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

## Related Classes

### Concept

Target classes specify sub-objects.

If a sub-object in a class also gets a class assigned to it, then this relates the target class to the target class of the sub-object. The target classes of the sub-objects are called the object’s *related classes*.

If a class does not fix the class of a related item, then any type of object could be assigned as the related item. If a class fixes the class of a related item, then the related item can only become an object of that class.

A class can also specify related *lists*. If a class is assigned to this related list, the related list can only contain items of this class. If no class is assigned to a related list, then the related list can contain objects of any class.

A related list can also be assigned *multiple* classes, meaning that items of a fixed set of classes can be put inside the related list.

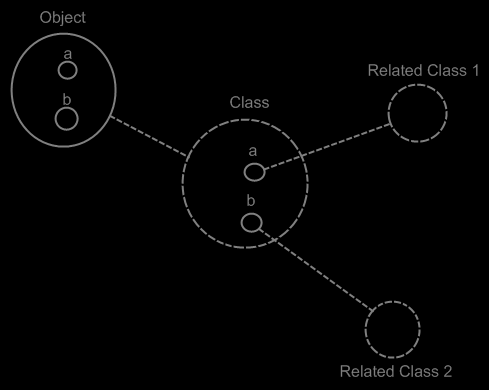
In that case one related list creates two related classes.

If a class’s related item does not have a class, the related item does not introduce a new related *class*.

### Diagram Notation

The concept of related classes is explained in the article *Related Classes*. This article only explains its expression in a diagram.

Below are displayed an object and its class. The class has two sub objects, each of which points to another class:

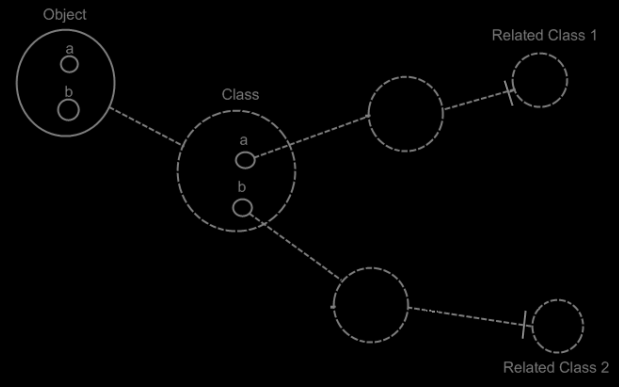


The Object automatically gets the same contents as the Class. The object’s sub-objects a and b could have been tied to Related Class 1 and Related Class 2, but they are not, because the sub-objects of Object and the sub-objects of Class are implicitly tied together by the tie between their parents Object and Class and the fact, that they have the same name, following the principle of *implicitly connected through parent* (will be explained in the article *Automatic Containment*).

For a big part it is true, that dashes uncover the structure of a system, while the solid lines uncover the system’s data. All the classes and their related classes and the lines between them are dashed. However, the sub-objects defined inside a class are not dashed, because they do not function as classes themselves. And also, the lines from object to their class are dashed. So it is not 100% true, that all the structure elements of the system are dashed, and all the data is drawn with a solid line, but it’s close to it.

When you want to see the structure of the system, and ignore the data of the system, you just have to look at the diagram from the following perspective: class structure = classes tied together.

The below is the same example, but now the classes get further redirected.



If the dashed lines do not emphasize the classes and relations enough, a coloring could be applied to the diagram, highlighting all the classes and their relations to other classes.

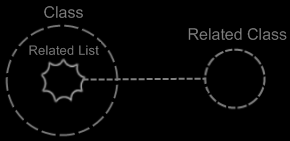
A class can also have a related list: a class holds a list of items of another class. A multiplicity of *many* is expressed in the diagram with a nonagon:



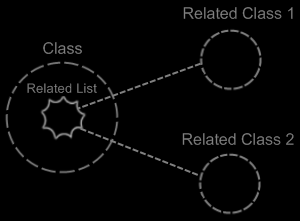
If the nonagon is placed inside a class, then the class specifies a list of items:



No class is assigned to the related list here, so the related list can contain objects of any class. If a class is assigned to this related list, the related list can only contain items of this class.



A related list can also be assigned *multiple* classes, meaning that items of a fixed set of classes can be put inside the related list.



In that case one related list defines two related classes.

If a class’s related item does not have a class, the class has a related item, that can be od any arbitrary class. So this related *item* does not introduce a new related *class*.

