Arrays

An array is a container that holds data (values) of one single type. For example, you can create an array that can hold 100 values of int type.
Array is a fundamental construct in Java that allows you to store and access large number of values conveniently.
How to declare an array?
Here's how you can declare an array in Java:
dataType[] arrayName;

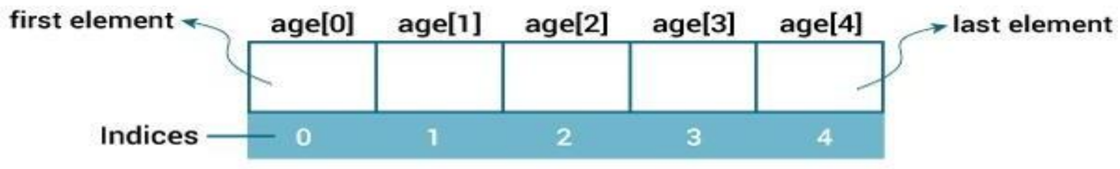
- *dataType* can be a primitive data type like: int, char, Double, byte etc. or an object (will be discussed in later chapters).

- *arrayName* is an identifier.

Let's take the above example again.
Double[] data;
Here, *data* is an array that can hold values of type Double.
But, how many elements can array this hold?
Good question! We haven't defined it yet. The next step is to allocate memory for array elements.
data = new Double[10];
The length of data array is 10. Meaning, it can hold 10 elements (10 Double values in this case).
Note, once the length of the array is defined, it cannot be changed in the program.

Let's take another example:
int[] age;
age = new int[5];
Here, *age* array can hold 5 values of type *int*.
It's possible to declare and allocate memory of an array in one statement. You can replace two statements above with a single statement.
int[] age = new int[5];

Java Array Index
You can access elements of an array by using indices. Let's consider previous example.
int[] age = new int[5];



Array age of length 5

The first element of array is age[0], second is age[1] and so on.
If the length of an array is *n*, the last element will be arrayName[n-1]. Since the length of *age* array is 5, the last element of the array is age[4] in the above example.
The default initial value of elements of an array is 0 for numeric types and false for boolean. We can demonstrate this:

```
class ArrayExample {
   public static void main(String[] args) {

      int[] age = new int[5];

      System.out.println(age[0]);
      System.out.println(age[1]);
      System.out.println(age[2]);
      System.out.println(age[3]);
      System.out.println(age[4]);
   }
}
```

When you run the program, the output will be:

```
0
0
0

0
0
```

There is a better way to access elements of an array by using looping construct (generally for loop is used).

```
class ArrayExample {
   public static void main(String[] args) {

      int[] age = new int[5];

      for (int i = 0; i < 5; ++i) {
         System.out.println(age[i]);
      }
   }
}
```

How to initialize arrays in Java?

In Java, you can initialize arrays during declaration or you can initialize (or change values) later in the program as per your requirement.

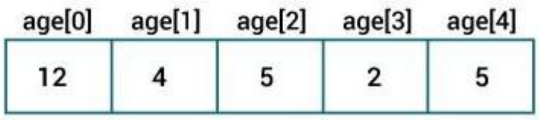
Initialize an Array during Declaration

Here's how you can initialize an array during declaration.

int[] age = {12, 4, 5, 2, 5};

This statement creates an array and initializes it during declaration.

The length of the array is determined by the number of values provided which is separated by commas. In our example, the length of age array is 5.



Let's write a simple program to print elements of this array.

```
class ArrayExample {
   public static void main(String[] args) {

      int[] age = {12, 4, 5, 2, 5};

      for (int i = 0; i < 5; ++i) {
         System.out.println("Element at index " + i +": " + age[i]);
      }
   }
}
```
When you run the program, the output will be:
```
Element at index 0: 12
Element at index 1: 4
Element at index 2: 5
Element at index 3: 2
Element at index 4: 5
```
How to access array elements?

You can easily access and alter array elements by using its numeric index. Let's take an example.

```
class ArrayExample {
   public static void main(String[] args) {

      int[] age = new int[5];

      // insert 14 to third element
      age[2] = 14;

      // insert 34 to first element
      age[0] = 34;

      for (int i = 0; i < 5; ++i) {
         System.out.println("Element at index " + i +": " + age[i]);
      }
   }
}
```
When you run the program, the output will be:
```
Element at index 0: 34
Element at index 1: 0
Element at index 2: 14
Element at index 3: 0
Element at index 4: 0
```

Example: Java arrays
The program below computes sum and average of values stored in an array of type int.

```
class SumAverage {
   public static void main(String[] args) {

      int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
      int sum = 0;
      Double average;

     for (int number: numbers) {
        sum += number;
     }

     int arrayLength = numbers.length;

     // Change sum and arrayLength to double as average is in double
     average = ((double)sum / (double)arrayLength);

     System.out.println("Sum = " + sum);
     System.out.println("Average = " + average);
   }
}
```

When you run the program, the output will be:

```
Sum = 36
Average = 3.6
```

Couple of things here.

- The for..each loop is used to access each elements of the array.

