Arrays are Objects

An array is an object container that holds fixed number of values of certain data type. The length of the array is established when it is created. Once created its length remain fixed. The specification says that array variable hold object references (just like class objects) i.e. they hold the reference of an object stored in heap memory.

Different Ways to Initialize Arrays

There are two ways:

1) **Using new operator:** When we want to give values later.
Example:
int[] arrayRefName = new int[10];
This will create an array in integers with size 10.

2) **Aggregate Initialization:** When we want to create array with given values
Example:
int[] arrayRefName = {1, 4, 5};
This will create an array of size 3 holding 1, 4 and 5.

Array Memory Allocation

Arrays just like objects are allocated memory in heap.

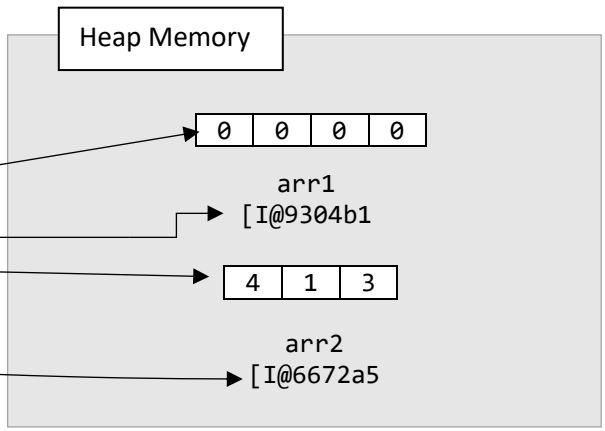
Arrays are allocated memory as soon as we initialize them **with values (aggregate initialization)** or **using new operator (giving it size).** However, if we have just declared an array and try to use this array we'll get a compile time error because our array isn't initialized.

Array reference variables holds the address (often called **a reference** in java terminology) of the actual object (array) stored in heap memory. We can understand it better through a memory diagram consider following code segment.

Example 1.1: Array Memory Allocation

Code:

```
1   int[] arr1 = new int[4];
2
3   int[] arr2 = {4,1,3};
4   System.out.println(arr1);
5   System.out.println(arr2);
```



Output:

```
[I@9304b1
[I@6672a5
```

Explanation: As you can see from the diagram we have created two objects (arrays) of int data type in heap memory.

The **new operator** creates the objects in memory and returns the reference of newly constructed objects. Same happens in case of aggregate initialization.

The **reference values** are stored in arr1 and arr2. All array variables hold references.

Note: Directly printing an array reference variable will print its reference value.

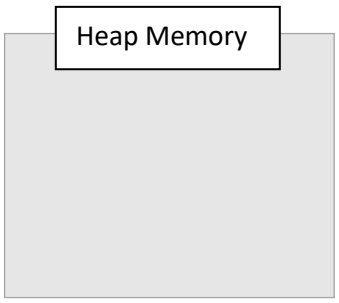
References are of this form:

[DataType@HexadecimalCode,

Where:

- Single left-bracket denotes 1D array, [[for 2D and so on…
- Datatype: I for integers, D for double, java.lang.String for Strings….

Example 1.2: Array Memory Allocation 2



Code:

```
1  int[] arr3;
2  char[] arr4;
```

Explanation: The program will not create any array in memory. The code merely declares two reference variables arr3 and arr4. Actual memory is allocated when we initialize our array.

Example 1.3: Common errors

Code:

```
1  int[] arr;
2  System.out.println(arr[0]);
```

Output:

```
Error in Line 2
```

Explanation: The above program will generate compile time error in line 2 because we haven't given any size to our array (not initialized). We cannot use an array without providing size first.

Example 1.4: null reference

Code:

```
1  int[] arr = null;
2  System.out.println(arr);
3  System.out.println(arr[0]);
```

Output:

```
null
NullPointerException at line 3
```

Explanation: arr holds null reference. This means it does not point to any valid object in memory. Directly, printing arr will give null.

Example 1.5: Error when Array size not given

Code:

```
1   int[] arr = new int[];
2   arr[0] = 100;
3   System.out.println(arr[0]);
```

Output:

```
Compile time error in line 1
```

Explanation: We cannot create an array with new without giving any size to it. Leaving empty [] will generate compile time error.

Array of Strings and other data types

We can create an array of any data type. Upon creation the array will contain default values depending on the data type of array.

