

JAVA



What you learn ?

Java

➤ **Array**

➤ **Single dimensional array**

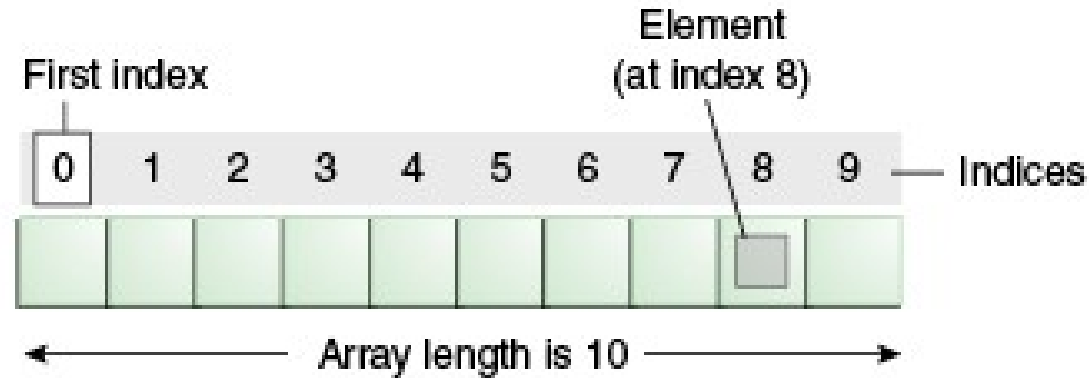
➤ **Multi dimensional array**

➤ **Array Class**

Array



Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.



Advantages-

- Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- Random access:** We can get any data located at an index position.

Disadvantages-

- Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Single Dimensional Array



//Java Program to illustrate the use of declaration, instantiation
//and initialization of Java array in a single line

```
class Testarray1{  
    public static void main(String args[]){  
        int a[]={33,3,4,5};//declaration, instantiation and initialization  
        //printing array  
        for(int i=0;i<a.length;i++)//length is the property of array  
            System.out.println(a[i]);  
    }  
}
```

Multi Dimensional Array



A multidimensional array is an array of arrays.

Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.

To create a two-dimensional array, add each array within its own set of curly braces:

Example:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };  
myNumbers[1][2] = 9;  
System.out.println(myNumbers[1][2]); // Outputs 9 instead of 7
```

Jagged Array

 If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

//Java Program to illustrate the jagged array


```
class TestJaggedArray{
    public static void main(String[] args){
        //declaring a 2D array with odd columns
        int arr[][] = new int[3][];
        arr[0] = new int[3];
        arr[1] = new int[4];
        arr[2] = new int[2];
        //initializing a jagged array
        int count = 0;
        for (int i=0; i<arr.length; i++) {
            for(int j=0; j<arr[i].length; j++) {
                arr[i][j] = count++;
            }
        }
    }
}
```

```
//printing the data of a jagged array
for (int i=0; i<arr.length; i++){
    for (int j=0; j<arr[i].length; j++){
        System.out.print(arr[i][j]+" ");
    }
    System.out.println();//new line
}
```

output:

```
0 1 2
3 4 5 6
7 8
```

Anonymous Array



As its name suggests, arrays having no name are called Anonymous arrays in java. This type of array is used only when we require an array instantly. We can also pass this array to any method without any reference variable.

Syntax:

```
new datatype[] {values separated by comma}
```

Example:

