|  |
| --- |
| Circle Language Spec: Parameters |

## Relations Between Commands & Objects

### Concept

A command can contain a set of related objects. Those are the parameters of the command. A command will use objects to do its work.

A parameter is an object related to a command.

The relation is bidirectional. When an object is a parameter, then the object also contains a reference back to the command.

There is complete exchangability between a command’s parameter and an object’s command.

Commands are special objects, that just happen to be executable. That is why the concept of relations applied to commands and objects, result in the concept of parameters.

You can view a system as being a flat set of procedures, that take objects as parameters. You can also view a system as being a set of objects, that have a set of commands inside of them them. Procedural and object oriented are completely interchangable in this system.

Multiple commands can have the same object filled in as a parameter.

Then the object will contain all of those commands.

These two perspectives on it are completely interchangable.

A command can contain multiple objects.

Then the command becomes part of all of those objects.

These two perspectives on it are completely interchangable.

In other programming languages, an executed command is considered part of *one* object, while in this new computer language the same execution of a command may as well be part of multiple objects. A command is part of any object, that is an argument of the command.

#### Object relations

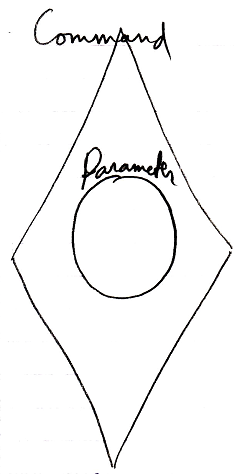
So when you’ve passed an object to a command, the command automatically becomes visible inside the object as a runnable command.

#### Class relations

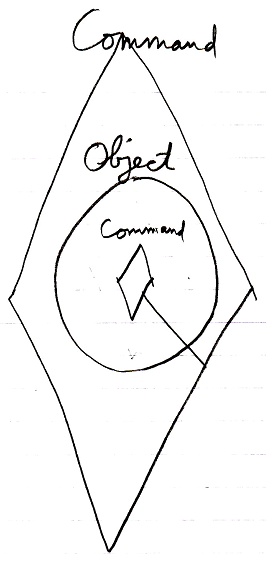
But in case of a command definition, a parameter is usually not filled in yet, but the class of the parameter *is* filled in. It is a relation between classes, instead of a relation between objects. This automatically makes the command definition available from the parameter’s class. There is complete exchangability between the class of a parameter and a command of a class.

### In a Diagram

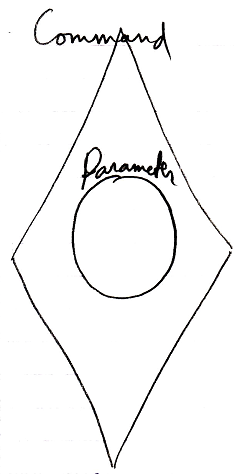
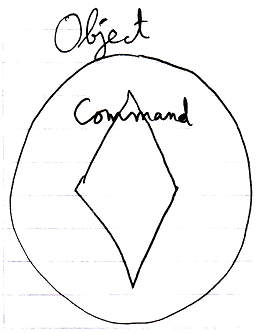
A parameter is an object related to a command.



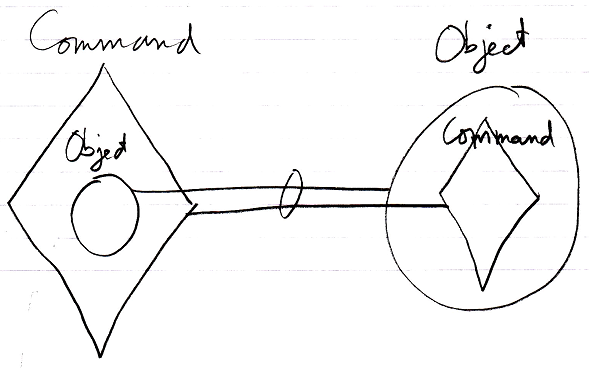
The relation is bidirectional. When an object is a parameter, then the object also contains a reference back to the command.



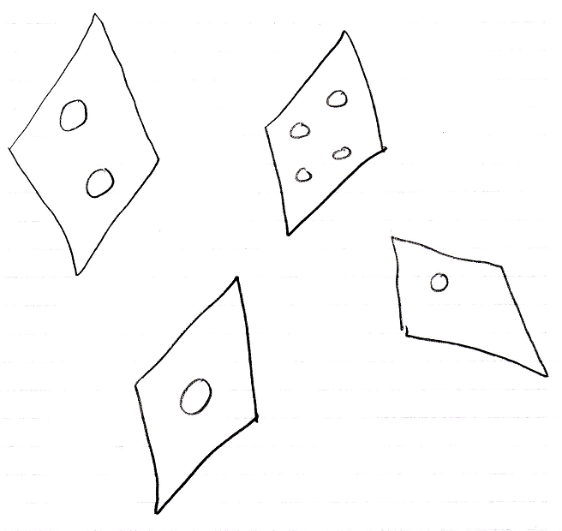
There is complete exchangeability between a command’s parameter and an object’s command.

