

Re: pass by reference

Source: <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2008-04/msg00657.html>

- *From:* Andreas Leitgeb <avl@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* 07 Apr 2008 13:57:52 GMT
-

PreScriptum: By now, I've read most of the later posts, and my focus has moved from insisting to use "by ref" in java-context to understanding the technical difference in detail.

Chris Smith <cdsmith@xxxxxxx> wrote:

... Pass by reference is a concept that has a particular meaning, independent of any other use of the word "reference" in a language.
[...]

Then either the word "reference" also needs a particular meaning, or the "pass by ref" is a cyclic definition, based on a meaning of "reference" that is derived from that concept...

Most seem to agree to that whatever a reference actually is, it must be one that references the **variable** (or lvalue) that was passed to it. Also the non-null-ness of a reference seems to be quite agreed on here.

I want to understand the crucial difference between C++'s way and Java's way in those special situations, where their technical effect coincides. There are of course other situations where the difference is obvious, but I want to understand those where it isn't.

If in C++ I've got a reference-parameter in a function f1, and pass this reference(which refers to a real variable one call-level upstairs) to another function f2 (again, by ref), then obviously f2 may modify the actual variable two levels up (which was passed to f1), and **not** modify the **reference** that lives in f1 and still refers to the actual object "var" in the upper level, afterwards.

If there's a thinko in this paragraph, please point it out.

If it's just unclear, see the following C++ code snippet.

```
struct mystruct { int i; } var;
void f2(mystruct& x2) { x2=*(new mystruct()); }
void f1(mystruct& x1) { f2(x1); } // <---- x1 is not changed, only var is.
int main() { f1(var); }
```

Re: pass by reference

- * Is the call to f2 from within f1 really "by reference" ?
- * Is "x1" (as argument to f2) actually an lvalue ?

If "x1" is just "seen" as an lvalue identical to "var", then why cannot a java ref be "seen" as "identical" to the object it points to? Just because it can be null?

```
f1(*(mystruct*)0); // is compileable! the inevitable runtime SIGSEGV
// only happens inside f2 at the assignment,
// f1 has no problem with a null-reference.
```

Just because Java-refs can be re-targetted by assignment?
Java's "." operator corresponds to C's "->" or: jRef.xyz == (*cPtr).xyz
If Java had an operator to tell an Object (addressed through the java-ref) to assume another object's value (as happens when assigning to a C++ reference), would it do "pass by ref", then? IOW: does the difference lie in the semantics of assignment?

i.e., a reference in C++ is simply another symbol, with a possibly different scope, that denotes the same object as the symbol it is initialized from.

```
int x[10]; for (int i=0; i<10; i++) { int &j=x[i]; j=42; }
Ref j doesn't need to be initialized from a symbol.
```

As I've tried to show, most of the artefacts that are used to disprove java's passing by reference, can be mimicked in C++ as well. The resolution of this contradiction can be either, that "passing by ref" depends on more than just a method's signature, or, that Java at least approaches some effects of pass-by-ref quite closely.

.