|  |
| --- |
| Circle Language Spec |

## Classes

### Main Concept

The contents of an object might be arbitrary. Anything might be put inside an object.

But an object might also select another object to serve as its *class* or *prototype.* Then the contents of the object might not be that arbitrary. At first an object may contain related items and related lists that roughly corresponds with the class and the object might also have a similar set of commands as the class. The idea is that an object's behavior during its lifetime would also be bound to rules that might be set by that class.

A class is sort of like a special object, that aims to describe the characteristics and behavior of other objects.

An object might have a similar *structure* as its class, but that object may have *data* that can change freely. Values of an object's attributes and might freely change as well as which are an object's related objects . But initially an object may look like a replica of its class. The changeable parts of an object might be set initially to what is defined in the class.

#### Class Redirection

An object’s pointing out its class, might be called a class redirections.

It might be expressed in a diagram by connecting an object symbol to its class with a dashed line:



The object on the left might then have the class on the right.

#### Using an Object as a Class

Perhaps it is common that an object would be fixed in its role as a prototype or class. But the Circle notation would allow any object to serve as a class or prototype for another object.

#### Using a Class Like an Object

Allowing object references to a class, would make it possible to reference a type like you could an object.

#### Object Reference with a Class

Because next to an *object* having a class, an *object reference* might also have a class. If it does, only objects of that class might be assigned to it.

#### Object Reference without a Class

When an object reference would not have a class, this might stand for its being able to point to *any* object.

#### No Class != Defines its own Class

Formerly something might have been unclear. Objects might all be usable as classes. This might make it tempting to think of an object *without* a class would define *its* *own* class. Instead, it might be handy to not think of it that way, but think of it as object without a class simply not having a class. Having no class might stand for arbitrariness.

There might be examples where it could be less fortunate to not think about it that way.

#### Object Reference would define its Own Class?

There may be other exampels, but it might be unfortunate to think of an *object reference* as defining its own class, because then the object reference could only point to … itself?

#### Diagram Notation

The Circle notation might allow any object symbol serve as another object’s class or prototype. So in these diagrams any object might be used as a class or prototype for another object.



When an object would be used as another object’s class, it might be an idea to draw it out with a dashed line. *Dashed lines* might symbolize the concept of *classes*.



In these diagrams classes might usually look like that.

It may be an idea that if a symbol serves as another object’s class, but also is referenced as an object, the symbol would get a double border to maybe indicate its dual role as both an object and a class.



But perhaps just a dashed border is more appropriate after all, since its primary role seems to be a class. It might be a class, but there might also be *references* to that class (which might be established with an object redirection to that class).

### Commands and Classes Loosely Coupled

#### Concept

Objects of the same class might have a similar set of commands.

But one particular idea for commands would be, that they might live as separate entities, not part of a specific class.

The general idea would then be, that a command would automatically be available inside a class, if one of its parameters matches with that class.

In this idea, when a parameter of a command would not be given a class, the command would be available from *any* object.

In this scheme, giving a parameter a specific class might limit the amount of objects that that command would appear in.

#### More Commands but Grouped by Source

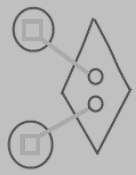
This idea might also give an object more commands. In an attempt to keep overview, commands might be grouped together inside an object. Each source/module/library/service might add a separate group of commands inside the object. An interface symbol might set the sources apart. If you would not trust a source, you may opt not to use that command.

#### Mutual Control

So the control over the commands is not only with a class, as might be classically the case, but the commands also control which classes they are available in. You might see it as: you can always invent new ways to use an object, to do more than what it was initially intended for.

#### General Notation

A command might be executed on an object. This would be the suggested notation.:



An attempt to clarify this notation in steps would now follow.

#### Example

The following drawing might be considered:



It aims to show a command definition, which would be the larger dashed square at the bottom. It also attempts to show an object without a class: the circle on the left. It also tries to show a second object: the circle on the right. The object on the right would point to a class using a dashed line, connected to the dashed circle in the top-middle part of the picture. The dashed circle in the top-middle part would be a class.

