**Java pass By Value – 2024**

Java is **strictly** pass-by-value, exactly as in C. The problem we’re facing here is statements like

In Java, Objects are passed by reference, and primitives are passed by value. This is half incorrect.

1. For Primitive types, parameters are pass-by-value
2. For Object types, the object reference is pass-by-value

**Passing Primitive Types**

**Primitive variables are directly stored in stack memory. Whenever any variable of primitive data type is passed as an argument, the actual parameters are copied to formal arguments and these formal arguments accumulate their own space in stack memory.**

Passing Object References

**Whenever an object is passed as an argument, an exact copy of the reference variable is created which points to the same location of the object in heap memory as the original reference variable.**

**As a result of this, whenever we make any change in the same object in the method, that change is reflected in the original object.** However, if we allocate a new object to the passed reference variable, then it won’t be reflected in the original object.

**public class** Test1 {  
 **public static void** swap(Object o1, Object o2) {  
 Object temp = o1;  
 o1 = o2;  
 o2 = temp;  
 }  
  
 **private static void** changeValue(Ball ball) {  
 ball.setColor("Red");  
 ball = **new** Ball("Green");  
 ball.setColor("Blue");  
 }  
  
 **public static void** main(String[] args) {  
 Ball red = **new** Ball("Red");  
 Ball blue = **new** Ball("Blue");  
 *swap*(red, blue);  
 System.***out***.println("After the swap method executes:");  
 System.***out***.println("`red` color value = " + red.getColor()); // Red  
 System.***out***.println("`blue` color value = " + blue.getColor()); // Blue  
 *changeValue*(blue);  
 System.***out***.println("After the changeValue method executes:");  
 System.***out***.println("`blue` color value = " + blue.getColor()); // Red  
 }  
}

**public class** Ball {  
 **private** String color;  
  
 **public** Ball(String color) {  
 **this**.color = color;  
 }

get()/set() methods  
}

The output shows that the swap() method didn’t swap the color values of the original objects. This helps to show that Java is pass by value, since the swap() method only acts upon copies of the original object reference values.

In case changeValue() method, **whenever we make any change in the same object in the method, that change is reflected in the original object.** However, if we allocate a new object to the passed reference variable, then it won’t be reflected in the original object.

Since the variables are just the reference to the objects, we get confused that we are passing the reference so Java is passed by reference. However, we are passing a copy of the reference and hence it’s pass by value.

In short, you can not modify value of any parameter passed, but you can call methods or change attributes of an object reference passed.

Data is shared between functions by passing parameters. Now, there are 2 ways of passing parameters:

* **passed by reference :** caller and callee use same variable for parameter.
* **passed by value :** caller and callee have two independent variables with same value.

References

<https://ananya281294.medium.com/java-passing-by-value-or-passing-by-reference-c75e312069ed>

<https://www.digitalocean.com/community/tutorials/java-is-pass-by-value-and-not-pass-by-reference>

<https://www.javadude.com/articles/passbyvalue.htm>