| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Arrays.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/ArrayList.html)   [**NEXT CLASS**](http://docs.google.com/java/util/BitSet.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Arrays.html)    [**NO FRAMES**](http://docs.google.com/Arrays.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#tyjcwt) |

## **java.util**

Class Arrays

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **java.util.Arrays**

public class **Arrays**extends [Object](http://docs.google.com/java/lang/Object.html)

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](http://docs.google.com/technotes/guides/collections/index.html).

**Since:** 1.2

| **Method Summary** | |
| --- | --- |
| static   | <T> [List](http://docs.google.com/java/util/List.html)<T> | | --- | | [**asList**](http://docs.google.com/java/util/Arrays.html#asList(T...))(T... a)            Returns a fixed-size list backed by the specified array. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(byte%5B%5D,%20byte))(byte[] a, byte key)            Searches the specified array of bytes for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(byte%5B%5D,%20int,%20int,%20byte))(byte[] a, int fromIndex, int toIndex, byte key)            Searches a range of the specified array of bytes for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(char%5B%5D,%20char))(char[] a, char key)            Searches the specified array of chars for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(char%5B%5D,%20int,%20int,%20char))(char[] a, int fromIndex, int toIndex, char key)            Searches a range of the specified array of chars for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(double%5B%5D,%20double))(double[] a, double key)            Searches the specified array of doubles for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(double%5B%5D,%20int,%20int,%20double))(double[] a, int fromIndex, int toIndex, double key)            Searches a range of the specified array of doubles for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(float%5B%5D,%20float))(float[] a, float key)            Searches the specified array of floats for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(float%5B%5D,%20int,%20int,%20float))(float[] a, int fromIndex, int toIndex, float key)            Searches a range of the specified array of floats for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(int%5B%5D,%20int))(int[] a, int key)            Searches the specified array of ints for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(int%5B%5D,%20int,%20int,%20int))(int[] a, int fromIndex, int toIndex, int key)            Searches a range of the specified array of ints for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(long%5B%5D,%20int,%20int,%20long))(long[] a, int fromIndex, int toIndex, long key)            Searches a range of the specified array of longs for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(long%5B%5D,%20long))(long[] a, long key)            Searches the specified array of longs for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(java.lang.Object%5B%5D,%20int,%20int,%20java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html)[] a, int fromIndex, int toIndex, [Object](http://docs.google.com/java/lang/Object.html) key)            Searches a range of the specified array for the specified object using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(java.lang.Object%5B%5D,%20java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html)[] a, [Object](http://docs.google.com/java/lang/Object.html) key)            Searches the specified array for the specified object using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(short%5B%5D,%20int,%20int,%20short))(short[] a, int fromIndex, int toIndex, short key)            Searches a range of the specified array of shorts for the specified value using the binary search algorithm. |
| static int | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(short%5B%5D,%20short))(short[] a, short key)            Searches the specified array of shorts for the specified value using the binary search algorithm. |
| static   | <T> int | | --- | | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(T%5B%5D,%20int,%20int,%20T,%20java.util.Comparator))(T[] a, int fromIndex, int toIndex, T key, [Comparator](http://docs.google.com/java/util/Comparator.html)<? super T> c)            Searches a range of the specified array for the specified object using the binary search algorithm. |
| static   | <T> int | | --- | | [**binarySearch**](http://docs.google.com/java/util/Arrays.html#binarySearch(T%5B%5D,%20T,%20java.util.Comparator))(T[] a, T key, [Comparator](http://docs.google.com/java/util/Comparator.html)<? super T> c)            Searches the specified array for the specified object using the binary search algorithm. |
| static boolean[] | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(boolean%5B%5D,%20int))(boolean[] original, int newLength)            Copies the specified array, truncating or padding with false (if necessary) so the copy has the specified length. |
| static byte[] | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(byte%5B%5D,%20int))(byte[] original, int newLength)            Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. |
| static char[] | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(char%5B%5D,%20int))(char[] original, int newLength)            Copies the specified array, truncating or padding with null characters (if necessary) so the copy has the specified length. |
| static double[] | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(double%5B%5D,%20int))(double[] original, int newLength)            Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. |
| static float[] | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(float%5B%5D,%20int))(float[] original, int newLength)            Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. |
| static int[] | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(int%5B%5D,%20int))(int[] original, int newLength)            Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. |
| static long[] | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(long%5B%5D,%20int))(long[] original, int newLength)            Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. |
| static short[] | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(short%5B%5D,%20int))(short[] original, int newLength)            Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. |
| static   | <T> T[] | | --- | | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(T%5B%5D,%20int))(T[] original, int newLength)            Copies the specified array, truncating or padding with nulls (if necessary) so the copy has the specified length. |
| static   | <T,U> T[] | | --- | | [**copyOf**](http://docs.google.com/java/util/Arrays.html#copyOf(U%5B%5D,%20int,%20java.lang.Class))(U[] original, int newLength, [Class](http://docs.google.com/java/lang/Class.html)<? extends T[]> newType)            Copies the specified array, truncating or padding with nulls (if necessary) so the copy has the specified length. |
| static boolean[] | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(boolean%5B%5D,%20int,%20int))(boolean[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static byte[] | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(byte%5B%5D,%20int,%20int))(byte[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static char[] | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(char%5B%5D,%20int,%20int))(char[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static double[] | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(double%5B%5D,%20int,%20int))(double[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static float[] | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(float%5B%5D,%20int,%20int))(float[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static int[] | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(int%5B%5D,%20int,%20int))(int[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static long[] | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(long%5B%5D,%20int,%20int))(long[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static short[] | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(short%5B%5D,%20int,%20int))(short[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static   | <T> T[] | | --- | | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(T%5B%5D,%20int,%20int))(T[] original, int from, int to)            Copies the specified range of the specified array into a new array. |
| static   | <T,U> T[] | | --- | | [**copyOfRange**](http://docs.google.com/java/util/Arrays.html#copyOfRange(U%5B%5D,%20int,%20int,%20java.lang.Class))(U[] original, int from, int to, [Class](http://docs.google.com/java/lang/Class.html)<? extends T[]> newType)            Copies the specified range of the specified array into a new array. |
| static boolean | [**deepEquals**](http://docs.google.com/java/util/Arrays.html#deepEquals(java.lang.Object%5B%5D,%20java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] a1, [Object](http://docs.google.com/java/lang/Object.html)[] a2)            Returns true if the two specified arrays are *deeply equal* to one another. |
| static int | [**deepHashCode**](http://docs.google.com/java/util/Arrays.html#deepHashCode(java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] a)            Returns a hash code based on the "deep contents" of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**deepToString**](http://docs.google.com/java/util/Arrays.html#deepToString(java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] a)            Returns a string representation of the "deep contents" of the specified array. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(boolean%5B%5D,%20boolean%5B%5D))(boolean[] a, boolean[] a2)            Returns true if the two specified arrays of booleans are *equal* to one another. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(byte%5B%5D,%20byte%5B%5D))(byte[] a, byte[] a2)            Returns true if the two specified arrays of bytes are *equal* to one another. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(char%5B%5D,%20char%5B%5D))(char[] a, char[] a2)            Returns true if the two specified arrays of chars are *equal* to one another. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(double%5B%5D,%20double%5B%5D))(double[] a, double[] a2)            Returns true if the two specified arrays of doubles are *equal* to one another. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(float%5B%5D,%20float%5B%5D))(float[] a, float[] a2)            Returns true if the two specified arrays of floats are *equal* to one another. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(int%5B%5D,%20int%5B%5D))(int[] a, int[] a2)            Returns true if the two specified arrays of ints are *equal* to one another. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(long%5B%5D,%20long%5B%5D))(long[] a, long[] a2)            Returns true if the two specified arrays of longs are *equal* to one another. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(java.lang.Object%5B%5D,%20java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] a, [Object](http://docs.google.com/java/lang/Object.html)[] a2)            Returns true if the two specified arrays of Objects are *equal* to one another. |