```
//One-dimensional integer type anonymous array  
new int[] {2,4,6,8};
```

```
// Multi-dimensional integer type anonymous array  
new int[][] { {1,3,5}, {2,4,6} };
```

Array Class



public class Arrays
extends Object

Arrays class is present in `java.util.Arrays`
Package.

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.

Syntax: In order to use Arrays

Benefits of Arrays Class in Java:

1. Arrays class makes it easier to perform common operations on an array.
2. No need to use complicated loops to work with an array.
3. No need to reinvent the wheel for operations such as binary search and sorting.

Method Summary

Methods	
Modifier and Type	Method and Description
static <T> List<T>	asList (T... a) Returns a fixed-size list backed by the specified array.
static int	binarySearch (byte[] a, byte key) Searches the specified array of bytes for the specified value using the binary search algorithm.
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.
static int	binarySearch (char[] a, char key) Searches the specified array of chars for the specified value using the binary search algorithm.
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.
static int	binarySearch (double[] a, double key) Searches the specified array of doubles for the specified value using the binary search algorithm.
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.
static int	binarySearch (float[] a, float key) Searches the specified array of floats for the specified value using the binary search algorithm.
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.
static int	binarySearch (int[] a, int key) Searches the specified array of ints for the specified value using the binary search algorithm.
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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

static <T,U> T[]

copyOf(U[] original, int newLength, **Class**<? extends T[]> newType)

Copies the specified array, truncating or padding with nulls (if necessary) so the copy has the specified length.

static boolean[]

copyOfRange(boolean[] original, int from, int to)

Copies the specified range of the specified array into a new array.

static byte[]

copyOfRange(byte[] original, int from, int to)

Copies the specified range of the specified array into a new array.

static char[]

copyOfRange(char[] original, int from, int to)

Copies the specified range of the specified array into a new array.

static boolean	deepEquals (Object[] a1, Object[] a2) Returns true if the two specified arrays are <i>deeply equal</i> to one another.
static int	deepHashCode (Object[] a) Returns a hash code based on the "deep contents" of the specified array.
static String	deepToString (Object[] a) Returns a string representation of the "deep contents" of the specified array.
static boolean	equals (boolean[] a, boolean[] a2) Returns true if the two specified arrays of booleans are <i>equal</i> to one another.
static boolean	equals (byte[] a, byte[] a2) Returns true if the two specified arrays of bytes are <i>equal</i> to one another.
static boolean	equals (char[] a, char[] a2) Returns true if the two specified arrays of chars are <i>equal</i> to one another.
static boolean	equals (double[] a, double[] a2) Returns true if the two specified arrays of doubles are <i>equal</i> to one another.
static boolean	equals (float[] a, float[] a2) Returns true if the two specified arrays of floats are <i>equal</i> to one another.
static boolean	equals (int[] a, int[] a2) Returns true if the two specified arrays of ints are <i>equal</i> to one another.
static boolean	equals (long[] a, long[] a2) Returns true if the two specified arrays of longs are <i>equal</i> to one another.
static boolean	equals (Object[] a, Object[] a2) Returns true if the two specified arrays of Objects are <i>equal</i> to one another.
static boolean	equals (short[] a, short[] a2) Returns true if the two specified arrays of shorts are <i>equal</i> to one another.
static void	fill (boolean[] a, boolean val) Assigns the specified boolean value to each element of the specified array of booleans.

static int	hashCode (char[] a) Returns a hash code based on the contents of the specified array.
static int	hashCode (double[] a) Returns a hash code based on the contents of the specified array.
static int	hashCode (float[] a) Returns a hash code based on the contents of the specified array.
static int	hashCode (int[] a) Returns a hash code based on the contents of the specified array.
static int	hashCode (long[] a) Returns a hash code based on the contents of the specified array.
static int	hashCode (Object[] a) Returns a hash code based on the contents of the specified array.
static int	hashCode (short[] a) Returns a hash code based on the contents of the specified array.
static void	sort (byte[] a) Sorts the specified array into ascending numerical order.
static void	sort (byte[] a, int fromIndex, int toIndex) Sorts the specified range of the array into ascending order.
static void	sort (char[] a) Sorts the specified array into ascending numerical order.
static void	sort (char[] a, int fromIndex, int toIndex) Sorts the specified range of the array into ascending order.
static void	sort (double[] a) Sorts the specified array into ascending numerical order.
static void	sort (double[] a, int fromIndex, int toIndex) Sorts the specified range of the array into ascending order.

static void	sort (short[] a, int fromIndex, int toIndex) Sorts the specified range of the array into ascending order.
static <T> void	sort (T[] a, Comparator <? super T> c) Sorts the specified array of objects according to the order induced by the specified comparator.
static <T> void	sort (T[] a, int fromIndex, int toIndex, Comparator <? super T> c) Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.
static String	toString (boolean[] a) Returns a string representation of the contents of the specified array.
static String	toString (byte[] a) Returns a string representation of the contents of the specified array.
static String	toString (char[] a) Returns a string representation of the contents of the specified array.
static String	toString (double[] a) Returns a string representation of the contents of the specified array.
static String	toString (float[] a) Returns a string representation of the contents of the specified array.
static String	toString (int[] a) Returns a string representation of the contents of the specified array.
static String	toString (long[] a) Returns a string representation of the contents of the specified array.
static String	toString (Object[] a) Returns a string representation of the contents of the specified array.
static String	toString (short[] a) Returns a string representation of the contents of the specified array.



Thanks

Anirudha Gaikwad