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

## Classes

### Concept

The contents of an object can be totally arbitrary. You can put anything inside an object. This is handy for users, who just want to group objects together into a parent object, much like they group together files in a folder.

However, an object can also select another object to function as its *prototype*. A prototype is also called a *class*. Classes describe the rules by which objects behave. Objects of the same class contain the same kinds related items and related lists and also support the same commands.

An object will have the same *structure* as its class, but not the same data. The values of the attributes can freely change for each object. *Which* objects are referenced is also different for each object. But initially the object will be an exact replica of the class. The class’s attribute values and object references only function as a default.

Any object can be used as a class. At first there was the idea, tha an object could be fixed in its role as a prototype, but in that case you could not establish a reference to a class anymore.

There used to be a misunderstanding about something. When an object does not have a class, it actually *does* *not* have a class. The object can however be *used* as a class. Formerly this was mistaken for an object’s *defining its own* class. But this is not true. An object without a class *does not* define its own class, *just* because it can be *used* as one. An object, that does not have a class, also does not define its own class; it simply has no class assigned to it. Its contents are totally arbitrary. *No class* stands for *arbitrariness*.

You have to keep that in mind. To understand why you have to keep that in mind, you need a prime example of a case in which it becomes a problem.

For instance: the article *Class Commands* introduces the concept of *commands and classes loosely coupled*. It says, that when a parameter gets a class, the command will be available in every object of that class. If a parameter has no class, it becomes a problem when you think of that, as the parameter’s defining its own class. Because in that case, the command will only be available from objects, who point out *that parameter* as their class. A strange situation. But the real situation is, that a parameter without a class, actually *has no* class. That adds the command to *any* object, because *no class* stands for *arbitrariness*.

An object can be assigned a class. An object reference can also be assigned a class. If an object reference does not have a class, then the object reference can point to *any* object. That is another example of how *no class* stands for *arbitrariness*. When an object or an object reference is assigned a class, you can not easily change that class. If an object has a class and you assign another class to it, it would erase the object’s original contents. If an object reference has a class, and it points to an object of that class, then when you change the class of the object reference, what happens to the target object, that has still has the original class? These are exceptional situations, for which the most practical behavior needs to be determined in the future.

### Diagram Notation

The principle of classes is explained in the article *Classes*. This article demonstrates its expression in a diagram.

Any object can serve as another object’s class. So any object can be the prototype for another object.



When you actually use an object as another object’s class, then its symbol is drawn with a dashed line. A dashed line stands for classes.



In a diagram a class will usually look like that.

If a symbol, that functions as another object’s class, is also referenced as an object, the symbol gets a double border, indicating its dual function as both an object and a class.



In fact it probably does not have a dual function, it is a class, but there are also *references* to the class (established with an object redirection to the class).

## Ideas

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