| static boolean | [**equals**](http://docs.google.com/java/util/Arrays.html#equals(short%5B%5D,%20short%5B%5D))(short[] a, short[] a2)            Returns true if the two specified arrays of shorts are *equal* to one another. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(boolean%5B%5D,%20boolean))(boolean[] a, boolean val)            Assigns the specified boolean value to each element of the specified array of booleans. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(boolean%5B%5D,%20int,%20int,%20boolean))(boolean[] a, int fromIndex, int toIndex, boolean val)            Assigns the specified boolean value to each element of the specified range of the specified array of booleans. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(byte%5B%5D,%20byte))(byte[] a, byte val)            Assigns the specified byte value to each element of the specified array of bytes. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(byte%5B%5D,%20int,%20int,%20byte))(byte[] a, int fromIndex, int toIndex, byte val)            Assigns the specified byte value to each element of the specified range of the specified array of bytes. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(char%5B%5D,%20char))(char[] a, char val)            Assigns the specified char value to each element of the specified array of chars. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(char%5B%5D,%20int,%20int,%20char))(char[] a, int fromIndex, int toIndex, char val)            Assigns the specified char value to each element of the specified range of the specified array of chars. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(double%5B%5D,%20double))(double[] a, double val)            Assigns the specified double value to each element of the specified array of doubles. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(double%5B%5D,%20int,%20int,%20double))(double[] a, int fromIndex, int toIndex, double val)            Assigns the specified double value to each element of the specified range of the specified array of doubles. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(float%5B%5D,%20float))(float[] a, float val)            Assigns the specified float value to each element of the specified array of floats. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(float%5B%5D,%20int,%20int,%20float))(float[] a, int fromIndex, int toIndex, float val)            Assigns the specified float value to each element of the specified range of the specified array of floats. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(int%5B%5D,%20int))(int[] a, int val)            Assigns the specified int value to each element of the specified array of ints. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(int%5B%5D,%20int,%20int,%20int))(int[] a, int fromIndex, int toIndex, int val)            Assigns the specified int value to each element of the specified range of the specified array of ints. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(long%5B%5D,%20int,%20int,%20long))(long[] a, int fromIndex, int toIndex, long val)            Assigns the specified long value to each element of the specified range of the specified array of longs. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(long%5B%5D,%20long))(long[] a, long val)            Assigns the specified long value to each element of the specified array of longs. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(java.lang.Object%5B%5D,%20int,%20int,%20java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html)[] a, int fromIndex, int toIndex, [Object](http://docs.google.com/java/lang/Object.html) val)            Assigns the specified Object reference to each element of the specified range of the specified array of Objects. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(java.lang.Object%5B%5D,%20java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html)[] a, [Object](http://docs.google.com/java/lang/Object.html) val)            Assigns the specified Object reference to each element of the specified array of Objects. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(short%5B%5D,%20int,%20int,%20short))(short[] a, int fromIndex, int toIndex, short val)            Assigns the specified short value to each element of the specified range of the specified array of shorts. |
| static void | [**fill**](http://docs.google.com/java/util/Arrays.html#fill(short%5B%5D,%20short))(short[] a, short val)            Assigns the specified short value to each element of the specified array of shorts. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(boolean%5B%5D))(boolean[] a)            Returns a hash code based on the contents of the specified array. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(byte%5B%5D))(byte[] a)            Returns a hash code based on the contents of the specified array. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(char%5B%5D))(char[] a)            Returns a hash code based on the contents of the specified array. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(double%5B%5D))(double[] a)            Returns a hash code based on the contents of the specified array. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(float%5B%5D))(float[] a)            Returns a hash code based on the contents of the specified array. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(int%5B%5D))(int[] a)            Returns a hash code based on the contents of the specified array. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(long%5B%5D))(long[] a)            Returns a hash code based on the contents of the specified array. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] a)            Returns a hash code based on the contents of the specified array. |
| static int | [**hashCode**](http://docs.google.com/java/util/Arrays.html#hashCode(short%5B%5D))(short[] a)            Returns a hash code based on the contents of the specified array. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(byte%5B%5D))(byte[] a)            Sorts the specified array of bytes into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(byte%5B%5D,%20int,%20int))(byte[] a, int fromIndex, int toIndex)            Sorts the specified range of the specified array of bytes into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(char%5B%5D))(char[] a)            Sorts the specified array of chars into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(char%5B%5D,%20int,%20int))(char[] a, int fromIndex, int toIndex)            Sorts the specified range of the specified array of chars into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(double%5B%5D))(double[] a)            Sorts the specified array of doubles into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(double%5B%5D,%20int,%20int))(double[] a, int fromIndex, int toIndex)            Sorts the specified range of the specified array of doubles into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(float%5B%5D))(float[] a)            Sorts the specified array of floats into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(float%5B%5D,%20int,%20int))(float[] a, int fromIndex, int toIndex)            Sorts the specified range of the specified array of floats into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(int%5B%5D))(int[] a)            Sorts the specified array of ints into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(int%5B%5D,%20int,%20int))(int[] a, int fromIndex, int toIndex)            Sorts the specified range of the specified array of ints into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(long%5B%5D))(long[] a)            Sorts the specified array of longs into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(long%5B%5D,%20int,%20int))(long[] a, int fromIndex, int toIndex)            Sorts the specified range of the specified array of longs into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] a)            Sorts the specified array of objects into ascending order, according to the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of its elements. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(java.lang.Object%5B%5D,%20int,%20int))([Object](http://docs.google.com/java/lang/Object.html)[] a, int fromIndex, int toIndex)            Sorts the specified range of the specified array of objects into ascending order, according to the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of its elements. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(short%5B%5D))(short[] a)            Sorts the specified array of shorts into ascending numerical order. |
| static void | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(short%5B%5D,%20int,%20int))(short[] a, int fromIndex, int toIndex)            Sorts the specified range of the specified array of shorts into ascending numerical order. |
| static   | <T> void | | --- | | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(T%5B%5D,%20java.util.Comparator))(T[] a, [Comparator](http://docs.google.com/java/util/Comparator.html)<? super T> c)            Sorts the specified array of objects according to the order induced by the specified comparator. |
| static   | <T> void | | --- | | [**sort**](http://docs.google.com/java/util/Arrays.html#sort(T%5B%5D,%20int,%20int,%20java.util.Comparator))(T[] a, int fromIndex, int toIndex, [Comparator](http://docs.google.com/java/util/Comparator.html)<? super T> c)            Sorts the specified range of the specified array of objects according to the order induced by the specified comparator. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(boolean%5B%5D))(boolean[] a)            Returns a string representation of the contents of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(byte%5B%5D))(byte[] a)            Returns a string representation of the contents of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(char%5B%5D))(char[] a)            Returns a string representation of the contents of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(double%5B%5D))(double[] a)            Returns a string representation of the contents of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(float%5B%5D))(float[] a)            Returns a string representation of the contents of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(int%5B%5D))(int[] a)            Returns a string representation of the contents of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(long%5B%5D))(long[] a)            Returns a string representation of the contents of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] a)            Returns a string representation of the contents of the specified array. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Arrays.html#toString(short%5B%5D))(short[] a)            Returns a string representation of the contents of the specified array. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Method Detail** |
| --- |

