Arrays

Introduction

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

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

Similarly, an array can be a collection of characters, Boolean, and Double.

Declaring Array Variables

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

Syntax:

```
datatype [] arrayRefVar;
datatype arrayRefVar []
```

```
Int [] arr;
Int arr []
```

Creating Array:

Declaring an array variable does not create an array (i.e. no space is reserved for an 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 the new keyword [array Size].

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

Combining the declaration of an array variable, creating an 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];
```

Array Indexes

To access different elements in an array -- all elements in the array are indexed, and indexing starts from 0. So if the array has five elements, the first index will be 0, and the 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. Trying to retrieve an element from an invalid index will give an **ArrayIndexOutOfBondsException**.

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

arr[0] = 1 arr[1] = 2 arr[2] = 3 arr[3] = 4 arr[4] = 5

Initialising an Array

In Single Line

Syntax of creating and initialising an array in a single line, dataType [] arrayRefVar =

{value0, value1, ..., value};

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

Loops

You can use loops to iterate through arrays. You can use several loops: for loop, while loop, and for each loop. Here's how you can use each of them to iterate through an array:

```
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();
      }
    }
```

For Each Loop

This is a special type of loop to access array elements of an array. But we can use this loop only to traverse an array. Nothing can be changed in the array using this loop.
The syntax is as follows:

```
for (datatype variable: array) {
    // code to be executed for each element in the array
}
```

Here, the data type is the data type of the elements in the array, the variable is a variable of that data type, and the array is the array you want to iterate through. The for-each loop automatically iterates through the array, assigning each element to the variable, and executes the code inside the loop for each element.

Here's an example of how you can use the enhanced for loop to iterate through an array:

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

How are Arrays Stored?

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

When an object is created 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
```

Reassigning references and Garbage collector

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

```
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 is referred by arr1.
Both arr and arr1 refer to the same arrays now.
}
}
```

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

Suppose you assign the arr1 reference variable to arr. In that case, no reference will be present for the 20 integer space created earlier, so the Garbage Collector can now free this memory block.

Passing Arrays to Functions

Passing Array as a function parameter

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

Passing Reference Type

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

Arrays are objects, and when you pass an array to a method, you're actually passing the reference to the array. This means any modifications made to the array within the method will affect the original array because it's the same underlying data structure being accessed.

```
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);
  }
}
```

In the provided code:

- The print method takes an array (which is a reference type) as an argument (pass-by-value), allowing it to print the contents of the array.
- The main method creates an array and calls the print method, passing the array as an argument (pass-by-value, where the value is the reference to the array).
- In summary, while Java is a "pass-by-value," the distinction is essential when working with reference types like arrays. You're passing a copy of the reference (address), and modifications to the object (or array) inside the method affect the original object because it's the same underlying object referenced.

Similarly, when we pass an array to the increment function shown below, the reference(address) to the array is passed, 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 the reference to the array was passed. Thus, the inside increment function arr refers to the same array created in the main. Hence, the changes by increment function are performed on the same array and will reflect in the main.