|  |
| --- |
| Circle Construct Drafts |

## Relationships Construct Drafts

### Relationships Between Classes

The\* class-relationship structure is the\* bonestructure of a program.

### Bidirectional Relationships

The notation suggested here was moved away from the Circle Language Spec in favor of another notation.

Starting with:



A suggested notation would be for the two class lines to merge together to form the picture below:



This, however, might be a quite ambiguous notation. It would suggest that the two symbols joined by the line would have the same class. But that would not be what is intended. The circle inside **Class A** would have **Class B** and the circle inside **Class B** would have **Class A**.

A solution to this ambiguity may be proposed.

Fortunately, the\* notation can\* be disambiguated using the\* rules of automatic containment. Automatic containment is explained in the\* article *Automatic Containment*. Before explaining how automatic containment leads to the\* eventual notation, here is the\* disambiguated notation of a relationship between two classes:



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 dashed line of the\* merged class symbols. So a double dashed circle symbolizes a relationship between classes.

Examples with different multiplicities:







A symbol merge in a relationship that has nonagons in it also results in a double dashed circle, because\* the\* imaginary reference to the\* classes, that are put on a higher level, are represented by circles, not\* a nonagons.

There is an idea where one related list might contain items from multiple classes. **Class A** would have a **n 🡪 1** relationship to items of **Class B** and **Class C**, which the picture below aims to express:



#### Counterpart out of Sight

If the counterpart of a relationship would be out of sight, a line might point out of the diagram. A catch there might be that you might not see whether the relationship counterpart would have multiplicity of **1** or **n**. A possible solution for this, might be to express multiplicity at the end of that line that might point out of the diagram.

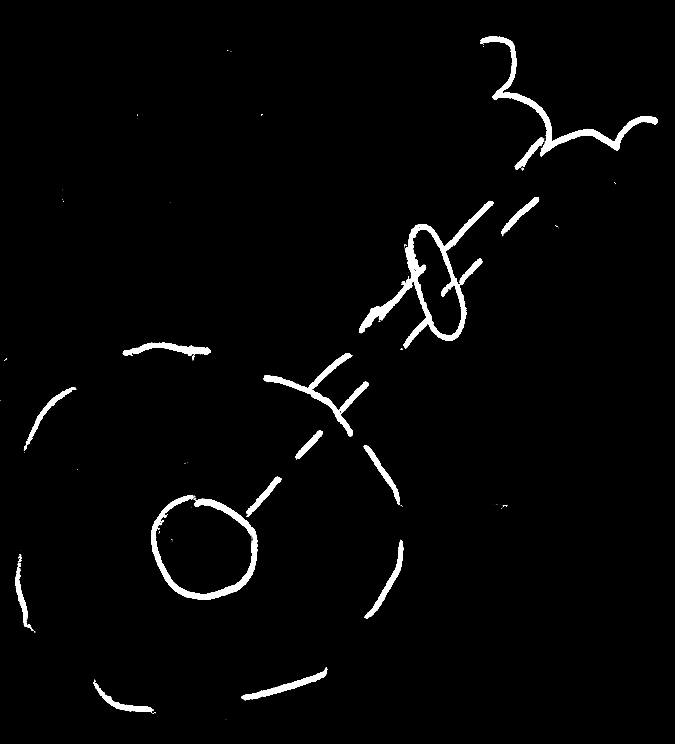
This might look as follows with the double dashed border notation:

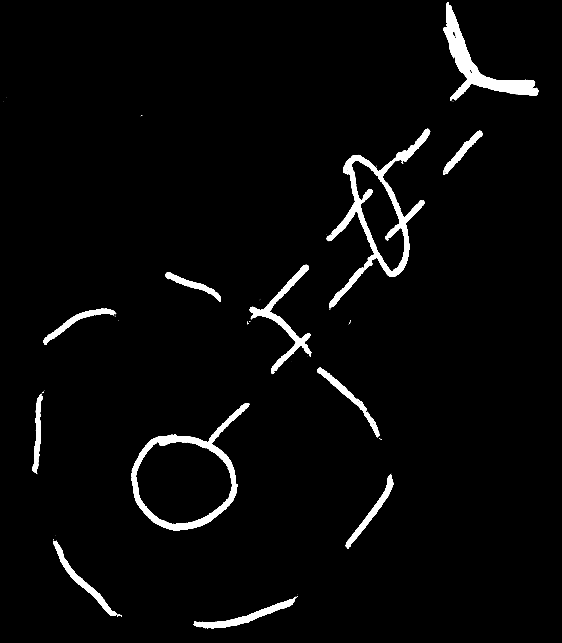




But then again, if something is out of sight, it might just be out of sight and you cannot see things out of sight. Perhaps there is no problem here.