### sort

public static void **sort**(long[] a)

Sorts the specified array of longs into ascending numerical order. The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sorted

### sort

public static void **sort**(long[] a,  
 int fromIndex,  
 int toIndex)

Sorts the specified range of the specified array of longs into ascending numerical order. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.)

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### sort

public static void **sort**(int[] a)

Sorts the specified array of ints into ascending numerical order. The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sorted

### sort

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

Sorts the specified range of the specified array of ints into ascending numerical order. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.)

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### sort

public static void **sort**(short[] a)

Sorts the specified array of shorts into ascending numerical order. The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sorted

### sort

public static void **sort**(short[] a,  
 int fromIndex,  
 int toIndex)

Sorts the specified range of the specified array of shorts into ascending numerical order. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.)

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### sort

public static void **sort**(char[] a)

Sorts the specified array of chars into ascending numerical order. The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sorted

### sort

public static void **sort**(char[] a,  
 int fromIndex,  
 int toIndex)

Sorts the specified range of the specified array of chars into ascending numerical order. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.)

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### sort

public static void **sort**(byte[] a)

Sorts the specified array of bytes into ascending numerical order. The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sorted

### sort

public static void **sort**(byte[] a,  
 int fromIndex,  
 int toIndex)

Sorts the specified range of the specified array of bytes into ascending numerical order. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.)

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### sort

public static void **sort**(double[] a)

Sorts the specified array of doubles into ascending numerical order.

The < relation does not provide a total order on all floating-point values; although they are distinct numbers -0.0 == 0.0 is true and a NaN value compares neither less than, greater than, nor equal to any floating-point value, even itself. To allow the sort to proceed, instead of using the < relation to determine ascending numerical order, this method uses the total order imposed by [Double.compareTo(java.lang.Double)](http://docs.google.com/java/lang/Double.html#compareTo(java.lang.Double)). This ordering differs from the < relation in that -0.0 is treated as less than 0.0 and NaN is considered greater than any other floating-point value. For the purposes of sorting, all NaN values are considered equivalent and equal.

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sorted

### sort

public static void **sort**(double[] a,  
 int fromIndex,  
 int toIndex)

Sorts the specified range of the specified array of doubles into ascending numerical order. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.)

The < relation does not provide a total order on all floating-point values; although they are distinct numbers -0.0 == 0.0 is true and a NaN value compares neither less than, greater than, nor equal to any floating-point value, even itself. To allow the sort to proceed, instead of using the < relation to determine ascending numerical order, this method uses the total order imposed by [Double.compareTo(java.lang.Double)](http://docs.google.com/java/lang/Double.html#compareTo(java.lang.Double)). This ordering differs from the < relation in that -0.0 is treated as less than 0.0 and NaN is considered greater than any other floating-point value. For the purposes of sorting, all NaN values are considered equivalent and equal.

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### sort

public static void **sort**(float[] a)

Sorts the specified array of floats into ascending numerical order.

The < relation does not provide a total order on all floating-point values; although they are distinct numbers -0.0f == 0.0f is true and a NaN value compares neither less than, greater than, nor equal to any floating-point value, even itself. To allow the sort to proceed, instead of using the < relation to determine ascending numerical order, this method uses the total order imposed by [Float.compareTo(java.lang.Float)](http://docs.google.com/java/lang/Float.html#compareTo(java.lang.Float)). This ordering differs from the < relation in that -0.0f is treated as less than 0.0f and NaN is considered greater than any other floating-point value. For the purposes of sorting, all NaN values are considered equivalent and equal.

