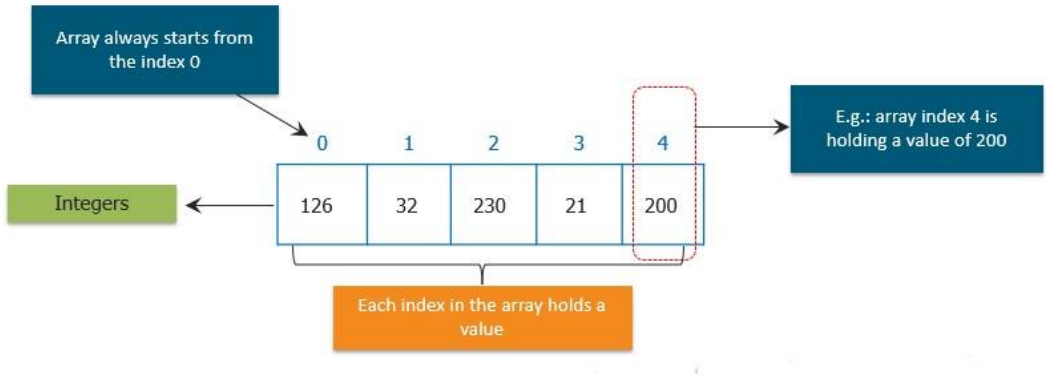
Array in Java

Array is a collection of elements of the same data type, arranged in a contiguous block of memory. Each element of an array can be accessed by its index or position within the array.

Arrays in Java are declared with a specific data type, followed by square brackets "[ ]" indicating the size of the array.



➔ Advantage of Array

- **Easy to use:** Arrays are easy to use and implement, making them a popular choice among programmers.
- **Fast access:** Arrays provide fast and efficient access to elements based on their index, which makes them ideal for storing and retrieving data quickly.
- **Memory efficiency:** Arrays are memory-efficient, as they store data in a contiguous block of memory, which makes them ideal for handling large amounts of data.
- **Easy to manipulate:** Arrays can be easily manipulated using loops, making it easy to perform operations on all elements of an array.

➔ Disadvantage of Array

- **Fixed size:** Arrays in Java are of fixed size, which means that the size of the array cannot be changed once it is initialized.
- **Lack of flexibility:** Arrays cannot be resized dynamically, which means that if you need to add or remove elements from an array, you need to create a new array with a different size.
- **Inefficient for certain operations:** Arrays are inefficient for certain operations, such as sorting and searching, as these operations can require a lot of computational power and time to execute.
- **Complex data types:** Arrays are not suitable for storing complex data types, such as objects and structures, as they can only store a single data type.

Types of Array

- **Single-Dimensional array**
- **Multi-Dimensional array**

➔ Single-Dimensional Array :

A single-dimensional array is a collection of elements of the same data type that are stored in a contiguous block of memory. Each element in the array is accessed using an index, which starts from 0 and goes up to the length of the array minus 1.

Declare an array in Java

In Java, here is how we can declare an array.

```
dataType[] arrayName;
```

- `dataType` - it can be primitive data types like `int`, `char`, `double`, `byte`, etc. or Java objects
- `arrayName` - it is an identifier

For example,

```
double[] data;
```

Here, `data` is an array that can hold values of type `double`.

But, how many elements can array this hold?

Good question! To define the number of elements that an array can hold, we have to allocate memory for the array in Java. For example,

```
// declare an array double[] data;

// allocate memory data = new double[10];
```

Here, the array can store **10** elements. We can also say that the **size or length** of the array is 10.

In Java, we can declare and allocate the memory of an array in one single statement.

For example,

```
double[] data = new double[10];
```

Initialize Arrays in Java

In Java, we can initialize arrays during declaration. For example,

```
//declare and initialize and array int[]
age = {12, 4, 5, 2, 5};
```

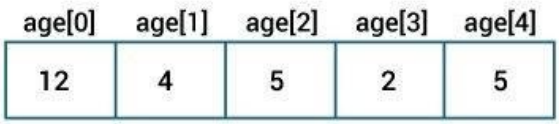
Here, we have created an array named age and initialized it with the values inside the curly brackets.

Note that we have not provided the size of the array. In this case, the Java compiler automatically specifies the size by counting the number of elements in the array (i.e. 5).

In the Java array, each memory location is associated with a number. The number is known as an array index. We can also initialize arrays in Java, using the index number. For example,

```
// declare an array
int[] age = new int[5];

// initialize array
age[0] = 12;
age[1] = 4; age[2]
= 5;
..
```



Java Arrays initialization

Note:

- Array indices always start from 0. That is, the first element of an array is at index 0.
- If the size of an array is `n`, then the last element of the array will be at index `n-1`.

Access Elements of an Array in Java

We can access the element of an array using the index number. Here is the syntax for accessing elements of an array,

```
// access array elements array[index]
```

Let's see an example of accessing array elements using index numbers.

Example: Access Array Elements

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

  // create an array
  int[] age = {12, 4, 5, 2, 5};

  // access each array elements
  System.out.println("Accessing Elements of Array:");
  System.out.println("First Element: " + age[0]);
  System.out.println("Second Element: " + age[1]);
  System.out.println("Third Element: " + age[2]);
  System.out.println("Fourth Element: " + age[3]);
  System.out.println("Fifth Element: " + age[4]);
 }
}
```

Output

```
Accessing Elements of Array:
First Element: 12
Second Element: 4
Third Element: 5
Fourth Element: 2
Fifth Element: 5
```

➔ Multi-Dimensional Array :

A multidimensional array is an array of arrays. Each element of a multidimensional array is an array itself. For example,

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

Here, we have created a multidimensional array named `a`. It is a 2dimensional array, that can hold a maximum of 12 elements,

	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]

2-dimensional Array

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

Let's take another example of the multidimensional array. This time we will be creating a 3-dimensional array. For example,

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

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

Initialize a 2d array in Java?

Here is how we can initialize a 2-dimensional array in Java.

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

As we can see, each element of the multidimensional array is an array itself. And also, unlike C/C++, each row of the multidimensional array in Java can be of different lengths.

	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]			

Example: 2-dimensional Array

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

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

      // calculate the length of each row
      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);
   }
}
```

Output:

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

In the above example, we are creating a multidimensional array named `a`.

Since each component of a multidimensional array is also an array (`a[0]`, `a[1]` and `a[2]` are also arrays).

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]);
         }
      }
   }
}
```

Output:

```
1
-2
3
-4
-5
6
9
7
```

We can also use the for...each loop to access elements of the multidimensional array. For example,

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

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

        // first for...each loop access the individual array
        // inside the 2d array
        for (int[] innerArray: a) {
            // second for...each loop access each element inside the row
            for(int data: innerArray) {
                System.out.println(data);
            }
        }
    }
}
```

Output:

```
1
-2
3
-4
-5
6
9
7
```

In the above example, we are have created a 2d array named `a`. We then used `for` loop and `for...each` loop to access each element of the array.

Initialize a 3d array in Java

Let's see how we can use a 3d array in Java. We can initialize a 3d array similar to the 2d array. For example,

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

Basically, a 3d array is an array of 2d arrays. The rows of a 3d array can also vary in length just like in a 2d array.

Example: 3-dimensional Array

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

      //  create  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);
                }
            }
        }
    }
}
```

Output:

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