



Arrays

Arrays

An **array** is a collection of variables of the same type. When you need to store a list of values, such as numbers, you can store them in an **array**, instead of declaring separate variables for each number.

To declare an **array**, you need to define the type of the elements with **square brackets**. For example, to declare an **array** of integers:

```
int[] arr;
```

The name of the **array** is **arr**. The type of elements it will hold is **int**.

Now, you need to define the **array**'s capacity, or the number of elements it will hold. To accomplish this, use the keyword **new**.

```
int[] arr = new int[5];
```

The code above declares an **array** of 5 integers. In an **array**, the elements are ordered and each has a specific and constant position, which is called an **index**.

To reference elements in an **array**, type the name of the **array** followed by the index position within a pair of square brackets.

Example:

```
arr[2] = 42;
```

This assigns a value of 42 to the element with 2 as its index.

Note that elements in the **array** are identified with **zero-based** index numbers, meaning that the first element's index is 0 rather than one. So, the maximum index of the **array** `int[5]` is 4.

Initializing Arrays

Java provides a shortcut for instantiating arrays of primitive types and strings. If you already know what values to insert into the **array**, you can use an **array literal**. Example of an **array** literal:

```
String[] myNames = { "A", "B", "C", "D"};  
System.out.println(myNames[2]);  
  
// Outputs "C"
```

Try It Yourself

Place the values in a **comma-separated** list, enclosed in curly braces. The code above automatically initializes an **array** containing 4 elements, and stores the provided values.

Sometimes you might see the square brackets placed after the **array** name, which also works, but the preferred way is to place the brackets after the **array**'s data type.

Array Length

You can access the length of an [array](#) (the number of elements it stores) via its **length** property.
Example:

```
int[] intArr = new int[5];
System.out.println(intArr.length);

//Outputs 5
```

Try It Yourself

Don't forget that in arrays, indexes start from 0. So, in the example above, the last index is 4.

Arrays

Now that we know how to set and get [array](#) elements, we can calculate the sum of all elements in an [array](#) by using loops.
The **for** loop is the most used loop when working with arrays, as we can use the **length** of the [array](#) to determine how many times to run the loop.

```
int [] myArr = {6, 42, 3, 7};
int sum=0;
for(int x=0; x<myArr.length; x++) {
    sum += myArr[x];
}
System.out.println(sum);

// 58
```

Try It Yourself

In the code above, we declared a variable **sum** to store the result and assigned it 0.
Then we used a **for** loop to iterate through the [array](#), and added each element's value to the variable.

The condition of the **for** loop is `x<myArr.length`, as the last element's index is `myArr.length-1`.

Enhanced for Loop

The **enhanced for loop** (sometimes called a "for each" loop) is used to traverse elements in arrays.
The advantages are that it eliminates the possibility of bugs and makes the code easier to read.
Example:

```
int[] primes = {2, 3, 5, 7};

for (int t: primes) {
    System.out.println(t);
}
```

```
/*  
2  
3  
5  
7  
*/
```

Try It Yourself

The **enhanced for loop** declares a variable of a type compatible with the elements of the [array](#) being accessed. The variable will be available within the **for** block, and its value will be the same as the current [array](#) element. So, on each iteration of the loop, the variable **t** will be equal to the corresponding element in the [array](#).

Notice the **colon** after the variable in the syntax.

Multidimensional Arrays

Multidimensional arrays are [array](#) that contain other arrays. The two-dimensional [array](#) is the most basic multidimensional [array](#).

To create multidimensional arrays, place each [array](#) within its own set of square brackets. Example of a two-dimensional [array](#):

```
int[] [] sample = { {1, 2, 3}, {4, 5, 6} };
```

This declares an [array](#) with two arrays as its elements.

To access an element in the two-dimensional [array](#), provide two indexes, one for the [array](#), and another for the element inside that [array](#).

The following example accesses the first element in the second [array](#) of sample.

```
int x = sample[1][0];  
System.out.println(x);  
  
// Outputs 4
```

Try It Yourself

The [array](#)'s two indexes are called **row index** and **column index**.

Multidimensional Arrays

You can get and set a multidimensional [array](#)'s elements using the same pair of square brackets. Example:

```
int[] [] myArr = { {1, 2, 3}, {4}, {5, 6, 7} };  
myArr[0][2] = 42;  
int x = myArr[1][0]; // 4
```

Try It Yourself

The above two-dimensional [array](#) contains three arrays. The first [array](#) has three elements, the second has a single element and the last of these has three elements.

In Java, you're not limited to just two-dimensional arrays. Arrays can be nested within arrays to as many levels as your program needs. All you need to declare an array with more than two dimensions, is to add as many sets of empty brackets as you need. However, these are harder to maintain. Remember, that all array members must be of the same type.

End.