|  |
| --- |
| Circle Language Spec: Relations |

## Class Referrers

### Concept

The *Referrers* article explained how an object can be made aware of its referrers. A *class* can also be made aware of the objects using it as a class.

Classes are implemented as a concept. That concept adds an object reference to the system interface. This object reference points which other object is its class. So oddly, an object reference, that points out the class, is already added to the class’s list of referrers. The classes are registered inside the same list of referrers as object referrers. This is actually just fine. The Referrers list is supposed to be a low-level view on the refererrers.

A class is usually only *used* as a class, and not also used as an object, so in practice, the Referrers list of a class, actually already *is* a list of class referrers. So a separate list of Class Referrers will not be implemented.

But if in the future there is a need to also maintain a separate list of class referrers, a separate Class Referrers concept could be implemented. In that case, when a related item’s *class* is set, the Related Item . Class . Set will update the target’s list of Class Referrers.

#### Not registering class referrers

The amount of referrers of a Number *object* may be small, but the amount of referrers of the Number *class* is humungous. The class will even have a Referrers list, when the class is not a created object, because Referrers applies to both symbols and objects.

You would want to turn the Referrers concept *off* for the Number class and *on* for Number objects. But the problem here is, that a class is a blueprint for an object. An object only supports Referrers, because the *class* supports it.

The first solution proposed was to simply not support the Referrers concept for classes that are widely used. But then for widely used classes, the Referrers concept never be supported. That is against the idea of supporting the Referrers concept by default.

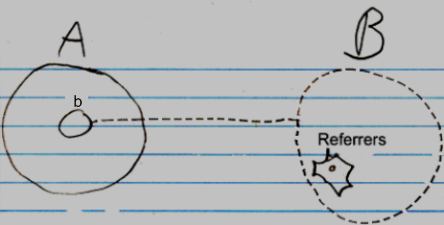
If you can not stop a class from supporting Referrers without stopping objects from supporting Referrers at the same time, then the Referrers concept will not be widely used anymore.

Therefore, you are going to have to specify for a symbol or object, that it is a non-practitioner of a concept. Derivation of objects will take over the specified concept, but not the non-practitioner aspect. Or perhaps instead of calling it non-practitioner, you could call it Objects Support Concept Referrers, or something.

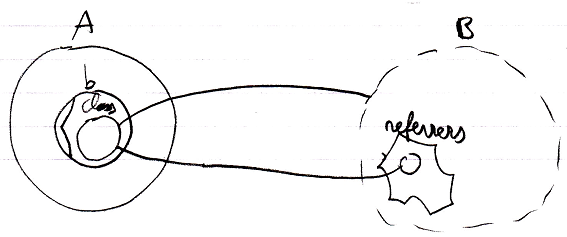
### Diagram Notation

< The notation of a reference to an object reference’s class needs to be determined in the future. >

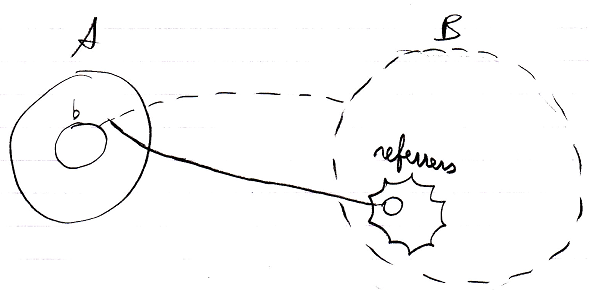
Because the concept of referrers simply also functions as the concept of class referrers, it can be displayed in a diagram the same way, except, that classes and class references are displayed with dashed lines.



The reference line of the item in the Referrers list is displayed, then it has to point to the class redirection of symbol b. There is no final notation yet for a to something else’s class. But a preliminary notation could either be a reference to the Class inside b’s system interface:



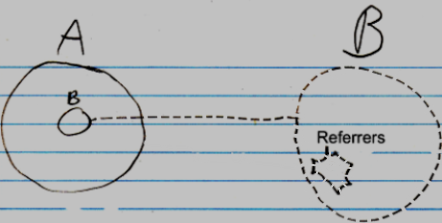
Or a reference line connnected to b’s class line:



The referrers are pointed at by solid lines, because they are just references to the objects, that use it as a class. No implicit notation of making the referrer lines *dashed* will be used here, because that will introduce too much ambiguity in the diagram notaiton.

As mentioned in the article *Referrers*, it is not clear yet under which circumstances the whole referrers list might be completely left out of the diagram.

If a class defines that its objects support Referrers, but the class itself won’t register its Referrers, then the Referrers list of the class will be drawn out with dashed lines.



Obviously, the inactive referrers list will not contain any object references.