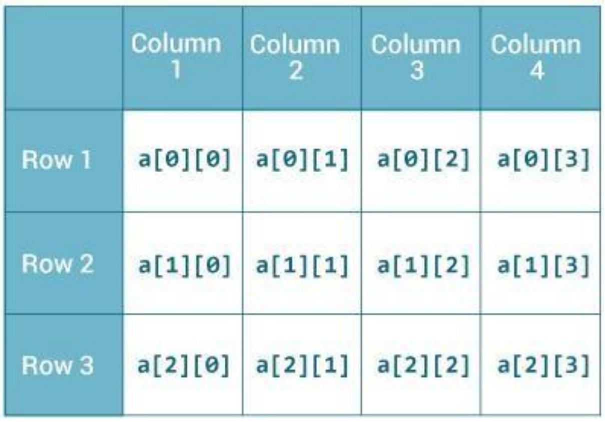
- To calculate the average, int values *sum* and *arrayLength* are converted into double since *average* is double. This is type casting. Learn more on *Java Type casting*.
- To get length of an array, length attribute is used. Here, numbers. length returns the length of *numbers* array.

Multidimensional Arrays

It's also possible to create an array of arrays known as multidimensional array. For example,

```
int[][] a = new int[3][4];
```

Here, a is a two-dimensional (2d) array. The array can hold maximum of 12 elements of type int.



	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Remember, Java uses zero-based indexing, that is, indexing of arrays in Java starts with 0 and not 1.

Similarly, you can declare a three-dimensional (3d) array. For example,

```
String[][][] personalInfo = new String[3][4][2];
```

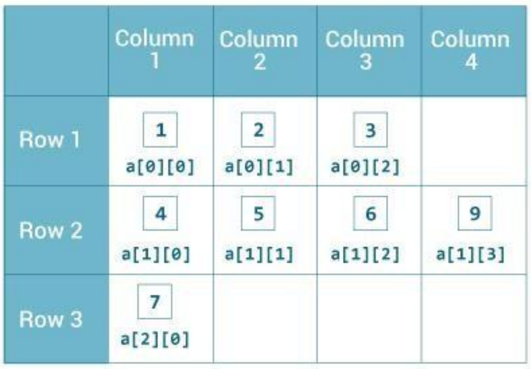
Here, *personalInfo* is a 3d array that can hold maximum of 24 (3*4*2) elements of type String.

In Java, components of a multidimensional array are also arrays.
If you know C/C++, you may feel like, multidimensional arrays in Java and C/C++ works in similar way. Well, it doesn't. In Java, rows can vary in length.
You will see the difference during initialization.

How to initialize a 2d array in Java?
Here's an example to initialize a 2d array in Java.

```
int[][] a = {                {1, 2, 3},      {4, 5, 6, 9},      {7}, };
```

As mentioned, each component of array *a* is an array in itself, and length of each rows is also different.



	Column 1	Column 2	Column 3	Column 4
Row 1	1 a[0][0]	2 a[0][1]	3 a[0][2]	
Row 2	4 a[1][0]	5 a[1][1]	6 a[1][2]	9 a[1][3]
Row 3	7 a[2][0]			

Let's write a program to prove it.

```
class MultidimensionalArray {
  public static void main(String[] args) {

    int[][] a = {           {1, 2, 3},        {4, 5, 6, 9},        {7},     };

    System.out.println("Length of row 1: " + a[0].length);
    System.out.println("Length of row 2: " + a[1].length);
    System.out.println("Length of row 3: " + a[2].length);
  }
}
```

When you run the program, the output will be:

```
Length of row 1: 3
Length of row 2: 4
Length of row 3: 1
```

Since each component of a multidimensional array is also an array (a[0], a[1] and a[2] are also arrays), you can use length attribute to find the length of each rows.

Example: Print all elements of 2d array Using Loop

```
class MultidimensionalArray {
  public static void main(String[] args) {

    int[][] a = {        {1, -2, 3},        {-4, -5, 6, 9},        {7},     };
   for (int i = 0; i < a.length; ++i) {
     for(int j = 0; j < a[i].length; ++j) {
       System.out.println(a[i][j]);
     }
   }
  }
}
```

It's better to use for..each loop to iterate through arrays whenever possible. You can perform the same task using for..each loop as:

```
class MultidimensionalArray {
  public static void main(String[] args) {

    int[][] a = {        {1, -2, 3},        {-4, -5, 6, 9},        {7},     };

   for (int[] innerArray: a) {
     for(int data: innerArray) {
       System.out.println(data);
     }
   }
  }
}
```

When you run the program, the output will be:

```
1
-2
3
-4
-5
```

6
9
7
How to initialize a 3d array in Java?
You can initialize 3d array in similar way like a 2d array. Here's an example:
// test is a 3d array
int[][][] test = {   {   {1, -2, 3},   {2, 3, 4}   },   {
{-4, -5, 6, 9},   {1},   {2, 3}   } };
Basically, 3d array is an array of 2d arrays.
Similar like 2d arrays, rows of 3d arrays can vary in length.

Example: Program to print elements of 3d array using loop

```
class ThreeArray {
  public static void main(String[] args) {

    // test is a 3d array
    int[][][] test = {            {              {1, -2, 3},              {2, 3, 4}            },            {
{-4, -5, 6, 9},              {1},             {2, 3}           }      };

    // for..each loop to iterate through elements of 3d array
    for (int[][] array2D: test) {
      for (int[] array1D: array2D) {
        for(int item: array1D) {
          System.out.println(item);
        }
      }
    }
  }
}
```

When you run the program, the output will be:

```
1
-2
3
2
3
4
-4
-5
6
9
1
2
```