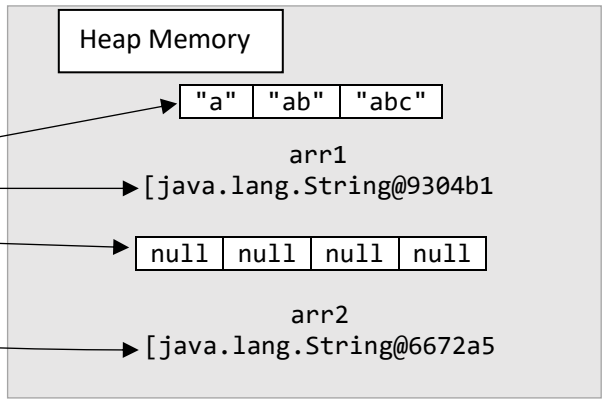
Data type	Default value
boolean	false
char	'\u0000'
int	0
double	0.0
Any reference type (eg. String)	null

We know String is a class in Java. So when we create an array of Strings we are actually creating an array of objects.

Example 2: Array of Strings

Code:

```
1   String[] arr1 = {"a", "ab", "abc"};
2
3   String[] arr2 = new String[4];
4   System.out.println(arr1[1]);
5   System.out.println(arr2[1]);
6
7   String str3 = arr1[2] + arr2[0];
```



```
8   System.out.println(str3);
9   System.out.println(arr1[2].length());
10  System.out.println(arr2[2].length());
```

Output:

```
ab
null
abcnull
3
NullPointerException at line 10
```

Explanation: Both String arrays are created in memory. However, arr2 contains null default value of String (objects). We can append a null value to another string however calling a string length() method on null will generate NullPointerException.

Array Length

Java provides a read only member named length that you can access to get the size of array. Length property is read only i.e. you cannot change its value, doing so will generate a compile time error.

Example 3.1: Using array length property

```
1   int[] arr1 = new int[10];
2   int[] arr2 = {1,4,6,7};
3
4   System.out.println(arr1.length);
5   System.out.println(arr2.length);
```

Output:

```
10
4
```

Example 3.2: Giving value to length property

```
1   int[] arr = {1,3,4};
2   arr.length = 10;
3   System.out.println(arr.length);
```

Output:

```
Compile Time Error at line 2
```

Explanation: We are trying to change value of length property which is read only. The compiler generates error stating length variable is final (constant). The length variable is initialized to a fixed value at the time of array initialization automatically.

Array Exceptions

We deal with majorly 2 exceptions related to array:

- NullPointerException

- ArrayIndexOutOfBoundsException

Example 4.1: null reference exception

Code:

```
1  int[] arr = null;
2  System.out.println(arr);
3  System.out.println(arr[0]);
```

Output:

```
null
Exception in line 3: NullPointerException
```

Explanation: arr holds null reference. This means it does not point to any valid object in memory. Directly, printing arr will give null. However, if we try to access ay index through null reference we'll get NullPointerException.

Example 4.2: index out of range

```
1  int[] arr = {7,9,1};
2  System.out.println(arr.length);
3  System.out.println(arr[5]);
```

Output:

```
3
ArrayIndexOutOfBoundsException at line 3
```

Explanation: Array indexes range from 0 to length – 1. If we try to access any index outside this range we'll get ArrayIndexOutOfBoundsException at runtime. Hence, any value less than 0 and greater than length – 1 will throw exception at runtime.

Example 4.3: Special Case - Array of size 0

```
1  int[] arr = new int[0];
2  System.out.println(arr.length);
3  System.out.println(arr[0]);
```

Output:

```
0
ArrayIndexOutOfBoundsException at line 3
```

Explanation: We can create an array of size 0 so the first line is perfectly correct. A size 0 array contains 0 elements. Hence when we try to access 0 index we get an exception at runtime. Accessing index 0 means our array contains at least 1 element.

Assigning array values

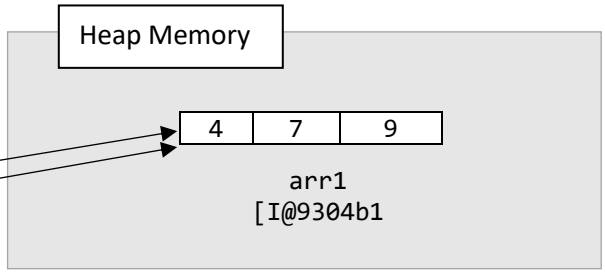
We can use assignment operator to copy a reference value from one array to another. After assignment both the array reference variables will point to same array object in memory. Using equality operator "==" on both variables will give true as answer.

Checking array equality: To check whether the two arrays are same or not we'll check their lengths and compare each element.

Remember: For objects '==' checks only reference value not the actual content of objects.

