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

## Relations Between Objects

### Concept

Relations between classes set the rules for how objects can be connected to eachother.

Relations between *objects* are the actual connections.

A relation between objects is always a relation between *two* individual objects. If one object refers to another, the other refers back to the first one.

From a 1 🡪 1 perspective this seems logical, but from an n 🡪 n perspective this may not seem logical.

A relation between two classes with each a multiplicity of n, creates a related list in both of the classes. Every object of those classes will contain a related list of related items.

Any object that as a relation to another object, gives the other object *one* relation back to the first object. So for each reference to an object, the other object contains a reference back. One reference inside an object is tied to one reference inside another object.

One object can relate to multiple objects, so an object *can* have a one-to-many relation to other objects, but one *related item* in one object always creates *one related item* inside the other object.

### Diagram Notation

The concept of relations between objects is explained in the article *Relations Between Objects*. The current article only explains its expression in a diagram.

The main rule about relations to object is: for each reference to an object, the other object contains a reference back.

When you draw out the separate counterparts of a relation between two objects it looks like this:



But to express the close relation between the two references, the two lines are merged, and disambiguated from normal object lines with a *relation symbol*:



The relation symbol is a double circle. The reason behind this notation, was already explained in the article *Relations in a Diagram*. The notation is accomplished by first taking the original picture with one class refering to another and the other refering back to the first class:



Then, an imaginary reference to each class is added to the diagram



Next, the class lines are merged, but also the class symbols are merged:

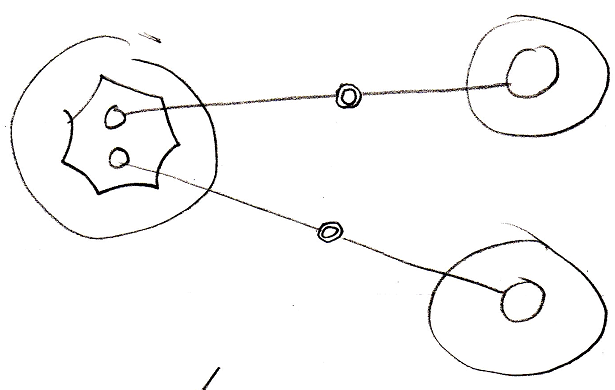


The notation would still be ambiguous, if it weren’t for the double line of the merged object symbols. So a double circle symbolizes a relation between objects.

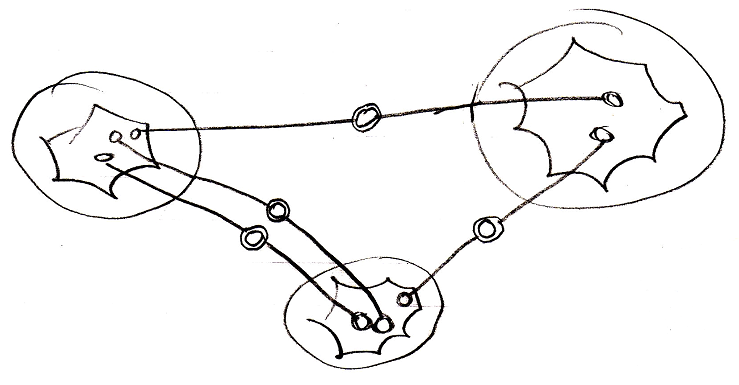
For relations between classes the relation symbol is a double *dashed* circle. For relations between objects, the relation symbol is a double circle drawn with *solid* lines.

In 1 🡪 n and n 🡪 n relations the rule, that each reference contains one reference back, also applies:

1 🡪 n:



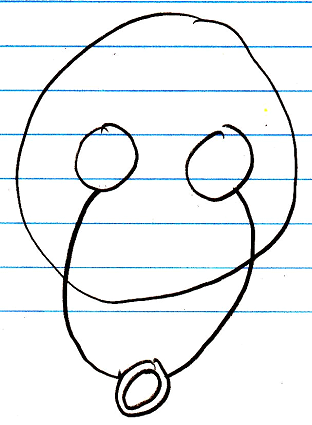
n 🡪 n:



It doesn’t matter whether an object reference is part of a list or not: every time it is still *two* individual object references, that are tied together.

### Object relating to itself

Sometimes an object relates to itself. In a diagram this looks as follows:



### Counterpart out of sight

When the counterpart of the relation is out of sight, a line should point out of the diagram. A catch there is, that you can’t see if the relation counterpart is part of a multiplicity of n or not. Therefore, the multiplicity is expressed at the end of th line pointing out of the diagram as follows:



### No reuse of merged imaginary references

If two imaginary references have merged, to become a relation symbol, then other references to the same objects won’t connect to an imaginary reference that has merged to become a relation symbol. Relations create their own imaginary references, that aren’t reused.

Here is a relation between two objects:



The two objects refer to eachother. This originally consisted of two distinct references:



Imaginary references were put on one level higher:



If other references to the same objects were also displayed in the diagram, then they would connect to the same imaginary references, put on a higher level:



When you merge the imaginary references to display that two references are part of a single relation, you will not connect all

references to the merged imaginary reference:



You will keep separate imaginary references for the other unary relations to the objects:

