Memory Allocation of Primitive, Non-primitive Data Types

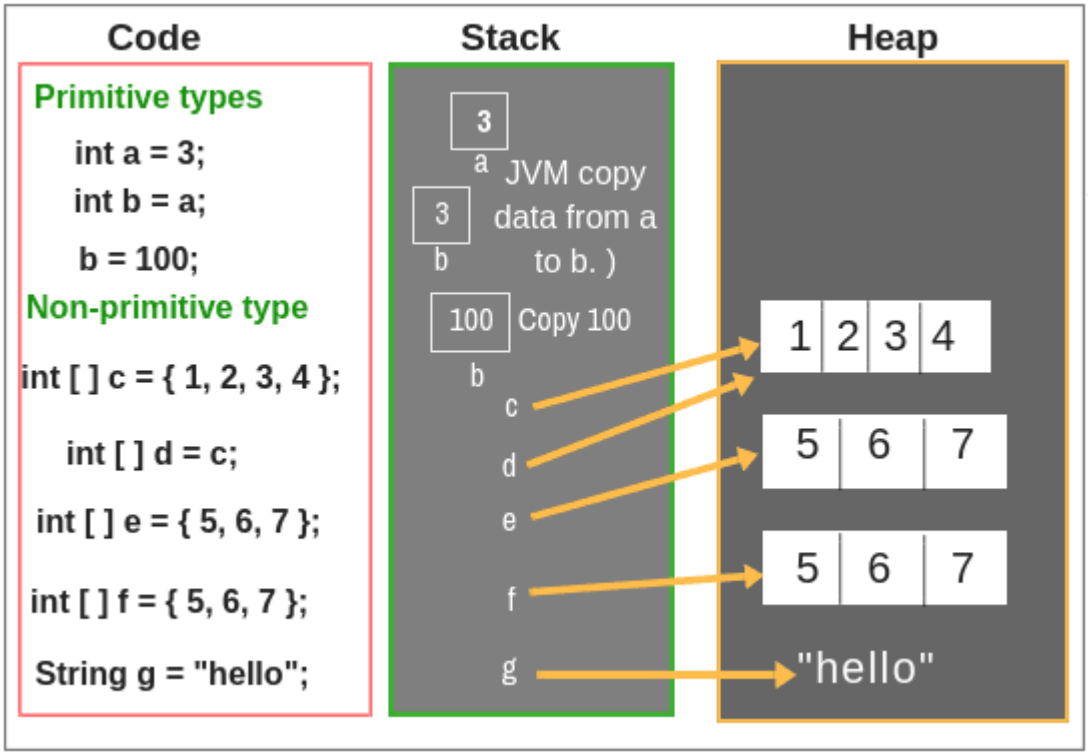
-Siva Yannam

➲ In Java, all data type for primitive type variables is stored on the stack.

➲ For reference data types, the stack holds a pointer to the object on the heap. When setting a reference type variable equal to another reference type variable, a copy of only the pointer is made.

➲ Certain object types cannot be manipulated on the heap.

Let's understand all these points by taking one simple example as shown in the below figure.



To understand all the above points, we have taken three blocks. The first block represents code, the second block represents the data structure stack memory and the third block is heap memory.

1. In the first line of code, we declared a primitive type variable int a = 3;. Since 'a' is a primitive type variable whose data type is int. So, all data will be allocated on the stack memory as shown in the second block of the above figure.

2. In the second line of code, we declared a variable type int b = a;. Since 'b' is a primitive type variable and 'a' is also a primitive type. Therefore, JVM just copies the data from a to b.

3. In the third line, we just modified the value of variable b equal to 100. In this case, JVM will store the entry 100 on the stack in the place of

4. Let's take some reference type variable i.e. Non-primitive type. In line 4, 'c' is a reference type variable that is declared as an integer array.

An integer array is an object. So, JVM will set c on the stack and point the pointer to an object on the heap.

This object is an array of size four with values 1, 2, 3, 4 in the index as shown in the above figure. The index starts from 0, 1, 2, and 3. Only data stored in c on the stack is a pointer.

5. In the fifth line, a variable d is also declared as integer array type and set equal to c. So in this case, JVM will copy the address of c's data to d in the stack memory. So, c and d both will point to the same object on the heap.

6. In the six-line of code, an integer array e is declared which is pointing a new object on the heap.

7. Again, an integer array f is created and pointing to the new object on the heap. In the above figure, you will notice that the values are the same but both e and f are pointing to the different objects on the heap. In this case, the address of e's data will not be copied into f in stack memory.

8. In the last line of code, a variable g is declared as a string data type which is the Non-primitive data type and it is pointing to the “hello” on the heap.

Thus, you have seen that all primitive data types are stored on the stack, and in the case of reference type, stack holds a pointer to the object on the heap. Here, we have described only basic ideas related to the concept of memory allocation of data types.

As you will move further and study core java, you will learn the memory concept in detail and concept will be cleared easily. If did not understand this tutorial, you can skip it and move for the further tutorial.

Hope that this tutorial has covered the basic points related to memory allocation of primitive and non-primitive data type in java.