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sorted

### sort

public static void **sort**(float[] a,  
 int fromIndex,  
 int toIndex)

Sorts the specified range of the specified array of floats into ascending numerical order. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.)

The < relation does not provide a total order on all floating-point values; although they are distinct numbers -0.0f == 0.0f is true and a NaN value compares neither less than, greater than, nor equal to any floating-point value, even itself. To allow the sort to proceed, instead of using the < relation to determine ascending numerical order, this method uses the total order imposed by [Float.compareTo(java.lang.Float)](http://docs.google.com/java/lang/Float.html#compareTo(java.lang.Float)). This ordering differs from the < relation in that -0.0f is treated as less than 0.0f and NaN is considered greater than any other floating-point value. For the purposes of sorting, all NaN values are considered equivalent and equal.

The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n\*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### sort

public static void **sort**([Object](http://docs.google.com/java/lang/Object.html)[] a)

Sorts the specified array of objects into ascending order, according to the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of its elements. All elements in the array must implement the [Comparable](http://docs.google.com/java/lang/Comparable.html) interface. Furthermore, all elements in the array must be *mutually comparable* (that is, e1.compareTo(e2) must not throw a ClassCastException for any elements e1 and e2 in the array).

This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort.

The sorting algorithm is a modified mergesort (in which the merge is omitted if the highest element in the low sublist is less than the lowest element in the high sublist). This algorithm offers guaranteed n\*log(n) performance.

**Parameters:**a - the array to be sorted **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the array contains elements that are not *mutually comparable* (for example, strings and integers).

### sort

public static void **sort**([Object](http://docs.google.com/java/lang/Object.html)[] a,  
 int fromIndex,  
 int toIndex)

Sorts the specified range of the specified array of objects into ascending order, according to the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of its elements. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.) All elements in this range must implement the [Comparable](http://docs.google.com/java/lang/Comparable.html) interface. Furthermore, all elements in this range must be *mutually comparable* (that is, e1.compareTo(e2) must not throw a ClassCastException for any elements e1 and e2 in the array).

This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort.

The sorting algorithm is a modified mergesort (in which the merge is omitted if the highest element in the low sublist is less than the lowest element in the high sublist). This algorithm offers guaranteed n\*log(n) performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the array contains elements that are not *mutually comparable* (for example, strings and integers).

### sort

public static <T> void **sort**(T[] a,  
 [Comparator](http://docs.google.com/java/util/Comparator.html)<? super T> c)

Sorts the specified array of objects according to the order induced by the specified comparator. All elements in the array must be *mutually comparable* by the specified comparator (that is, c.compare(e1, e2) must not throw a ClassCastException for any elements e1 and e2 in the array).

This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort.

The sorting algorithm is a modified mergesort (in which the merge is omitted if the highest element in the low sublist is less than the lowest element in the high sublist). This algorithm offers guaranteed n\*log(n) performance.

**Parameters:**a - the array to be sortedc - the comparator to determine the order of the array. A null value indicates that the elements' [natural ordering](http://docs.google.com/java/lang/Comparable.html) should be used. **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the array contains elements that are not *mutually comparable* using the specified comparator.

### sort

public static <T> void **sort**(T[] a,  
 int fromIndex,  
 int toIndex,  
 [Comparator](http://docs.google.com/java/util/Comparator.html)<? super T> c)

Sorts the specified range of the specified array of objects according to the order induced by the specified comparator. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be sorted is empty.) All elements in the range must be *mutually comparable* by the specified comparator (that is, c.compare(e1, e2) must not throw a ClassCastException for any elements e1 and e2 in the range).

This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort.

The sorting algorithm is a modified mergesort (in which the merge is omitted if the highest element in the low sublist is less than the lowest element in the high sublist). This algorithm offers guaranteed n\*log(n) performance.

**Parameters:**a - the array to be sortedfromIndex - the index of the first element (inclusive) to be sortedtoIndex - the index of the last element (exclusive) to be sortedc - the comparator to determine the order of the array. A null value indicates that the elements' [natural ordering](http://docs.google.com/java/lang/Comparable.html) should be used. **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the array contains elements that are not *mutually comparable* using the specified comparator. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### binarySearch

public static int **binarySearch**(long[] a,  
 long key)

Searches the specified array of longs for the specified value using the binary search algorithm. The array must be sorted (as by the [sort(long[])](http://docs.google.com/java/util/Arrays.html#sort(long%5B%5D)) 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 searchedkey - 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.

### binarySearch

public static int **binarySearch**(long[] a,  
 int fromIndex,  
 int toIndex,  
 long key)

Searches a range of the specified array of longs for the specified value using the binary search algorithm. The range must be sorted (as by the [sort(long[], int, int)](http://docs.google.com/java/util/Arrays.html#sort(long%5B%5D,%20int,%20int)) method) prior to making this call. If it is not sorted, the results are undefined. If the range contains multiple elements with the specified value, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched for **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### binarySearch

public static int **binarySearch**(int[] a,  
 int key)

Searches the specified array of ints for the specified value using the binary search algorithm. The array must be sorted (as by the [sort(int[])](http://docs.google.com/java/util/Arrays.html#sort(int%5B%5D)) 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 searchedkey - 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.

### binarySearch

public static int **binarySearch**(int[] a,  
 int fromIndex,  
 int toIndex,  
 int key)

Searches a range of the specified array of ints for the specified value using the binary search algorithm. The range must be sorted (as by the [sort(int[], int, int)](http://docs.google.com/java/util/Arrays.html#sort(int%5B%5D,%20int,%20int)) method) prior to making this call. If it is not sorted, the results are undefined. If the range contains multiple elements with the specified value, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched for **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### binarySearch

public static int **binarySearch**(short[] a,  
 short 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[])](http://docs.google.com/java/util/Arrays.html#sort(short%5B%5D)) 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 searchedkey - 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.

### binarySearch

public static int **binarySearch**(short[] a,  
 int fromIndex,  
 int toIndex,  
 short key)

Searches a range of the specified array of shorts for the specified value using the binary search algorithm. The range must be sorted (as by the [sort(short[], int, int)](http://docs.google.com/java/util/Arrays.html#sort(short%5B%5D,%20int,%20int)) method) prior to making this call. If it is not sorted, the results are undefined. If the range contains multiple elements with the specified value, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched for **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### binarySearch

public static int **binarySearch**(char[] a,  
 char key)

Searches the specified array of chars for the specified value using the binary search algorithm. The array must be sorted (as by the [sort(char[])](http://docs.google.com/java/util/Arrays.html#sort(char%5B%5D)) 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 searchedkey - 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.

