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

## Dual & Unary

### Concept

A unary relation is a relation with only one direction: one class relates to another class, but the other class doesn’t relate back to the first class.

A dual relation is a bi-directional relation: one class relates to another class, and the other class relates back to the first class.

In most cases, it is best to make a relation dual. You don’t even have to give the the backward related item a name, just let it sit there, until you find a name for it.

Bi-directional relations were already introduced by the article *Relations*. What is left to cover is the reason why to make a relation dual or unary.

### Ridiculous to maintain backward relation

Only if storage of a relation counterpart results in a ridiculous amount of data, that you don't even use, then you may want to ommit the backward relation.

This is the case when the target class of the relation is very generally used. The key example for this is a *Number*. A number is used by too many other classes, so it is ridiculous to give a Number a related list for every class that uses Numbers. To determine if a backward relation is ridiculous to maintain you could also consider the following:

- No functional correspondance with anything particular

- Too many objects will refer to this.

This is all very subjective, but I can’t give a more exact definition for it.

### Unable to program class

Another reason for not keeping the backward relation, is that you may not be able to program the target class, because somebody else authored it. But there’s a way to go around this: use inheritance to create a derived class, relate to the derived class, storing the backward relations inside the derived class. Then the original class is not burdoned with extra related lists. The derived class is an extension of the original class.

### The Referrers Concept

A Number class could choose to support the Referrers concept. This will give a Number object only one list of all referrers, instead of a separate list for every class that uses Numbers. Numbers may be used by many classes, but an individual Number object, is never used much. It is not a lot of data to register inside an Number object, which objects refer to that particular Number.

But then the Number class will also register all its *class referrers*, which is undoable, because a humungous amount of objects refer to this class. But a solution for this was already proposed by the article *Class Referrers*. You can choose for a class to not register its class referrers, while objects do register their referrers.

### No exact formula

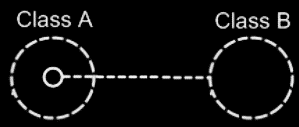
I admit, that I would like to give an exact formula for when a backward relation should or should not be maintained. But for now, I can only give a functional description of when it is ridiculous to maintain a backward relation. A programmer is going to have to determine it, when a relation should be unary.

### Diagram Notation

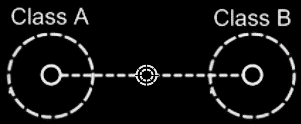
The concept of dual and unary is explained in the articles *Dual & Unary, Related Classes* and *Relations*.

The expression of dual and unary in a diagram has already been introduced in the articles *Related Classes in a Diagram* and *Relations in a Diagram*. This article only puts the diagram expressions of dual and unary next eachother for a comparison.

This is a unary relation between Class A and Class B:



This is a dual relation between Class A and Class B:



## Ideas

Relations,

New thing: what I should consider in the future, is that a relation counterpart can be completely derived from the other relation counterpart. Therefore, you might make a relation dual, so the counterpart usable, but not STORE it, but derived it somehow. That way you can use all relation counterparts, just not store the ridiculously large ones. > No, because then you’d have to scan the whole internet for referrers.

JJ