Example 5.1: Array assignment

Code:




```
1   int[] arr1 = {4, 7, 9};
2
3   int[] arr2 = arr1;
4   System.out.println(arr2.length);
5   System.out.println(arr1 == arr2);
```

Output:

```
3
true
```

Example 5.2: Array assignment 2

```
1   int[] a1 = new int[3];
2   int[] a2 = a1;
3   int[] a3 = a2;
4
5   a2[0] = 5;
6   a3[0] = 10;
7   a2[1] = 12
8   a3[2] = 4
9
10  for(int i = 0; i < a1.length; i++)
11  {
12      int n = a1[i];
13      System.out.println(n);
14  }
```

Output:

```
10
12
4
```

Explanation: All arrays (a1, a2 and a3) point to the same object in memory. Hence, changes made through any of them will be done on the same object.

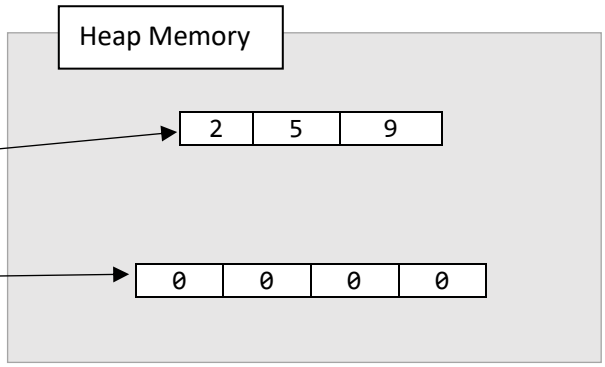
Changing Array Size

We cannot change the size of array after creation. However we can pass a new array to our reference variable, doing so will create a new array in memory and we won't be able to retrieve the data back.

Example 6: Changing array size

Code:

```
1  int[] arr = {2,5,9};
2
3  arr = new int[4];
4
5
6  for(int i = 0; i < arr.length; i++)
7  {
8     int n = arr[i];
9     System.out.println(n);
10 }
```



Output:

```
0
0
0
0
```

Explanation: In the above program we are giving new size to our array. However, this does not change the size of existing object. Instead it creates a new array. Further our reference variable now starts pointing to new object in memory. This means the old array contents are lost and cannot be retrieved.

Problem 1	**Arrays are fixed size data structures. Explain with example.**

Problem 2	**Given two arrays as input. Print "true" if both the arrays are same i.e. they contain same number of elements and all elements at corresponding locations are same. Note: Do not compare two array reference variables using '==' for equality.**	
Test Cases	**Input**	**Output**
	arr1 = {1,2,12,5} arr2 = {1,2,12,5} Note: both arrays have different reference values	true
	arr1 = {1,5} arr2 = {1,5,9}	false

Problem 3	How many objects will be created in each of the following segments? Explain with a diagram for each.

Code 3.1

```
1  int[] arr1 = {1,3,5};
2  int[] arr2 = new int[3];
```

Code 3.2

```
1  int[] a1, a2, a3;
2  a1 = new int[3];
```

Code 3.3

```
1  int[] a1, a2, a3;
2  a1 = new int[3];
3  a2 = a1;
4  a3 = a1;
```

Problem 4	Give output for each of the following code segments. Explain your answer using memory diagram.

Code 4.1

```
1  int[] a1 = new int[3];
2  int len = a1.length;
3  a1.length = len + 1;
4  System.out.println(a1.length);
```

Code 4.2

```
1  int[] b = {-1,7,8};
2  System.out.println(b[-1]);
```

Code 4.3

```
1  int[] a = new int[0];
2  a[0] = 10;
3  System.out.println(a[0]);
```

Code 4.4

```
1  int[] arr1 = new int[];
2  arr1[0] = 10;
3  System.out.println(arr1[0]);
```

Code 4.5

```
1  boolean[] b = new boolean[3];
2  for(int i = 0; i < b.length; i++)
3  {
4     if(i%2 == 0)
5     {
6           b[i] = true;
7     }
8     System.out.println(b[i]);
9  }
```

Code 4.6

```
1  String[] str1 = new String[2];
2  String[] str2 = {null, null};
3  System.out.println(str1.length);
4  System.out.println(str1[0].length());
5  System.out.println(str1 == str2);
```

Code 4.7

```
1  int[] arr1 = {3,5,6};
2  int[] arr2 = arr1;
3  arr2[1] = 10;
4  int[] arr1 = new int[2];
5  System.out.println(arr1[1]);
6  System.out.println(arr2[1]);
```

Code 4.8

```
1  int[] a1 = {2,3,4};
2  int[] a2 = {2,3,4};
3  System.out.println(a1==a2);
```