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

## Multiplicity

Multiplicity as a coding principle is the distinction between single and multiple.

A related object can be a single object, but related objects can also be part of a list. In that case the list of objects gets a name, but the individual objects do not.

A single related object is also called a *related item*, while a list of related objects is called a *related list*.

A single related object can hold a reference to one object, but the reference does not have to be filled in, in case of which the reference points to no object at all. So a related item has a multiplicity of between 0 and 1. In some other programming languages this is expressed as: 0 .. 1 . A related *list* can be filled with zero or more related objects. In some other programming languages this is expressed as: 0 .. \* .

The two types of multiplicity are also called 1 and *n*. A related item is often called a 🡪 1 (say: ‘to one’) related item. A related list is often called a 🡪 n (say: ‘to en’) related list.

Other types of multiplicity (such as 1 .. \*) are not part of the coding principle of multiplicity. The fundamentally necessary types of multiplicity are 1 and n . Other types of multiplicity can be implemented as a *concept* programmable in the new computer language itself, rather than a *principle* that the system cannot do without.

In Computer Language 0.9 there was also a multiplicity called x. X was a list with a fixed number of items in its collection: for instance a list of exactly three items. Each of the items had a name. The idea was abandoned, because the effects of synchronizing the two sides of the relation were unpredictable (see the article *Relation Synchronization*), and 🡪 x could could easily be replaced by for instance three separate related items.

If an object contains a related item, it is displayed in the diagram as just a circle contained by another circle.



If an object contains a related list, the related list is displayed inside an object as a *nonagon*.



The reason this shape is used, to express a related list, is because it has many corners, symbolizing a multitude. It is also like circles would fit in between the pointy corners. This way it seems to be the inversion of a circle: the opposite of a singular object.

The items the related list are placed inside the nonagon again.



## Ideas

Multiplicity,

A collection symbol only makes

sense when the class of an object references is fixed,

because when classes are not fixed, you can already

put any amount of objects inside another object,

but with a fixed class this is not the case.

JJ

Multiplicity,

2010-05-15

Actually, one of the points of putting objects inside a separate list is that the list can centrally control certain aspects that each item of the list must conform to.

JJ