### binarySearch

public static int **binarySearch**(char[] a,  
 int fromIndex,  
 int toIndex,  
 char key)

Searches a range of the specified array of chars for the specified value using the binary search algorithm. The range must be sorted (as by the [sort(char[], int, int)](http://docs.google.com/java/util/Arrays.html#sort(char%5B%5D,%20int,%20int)) method) prior to making this call. If it is not sorted, the results are undefined. If the range contains multiple elements with the specified value, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched for **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### binarySearch

public static int **binarySearch**(byte[] a,  
 byte key)

Searches the specified array of bytes for the specified value using the binary search algorithm. The array must be sorted (as by the [sort(byte[])](http://docs.google.com/java/util/Arrays.html#sort(byte%5B%5D)) 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 searchedkey - 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.

### binarySearch

public static int **binarySearch**(byte[] a,  
 int fromIndex,  
 int toIndex,  
 byte key)

Searches a range of the specified array of bytes for the specified value using the binary search algorithm. The range must be sorted (as by the [sort(byte[], int, int)](http://docs.google.com/java/util/Arrays.html#sort(byte%5B%5D,%20int,%20int)) method) prior to making this call. If it is not sorted, the results are undefined. If the range contains multiple elements with the specified value, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched for **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### binarySearch

public static int **binarySearch**(double[] a,  
 double key)

Searches the specified array of doubles for the specified value using the binary search algorithm. The array must be sorted (as by the [sort(double[])](http://docs.google.com/java/util/Arrays.html#sort(double%5B%5D)) 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. This method considers all NaN values to be equivalent and equal.

**Parameters:**a - the array to be searchedkey - 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.

### binarySearch

public static int **binarySearch**(double[] a,  
 int fromIndex,  
 int toIndex,  
 double key)

Searches a range of the specified array of doubles for the specified value using the binary search algorithm. The range must be sorted (as by the [sort(double[], int, int)](http://docs.google.com/java/util/Arrays.html#sort(double%5B%5D,%20int,%20int)) method) prior to making this call. If it is not sorted, the results are undefined. If the range contains multiple elements with the specified value, there is no guarantee which one will be found. This method considers all NaN values to be equivalent and equal.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched for **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### binarySearch

public static int **binarySearch**(float[] a,  
 float key)

Searches the specified array of floats for the specified value using the binary search algorithm. The array must be sorted (as by the [sort(float[])](http://docs.google.com/java/util/Arrays.html#sort(float%5B%5D)) 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. This method considers all NaN values to be equivalent and equal.

**Parameters:**a - the array to be searchedkey - 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.

### binarySearch

public static int **binarySearch**(float[] a,  
 int fromIndex,  
 int toIndex,  
 float key)

Searches a range of the specified array of floats for the specified value using the binary search algorithm. The range must be sorted (as by the [sort(float[], int, int)](http://docs.google.com/java/util/Arrays.html#sort(float%5B%5D,%20int,%20int)) method) prior to making this call. If it is not sorted, the results are undefined. If the range contains multiple elements with the specified value, there is no guarantee which one will be found. This method considers all NaN values to be equivalent and equal.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched for **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### binarySearch

public static int **binarySearch**([Object](http://docs.google.com/java/lang/Object.html)[] a,  
 [Object](http://docs.google.com/java/lang/Object.html) key)

Searches the specified array for the specified object using the binary search algorithm. The array must be sorted into ascending order according to the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of its elements (as by the [sort(Object[])](http://docs.google.com/java/util/Arrays.html#sort(java.lang.Object%5B%5D)) method) prior to making this call. If it is not sorted, the results are undefined. (If the array contains elements that are not mutually comparable (for example, strings and integers), it *cannot* be sorted according to the natural ordering of its elements, hence results are undefined.) If the array contains multiple elements equal to the specified object, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedkey - 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. **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the search key is not comparable to the elements of the array.

### binarySearch

public static int **binarySearch**([Object](http://docs.google.com/java/lang/Object.html)[] a,  
 int fromIndex,  
 int toIndex,  
 [Object](http://docs.google.com/java/lang/Object.html) key)

Searches a range of the specified array for the specified object using the binary search algorithm. The range must be sorted into ascending order according to the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of its elements (as by the [sort(Object[], int, int)](http://docs.google.com/java/util/Arrays.html#sort(java.lang.Object%5B%5D,%20int,%20int)) method) prior to making this call. If it is not sorted, the results are undefined. (If the range contains elements that are not mutually comparable (for example, strings and integers), it *cannot* be sorted according to the natural ordering of its elements, hence results are undefined.) If the range contains multiple elements equal to the specified object, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched for **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the search key is not comparable to the elements of the array within the specified range. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### binarySearch

public static <T> int **binarySearch**(T[] a,  
 T key,  
 [Comparator](http://docs.google.com/java/util/Comparator.html)<? super T> c)

Searches the specified array for the specified object using the binary search algorithm. The array must be sorted into ascending order according to the specified comparator (as by the [sort(T[], Comparator)](http://docs.google.com/java/util/Arrays.html#sort(T%5B%5D,%20java.util.Comparator)) method) prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements equal to the specified object, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedkey - the value to be searched forc - the comparator by which the array is ordered. A null value indicates that the elements' [natural ordering](http://docs.google.com/java/lang/Comparable.html) should be used. **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. **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the array contains elements that are not *mutually comparable* using the specified comparator, or the search key is not comparable to the elements of the array using this comparator.

### binarySearch

public static <T> int **binarySearch**(T[] a,  
 int fromIndex,  
 int toIndex,  
 T key,  
 [Comparator](http://docs.google.com/java/util/Comparator.html)<? super T> c)

Searches a range of the specified array for the specified object using the binary search algorithm. The range must be sorted into ascending order according to the specified comparator (as by the [sort(T[], int, int, Comparator)](http://docs.google.com/java/util/Arrays.html#sort(T%5B%5D,%20int,%20int,%20java.util.Comparator)) method) prior to making this call. If it is not sorted, the results are undefined. If the range contains multiple elements equal to the specified object, there is no guarantee which one will be found.

