|  |
| --- |
| Circle Language Spec: Classes |

## Class Commands

### Concept

Objects of the same class have the same set of commands.

However, commands are separate entities, not defined by a class.

When a parameter of a command is not given a specific class, then the command is available from any object.

This gives an object a lot of commands. But this will not result in a mess, because commands are nicely grouped together inside the object. Each module creates its own group of commands inside an object. If you don’t trust a module: don’t use the command.

So when the class of a command parameter is not fixed, the command is available from any object. When the class of a command parameter *is* fixed, this makes the command only available from objects of that class. This considerably limits the amount of objects that get the command.

In that sense a *command* defines behavior of a class. Which makes sense from a real-world point of view, because you can always invent new ways to use an object.

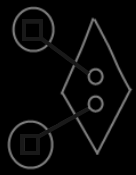
A command is available from any object that has anything to do with the command.

### Diagram Notation

The principle of class commands is explained in the article *Class Commands*. This article demonstrates their expression in a diagram.

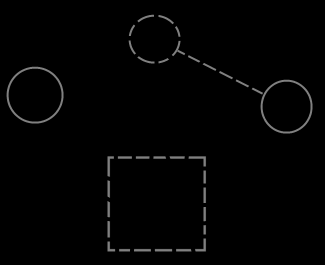
The article *Command Arguments in a Diagram* demonstrates, how a command argument also makes the command part of an object.

A command is executed on an object.

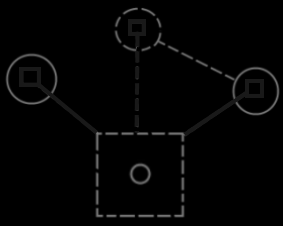


If a command definition does does not fix the class of one of its parameters, then the command will be immediately available from *any* object. The command will also be visible in *any* class.

Here is an example of a command definition, two objects and a class. One of the objects has that class.



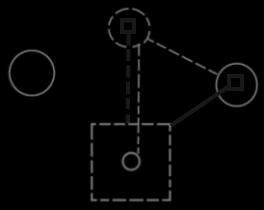
When a parameter is added to the command, and the parameter does not have a class assigned to it yet, the following happens:



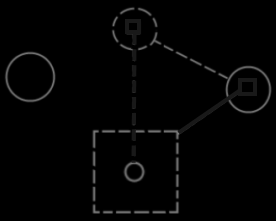
Because the command definition got a parameter with no class assigned to it, the command immediately becomes available from any object or class in the system. The command symbols inside the objects and the class are tied to the the definition, to indicate mutuality of definition. Immediately the command is very present in the system, because it can be executed on any object.

In theory, all the lines between the squares could have been drawn with a solid line. But the *class* and the *command definition* are tied together with a dashed line, because it is a relation between structure elements. Now all structure elements and their relations are drawn out with dashed lines. It’s more intuitive that way.

If you assign a class to the parameter, then the command will only be available from objects of that class. The command will also only be visible inside just that class, not just any class.

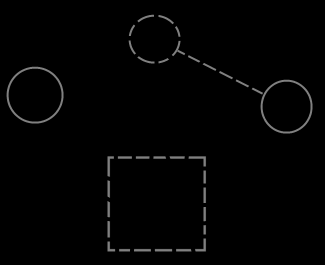


The connection between the class and the command definition is now crowded with two lines. The lines are merged together, to express the tight bond between the command parameter and the class command.

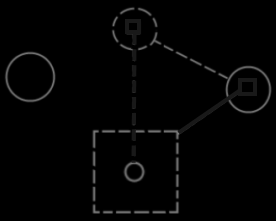


This also better expresses, that the two directions of the bidirectional relation between the command and the class are linked.

So in short, this:



When adding a parameter with a class, turns into this:



What is visualized is, that the system got expanded with a connection between the class and the command, which also added the command to the only object of that class.