Integer` instances representing the elements of `a` in the same order. If `a` is `null`, this method returns 0.

Parameters:

`a` the array whose hash value to compute

Returns: a content-based hash code for `a`

11. public static String deepToString (Object[] a)

Returns a string representation of the "deep contents" of the specified array. If the array contains other arrays as elements, the string representation contains their contents and so on. This method is designed for converting multidimensional arrays to strings.

The string representation consists of a list of the array's elements, enclosed in square brackets (`"[]"`). Adjacent elements are separated by the characters `", "` (a comma followed by a space). Elements are converted to strings as by `String.valueOf(Object)`, unless they are themselves arrays. If an element `e` is an array of a primitive type, it is converted to a string as by invoking the appropriate overloading of `Arrays.toString(e)`. If an element `e` is an array of a reference type, it is converted to a string as by invoking this method recursively.

To avoid infinite recursion, if the specified array contains itself as an element, or contains an indirect reference to itself through one or more levels of arrays, the self-reference is converted to the string `"[...]"`. For example, an array containing only a reference to itself would be rendered as `"[[...]]"`.

This method returns `"null"` if the specified array is `null`.

Parameters:

`a` the array whose string representation to return

Returns: a string representation of `a`

See also:
toString(Object[])

12. public static boolean deepEquals (Object[] a1, Object[] a2)

Returns `true` if the two specified arrays are *deeply equal* to one another. Unlike the `equals(Object[],Object[])` method, this method is appropriate for use with nested arrays of arbitrary depth.

Two array references are considered deeply equal if both are `null`, or if they refer to arrays that contain the same number of elements and all corresponding pairs of elements in the two arrays are deeply equal.

Two possibly `null` elements `e1` and `e2` are deeply equal if any of the following conditions hold:

- `e1` and `e2` are both arrays of object reference types, and `Arrays.deepEquals(e1, e2) would return true`
- `e1` and `e2` are arrays of the same primitive type, and the appropriate overloading of `Arrays.equals(e1, e2)` would return true.
- `e1 == e2`
- `e1.equals(e2)` would return true.

Note that this definition permits `null` elements at any depth.

If either of the specified arrays contain themselves as elements either directly or indirectly through one or more levels of arrays, the behavior of this method is undefined.

Parameters:

`a1` one array to be tested for equality
`a2` the other array to be tested for equality

Returns: `true` if the two arrays are equal

See also:
equals(Object[],Object[]), Objects.deepEquals(Object, Object)

13. public static int deepHashCode (Object[] a)

Returns a hash code based on the "deep contents" of the specified array. If the array contains other arrays as elements, the hash code is based on their contents and so on, ad infinitum. It is therefore unacceptable to invoke this method on an array that contains itself as an element, either directly or indirectly through one or more levels of arrays. The behavior of such an invocation is undefined.

For any two arrays `a` and `b` such that `Arrays.deepEquals(a, b)`, it is also the case that `Arrays.deepHashCode(a) == Arrays.deepHashCode(b)`.

The computation of the value returned by this method is similar to that of the value returned by `List.hashCode()` on a list containing the same elements as `a` in the same order, with one difference: If an element `e` of `a` is itself an array, its hash code is computed not by calling `e.hashCode()`, but as by calling the appropriate overloading of `Arrays.hashCode(e)` if `e` is an array of a primitive type, or as by calling `Arrays.deepHashCode(e)` recursively if `e` is an array of a reference type. If `a` is `null`, this method returns 0.

Parameters:

`a` the array whose deep-content-based hash code to compute

Returns: a deep-content-based hash code for `a`

See also:
hashCode(Object[])

14. public static void parallelSort (int[] a)

Sorts the specified array into ascending numerical order.

Parameters:

`a` the array to be sorted

Since: 1.8

@implNote The sorting algorithm is a parallel sort-merge that breaks the array into sub-arrays that are themselves sorted and then merged. When the sub-array length reaches a minimum granularity, the sub-array is sorted using the appropriate `Arrays.sort` method. If the length of the specified array is less than the minimum granularity, then it is sorted using the appropriate `Arrays.sort` method. The algorithm requires a working space no greater than the size of the original array. The `ForkJoin common pool` is used to execute any parallel tasks.

15. public static void parallelPrefix (int[] array, IntBinaryOperator op)

Cumulates, in parallel, each element of the given array in place, using the supplied function. For example if the array initially holds `[2, 1, 0, 3]` and the operation performs addition, then upon return the array holds `[2, 3, 3, 6]`. Parallel prefix computation is usually more efficient than sequential loops for large arrays.

Parameters:

`array`	the array, which is modified in-place by this method
`op`	a side-effect-free, associative function to perform the cumulation

Exceptions:

`NullPointerException`	if the specified array or function is null

16. public static void parallelSetAll (int[] array, IntUnaryOperator generator)

Set all elements of the specified array, in parallel, using the provided generator function to compute each element.

If the generator function throws an exception, an unchecked exception is thrown from `parallelSetAll` and the array is left in an indeterminate state.

Parameters:

`array`	array to be initialized
`generator`	a function accepting an index and producing the desired value for that position

Exceptions:

`NullPointerException`	if the generator is null

17. public static void setAll (int[] array, IntUnaryOperator generator)

Set all elements of the specified array, using the provided generator function to compute each element.

If the generator function throws an exception, it is relayed to the caller and the array is left in an indeterminate state.

Parameters:

`array` array to be initialized
`generator` a function accepting an index and producing the desired value for that position

Exceptions:

`NullPointerException` if the generator is null

18. public static OfInt spliterator (int[] array)

Returns a `Spliterator.OfInt` covering all of the specified array.

The spliterator reports `Spliterator.SIZED`, `Spliterator.SUBSIZED`, `Spliterator.ORDERED`, and `Spliterator.IMMUTABLE`.

Parameters:

`array` the array, assumed to be unmodified during use

Returns: a spliterator for the array elements

Package java.util.*

Arrays

Original API Documentation