Java Arrays

In Java, **an array is a data structure that can store a fixed-size sequence of elements of the same data type. An array is an object**, which means it can be assigned to a variable, passed as a parameter to a method, and returned as a value from a method.

Arrays in Java are zero-indexed, which means that the first element in an array has an index of 0, the second element has an index of 1, and so on. The length of an array is fixed when it is created and cannot be changed later.

Java arrays can store elements of any data type, including primitive types such as int, double, and boolean, as well as object types such as String and Integer. Arrays can also be multidimensional, meaning that they can have multiple rows and columns.

Creating an array

To create an array in a single statement, you first declare the type of the array, followed by the name of the array, and then the values of the array enclosed in curly braces, separated by commas.

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

String[] names = {"John", "Mary", "David", "Sarah"};
```

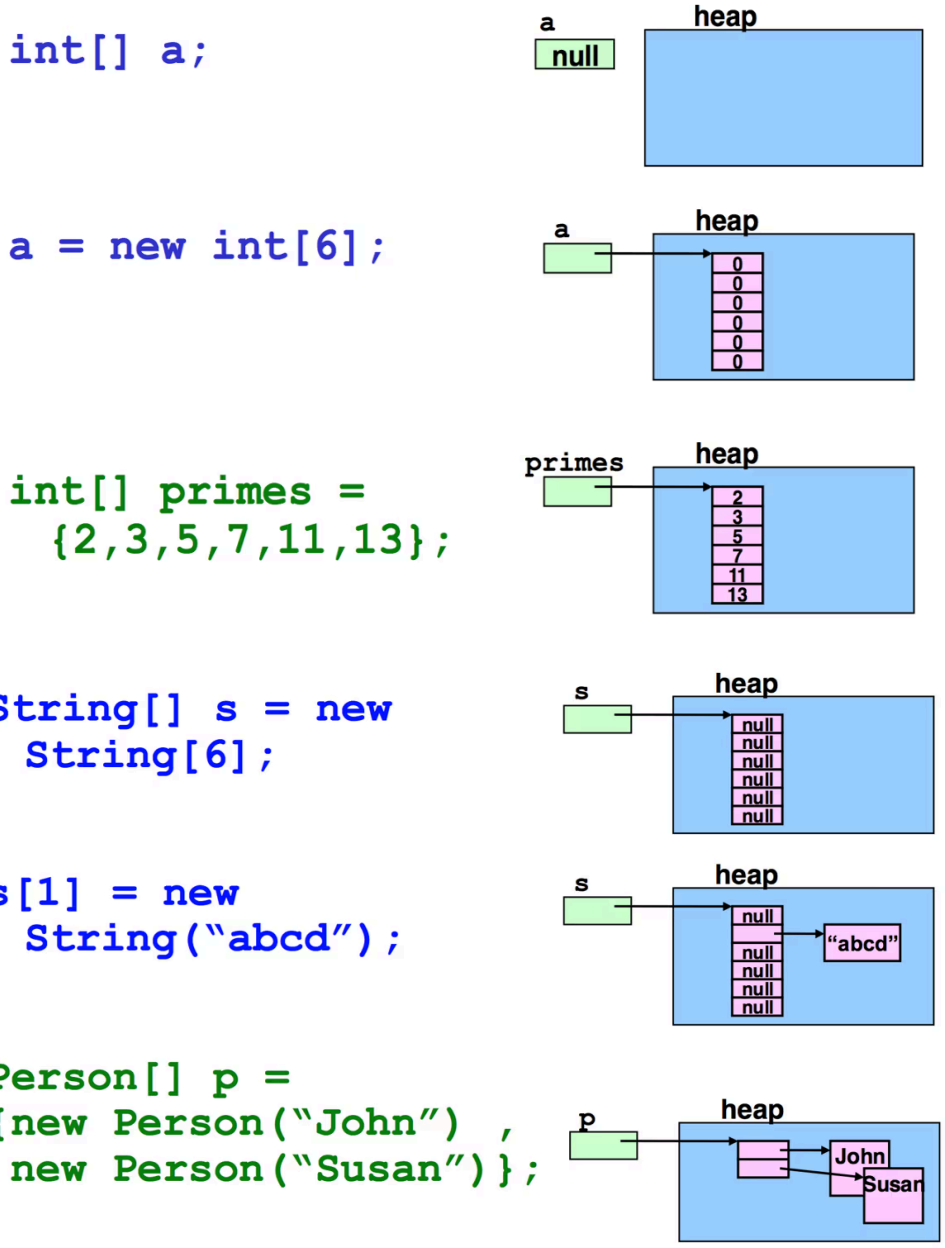
Array declaration and initialization in separate statements is another way to create an array in Java. This approach involves declaring an array variable and then initializing it with values in a separate statement.

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

String[] names;
names = {"John", "Mary", "David", "Sarah"};
```

Array declaration with default values is a way to create an array in Java and initialize it with default values of the specified data type. This approach is useful when you need to create an array with a fixed size, but you don't have specific values to initialize it with.

```java
int[] numbers = new int[5]; // all zeros

String[] names = new String[4]; // all null values
```

Arrays in memory



Multi-Dimensional arrays

A multidimensional array is an array that contains other arrays. In Java, you can create multidimensional arrays with two or more dimensions. A two-dimensional array is an array of arrays, while a three-dimensional array is an array of arrays of arrays, and so on.

```
int[][] matrix;
```

In this example, we declare a two-dimensional array of integers named `matrix`, but we haven't initialized it with any values yet.

To initialize the array with values, we use the `new` keyword followed by the type of data the array will hold, enclosed in square brackets, and the number of rows and columns the array will contain.

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

In this example, we initialize the `matrix` array with 3 rows and 4 columns.