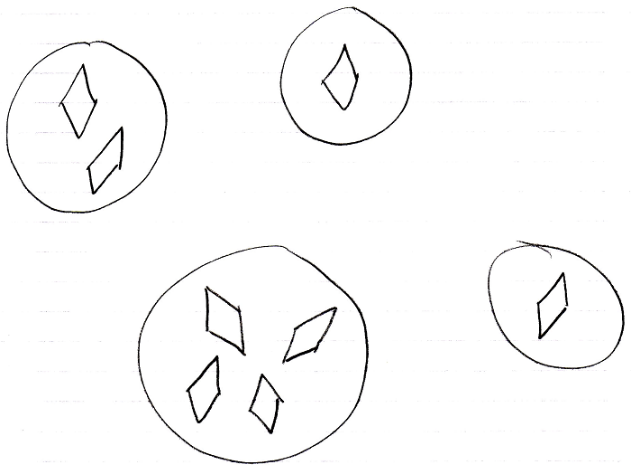
Commands are special objects, that just happen to be executable. That is why the concept of relations applied to commands and objects, result in the concept of parameters.



You can view a system as being a flat set of procedures, that take objects as parameters.

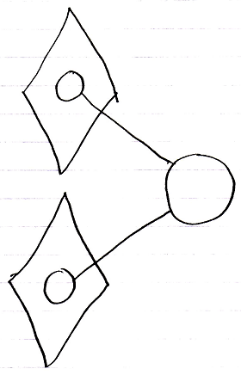


You can also view a system as being a set of objects, that have a set of commands inside of them them.

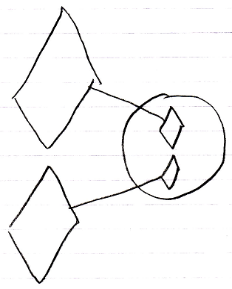


Procedural and object oriented are completely interchangable in this system.

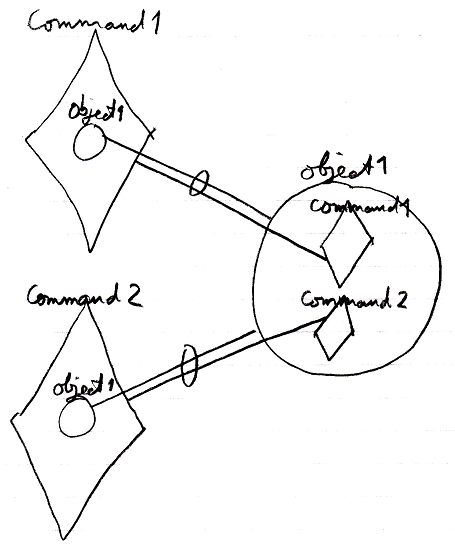
Multiple commands can have the same object as a parameter.



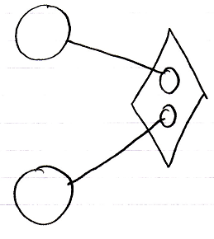
Then the object will contain all of those commands.



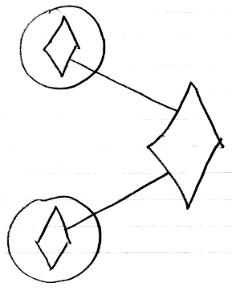
These two perspectives on it are completely interchangable.



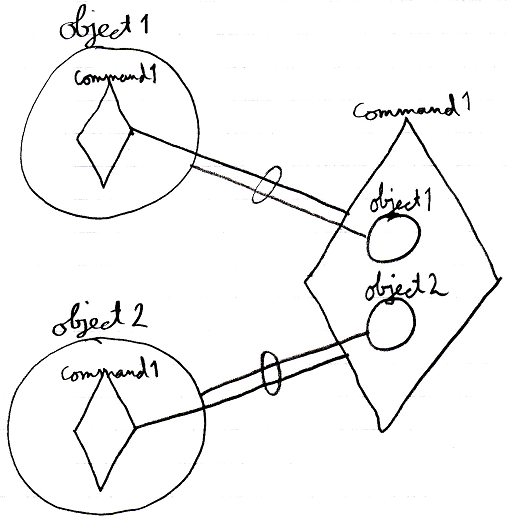
A command can contain multiple objects.



Then the command becomes part of all of those objects.



These two perspectives on it are completely interchangable.



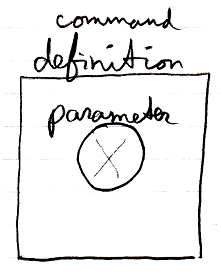
In this new computer language both points of view, commands being part of objects, and commands being independent entities with parameters, are equally visible in the language.

