Lesson 4.1

Array

By the end of this lesson you will learn:

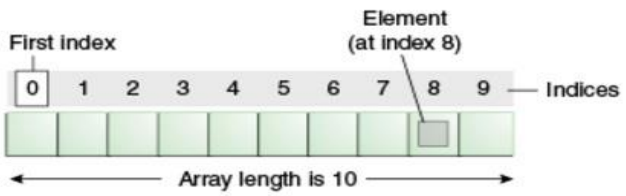
- Array
- 1D Array Declaration
- 2D Array Declaration
- Accessing array elements
- Array of Objects
- Manipulating an Array of Objects

Array is a very powerful feature which provides a very convenient way to group related or similar information. In java it works differently than any other language like, C/C++.

Array

An array is a group of similar data type which are referred by a common name where each element works like an individual variable. It is index-based, and the first element of the array is stored at the index 0 and the last index will be SIZE_OF_ARRAY-1. So, it is a data structure where we can store fixed set of similar type of data.

In java, we can store primitives like, int, float, double, char, etc. as well as object (which is non-primitives) references of a class depending on the definition of array. In case of primitives data types, the actual values are stored in contiguous memory locations. In case of objects of a class, the actual objects are stored in heap segment.



An array of 10 elements.

One-Dimensional/ 1D Array

A one-dimensional array is a list of same-typed variables. To create an array, we first must create an array variable of the desired type.

There are ***four approaches*** to declare a 1D array.

Declaring 1D Array : 1st Approach

```
int arr1[ ] = new int [5];

        Or,

int [ ]arr1 = new int [5];
```

So, here we have declared an integer type of array named arr1. The square bracket[] indicates that it is an array declaration not a variable. Also, it can be noted that the **[ ]** symbol (better known as Array Notation) can be placed both before and after the name of the array. The keyword "new" is used to allocate memory for the array. int [5] denotes that memory need to be allocated for 5 integer values.

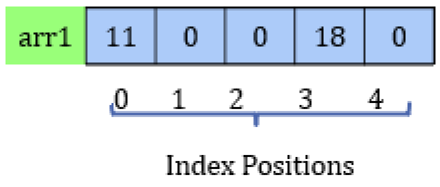
Memory Representation and Initialization of an Array: 1st Approach

This approach is used when we know the size of the array, but we do not know the elements of the array. As we do not know the elements we cannot initialize the array yet.

According to the concept of default values, all the index positions will be initialized as 0. Now, if we want to initialize a value in a particular index, we can do it like this:

```
arr1 [0] = 11;

arr1 [3] = 18;
```



Similarly, we can also access the elements of the array using their index positions.

Declaring 1D Array : 2nd Approach

```
int [ ]arr2;
.
.
.
arr2 = new int [size];
```

Here,

int is the type of array and arr2 is the name of the array. The [ ] symbol denotes that it is an array, not a variable. **size** is the size of array, which was computed somewhere in between the two statements.

Memory Representation and Initialization of an Array: 2nd Approach

This approach is used when we neither know the size nor the elements of the array. So, initially we cannot allocate memory for the array. Memory allocation is done after knowing the value of size.

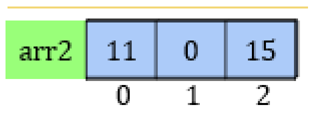
So, for the size calculation,

int arr2[ ];

int size = arr1[3] / 6;

arr2 = new int [size];

size = 18 / 6 = 3



According to the concept of default values, all the index positions will be initialized as 0. Now, if we want to initialize a value in a particular index, we can do it like this:

arr2 [0] = 11;

arr2 [2] = 15;

Similarly, we can also access the elements of the array using their index positions.

Declaring and Initializing 1D Array : 3rd Approach

int arr3[ ] = new int [ ] {11,22,33,44};

Or,

int [ ]arr3 = new int [ ] {11,22,33,44};

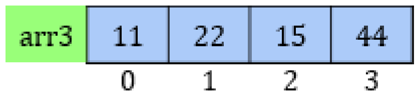
Here,

int is the type of array and **arr3** is the name of the array. The **[ ]** symbol denotes that it is an array, not a variable. **new** keyword allocates memory for the array. **int [ ] {11,22,33,44}** denotes that memory will be allocated for four integers (as there are four integers inside the curly braces) and the value 11, 22, 33 and 44 will be initialized in the array.

This approach is used when we do not know the size of the array, but we know the elements of the array. The size gets computed automatically from the number of elements of the array.

Memory Representation of an Array: 3rd Approach

int arr3[ ] = new int [ ] {11,22,33,44};



If we want to change a value in a particular index, we can do it like this:

arr3 [2] = 15;

Similarly, we can also access the elements of the array using their index positions.

Declaring and Initializing 1D Array : 4th Approach

This approach is NOT RECOMMENDED.

