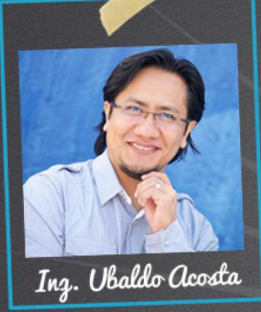



JAVA FUNDAMENTALS COURSE

**PASS BY VALUE AND
PASS BY REFERENCE IN JAVA**

Por el experto: Ing. Ubaldo Acosta



Ing. Ubaldo Acosta



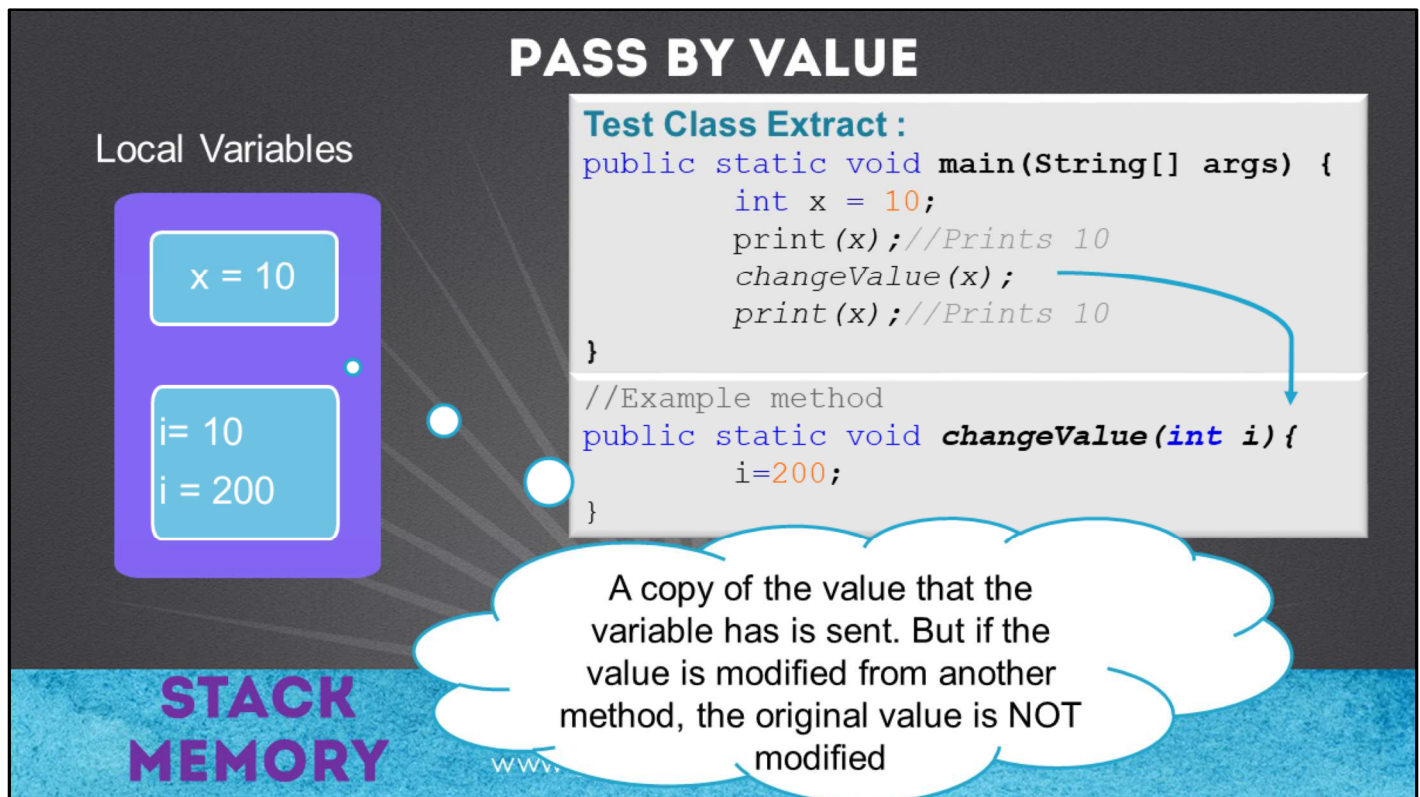
Experiencia y Conocimiento para tu vida

JAVA FUNDAMENTALS COURSE
www.globalmentoring.com.mx

Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson.

We are going to study the subject of Pass by value and reference in Java.