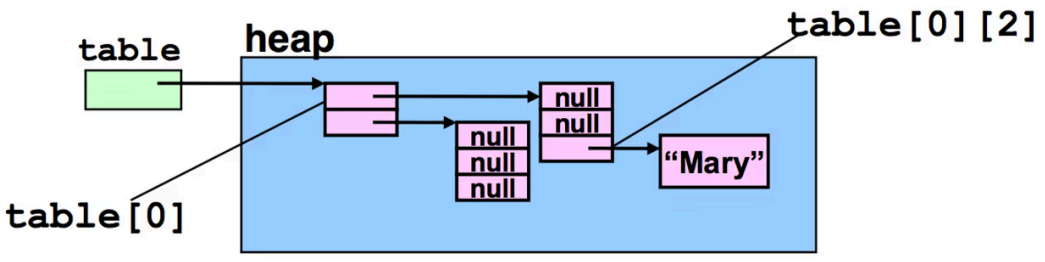
Here's another example of declaring a two-dimensional array of integers filled with initial values:

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

Alternatively, we declare a two-dimensional array of String named `table` with 2 rows and 3 columns. Since we haven't specified any values, each element in the array is initialized with the default value of `null`. Afterward, the last element of the first row is initialized with a reference to a String object containing "Mary".

```
String[][] table = new String[2][3];
table[0][2] = "Mary";
```

Note well: in Java, rows in multidimensional arrays are separate objects and are not contiguous in memory. As a consequence, switch rows in an extremely fast operation which does not require copies in memory.



Assigning and accessing elements

An element of an Array is referred to by its index. In the example below we create an Array to hold 3 integers, and then assign values to indices 0 and 2. After that we print the values.

```
int[] numbers = new int[3];
numbers[0] = 2;
numbers[2] = 5;

System.out.println(numbers[0]);
System.out.println(numbers[2]);
```

The index is an integer, and its value is between [0, length of the Array - 1]. For example an Array to hold 5 elements has indices 0, 1, 2, 3, 4.

```
Scanner reader = new Scanner(System.in);

int[] numbers = new int[5];
numbers[0] = 42;
numbers[1] = 13;
numbers[2] = 12;
numbers[3] = 7;
numbers[4] = 1;

System.out.println("Which index should we access? ");
int index = Integer.valueOf(reader.nextLine());

int number = numbers[index];
System.out.println(number);
```

Iterating over an array

You can find the size of the array through the associated attribute `length`. *Note well, that `length` is an attribute not a method call, so `numbers.length()` doesn't work.*

You can iterate over the array in various ways:

```
int[] numbers = new int[4];


int index = 0;
while (index < numbers.length) {
    System.out.println(numbers[index]);
    index = index + 1;
}
```

or

```
for (index = 0; i < numbers.length; index++) {
    System.out.println(numbers[index]);
}
```

or

```
for (int number : numbers) {
  System.out.println(number);
}
```

If the index is pointing outside the Array, i.e. the element doesn't exist, we get an **ArrayIndexOutOfBoundsException**. This error tells, that the Array doesn't contain the given index.

The next example is a program that initially asks the user to enter how many numbers, and then enter the numbers. Finally, it prints back the numbers in the same order. The numbers are stored in an Array.

```
Scanner reader = new Scanner(System.in);
System.out.print("How many numbers? ");
int howMany = reader.nextInt();

int[] numbers = new int[howMany];

System.out.println("Enter the numbers:");
for (int i = 0; i < numbers.length; i++) {
   numbers[i] = reader.nextInt();
}

System.out.println("Here are the numbers again:");
for (int i = 0; i < numbers.length; i++) {
   System.out.println(numbers[i]);
}
```

Passing arrays to methods

You can use arrays as a parameter of a method just like any other variable. When you use array as a parameter of a method, the method receives a copy of the reference to the array.

```
public static void listElements(int[] integerArray) {
    for (int n : integerArray) {
        System.out.println(n);
    }
}
```

```
int[] numbers = {1, 2, 3}
listElements(numbers);

// or

listElements(new int[]{1, 2, 3});
```

As noticed earlier, you can freely choose the name of the parameter inside the method, the name doesn't have to be the same as the name of the variable when you call the method. In the example above, we call the array `integerArray`, meanwhile the caller of the method has named the same array `numbers`.

Array is an object, so when you change the array inside the method, the changes persist after the execution of the method.

```
public static void fill(int[] array, int value) {
    for (int i = 0; i < array.lenght; i++) {
        array[i] = value;
    }
}
```

Array Utilities

java.util.Arrays

`java.util.Arrays` contains various methods for manipulating arrays such as **sorting, searching, filling, printing or being viewed as Collections**

- copyOf() / copyOfRange
- fill()
- equals() / deepEquals()
- hashCode() / deepHashCode()
- sort()
- binarySearch()
- toString() / deepToString()
- asList()
- setAll()

System.arraycopy()

`System.arraycopy()` copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. The number of components copied is equal to the length argument. Its use is advised because of its speed and simplicity!

```
int[] v1 = new int[8];
int[] v2 = new int[8];

Arrays.fill(v1, 10);
Arrays.fill(v2, 12);

// manual array copy
for (int i = 0; i < 4; i++) {
    v2[i+4] = v1[i];
}

// system array copy
// produces the same result as the manual version
// System.arraycopy(v1, 0, v2, 4, 4);

System.out.println(Arrays.toString(v1));
// 10,10,10,10,10,10,10,10

System.out.println(Arrays.toString(v2));
// 12,12,12,12,10,10,10,10
```

Resources

- https://www.baeldung.com/java-arrays-guide
- https://www.baeldung.com/java-util-arrays