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

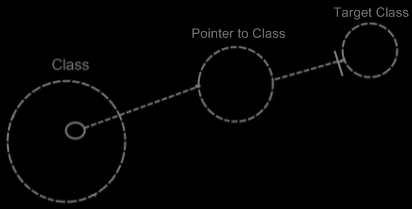
## Relation to a Pointer

### Concept

As covered by the article *Related Classes*, you can also establish a unary relation with a *pointer* to another class. This is not so common, but it is possible all the same. This is mostly applied, to allow a class to make a sub-object’s class *adjustable*. It is important to consider, that everything inside a pointer is really part of the *target class*, but a pointer itself is usable individually, independent from the target class. This is well visualized in the article *Relation to a Pointer in a Diagram.* To make a relation to a pointer dual, you have to give the target class a relation back to the first class. The first class relates to the pointer, but the target class relates back to the first class. This automatically gives the pointer a relation back to the first class. This creates a dual relation between the first class and the pointer to a class, but only a unary backwards relation between the target class and the first class. This is because the first class does not directly refer to the target class, but the target class does directly refer back to it. You should see it in a diagram. That will make it much clearer.

### Diagram Notation

You can also establish a unary relation with a *pointer* to another class. This is not so common, but it is possible all the same.

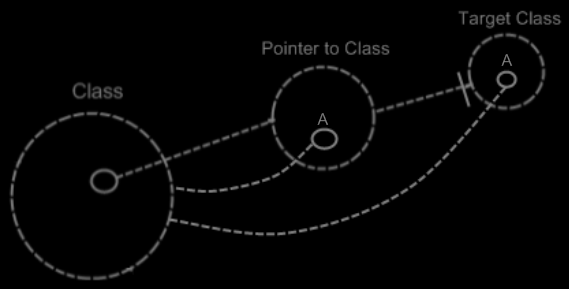


This is mostly applied, to allow a class to make a sub-object’s class *adjustable*.

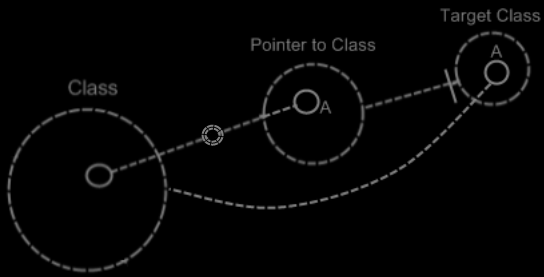
It is important to consider, that everything inside a pointer is really part of the *target class*, but a pointer itself is usable individually, independent from the target class.

To make a relation to a pointer dual, you have to give the target class a relation back to the first class.

The relation back can be displayed in both symbols, that represent the target class:



The two unary relations between Class and Pointer to Class melt together to a single dual relation. But the unary relation from the Target Class to the Class stays unary, because Class does not directly relate to Target Class:



The notation for a dual relation was covered by the article *Relations in a Diagram*.

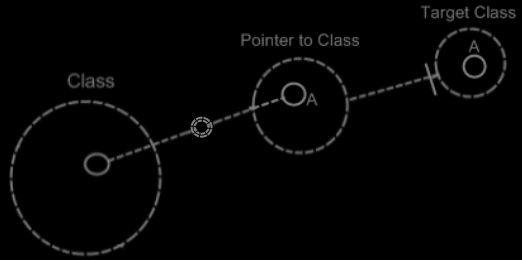
So only Class and Pointer to Class get a dual relation to eachother.

Target Class keeps a unary relation to Class. Funny enough, that unary relation is part of the dual relation between Class and Pointer to Class. The dual relation actually consists of:

- Class relates to Pointer to Class

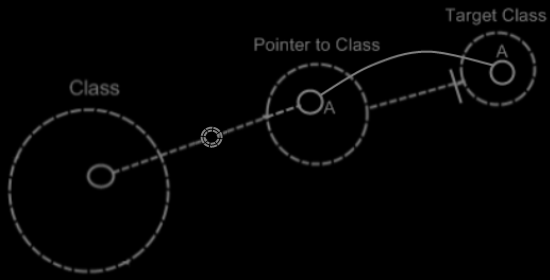
- Target Class relates back to Class

The connection between Target Class and Class is already implied by the connection between Pointer to Class and Class. You’re allowed to leave out of the diagram then:



Target Class and Class are already implicitly related to eachother through the pointer to the target class.

In all the diagrams above, that display the backward relation, the sub-symbols of Pointer to Class and Target Class were given a name: A. This was done, because there was no line in the diagram to indicate that they were the same sub-object. Officially, when symbols share an aspect, in that they are equal in object, class, interface or definition, they should be tied together with a line. Officially an object line should have been connecting both symbols of A:



But similarity in aspect can also be implied by a *name* and the *connection between parents*. This kind of implicit connection is explained in the article *Automatic Containment*.

The only point to implicit connection through parent is to make the diagram clearer.

## Ideas

Relations,

Relations to Pointers,

2008-09-25

Pointers (references to related objects)

A relation between a *pointer to an object* and a *command*. The pointer is a totally different entity, than the object itself.

> 2008-10-01 I’d think, that this will add related objects to the system interface, so related objects to a related item system object, instead of related objects to the target object of the related item system object.  
This is a relations issue: relations to pointers in particuler.

I will need to look at *System Objects* to see what a pointer actually was: it was a relation to a related item, instead of a relation to an object independent of any other container.

JJ