**Parameters:**a - the array to be searchedfromIndex - the index of the first element (inclusive) to be searchedtoIndex - the index of the last element (exclusive) to be searchedkey - the value to be searched forc - the comparator by which the array is ordered. A null value indicates that the elements' [natural ordering](http://docs.google.com/java/lang/Comparable.html) should be used. **Returns:**index of the search key, if it is contained in the array within the specified range; 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 in the range greater than the key, or toIndex if all elements in the range 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. **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the range contains elements that are not *mutually comparable* using the specified comparator, or the search key is not comparable to the elements in the range using this comparator. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length**Since:** 1.6

### equals

public static boolean **equals**(long[] a,  
 long[] a2)

Returns true if the two specified arrays of longs 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 equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal

### equals

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 equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal

### equals

public static boolean **equals**(short[] a,  
 short[] a2)

Returns true if the two specified arrays of shorts 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 equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal

### equals

public static boolean **equals**(char[] a,  
 char[] a2)

Returns true if the two specified arrays of chars 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 equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal

### equals

public static boolean **equals**(byte[] a,  
 byte[] a2)

Returns true if the two specified arrays of bytes 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 equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal

### equals

public static boolean **equals**(boolean[] a,  
 boolean[] a2)

Returns true if the two specified arrays of booleans 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 equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal

### equals

public static boolean **equals**(double[] a,  
 double[] a2)

Returns true if the two specified arrays of doubles 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.

Two doubles d1 and d2 are considered equal if:

new Double(d1).equals(new Double(d2))

(Unlike the == operator, this method considers NaN equals to itself, and 0.0d unequal to -0.0d.)

**Parameters:**a - one array to be tested for equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal**See Also:**[Double.equals(Object)](http://docs.google.com/java/lang/Double.html#equals(java.lang.Object))

### equals

public static boolean **equals**(float[] a,  
 float[] a2)

Returns true if the two specified arrays of floats 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.

Two floats f1 and f2 are considered equal if:

new Float(f1).equals(new Float(f2))

(Unlike the == operator, this method considers NaN equals to itself, and 0.0f unequal to -0.0f.)

**Parameters:**a - one array to be tested for equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal**See Also:**[Float.equals(Object)](http://docs.google.com/java/lang/Float.html#equals(java.lang.Object))

### equals

public static boolean **equals**([Object](http://docs.google.com/java/lang/Object.html)[] a,  
 [Object](http://docs.google.com/java/lang/Object.html)[] a2)

Returns true if the two specified arrays of Objects are *equal* to one another. The 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. Two objects e1 and e2 are considered *equal* if (e1==null ? e2==null : e1.equals(e2)). In other words, the 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 equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal

### fill

public static void **fill**(long[] a,  
 long val)

Assigns the specified long value to each element of the specified array of longs.

**Parameters:**a - the array to be filledval - the value to be stored in all elements of the array

### fill

public static void **fill**(long[] a,  
 int fromIndex,  
 int toIndex,  
 long val)

Assigns the specified long value to each element of the specified range of the specified array of longs. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### fill

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 filledval - the value to be stored in all elements of the array

### fill

public static void **fill**(int[] a,  
 int fromIndex,  
 int toIndex,  
 int val)

Assigns the specified int value to each element of the specified range of the specified array of ints. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### fill

public static void **fill**(short[] a,  
 short val)

Assigns the specified short value to each element of the specified array of shorts.

**Parameters:**a - the array to be filledval - the value to be stored in all elements of the array

### fill

public static void **fill**(short[] a,  
 int fromIndex,  
 int toIndex,  
 short val)

Assigns the specified short value to each element of the specified range of the specified array of shorts. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### fill

public static void **fill**(char[] a,  
 char val)

Assigns the specified char value to each element of the specified array of chars.

**Parameters:**a - the array to be filledval - the value to be stored in all elements of the array

### fill

public static void **fill**(char[] a,  
 int fromIndex,  
 int toIndex,  
 char val)

Assigns the specified char value to each element of the specified range of the specified array of chars. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### fill

public static void **fill**(byte[] a,  
 byte val)

Assigns the specified byte value to each element of the specified array of bytes.

**Parameters:**a - the array to be filledval - the value to be stored in all elements of the array

### fill

public static void **fill**(byte[] a,  
 int fromIndex,  
 int toIndex,  
 byte val)

Assigns the specified byte value to each element of the specified range of the specified array of bytes. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### fill

public static void **fill**(boolean[] a,  
 boolean val)

Assigns the specified boolean value to each element of the specified array of booleans.

**Parameters:**a - the array to be filledval - the value to be stored in all elements of the array

### fill

public static void **fill**(boolean[] a,  
 int fromIndex,  
 int toIndex,  
 boolean val)

Assigns the specified boolean value to each element of the specified range of the specified array of booleans. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### fill

public static void **fill**(double[] a,  
 double val)

Assigns the specified double value to each element of the specified array of doubles.

**Parameters:**a - the array to be filledval - the value to be stored in all elements of the array

### fill

public static void **fill**(double[] a,  
 int fromIndex,  
 int toIndex,  
 double val)

Assigns the specified double value to each element of the specified range of the specified array of doubles. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### fill

public static void **fill**(float[] a,  
 float val)

Assigns the specified float value to each element of the specified array of floats.

**Parameters:**a - the array to be filledval - the value to be stored in all elements of the array

### fill

public static void **fill**(float[] a,  
 int fromIndex,  
 int toIndex,  
 float val)

Assigns the specified float value to each element of the specified range of the specified array of floats. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length

### fill

public static void **fill**([Object](http://docs.google.com/java/lang/Object.html)[] a,  
 [Object](http://docs.google.com/java/lang/Object.html) val)

Assigns the specified Object reference to each element of the specified array of Objects.

**Parameters:**a - the array to be filledval - the value to be stored in all elements of the array **Throws:** [ArrayStoreException](http://docs.google.com/java/lang/ArrayStoreException.html) - if the specified value is not of a runtime type that can be stored in the specified array

### fill

public static void **fill**([Object](http://docs.google.com/java/lang/Object.html)[] a,  
 int fromIndex,  
 int toIndex,  
 [Object](http://docs.google.com/java/lang/Object.html) val)

Assigns the specified Object reference to each element of the specified range of the specified array of Objects. The range to be filled extends from index fromIndex, inclusive, to index toIndex, exclusive. (If fromIndex==toIndex, the range to be filled is empty.)

