

Java SE & Spring

Module 1: Java SE

05.Arrays



Arrays

- An array is a collection of similar types of data.
- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- For example, if we want to store the names of 100 people then we can create an array of the string type that can store 100 names.
- Array indexes start with 0: [0] is the first element. [1] is the second element, etc. So, an array's n^{th} element index is [n-1].

Declaring arrays

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

Syntax:

```
dataType[] arrayName;
```

- **dataType** - it can be primitive data types like **int**, **char**, **double**, **byte**, etc. or Java objects (will be discussed in next topic)
- **arrayName** - it is an identifier

Example:

```
int[] masses;
```

Here, **ages** is an array that can hold values of type int.

How many elements can array hold?

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

```
1 // declare an array
2 int[] masses;
3
4 // allocate memory
5 masses = new double[10];
```

We can declare and allocate memory of an array in one single statement(Most of the time, this one is preferred):

```
int[] masses = new double[10];
```

Initializing arrays

- We can initialize arrays during declaration:

```
1 //declare and initialize and array
2 int[] masses= {62, 54, 75, 52, 85, 79};
```

- We have created an array named masses and initialized it with the values inside the curly brackets {}.

As you notice, 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. 6). Arrays can also be initialized, using the index number:

```
1 // declare an array
2 int[] masses = new int[5];
3 // initialize array
4 masses[0] = 62;
5 masses[1] = 54;
6 masses[3] = 75;
```

Initializing arrays- Behind the scenes

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

0	masses[0]
0	masses[1]
0	masses[2]
0	masses[3]
0	masses[4]

```
masses[0] = 62;  
masses[1] = 54;  
masses[3] = 75;
```

62	masses[0]
54	masses[1]
0	masses[2]
75	masses[3]
0	masses[4]

Accessing array elements

We can access the element of an array using the index number.

Syntax:

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

Example:

```
1  int[] age = {12, 4, 5, 2, 5};  
2  
3  // access each array elements  
4  System.out.println("Accessing Elements of Array:");  
5  System.out.println("First Element: " + age[0]);  
6  System.out.println("Second Element: " + age[1]);  
7  System.out.println("Third Element: " + age[2]);  
8  System.out.println("Fourth Element: " + age[3]);  
9  System.out.println("Fifth Element: " + age[4]);
```

Further reading: <https://www.programiz.com/java-programming/arrays>

Looping through array elements

We can also loop through each element of the array rather than get one by one.

```
1  int[] ages = {12, 4, 5, 2, 5};
2
3  // looping through array elements via classical for loop
4  for(int i = 0; i < ages.length; i++){
5      System.out.println(ages[i]);
6  }
7
8  // looping through array elements via enhanced for loop (foreach loop)
9  for (int age : ages) {
10     System.out.println("Age = " + age);
11 }
```

Question: What if we write `i <= ages.length` rather than the current one at line 4?

Try to guess 😊, the answer is in the next slide.

ArrayIndexOutOfBoundsException

```
int[] ages = {12, 4, 25, 32, 45};
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]

```
i <= ages.length
```

12	masses[0]	← i=0
4	masses[1]	
25	masses[2]	
32	masses[3]	
45	masses[4]	


ArrayIndexOutOfBoundsException

```
int[] ages = {12, 4, 25, 32, 45};
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]

```
i <= ages.length
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]



ArrayIndexOutOfBoundsException

```
int[] ages = {12, 4, 25, 32, 45};
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]

```
i <= ages.length
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]

← i=2

ArrayIndexOutOfBoundsException

```
int[] ages = {12, 4, 25, 32, 45};
```

12	ages[0]
4	ages[1]
25	ages[2]
32	ages[3]
45	ages[4]

```
i <= ages.length
```

12	ages[0]
4	ages[1]
25	ages[2]
32	ages[3]
45	ages[4]

← i=3


ArrayIndexOutOfBoundsException

```
int[] ages = {12, 4, 25, 32, 45};
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]

```
i <= ages.length
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]

 i=4

ArrayIndexOutOfBoundsException

```
int[] ages = {12, 4, 25, 32, 45};
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]

At last iteration the i will be 5

```
i <= ages.length
```

12	masses[0]
4	masses[1]
25	masses[2]
32	masses[3]
45	masses[4]
?	masses[5]



Multidimensional Arrays

A multidimensional array is an array of arrays. Each element of a multidimensional array is an array itself. In such case, data is stored in row and column based index (also known as matrix form). Multidimensional arrays are used if we want to put arrays inside an array. For example: We have 4 students, each individual student has 3 exams. We can hold this exam results in a 2 dimensional array like below:

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

With this way we can hold exam results more effectively rather than creating an array consists 12 elements.

Multidimensional Arrays

Our 2-dimensional exams array will be look like this:

	col1	col2	col3
row1	<code>exams[0][0]</code>	<code>exams[0][1]</code>	<code>exams[0][2]</code>
row2	<code>exams[1][0]</code>	<code>exams[1][1]</code>	<code>exams[1][2]</code>
row3	<code>exams[2][0]</code>	<code>exams[2][1]</code>	<code>exams[2][2]</code>
row4	<code>exams[3][0]</code>	<code>exams[3][1]</code>	<code>exams[3][2]</code>

Initializing Multidimensional Arrays

```
1 int[][] exams = {  
2     {70, 58, 83},  
3     {74, 95, 60},  
4     {37},  
5     {67, 49}  
6 };
```

	col1	col2	col3
row1	70 exams[0][0]	58 exams[0][1]	83 exams[0][2]
row2	74 exams[1][0]	95 exams[1][1]	60 exams[1][2]
row3	37 exams[2][0]	? exams[2][1]	? exams[2][2]
row4	67 exams[3][0]	49 exams[3][1]	? exams[3][2]



Practice Time: Let's Code Together!

📌 Exercise:

Create an array for exam scores of a class that contains 13 students.

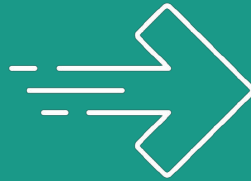
1. Initialize the array (fill the array with exam scores)
2. Calculate the total scores of the class, and print it out to the console.
3. Calculate average point of the class, and print it out to the console.
4. If the average score is greater than 70, print “The class is successful, average score is: {average}”,
otherwise print “The class should study harder, average score is: {average}”

🎁 Bonus Exercise:

Print out all elements of the array to the console in slide 18 (Initializing Multidimensional Arrays).



Questions ?



Next: Object Oriented Programming (OOP)