|  |
| --- |
| Circle Language Spec: Objects |

## Target Objects

An object reference can point to another object reference, which points to another object reference and so on. The first object found in this redirection, that does not refer to another object again, is called the *target object*. Even though any of the object *references* can be used like it is the object itself, the *target object* is considered the real object and not just a reference to it.

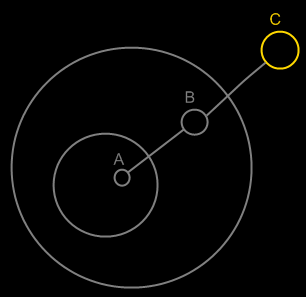
The term target object is also used to denote the direct reference target, not necessarily the final target. What kind of target is denoted, will be clear from the context.

### Compared to C++

In C++ you had to specify in advance the number of pointer redirections of a variable. In the new computer language a symbol can follow any amount of indirections, from zero to infinity. You don’t specify the amount of redirections in advance. You can just add a redirection by turning the target object into a pointer.

### In a Diagram

The target object is the last point in a string of object reference redirections.



Symbol A is an object reference to symbol B. Symbol B is an object reference to symbol C. Symbol C is the target object of both symbols A and B.

The idea of target objects is also a way to make a single symbol

in the diagram represent the actual object, whereas the others are just seen as references to the object: to have the actual object only represented by a single symbol in the diagram.

## Ideas

Objects,

Target,

2008-07-26

I need to rename the term Target Object, Target Class and Target Interface to Final Object Target, Final Class Target and Final Interface Target, because I’m not targeting an object, class or interface, but I’m targeting an object reference representing an object, class or interface.

Also the term object target is the same as direct object target. That also counts for classes and interfaces.

The term Target Object, Target Class and Target Interface have less of a use now. But the way they are used now is misleading.

JJ

### Out of the original Symbol documentation

#### Object Trace

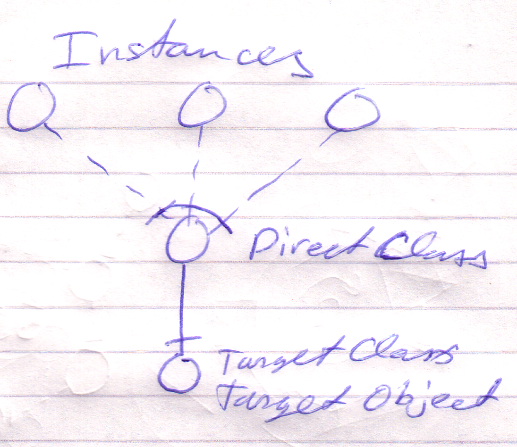
< 2008-10-06 Probably not right anymore. >

To find the target object, you’d expect to only follow object lines. However, there’s a pitfall: a situation that does not occur a lot, though.

If a type line points to a symbol with an object line, the type is a single object.

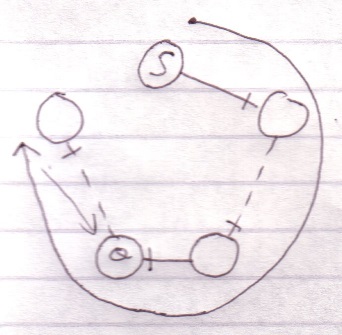


Each instance of the type is actually the same object.



Therefore, a type line can redirect the object of the symbol. Therefore, type lines need to be followed to find the object.

The last symbol pointed to by an object line is the object.



This kind of redirectioning is called an *object trace*.

Delegating the object aspect is the main type of object redirection.

# From the old Symbol documentation

In C++ bepaal je de redirection diepte vooraf:

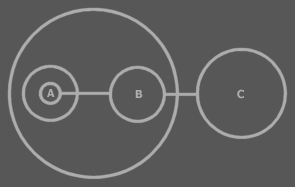
Int \*\*\*TripleRedirected

In Symbol kan je de redirection diepte achteraf bepalen

Als je in C++ een object referenties toewijst aan een object referentie, dan wijs je niet naar de object referentie, maar naar het target object. Symbol heeft meer structurering hier.

#### Multiple Redirection and Final Targets

If an object symbol has an object line to a symbol that again has an object line, there is redirected until a symbol without an object line is encountered: the *target object*.



C is the target object of A and B.

The target object symbol is regarded to represent the object for real. The other symbols are references to the object.

The same way there are symbols serving as a *target type* or a *target interface*. Also a procedure has an interface target. A procedure also has a call target and reference target. In both those cases reference lines are followed.