Lawrence M. Brown
ktee@erols.com

# Java Arrays

## 1. Introduction

In Java arrays are objects that represent a group of contiguous memory locations of the same type and referred to by the same name. The element type for an array can be any primitive data type or object.

## 2. Declaring and Allocating Arrays

Arrays can be created using the new operator or by declaring a static initializer. The new operator specifies the size of the array.

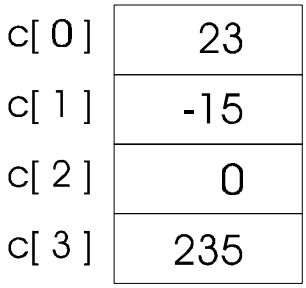
```
int[] c, d;                    // declare references to array c and d
c = new int [ 4 ];             // allocate memory for the 4 elements in array c
d = new int [ 125 ];           // allocate memory for the 125 elements in array d

byte[] b = new byte [ 100 ];   // declare and allocate memory for array b
```

- An array intializer may be any arbitrary expression.

```
int c = 15;
int[] primes = { 1, 2, 3, 5, 7, 9, 11 };
int[] a = { 1, primes[2], c, (int)Math.pow(2,2) };
```

- The first element in every array is the *zeroth* element.




- When memory is allocated for an array, the elements are automatically initialized to their default values: zero for all numeric primitive data types, false for boolean variables and null for references.

- As with all objects, if the array reference (object reference) is assigned a null value, then the garbage collector will reclaim the dereferenced (unused) memory.

## 3. Accessing Array Elements

Placing an integer-valued expression inside square brackets can access an array element, where the brackets are proceeded by the name of the array.

Accessing Array Elements Example

```
int c = 15, j = 1;

int[] primes = { 1, 2, 3, 5, 7, 9, 11 };
int[] a = { 1, primes[2], c, (int)Math.pow(2,2) };

System.out.println( "\n Before: \na[0] is " + a[0] );
System.out.println( "a[1] is " + a[1] \n "a[2] is " + a[2] );
System.out.println( "a[3] is " + a[3] );

a[ 0 ] = 0;

a[ j++ + a[0] ] = 1;          // a[1 + 0] = a[1] = 1, j = 2
a[ j-- ]++;                   // a[2] = a[2] + 1 = 16, j=1
a[ 3 ] %= 5;                  // a[3] = a[3] % 5 = 4 % 5 = 4;

//  a[ 0 ] = a[3] + a[1] - a[0] = 4 + 1 - 0 = 5, j=0
a[ j-1 ] = a[3] + a[j] - a[--j];

System.out.println( "\n After: \n a[0] is " + a[0] );
System.out.println( "a[1] is " + a[1] \n "a[2] is " + a[2] );
System.out.println( "a[3] is " + a[3] );
```

Output

```
Before:
a[0] is 1
a[1] is 3
a[2] is 15
a[3] is 4

After:
a[0] is 5
a[1] is 1
a[2] is 16
a[3] is 4
```

- When accessing an array element, if the index is less than zero or greater than or equal to the array length then an ArrayIndexOutOfBoundsException is thrown.
- Arrays have a length data field (read only), specifying the number of elements in the array. The length data field can be accessed through the dot operator.

```
for ( int k=0; k < a.length ; k++ )
        System.out.println( a[ k ] );
```

- The bounds of an array are integers between 0 and length - 1

# 4. Copying Arrays

## System method arraycopy()

Arrays can be copied using the Java System method arraycopy():

public static native void **arraycopy**(Object src, int src_position, Object dst, int dst_position, int length)

> Copies a region of the source array, src, beginning at the array cell src_position, to the destination array, dst, beginning at the cell dst_position in the destination. The number of cells copied is equal to the length argument.

Parameters:

| | |
|---|---|
| src | - the source array. |
| src_position | - start position (first cell to copy) in the source array. |
| dst | - the destination array. |
| dst_position | - start position in the destination array. |
| length | - the number of array elements to be copied. |

**Note**: arraycopy does not allocate memory for the destination array; the memory must already be allocated.

Example, arraycopy()

```
int[] primes = { 1, 2, 3, 5, 7, 9, 11 };
int[] c = new int[ primes.length ];

System.arraycopy( primes, 0, c, 0, primes.length);      // copy array primes to array c
```

## Cloning an Array

- By default, all Java arrays support the **clone** method.

Example, Cloning Arrays

```
public class ArrayClone
{
    public static void main( String[] args )
    {
      int[] primes = { 1, 2, 3, 5, 7, 11, 13, 17 };
      int[] backup;

      backup = (int[]) primes.clone();
      backup[0] = 0;

      System.out.println( "Primes: " );
      for( int i=0 ; i < primes.length ; i++ )
        System.out.print( " " + primes[i] );
```

- Declare reference to an int array.
- Call the clone() method for the Java array.
- Cast Object to array.
- Set the first element of the cloned array to zero.

```
        System.out.println( "\nbackup: With first Element Modified" );
        for( int j=0 ; j < backup.length ; j++ )
           System.out.print(" " + backup[j] );
    }
}
```

**Output:**

```
Primes:
1 2 3 5 7 11 13 17

backup: : With first Element Modified
0 2 3 5 7 11 13 17
```

## 5. Multidimensional Arrays

Multidimensional arrays are implemented as arrays of arrays. Multidimensional arrays are declared by appending the appropriate number of bracket pairs after the array name.

```
int[][] twoD = new int[512][128];              // integer array 512 x 128 elements

char[][][] threeD  = new char[8][16][24];      // character array 8 x 16 x 24

String[][] cats = {       { "Caesar",      "blue-point" },        // String array  4 rows x 2 columns
                          { "Kristin",     "blue-cream-point" },
                          { "Yodi",        "red-point"},
                          { "Heather",     "seal-point"}
                  };

double[][] identity =  {  { 1.0, 0.0, 0.0 },              // 3 x 3 identity array
                          { 0.0, 1.0, 0.0 },
                          { 0.0, 0.0, 1.0 }
                       };
```

- The number of elements in each direction need not be specified in the declaration.
- The number of elements in each direction need not be equal.

```
int[][]  irregular   = {     { 1 },
                             { 2, 3 },
                             { 4, 5, 6, 7 },
                             { 0 }
                       };
```

Example, Triangle Array

```
// declaring a triangle array

int[][] triangle = new int[10][];

for ( int j = 0; j<triangle.length ; j++ )
        triangle[ j ] = new int[ j + 1 ];


for ( int i = 0 ; i < triangle.length ; i++ )
{       for ( int j = 0 ; j < triangle[i].length ; j++ )
                System.out.print( triangle[i][j] + "  " );
        System.out.println();
}
```

Output

```
0
0 0
0 0 0
0 0 0 0
0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

- Java permits you to mix and break apart array declarations. This practice is confusing and NOT recommended.

- Java also allows placement of the array brackets with the variable name. This practice is also NOT recommended.

```
int[]  oneD, twoD[];

int  threeD[][][];
```

In the above example, oneD is a single array of integers, twoD is an array of integer arrays.