**Parameters:**a - the array to be filledfromIndex - the index of the first element (inclusive) to be filled with the specified valuetoIndex - the index of the last element (exclusive) to be filled with the specified valueval - the value to be stored in all elements of the array **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromIndex > toIndex [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if fromIndex < 0 or toIndex > a.length [ArrayStoreException](http://docs.google.com/java/lang/ArrayStoreException.html) - if the specified value is not of a runtime type that can be stored in the specified array

### copyOf

public static <T> T[] **copyOf**(T[] original,  
 int newLength)

Copies the specified array, truncating or padding with nulls (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 null. Such indices will exist if and only if the specified length is greater than that of the original array. The resulting array is of exactly the same class as the original array.

**Parameters:**original - the array to be copiednewLength - the length of the copy to be returned **Returns:**a copy of the original array, truncated or padded with nulls to obtain the specified length **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOf

public static <T,U> T[] **copyOf**(U[] original,  
 int newLength,  
 [Class](http://docs.google.com/java/lang/Class.html)<? extends T[]> newType)

Copies the specified array, truncating or padding with nulls (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 null. Such indices will exist if and only if the specified length is greater than that of the original array. The resulting array is of the class newType.

**Parameters:**original - the array to be copiednewLength - the length of the copy to be returnednewType - the class of the copy to be returned **Returns:**a copy of the original array, truncated or padded with nulls to obtain the specified length **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null [ArrayStoreException](http://docs.google.com/java/lang/ArrayStoreException.html) - if an element copied from original is not of a runtime type that can be stored in an array of class newType**Since:** 1.6

### copyOf

public static byte[] **copyOf**(byte[] 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 (byte)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 copiednewLength - 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 **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOf

public static short[] **copyOf**(short[] 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 (short)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 copiednewLength - 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 **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOf

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 copiednewLength - 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 **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOf

public static long[] **copyOf**(long[] 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 0L. 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 copiednewLength - 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 **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOf

public static char[] **copyOf**(char[] original,  
 int newLength)

Copies the specified array, truncating or padding with null characters (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 '\\u000'. 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 copiednewLength - the length of the copy to be returned **Returns:**a copy of the original array, truncated or padded with null characters to obtain the specified length **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOf

public static float[] **copyOf**(float[] 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 0f. 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 copiednewLength - 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 **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOf

public static double[] **copyOf**(double[] 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 0d. 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 copiednewLength - 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 **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOf

public static boolean[] **copyOf**(boolean[] original,  
 int newLength)

Copies the specified array, truncating or padding with false (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 false. 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 copiednewLength - the length of the copy to be returned **Returns:**a copy of the original array, truncated or padded with false elements to obtain the specified length **Throws:** [NegativeArraySizeException](http://docs.google.com/java/lang/NegativeArraySizeException.html) - if newLength is negative [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

public static <T> T[] **copyOfRange**(T[] 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 null 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.

The resulting array is of exactly the same class as the original array.

**Parameters:**original - the array from which a range is to be copiedfrom - the initial index of the range to be copied, inclusiveto - 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 nulls to obtain the required length **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

public static <T,U> T[] **copyOfRange**(U[] original,  
 int from,  
 int to,  
 [Class](http://docs.google.com/java/lang/Class.html)<? extends T[]> newType)

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 null 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. The resulting array is of the class newType.

**Parameters:**original - the array from which a range is to be copiedfrom - the initial index of the range to be copied, inclusiveto - the final index of the range to be copied, exclusive. (This index may lie outside the array.)newType - the class of the copy to be returned **Returns:**a new array containing the specified range from the original array, truncated or padded with nulls to obtain the required length **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null [ArrayStoreException](http://docs.google.com/java/lang/ArrayStoreException.html) - if an element copied from original is not of a runtime type that can be stored in an array of class newType.**Since:** 1.6

### copyOfRange

public static byte[] **copyOfRange**(byte[] 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 (byte)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 copiedfrom - the initial index of the range to be copied, inclusiveto - 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 **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

public static short[] **copyOfRange**(short[] 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 (short)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 copiedfrom - the initial index of the range to be copied, inclusiveto - 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 **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

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 copiedfrom - the initial index of the range to be copied, inclusiveto - 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 **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

public static long[] **copyOfRange**(long[] 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 0L 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 copiedfrom - the initial index of the range to be copied, inclusiveto - 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 **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

public static char[] **copyOfRange**(char[] 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 '\\u000' 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 copiedfrom - the initial index of the range to be copied, inclusiveto - 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 null characters to obtain the required length **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

public static float[] **copyOfRange**(float[] 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 0f 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 copiedfrom - the initial index of the range to be copied, inclusiveto - 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 **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

public static double[] **copyOfRange**(double[] 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 0d 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 copiedfrom - the initial index of the range to be copied, inclusiveto - 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 **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### copyOfRange

public static boolean[] **copyOfRange**(boolean[] 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 false 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 copiedfrom - the initial index of the range to be copied, inclusiveto - 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 false elements to obtain the required length **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length() [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if from > to [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if original is null**Since:** 1.6

### asList

public static <T> [List](http://docs.google.com/java/util/List.html)<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()](http://docs.google.com/java/util/Collection.html#toArray()). The returned list is serializable and implements [RandomAccess](http://docs.google.com/java/util/RandomAccess.html).

