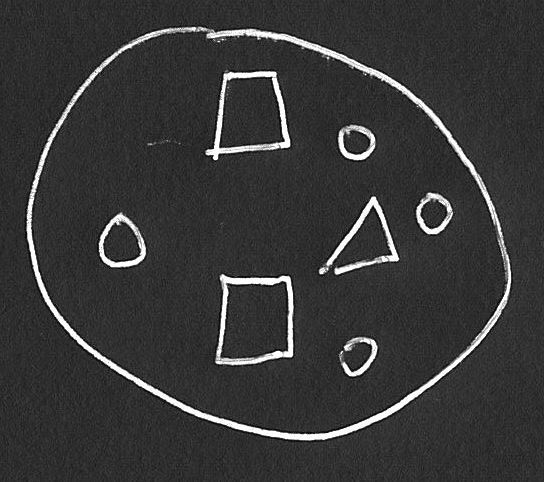
|  |
| --- |
| Circle Language Spec |

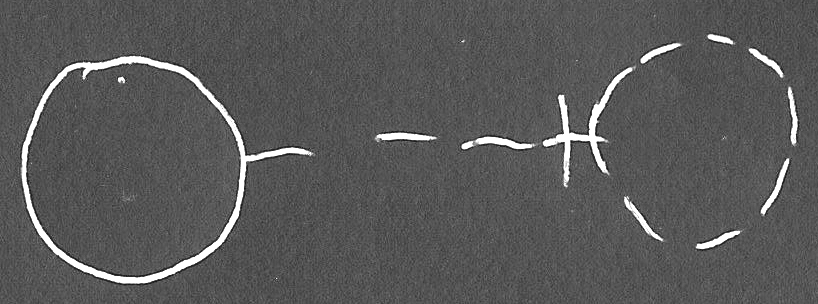
## Classes

### 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:*



The object on the left might point at its class on the right with a dashed line.

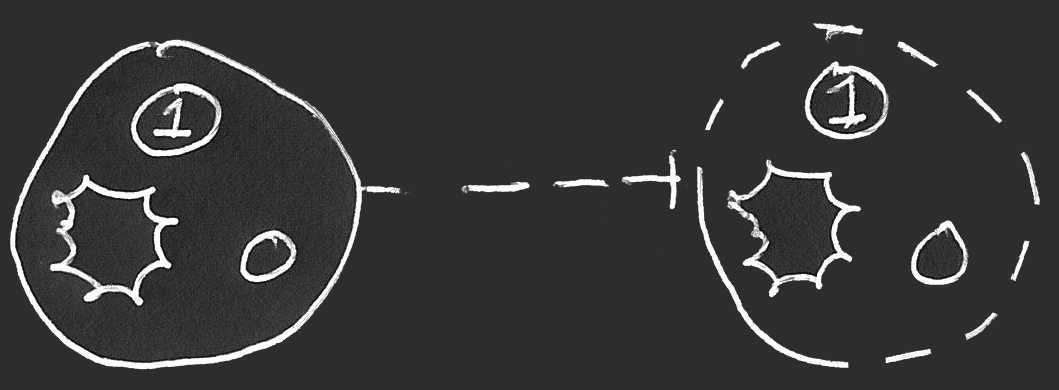
Then the contents of the object might not be that arbitrary:



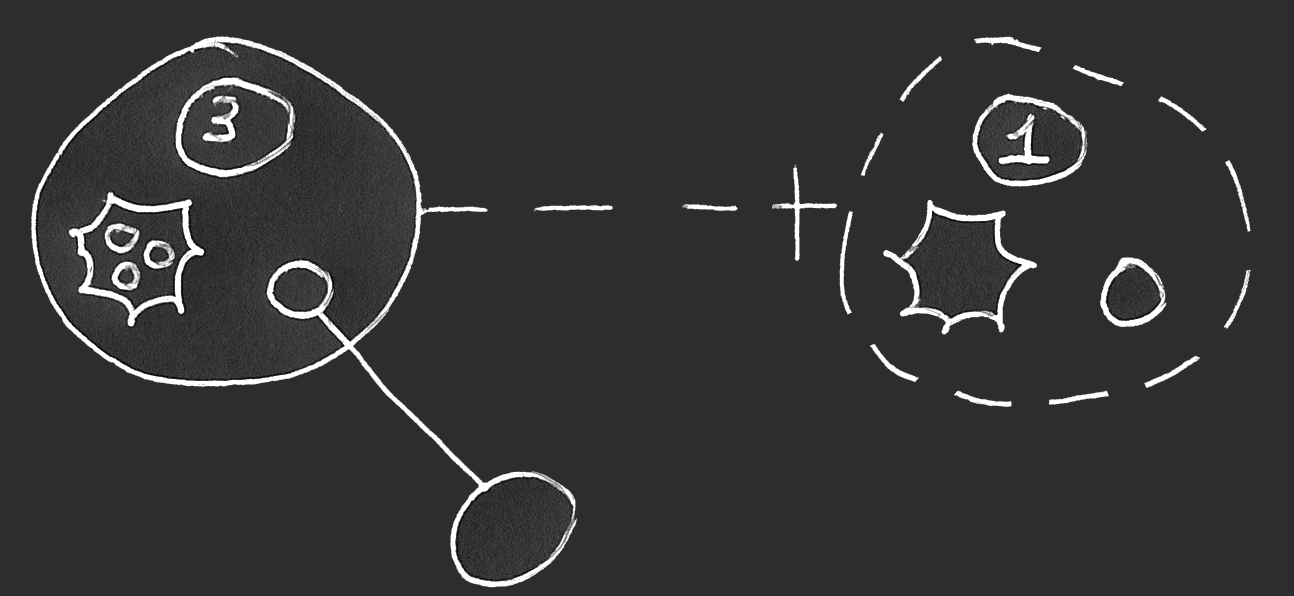
The object on the left may have similar contents as its class on the right. At first an object may contain related items and related lists that roughly corresponds with the class and the object might also have a similar set of commands as the class.

An idea would be that an object's behavior during its lifetime might be governed by rules set by this class. A class would be sort of like a special object, that aims to describe characteristics and behavior of other objects.

Initially an object may look like a replica of its class. Changeable parts of an object might be set initially to what would be defined in the class:



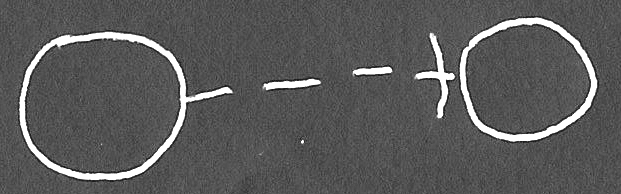
An object might have a similar *structure* as its class, but that object may have *data* that can change freely. Values of an object's attributes and might freely change as well as related objects:



While the structure of the larger circle at the right might seem somewhat similar to the contents of its class on the right, the data seems to have been changed.

### Class Redirection

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



The object on the left would have the class on the right. An object’s pointing out its class, could be called class redirection. The usage of the dashed line would have a specific meaning here. *Dashed lines* might symbolize the concept of *classes*.

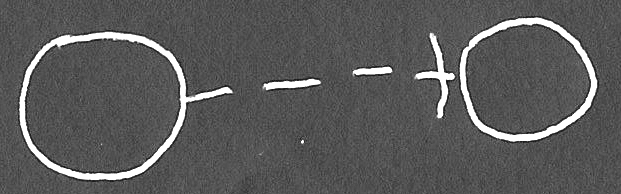
### Using Dashed Shapes

*Dashed shapes* might be used to denote classes:



But using dashed *shapes* might be optional.

Here an object would be used as another object’s class:

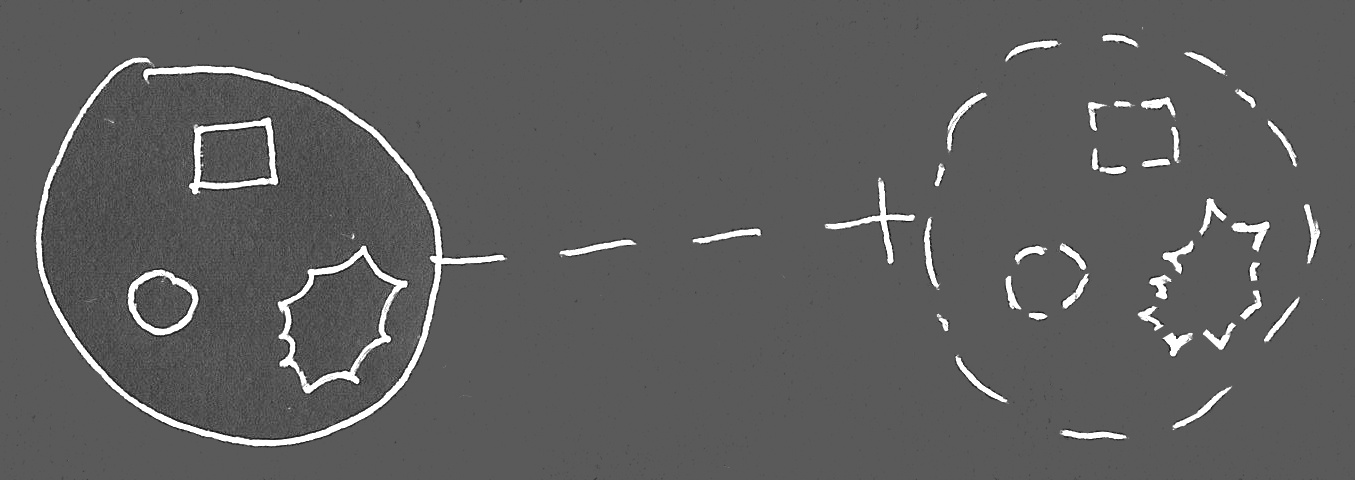


It might be an idea to draw the class on the right out with a dashed line:



But it might be optional.

Sometimes other parts of the static (or 'class') structure might be drawn with dashed lines too, to keep the static structure and object structure visually distinct.



But the rules do not seem to be that precise yet when it comes to shapes drawn with different line styles. The rules for pointing lines are a bit stricter: dashed would point out a class.

### Using an Object as a Class

Perhaps it might be common that an object would be fixed in its role as a prototype or class. This dashed shape notation might be useful as specific in meaning to indicate that it's *only* *usable* as a class or simply that it *is* a class and only a class.

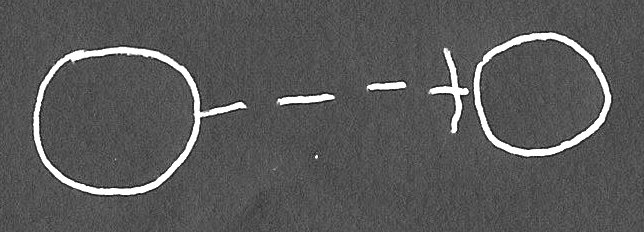
To express an object's (fixed) role as a class, a dashed line might be used to draw its symbol:



Here one including an object circle with a class redirection:

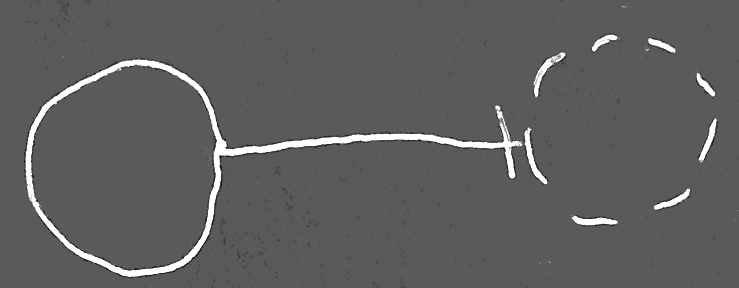


But the Circle notation itself would allow any object to serve as a class or prototype for another object.



### Using a Class Like an Object

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



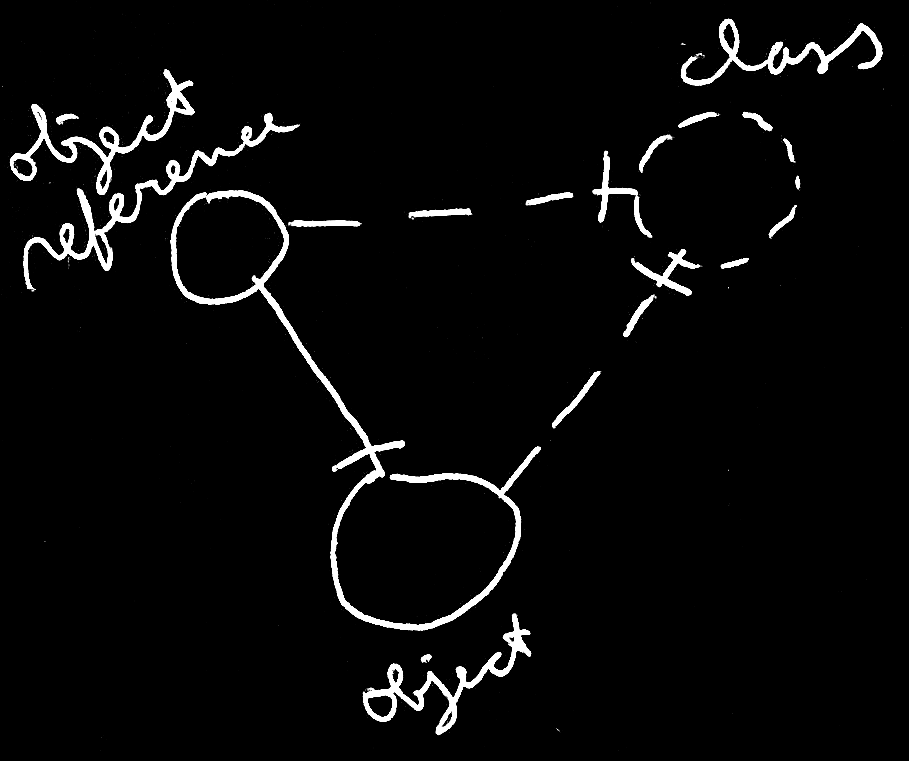
That the line in the middle would be *solid* might make it an object reference.

