**Java is Pass by Value and Not Pass by Reference**

JULY 27, 2016 BY [PANKAJ](http://www.journaldev.com/author/pankaj) [35 COMMENTS](http://www.journaldev.com/3884/java-is-pass-by-value-and-not-pass-by-reference#comments)

One of the biggest confusion in Java programming language is whether java is **Pass by Value** or **Pass by Reference**. I ask this question a lot in interviews and still see interviewee confused with it. So I thought to write a post about it to clarify all the confusions around it.

First of all we should understand what is meant by pass by value or pass by reference.

* **Pass by Value**: The method parameter values are copied to another variable and then the copied object is passed, that’s why it’s called pass by value.
* **Pass by Reference**: An alias or reference to the actual parameter is passed to the method, that’s why it’s called pass by reference.

Java is always Pass by Value and not pass by reference, we can prove it with a simple example.

Let’s say we have a class Balloon like below.

package com.journaldev.test;

public class Balloon {

private String color;

public Balloon(){}

public Balloon(String c){

this.color=c;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

}

And we have a simple program with a generic method to swap two objects, the class looks like below.

package com.journaldev.test;

public class Test {

public static void main(String[] args) {

Balloon red = new Balloon("Red"); //memory reference 50

Balloon blue = new Balloon("Blue"); //memory reference 100

swap(red, blue);

System.out.println("red color="+red.getColor());

System.out.println("blue color="+blue.getColor());

foo(blue);

System.out.println("blue color="+blue.getColor());

}

private static void foo(Balloon balloon) { //baloon=100

balloon.setColor("Red"); //baloon=100

balloon = new Balloon("Green"); //baloon=200

balloon.setColor("Blue"); //baloon = 200

}

//Generic swap method

public static void swap(Object o1, Object o2){

Object temp = o1;

o1=o2;

o2=temp;

}

}

When we execute above program, we get following output.

red color=Red

blue color=Blue

blue color=Red

If you look at the first two lines of the output, it’s clear that swap method didn’t worked. This is because Java is pass by value, this swap() method test can be used with any programming language to check whether it’s pass by value or pass by reference.

Let’s analyze the program execution step by step.

Balloon red = new Balloon("Red");

Balloon blue = new Balloon("Blue");

When we use **new** operator to create an instance of a class, the instance is created and the variable contains the reference location of the memory where object is saved. For our example, let’s assume that “red” is pointing to 50 and “blue” is pointing to 100 and these are the memory location of both Balloon objects.

Now when we are calling swap() method, two new variables o1 and o2 are created pointing to 50 and 100 respectively.

So below code snippet explains what happened in the swap() method execution.

public static void swap(Object o1, Object o2){ //o1=50, o2=100

Object temp = o1; //temp=50, o1=50, o2=100

o1=o2; //temp=50, o1=100, o2=100

o2=temp; //temp=50, o1=100, o2=50

} //method terminated

Notice that we are changing values of o1 and o2 but they are copies of “red” and “blue” reference locations, so actually there is no change in the values of “red” and “blue” and hence the output.

If you have understood this far, you can easily understand the cause of confusion. Since the variables are just the reference to the objects, we get confused that we are passing the reference so java is pass by reference. However we are passing a copy of the reference and hence it’s pass by value. I hope it clear all the doubts now.

Now let’s analyze foo() method execution.

private static void foo(Balloon balloon) { //baloon=100

balloon.setColor("Red"); //baloon=100

balloon = new Balloon("Green"); //baloon=200

balloon.setColor("Blue"); //baloon = 200

}

The first line is the important one, when we call a method the method is called on the Object at the reference location. At this point, balloon is pointing to 100 and hence it’s color is changed to Red.

In the next line, ballon reference is changed to 200 and any further methods executed are happening on the object at memory location 200 and not having any effect on the object at memory location 100. This explains the third line of our program output printing **blue color=Red**.