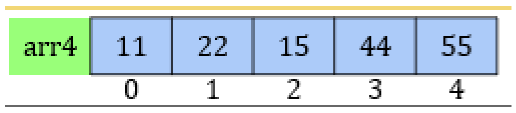
int arr4[ ] = {11,22,33,44,55};

Or,

int [ ]arr4 = {11,22,33,44,55};

Here,

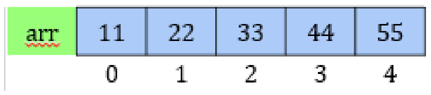
int is the type of array and **arr4** is the name of the array. The **[ ]** symbol denotes that it is an array, not a variable. Memory will be allocated for five integers (as there are five integers inside the curly braces) and the value 11, 22, 33, 44 and 55 will be initialized in the array.



Printing array elements

We can simply print all the elements of an array using a loop. Let's assume that we have the following array:

```
int arr[ ] = new int [ ] {11,22,33,44,55};
```



Using following any of the code snippets below one can print all the stored elements in an array. Here length attribute of any array represents the size of the array.

```
for(int i=0; i<arr.length; i++){                    // using for loop
   System.out.println(arr[i]);
}
int i=0;                                            // using while loop
while(i<arr.length){
   System.out.println(arr[i]);
   i++;
}
int i=0;                                            // using do- while loop
do{
   System.out.println(arr[i]);
   i++;
}
while(i<arr.length);
```

Two-Dimensional Array/ 2D Array:

Declaring 2D Array

1st Approach:

```
int arr5[ ][ ] = new int [3][3];
```

2nd Approach:

```
int arr6[ ][ ];
row = arr1[3]/9;
col = arr1[3]/6;
arr6 = new int [row][col];
```

3rd Approach:

```
int arr7 [ ][ ] = new int [ ][ ]{{1,2}, {3,4}};
```

4th Approach:

```
int arr8[ ][ ] = {{7,8},{9,5}};                // This approach is NOT RECOMMENDED.
```

Here,

The [ ][ ] symbol denotes that it is a 2D array. The [ ][ ] symbols can also be written before the name of the array. The value in the 1st [ ] represents the number of rows and the value in 2nd [ ] represents the number of columns.

Memory Representation of 2D Array

1st Approach

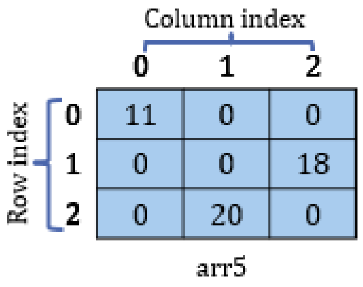
int arr5[ ][ ] = new int [3][3];

As, no elements has been initialized, all the index positions will be initialized by the default value 0. As, no elements has been initialized, all the index positions will be initialized by the default value 0.

Now, if we want to initialize a value in a particular index, we can do it like this:

```
arr5 [0][0] = 11;
arr5 [1][2] = 18;
```

arr5 [2][1] = 20;



2nd Approach:

int arr6[ ][ ];

row = arr1[3]/9;

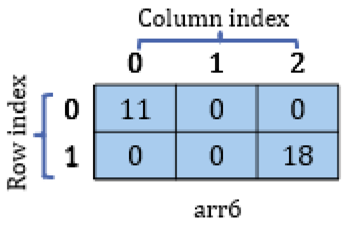
col = arr1[3]/6;

arr6 = new int [row][col];

As, no elements has been initialized, all the index positions will be initialized by the default value 0. Now, if we want to initialize a value in a particular index, we can do it like this:

arr6 [0][0] = 11;

arr6 [1][2] = 18;



3rd Approach

int arr7 [ ][ ] = new int [ ][ ]{{1,2}, {3,4}};

int [ ][ ]{{1,2}, {3,4}} denotes that there are two rows as there are two pairs of inner curly braces and there are two columns as each pair of inner curly braces have two elements.

Now, if we want to initialize a value in a particular index, we can do it like this:

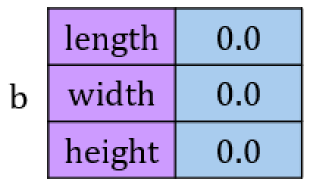
arr7 [0][0] = 11;

arr7 [1][1] = 18;

Array of Objects

The following statement creates an object of the Box class we studied previously:

Box b = new Box( );

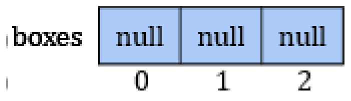


Here, b can store data for only one Box object. So, if we need to store information of multiple boxes then how can we store?

The solution would be an array of Box.

