I always thought Java was **pass-by-reference**.
However, I've seen a couple of blog posts (for example, this blog) that claim that it isn't.
I don't think I understand the distinction they're making.

What is the explanation?

- 564
I believe that much of the confusion on this issue has to do with the fact that different people have different definitions of the term "reference". People coming from a C++ background assume that "reference" must mean what it meant in C++, people from a C background assume "reference" must be the same as "pointer" in their language, and so on. Whether it's correct to say that Java passes by reference really depends on what's meant by "reference". – Gravity Jul 30 '11 at 7:23
- 96
I try to consistently use the terminology found at the Evaluation Strategy article. It should be noted that, even though the article points out the terms vary greatly by community, it stresses that the semantics for `call-by-value` and `call-by-reference` differ in a very crucial way. (Personally I prefer to use `call-by-object-sharing` these days over `call-by-value[-of-the-reference]`, as this describes the semantics at a high-level and does not create a conflict with `call-by-value`, which is the underlying implementation.) – user166390 Dec 15 '11 at 6:12
- 47
@Gravity: Can you go and put your comment on a HUGE billboard or something? That's the whole issue in a nutshell. And it shows that this whole thing is semantics. If we don't agree on the base definition of a reference, then we won't agree on the answer to this question :) – MadConan Nov 12 '13 at 20:58
- 22
I think the confusion is "pass by reference" versus "reference semantics". Java is pass-by-value with reference semantics. – spraff Mar 27 '14 at 13:54
- 8
@Gravity, while you're absolutely correct in that folks coming from C++ will instinctively have a different intuition set regarding the term "reference", I personally believe the body is more buried in the "by". "Pass by" is confusing in that it is absolutely distinct from "Passing a" in Java. In C++, however it colloquially is not. In C++ you can say "passing a reference", and it's **understood that it will pass the `swap(x,y)` test**. – user4229245 Mar 23 '15 at 13:44

Java is always **pass-by-value**. Unfortunately, they decided to call the location of an object a "reference". When we pass the value of an object, we are passing the reference to it. This is confusing to beginners.
It goes like this:

```
public static void main(String[] args) {
  Dog aDog = new Dog("Max");
  // we pass the object to foo
  foo(aDog);
  // aDog variable is still pointing to the "Max" dog when foo(...) returns
```

```
  aDog.getName().equals("Max"); // true
  aDog.getName().equals("Fifi"); // false
}

public static void foo(Dog d) {
  d.getName().equals("Max"); // true
  // change d inside of foo() to point to a new Dog instance "Fifi"
  d = new Dog("Fifi");
  d.getName().equals("Fifi"); // true
}
```

In the example above `aDog.getName()` will still return `"Max"`. The value `aDog` within `main` is not changed in the function `foo` with the `Dog` `"Fifi"` as the object reference is passed by value. If it were passed by reference, then the `aDog.getName()` in `main` would return `"Fifi"` after the call to `foo`.
Likewise:

```
public static void main(String[] args) {
  Dog aDog = new Dog("Max");
  foo(aDog);
  // when foo(...) returns, the name of the dog has been changed to "Fifi"
  aDog.getName().equals("Fifi"); // true
}

public static void foo(Dog d) {
  d.getName().equals("Max"); // true
  // this changes the name of d to be "Fifi"
  d.setName("Fifi");
}
```

In the above example, `Fifi` is the dog's name after call to `foo(aDog)` because the object's name was set inside of `foo(...)`. Any operations that `foo` performs on `d` are such that, for all practical purposes, they are performed on `aDog` itself (except when `d` is changed to point to a different `Dog`instance like `d = new Dog("Boxer")`).

- 205
  Isn't it slightly confusing the issue with internal details? There's no conceptual difference between 'passing a reference' and 'passing the value of a reference', assuming that you mean 'the value of the internal pointer to the object'. – izb Sep 8
- 326
  But there is a subtle difference. Look at the first example. If it was purely pass by reference, aDog.name would be "Fifi". It isn't - the reference you are getting is a value reference that if overwritten will be restored when exiting the function. – erlando Sep 11 '08 at 6:55
- 259
  @Lorenzo: No, in Java everything is passed by value. Primitives are passed by value, and object references are passed by value. The objects themselves are never passed to a method, but the objects are always in the heap and only a reference to the object is passed to the method. – Esko Luontola Feb 12 '09 at 23:02
- 250
  My attempt at a good way to visualize object passing: Imagine a balloon. Calling a fxn is like tieing a second string to the balloon and handing the line to the fxn. parameter = new Balloon() will cut that string and create a new balloon (but this has no effect on the original balloon). parameter.pop() will still pop it though because it follows the string to the same, original balloon. Java is pass by value, but the value passed is not deep, it is at the highest level, i.e. a primitive or a

pointer. Don't confuse that with a deep pass-by-value where the object is entirely cloned and passed. – dhackner Oct 20 '10 at 16:38

- 119
  What's confusing is that object references are actually pointers. In the beginning SUN called them pointers. Then marketing informed that "pointer" was a bad word. But you still see the "correct" nomenclature in NullPointerException. – Prof. Falken Feb 9 '11 at 9:46