|  |
| --- |
| 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 anymore. 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. Also, the idea is that an object's behavior during its lifetime would be defined 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 might freely change. *Which* objects are referenced might also be different for each object. 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.

#### 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 assigned to it. Having no class might stand for arbitrariness.

There might be some examples where it could become a problem to not think about it that way.

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

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?

#### 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.

<Perhaps a picture.>

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.

<Perhaps a picture.>

#### 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).

### 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.

### 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 any specific class.

The general idea then would 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.

#### Many Commands Grouped by Source

This idea might quickly give an object many commands. In this idea, apparently all it takes for an object to support a command, is for the object's class only has to match any command's parameters. And if a parameter is not given a class, it would even add the command to any object. 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. Those groups of commands might be set apart as separate interfaces. If you would not trust a source, you might opt to not use that command.

#### Other Issues

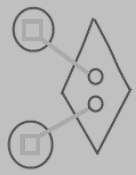
In that sense\* a *command* defines behavior of a class. Which makes sense\* from a real-world point of view, because\* you\* can always invent new\* ways to use an object.

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

#### Diagram Notation

The\* 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 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.



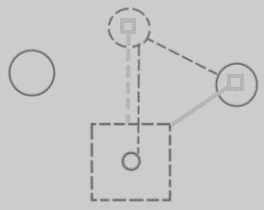
When a parameter is added to the\* command, and the\* parameter does not\* have a class assigned to it yet\*, the\* following happens:



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\* 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 is 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.

## Ideas about Classes Main Concept

Pointer to class of,

2008-08-17

Consider the\* notation of pointing to the\* class of an object reference, used in the\* article Class Referrers in a Diagram.

I need a notation for explicitly referring to a pointer or to the\* class of an object or to the\* class of an object reference.

Do consider that the\* target object in a diagram really needs to represents the\* object. You\* should not think of it as an object reference, because that will make it harder to see through the\* system.

JJ

2004,

Every of those objects has a type. The\* type determines the\* contents of the\* symbol. Every object of the\* same type has the\* same contents and the\* contents of these objects changes simultaniously as you\* edit it.

< 2008-10-12 That is no longer true. Objects of the\* same class can have different contents. But what does change simultaneously, when you\* edit the\* class? >

JJ

Classes,

2009-05-12

Another synonym for class is *type*.

You\* have to mention this somewhere.

JJ

Classes,

2008-11-13

If you\* can see object usage, you\* can not see class-sub-object usage.

You\*'d have to look at the\* usage of the\* sub-objects of the\* objects of that class,

to see the\* class's sub-object usage. Indirectly you\* will be able to see the\* dependency on a class's sub-object.

Doesn't a sub-object have a reference to the\* class's sub-object or does the\* parent

object only have a reference to the\* class?

JJ

Classes,

2008-11-26

The\* remark below might give you\* a clue about an exact sum-up of the\* uses of classes. One of the\* uses is having more than one of something. Another use is to selectively have none at all of something, so only a selection of things. Another use is being able to more easily reorganize separate units, if they are separate objects. Another use is being able to reference the\* same thing from multiple places. That's not a use of classes, but a use of objects. Perhaps all of this is the\* use of objects, not necessarily the\* use of classes.

Om van projectfases losse units van te maken, in plaats van één document, kun je makkelijker de units schuiven en rangschikken en slechts een gedeelte van de fases gebruiken, en een fase meerdere malen hergebruiken, eigenlijk precies zoals je dat met classes doet.

JJ

Classes

2009-05-12

I do not know yet how to ventilate changes to classes to their derived objects.

JJ

## Ideas about Class Reference

Class reference,

2008-07-30

Right now I define class reference as being an object’s specification of what is its class.

But accidently I used class reference as a pointer to a class, as being analogus to a command reference.

JJ

### Out of the\* original Symbol documentation

An object symbol can also serve as its own type:



Then it is an object that defines its own type. But if it has a type line, it redirects its type to another symbol. Then it is no longer its own type, but a mere object from an existing type, also called an instance of a type. The\* target of the\* type line is regarded the\* type itself.



A is an instance, B is the\* type.