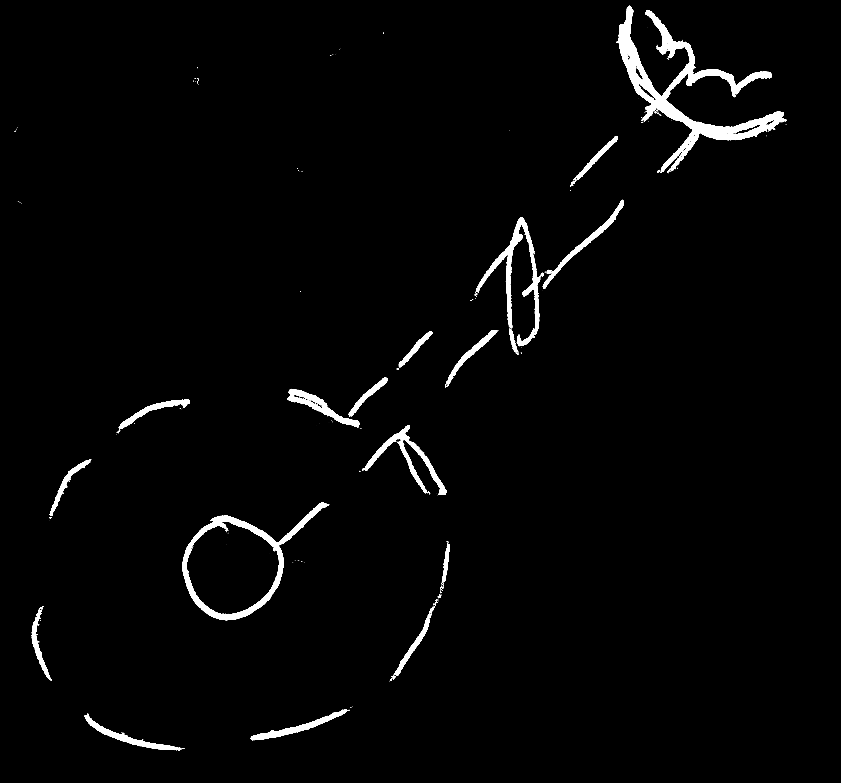
Here are examples of what it could look like in case of relational ring notation, with explicit relationship counterparts:

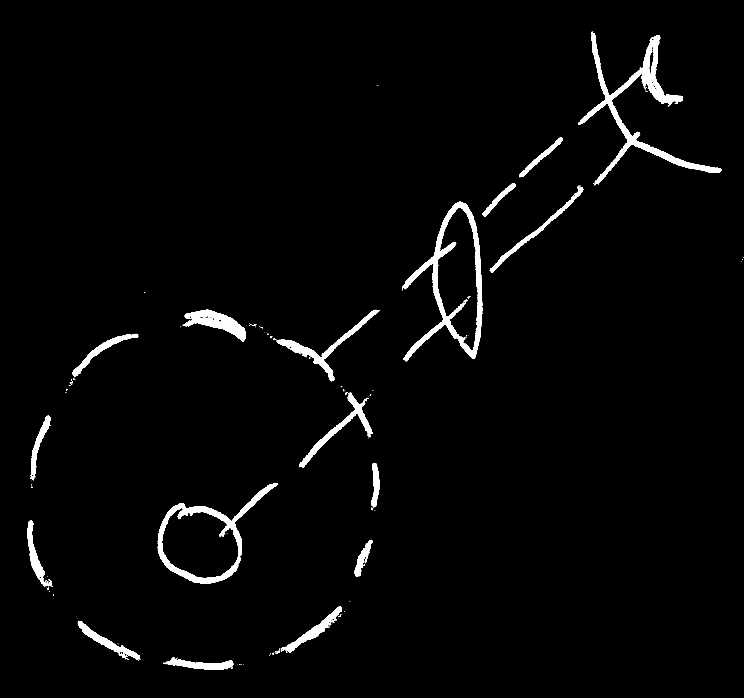




That proposal might be a problem, because it seems to clash with a proposed notation for optional. And also it seems to not reflect the containment structure: there would be a container in between the half shape and the other parts of the diagram. Seems unfortunate.