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:**a - the array by which the list will be backed **Returns:**a list view of the specified array

### hashCode

public static int **hashCode**(long[] a)

Returns a hash code based on the contents of the specified array. For any two long 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](http://docs.google.com/java/util/List.html#hashCode()) method on a [List](http://docs.google.com/java/util/List.html) containing a sequence of [Long](http://docs.google.com/java/lang/Long.html) 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**Since:** 1.5

### hashCode

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](http://docs.google.com/java/util/List.html#hashCode()) method on a [List](http://docs.google.com/java/util/List.html) containing a sequence of [Integer](http://docs.google.com/java/lang/Integer.html) 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**Since:** 1.5

### hashCode

public static int **hashCode**(short[] a)

Returns a hash code based on the contents of the specified array. For any two short 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](http://docs.google.com/java/util/List.html#hashCode()) method on a [List](http://docs.google.com/java/util/List.html) containing a sequence of [Short](http://docs.google.com/java/lang/Short.html) 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**Since:** 1.5

### hashCode

public static int **hashCode**(char[] a)

Returns a hash code based on the contents of the specified array. For any two char 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](http://docs.google.com/java/util/List.html#hashCode()) method on a [List](http://docs.google.com/java/util/List.html) containing a sequence of [Character](http://docs.google.com/java/lang/Character.html) 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**Since:** 1.5

### hashCode

public static int **hashCode**(byte[] a)

Returns a hash code based on the contents of the specified array. For any two byte 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](http://docs.google.com/java/util/List.html#hashCode()) method on a [List](http://docs.google.com/java/util/List.html) containing a sequence of [Byte](http://docs.google.com/java/lang/Byte.html) 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**Since:** 1.5

### hashCode

public static int **hashCode**(boolean[] a)

Returns a hash code based on the contents of the specified array. For any two boolean 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](http://docs.google.com/java/util/List.html#hashCode()) method on a [List](http://docs.google.com/java/util/List.html) containing a sequence of [Boolean](http://docs.google.com/java/lang/Boolean.html) 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**Since:** 1.5

### hashCode

public static int **hashCode**(float[] a)

Returns a hash code based on the contents of the specified array. For any two float 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](http://docs.google.com/java/util/List.html#hashCode()) method on a [List](http://docs.google.com/java/util/List.html) containing a sequence of [Float](http://docs.google.com/java/lang/Float.html) 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**Since:** 1.5

### hashCode

public static int **hashCode**(double[] a)

Returns a hash code based on the contents of the specified array. For any two double 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](http://docs.google.com/java/util/List.html#hashCode()) method on a [List](http://docs.google.com/java/util/List.html) containing a sequence of [Double](http://docs.google.com/java/lang/Double.html) 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**Since:** 1.5

### hashCode

public static int **hashCode**([Object](http://docs.google.com/java/lang/Object.html)[] a)

Returns a hash code based on the contents of the specified array. If the array contains other arrays as elements, the hash code is based on their identities rather than their contents. It is therefore acceptable to invoke this method on an array that contains itself as an element, either directly or indirectly through one or more levels of arrays.

For any two 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 equal to the value that would be returned by Arrays.asList(a).hashCode(), unless a is null, in which case 0 is returned.

**Parameters:**a - the array whose content-based hash code to compute **Returns:**a content-based hash code for a**Since:** 1.5 **See Also:**[deepHashCode(Object[])](http://docs.google.com/java/util/Arrays.html#deepHashCode(java.lang.Object%5B%5D))

### deepHashCode

public static int **deepHashCode**([Object](http://docs.google.com/java/lang/Object.html)[] 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()](http://docs.google.com/java/util/List.html#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**Since:** 1.5 **See Also:**[hashCode(Object[])](http://docs.google.com/java/util/Arrays.html#hashCode(java.lang.Object%5B%5D))

### deepEquals

public static boolean **deepEquals**([Object](http://docs.google.com/java/lang/Object.html)[] a1,  
 [Object](http://docs.google.com/java/lang/Object.html)[] a2)

Returns true if the two specified arrays are *deeply equal* to one another. Unlike the [equals(Object[],Object[])](http://docs.google.com/java/util/Arrays.html#equals(java.lang.Object%5B%5D,%20java.lang.Object%5B%5D)) 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 equalitya2 - the other array to be tested for equality **Returns:**true if the two arrays are equal**Since:** 1.5 **See Also:**[equals(Object[],Object[])](http://docs.google.com/java/util/Arrays.html#equals(java.lang.Object%5B%5D,%20java.lang.Object%5B%5D))

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**(long[] 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(long). Returns "null" if a is null.

**Parameters:**a - the array whose string representation to return **Returns:**a string representation of a**Since:** 1.5

### toString

public static [String](http://docs.google.com/java/lang/String.html) **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**Since:** 1.5

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**(short[] 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(short). Returns "null" if a is null.

**Parameters:**a - the array whose string representation to return **Returns:**a string representation of a**Since:** 1.5

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**(char[] 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(char). Returns "null" if a is null.

**Parameters:**a - the array whose string representation to return **Returns:**a string representation of a**Since:** 1.5

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**(byte[] 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(byte). Returns "null" if a is null.

**Parameters:**a - the array whose string representation to return **Returns:**a string representation of a**Since:** 1.5

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**(boolean[] 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(boolean). Returns "null" if a is null.

**Parameters:**a - the array whose string representation to return **Returns:**a string representation of a**Since:** 1.5

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**(float[] 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(float). Returns "null" if a is null.

**Parameters:**a - the array whose string representation to return **Returns:**a string representation of a**Since:** 1.5

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**(double[] 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(double). Returns "null" if a is null.

**Parameters:**a - the array whose string representation to return **Returns:**a string representation of a**Since:** 1.5

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**([Object](http://docs.google.com/java/lang/Object.html)[] a)

Returns a string representation of the contents of the specified array. If the array contains other arrays as elements, they are converted to strings by the [Object.toString()](http://docs.google.com/java/lang/Object.html#toString()) method inherited from Object, which describes their *identities* rather than their contents.

The value returned by this method is equal to the value that would be returned by Arrays.asList(a).toString(), unless a is null, in which case "null" is returned.

**Parameters:**a - the array whose string representation to return **Returns:**a string representation of a**Since:** 1.5 **See Also:**[deepToString(Object[])](http://docs.google.com/java/util/Arrays.html#deepToString(java.lang.Object%5B%5D))

### deepToString

public static [String](http://docs.google.com/java/lang/String.html) **deepToString**([Object](http://docs.google.com/java/lang/Object.html)[] 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**Since:** 1.5 **See Also:**[toString(Object[])](http://docs.google.com/java/util/Arrays.html#toString(java.lang.Object%5B%5D))

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Arrays.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/ArrayList.html)   [**NEXT CLASS**](http://docs.google.com/java/util/BitSet.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Arrays.html)    [**NO FRAMES**](http://docs.google.com/Arrays.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#tyjcwt) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).