#### Object relations

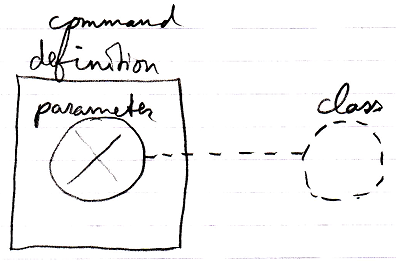
So when you’ve passed an object to a command, the command automatically becomes visible inside the object as a runnable command.

#### Class relations

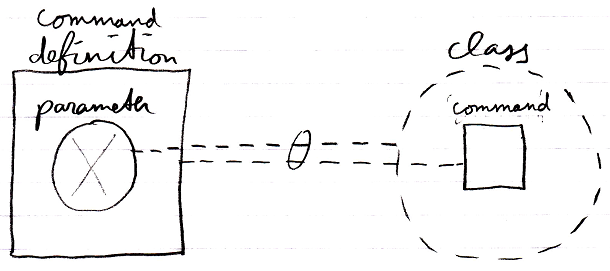
But in case of a command definition, a parameter is usually not filled in yet.



But the class of the parameter is determined.



It is a relation between classes, instead of a relation between objects. This automatically makes the command definition available from the parameter’s class.



There is complete exchangability between the class of a parameter and a command of a class.

## Use this

When you program a command, that does something, the action is immediately available in the context of any type of object, that has anything to do with that command.

This makes it so that the very same command can be part of several classes. This makes commands independent entities, that are not tied to a class.

A command can be called on each object, whose class is a parameter of the command.

When you call a command on an object, the call points to the command of an object, but the object is also visible as a parameter of the call.

### Out of Commands & Classes Loosely Coupled

#### Inactive command references

In the explanations above only command calls were considered, not inactive command references. Inactive command references don’t work the same as calls.

Calls have parameters filled in. Inactive command references don't have parameters filled in. So they also don’t call upon objects. Only as a command reference gets called again, objects assigned to the call’s parameters.

The object of the parameter is not determined yet in an inactive command reference. However, the class of the parameter may be determined after all. Therefore, the parameters in an inactive command references may be tied to a command inside class with a class line.

#### Local variables

Local variables, as they are called in contemporary programming languages, are a lot like parameters. Unlike parameters, they are private to the command, rather than publically written or read. Local variables are private contents of a command. Since they can not be publically set, and you can not determine what object the private variable will hold, the command also can not become part of any arbitrary object of the class of the local variable.

So the command does not become part of each object of the class of that local variable, because the object of that local variable can not be freely appointed.

The local variable *might* make the command part of the class anyway, but then as a static command, that can be called on the class, but not on any arbitrary object.

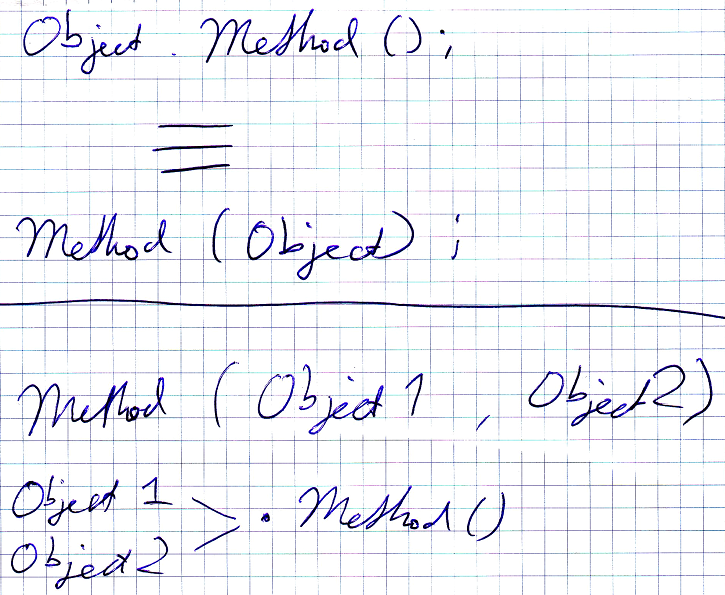
> 2008-09-06 Not true. Just wait what it will look like relations between objects and commands.

## Ideas

Commands,

± 2008

The idea of multi-methods and exchangeability between class methods and method parameters is the expressed in textual pseudo-code below:



JJ

## From the original Symbol documentation

### Parameter and Argument

Parameters and arguments are the *public* members of a procedure. In a square they’re called parameters. In a diamond they’re called arguments.



A parameter is never created. An argument is an object that exists when the diamond executes. Therefore, an argument is a real created object, while a parameter is only a blue-print for it.

A parameter is a public member of a definition.

An argument is a public member of a diamond in execution.

* Definitions of the terms parameter and argument

#### Brainstorm

##### Other

Creation and state read and write only apply to object parameters, not to parameters that are procedures.