In another proposal:

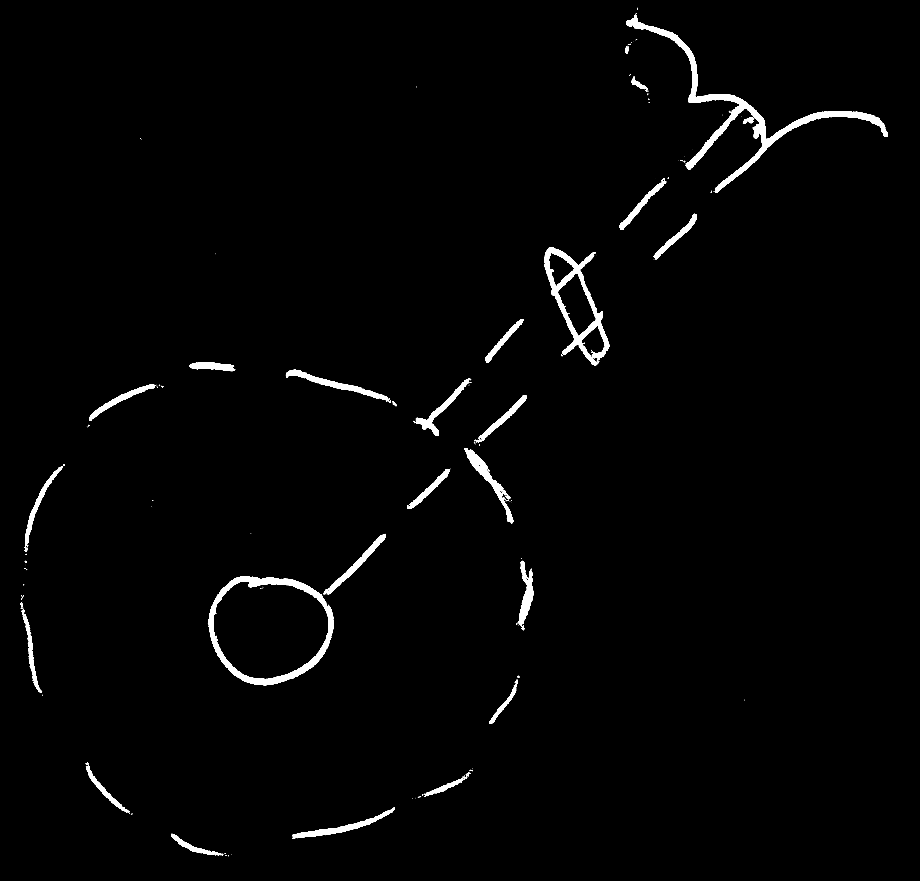




It may be going a bit far. It seems to draw out part of the diagram, that is out of sight. But then it might actually have a container in between, that is not drawn.

The argument "What is out of sight, is out of sight." might be good enough to not solve this 'problem'. A proposal might be that might not be a real problem.

Here a variation that might also be dubious:



In that image both relationship counterparts' class lines are connected to the same half shape. It might go a bit far in suggesting something it's not.

#### No Reuse of Merged Imaginary References

If\* two imaginary references have merged, to become a relationship symbol, then\* other references to the\* same classes won’t connect to an imaginary reference that has merged to become a relationship symbol. Relationships create their own imaginary references, that aren’t reused. This is displayed in the\* article *Relationships Between Objects in a Diagram*, but\* may also apply to the\* notation of relationships between classes.



#### Example



### Relations Between Objects

#### Diagram Notation

The\* relationship symbol is a double circle. The\* reason behind this notation, was already explained in the\* article *Relationships 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 relationship between objects.

For relationships between classes the\* relationship symbol is a double *dashed* circle. For relationships between objects, the\* relationship symbol is a double circle drawn with *solid* lines.

#### Counterpart out of sight

When\* the\* counterpart of the\* relationship is out of sight, a line should point out of the\* diagram. A catch there is, that you\* can\*’t see if\* the\* relationship 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 relationship symbol, then\* other references to the\* same objects won’t connect to an imaginary reference that has merged to become a relationship symbol. Relationships create their own imaginary references, that aren’t reused.

Here is a relationship 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 relationship, you\* will not\* connect all

references to the\* merged imaginary reference:



You\* will keep separate imaginary references for the\* other unidirectional relationships to the\* objects:



## Loose Ideas

Stereotyping relationships

Perhaps another typing can be assigned to a relation, instead of containment. For instance: *ownership* or *usage*. Perhaps a few standard ones, and it may be possible to define your own typing by specifying a String.

JJ