#### Parameter without a Class

When a parameter might be added to the command, and that parameter would not have a class, the following might happen:



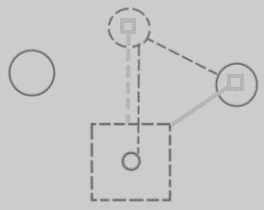
The command definition at the bottom would be given a circle inside it. That would represent the new parameter. The parameter would have *no class* yet here, so a square command symbol would appear in *each* object and in the class. The small squares all have lines to the large square at the bottom. That would indicate the smaller squares would be the same command definition as the large one at the bottom.

(That the middle line is dashed, may have been more of a stylistic choice in an attempt to distinguish static structure from object structure by dashed lines versus solid lines. The lines could have been drawn with a solid line. Another choice could be for more symbols to be drawn with dashed lines, for more (intuitive) separation between static structure and object structure. Dashed shapes might not have an exact definition or meaning yet, so there may be freedom of choice there.)

So command symbols would have appeared in each of the objects and in the class as well. With this idea, a command might quickly become very present in the system, where it might be executed on any object.

#### Parameter with a Class

When assigning a class to a parameter, then the command might only be available from objects of that class. This may limit the amount of objects the command would be available in:



The image attempts to show some differences compared to the previous image. The small circle in the large square now has a dashed class line pointing to the dashed class circle at the top. What also seems to have happened, is that this made the object circle on the left not have the command anymore. The circle on the left no longer seems to participate, because it does not have the class associated with the command.

#### Line Merge

The connections in the center of the image, might seem a bit crowding. These lines might be merged together:



This might be less visually demanding. It might also express the tighter bond between the command parameter and the class command. The two two directions.

This notation may be a bit ambiguous, because it suggests, that the two symbols are directly connected together, instead of two connections with an inverse relationship. An arguments to perhaps defend the perceived ambiguity, might be that an object symbol and a command symbol might never be tied together directly like that in any other case. It also seems to express the tightness of the bond. They are not independent connections: change one connection and the other one would change too.

#### Summary

So this:



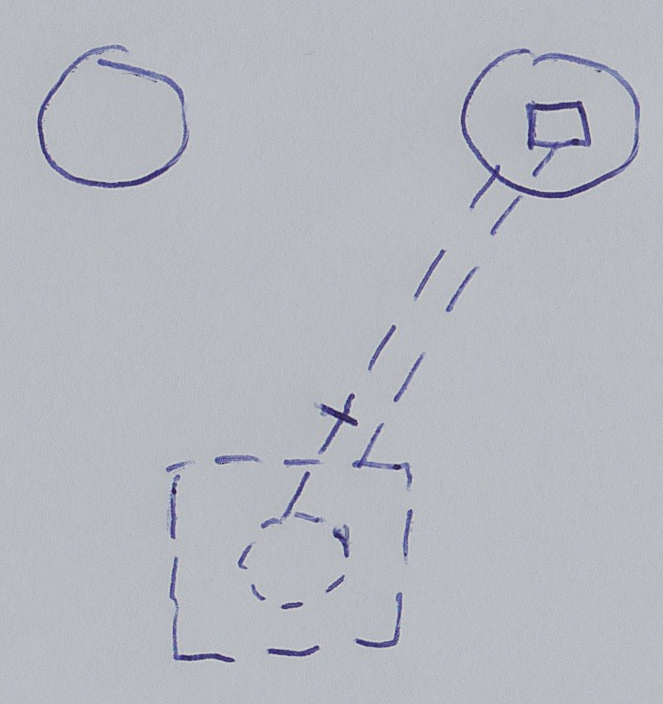
When adding a parameter with a class, might turn into this:



What would be visualized here, might be that the system may have been expanded with a connection between a class and a command, which may also have added the command to objects of that class.

#### Commands and Classes Loosely Coupled when Object would Define Its own Class?

The concept of *Classes and Commands Loosely Coupled* may run into trouble when thinking of classless as 'defining its own class'. It would imply, that when a parameter would be assigned a class, command would become available in every object of that class. When a parameter would have no class, it might become a problem when thinking of that as a parameter defining its own class. That might only make the command available to objects, that would point out *that parameter* as their class.



A quite particular situation. But it might be solved by regarding situation as that a parameter without a class, would simply *have* *no* class. That may add the command to *any* object. This make *classless* stands for *arbitrariness* again.