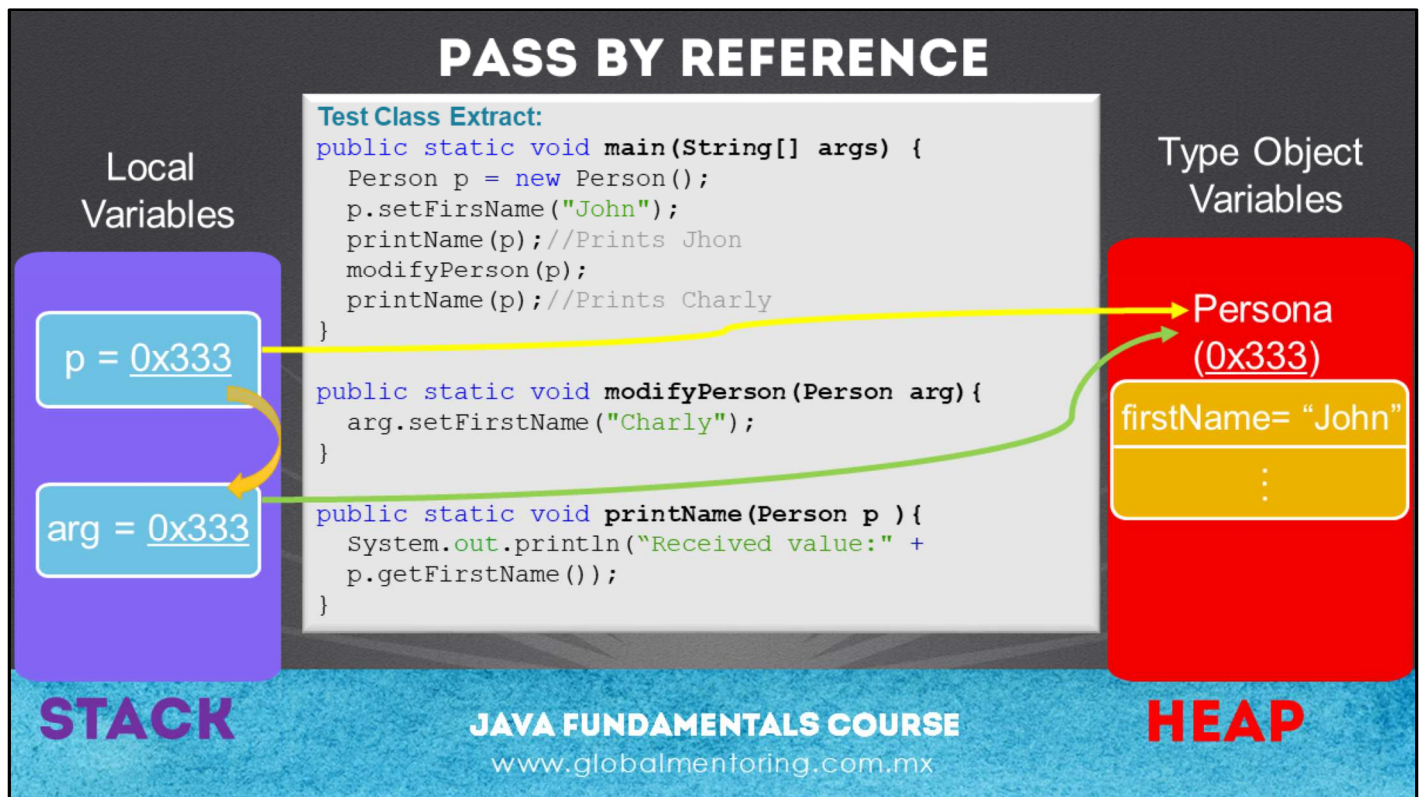
Are you ready? Come on!



So far we have used primitive types when sending values to our methods, this is known as pass by value, because as we saw in previous lessons, we are only copying the value of the primitive type, but each variable contains its own copy.

For example, in the code shown we can see an example of pass by value, also known as values of primitive type. What we see in the code is that when creating a variable, in this case x, and assigning it a value, for example 10, this variable, when modified in another method, its original value is not modified, since it only passes a copy of its original value.

The way to change the value of the variable x would be to change it within the same method. However, in this example the method called changeValue tries to make a change to the variable x, but since it is beyond the scope of this method to access the x variable, you can only change the value of its local variable called i, but the original variable 'x' is not modified.



Besides the pass by value concept, in Java we have the concept of pass by reference, basically we will use objects as parameters instead of primitive types. When using an object as a parameter and sending it as an argument to a method, what we are doing is sending the reference of the object that we want to use, and instead of using a copy of the value of the object, what we are really doing is pointing to the same object, with the purpose of modifying it directly, without having to make a copy.

This is because imagine an object that contains 50 or more attributes, and if a copy of the object is made, what we would actually be doing is generating a copy of these 50 values. Therefore, when we work with objects, and send them as arguments to a method, **we are only sending the value of the reference** where this object is located, so that we can directly modify the object, either from the original variable that points to the object, or from the local variable that works as an argument in the called method.

In the code we can observe the variable `p` of type `Person`, this class we will create it in the section of exercises, but basically it has an attribute called `firstName` of type `String`, and what we can observe is that the value of the attribute name is modified, this is due to the local variable called `arg` that points to the same object created in the Heap memory of `Person` type, and therefore when modifying the attribute of the same object, then the object originally created is modified, and we can access this change even from other methods .

This leads to discussions, since it is said that Java only passes parameters by value, and not by reference, and in some way this is true, since as we can see in the stack memory, the value of `p` is "copied" to the variable `arg` when calling the method `modifyPerson`, but because what receives `arg` is the memory address of the object `Person` originally created, then the variable `arg` can access the attributes and methods of the variable `p`, and in this way is possible modify the state or content of the original object. And after finishing the method `modifyPerson` and returning to the main method it is possible to access these changes, and instead of printing `John` which was the original name, the value of `Charly` is sent to the console, which is the value modified by the variable `arg`.

In this way we can see how in Java when we work with objects, what we are really providing is the value of the memory reference, in this case the value of `0x333`, and with this reference we access the object directly, and so we can modify it. Finally, we can access these modifications even outside the method where the object was originally created, or from the method where the object was created.

ONLINE COURSE

JAVA FUNDAMENTALS

Author: Ubaldo Acosta



JAVA FUNDAMENTALS COURSE

www.globalmentoring.com.mx