Box boxes[ ] = new Box [3];

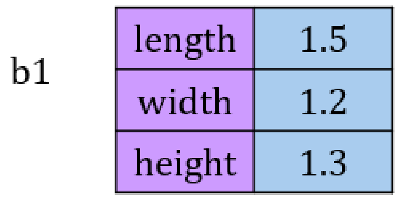
As, no objects has been initialized in the boxes array, the default value for each of the indexes will be ***null***.



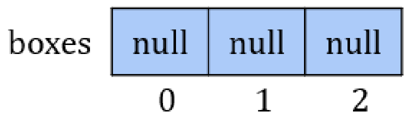
We can create object of Box class referring the index of the boxes array or we can create object of Box class and assign them in the boxes array.

Let's assume we have created an object of Box using its parameterized constructor and we have an array of Box.

Box b1 = new Box(1.5,1.2,1.3);

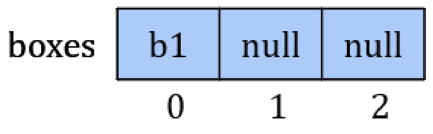


Box boxes[ ] = new Box [3];



Now, if we want to assign the b1 object in the boxes[0] position then it will be like below,

boxes[0] = b1;

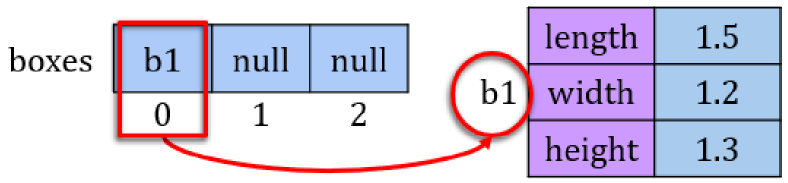


So, now, the following two statements will yield the same output:

System.out.println(b1.getLength( ));

System.out.println(boxes[0].getLength( ));

For both statements, the output will be 1.5, the reason is explained below.



Memory location of boxes[0] holds the reference b1.

So, they both refers to the same object.

We can also create an object and refer that object directly by using an index of the array.

boxes[1] = new Box(2.7,2.5,1.2);

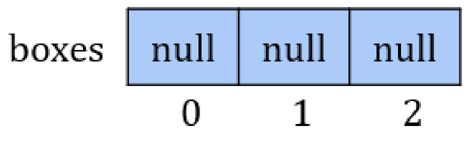
Now, we can use boxes[1] to access that object. For Example:

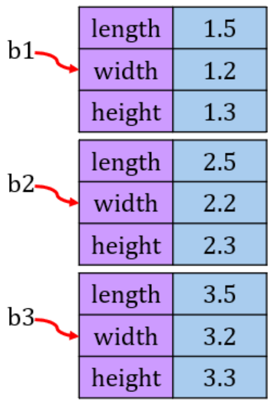
System.out.println(boxes[1].getLength( ));

Then the output will be 2.7

Manipulating an Array of Objects

Let's Assume that we have 3 objects of box class and an array of Box class.





Now, if we want to assign the objects in the array. Then here is the solution.

<u>Solution:</u>

We can only assign these objects in the array if there is *null* value in any of the indexes. The followings steps can be followed to assign an object in the array:

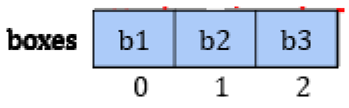
- Start from the 1st index of the array and check the value of that index.
- If it is *null*, assign the object and exit. Else, go to the next index.
- Repeat until the last index.

Now, step by step the whole solution is given below:

Start from the 1st index of the array and check the value of that index.

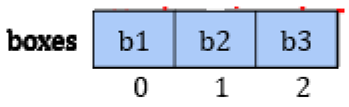
- If it is ***null***, assign the object and exit. Else, go to the next index.
- Repeat until the last index.

Let's assign all the three objects referred by b1, b2 and b3 respectively, by following above steps,



Removing an object from array:

Let's Assume that we have an array of Box class and 3 objects of Box is already assigned in the array. Now, if we want to remove the object b2 from the array then the solution is given below,

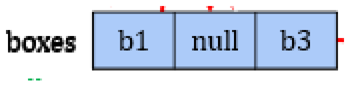


<u>Solution</u>:

We can only assign these objects in the array if there is *b2* in any of the indexes. The followings steps can be followed to remove an object from the array:

- Start from the 1st index of the array and check the value of that index.
- If it is *b2*, assign the null and exit. Else, go to the next index.
- Repeat until the last index.

After removing b2,



Printing all the data:

Let's Assume that we want to print the data of the boxes stored inside the array.

<u>Solution</u>:

We cannot print data from an index if there is ***null*** in that index*.* The followings steps can be followed to print data of the boxes stored in the array:

- Start from the 1st index of the array and check the value of that index.
- If it is not ***null***, print data and go to next index. Else, ignore and go to next index.
- Repeat until the last index.