



Java Foundation with Data Structures

Lecture 6: Arrays

Array Description

What are arrays?

In cases where there is a need to use several variables of same type, for storing, example, names or marks of 'n' students we use a data structure called arrays. Arrays are basically collection of fixed number of elements of a single type. Using arrays, saves us from the time and effort required to declare each of the element of array individually.

The length of an **array** is established when the **array** is created. After creation, its length is fixed. For example: {1,2,3,4,5} is an array of integer.

Similarly, an array can be a collection of character, Boolean, Double as well.

Declaring Array Variables

To use an array in a program, you must declare a variable to refer the array, and you must specify the type (which once specified can't be changed) of array the variable can reference. Here is the syntax for declaring an array variable –

Syntax

```
datatype [] arrayRefVar; // preferred way.
```

OR

```
datatype arrayRefVar [];
```

Example:

```
int [] arr;
```

OR

```
int arr [];
```

Creating Array

Declaring array variable does not create array (i.e. no space is reserved for array).

Here is the **syntax** for creating an array –

```
arrayRefVar = new datatype [array Size];
```

Example:

```
arr=new int [20];
```

The above statement does two things –

- It creates an array using **new** keyword [array Size].

- It assigns the reference of the newly created array to the variable arrayRefVar.

Combining declaration of array variable, creating array and assigning the reference of the array to the variable can be combined in one statement, as shown below –

```
datatype [] arrayRefVar = new datatype [array Size];
```

Example:

```
int [] arr = new int [20]; // This statement will allocate contiguous space for 20 integers
```

Array Indexes

In order to access different elements in an array - all elements in array are indexed and indexing starts from 0. So if there are 5 elements in array then first index will be 0 and last one will be 4. Similarly, if we have to store n values in an array, then indexes will range from 0 to n-1.

```
int [] arr = { 1 , 2 , 3 , 4 , 5 };
               ↑   ↑   ↑   ↑   ↑
            arr[0] arr[1] arr[2] arr[3] arr[4]
```

- Trying to retrieve an element from an invalid index will give an **ArrayIndexOutOfBoundsException**.

Initializing an Array

In Single line

Syntax of creating and initializing an array in single line ...

```
dataType [] arrayRefVar = {value0, value1, ..., value};
```

Example:

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

Using loop

Array can be initialized using loop.

Example:

```
public static void main(String[] args) {
    int [] arr = new int [20];
    Scanner Scan = new Scanner(System.in);
```

```

        for (int i=0 ; i<arr.length; i++){
            arr[i]=Scan.nextInt();
        }
    }
}

```

Default values for different data types with which an array is initialized

All the elements of the arrays after creating arrays are initialized to default values (if we not initialize them while creation). Following is a table showing default values for various data_types.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

For-Each Loop

This is a special type of loop to access array elements of array. But this loop can be used only to traverse array, nothing can be changed in array using this loop.

For Example:

```

public class Solutions {
    public static void main (String [] args) {
        int [] arr= {10,20,30,40,50};
        for (int i:arr)
        {
            System.out.print(i+" ");
        }
    }
}

```

Output:

1 2 3 4 5

How arrays are stored?

Arrays in Java store one of two things: either primitive values (int, char,) or references (a.k.a pointers).

When an object is created by using “new”, memory is allocated on the heap and a reference is returned. This is also true for arrays, since arrays are objects.

```
int arr [] = new int [10]; //here arr is a reference to the array and not the name of the array.
```

Reassigning references and Garbage collector

All the reference variables (not final) can be reassigned again and again but their data type to whom they will refer is fixed at the time of their declaration.

For Example:

```
public class Solutions {  
    public static void main (String [] args) {  
        int [] arr=new int [20]; // HERE arr is a reference not array name...  
        int [] arr1=new int [10];  
        arr=arr1; // We can re-assign arr to the arrays which referred by arr1. Both arr and arr1 refer to the same arrays now.  
    }  
}
```

In Above example, we create two reference variables **arr** and **arr1**. So now there are also 2 objects in the garbage collection heap.

If you assign arr1 reference variable to arr, then no reference will be present for the 20 integer space created earlier, so this block of memory can now be freed by Garbage collector.

Garbage collector

Live objects(We can think of them as blocks of memory for now) are tracked and everything else designated garbage. As you'll see, this fundamental misunderstanding can lead to many performance problems.

- When an object is no longer used, the garbage collector reclaims the underlying memory and reuses it for future object allocation. This means

there is no explicit deletion and no memory is given back to the operating system.

New objects are simply allocated at the end of the used heap

Passing Arrays to Functions

Passing Array as function parameter

In Java programming language the parameter passing is always, ALWAYS, made by value. Whenever we create a variable of a type, we pass a copy of its value to the method.

Passing Reference Type // In case of array reference of array is passed

As we studied in lecture that we store references of Non-Primitive data types and access them via references, So in such cases the references of Non-Primitive's are passed to function.

For Example:

```
public class Solutions {  
  
    public static void print(int [] arr)  
    {  
        for (int i=0;i<5;i++)  
        {  
            System.out.print(arr[i]+" ");  
        }  
    }  
    public static void main (String [] args) {  
        int [] arr= {1,2,3,4,5};  
        print(arr); //--> Reference to array is passed  
    }  
}
```

Similarly, When we pass a an array to the increment function shown below then the reference(address) to the array is passed and not the array itself.

```
public class Solutions {  
  
    public static void increment (int [] arr)  
    {  
        for (int i=0;i<5;i++)  
        {
```

```

        arr[i]++;
    }
}
public static void main (String [] args) {
    int [] arr= {1,2,3,4,5};
    increment(arr);
    for (int i=0;i<5;i++)
    {
        System.out.print(arr[i]+" ");
    }
}
}

```

Output:

2 3 4 5 6

Here reference to the array was passed. Thus inside increment function arr refers to the same array which was created in main. Hence the changes by increment function are performed on the same array and they will reflect in main.

Now, lets change code for increment function a little and make arr point to another array as shown in example given below.

```

public class Solutions {

    public static void increment (int [] arr)
    {
        int [] arr1= {1,2,3,4,5};
        arr=arr1;
        for (int i=0;i<5;i++)
        {
            arr[i]++;
        }
    }

    public static void main (String [] args) {
        int [] arr= {1,2,3,4,5};
        increment(arr);
        for (int i=0;i<5;i++)
        {
            System.out.print(arr[i]+" ");
        }
    }
}

```

```

    }
}

```

Output:

1 2 3 4 5

Here the changes done in main didn't reflect. Although here as well the reference to the array was passed, but in the first line inside function we created another array of size 5 and made arr refer to that array (without affecting array created in main). Thus the changes this time won't reflect.

Returning Array from a method

Similarly, as we pass reference as a function parameter we will return reference too in case of array.

For Example:

```

class ArrayUse{
    public static void main(String[] args){
        int[] A = numbers();
    }

    public static int[] numbers(){
        int[] A = new int[3];
        A[0] = 2;
        A[1] = 3;
        A[2] = 4;
        return A;
    }
}

```