Or possibly this expresses it more clearly:



### Object Reference with a Class

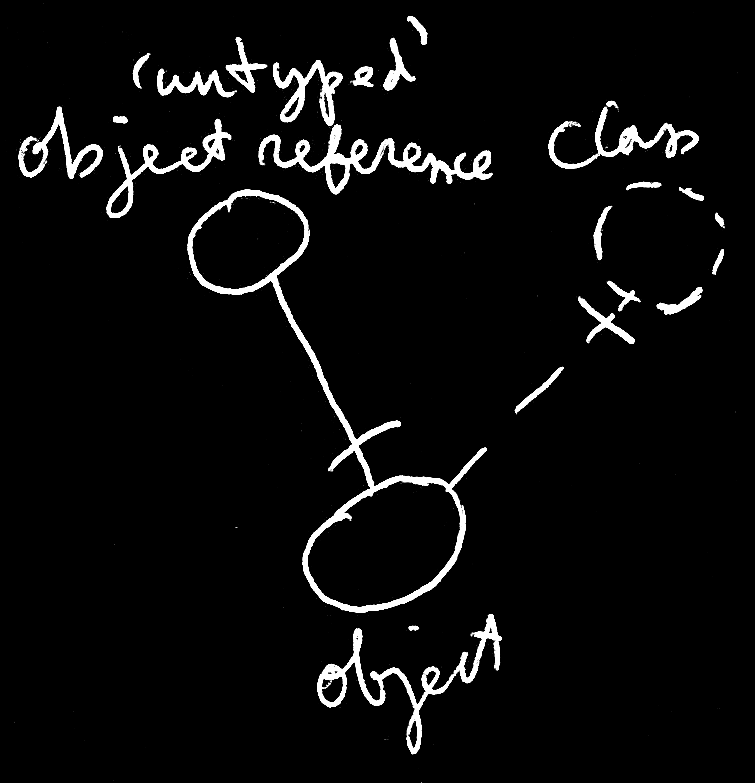
Next to an *object* having a class, an *object reference* might also have a class. Then only objects of that class might be assigned to it.



The object reference on the left would point out its class on the right. The object at the bottom would also point out its class on the right. The solid line would be a valid for the object reference to point to the object, since they seem to have the same class.

### 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.



The 'untyped' object reference at the top-left would be able to point to objects of any class. It would point to an object at the bottom, which seems to have a class, that the 'untyped' object reference may not explicitly point out.

One interpretation might be that an 'untyped' object reference could take on the class of an object it may point to. So indirectly it *would* have a class.

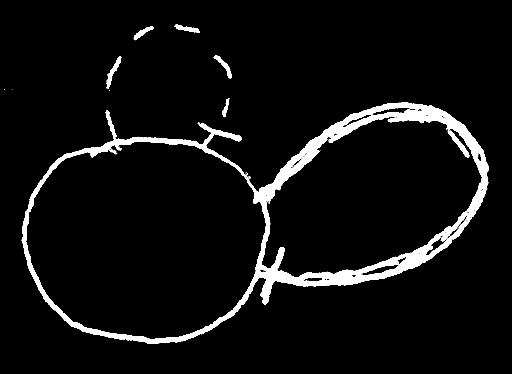
### No Class != Defines its own Class

Something might have been unclear at one point. 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.

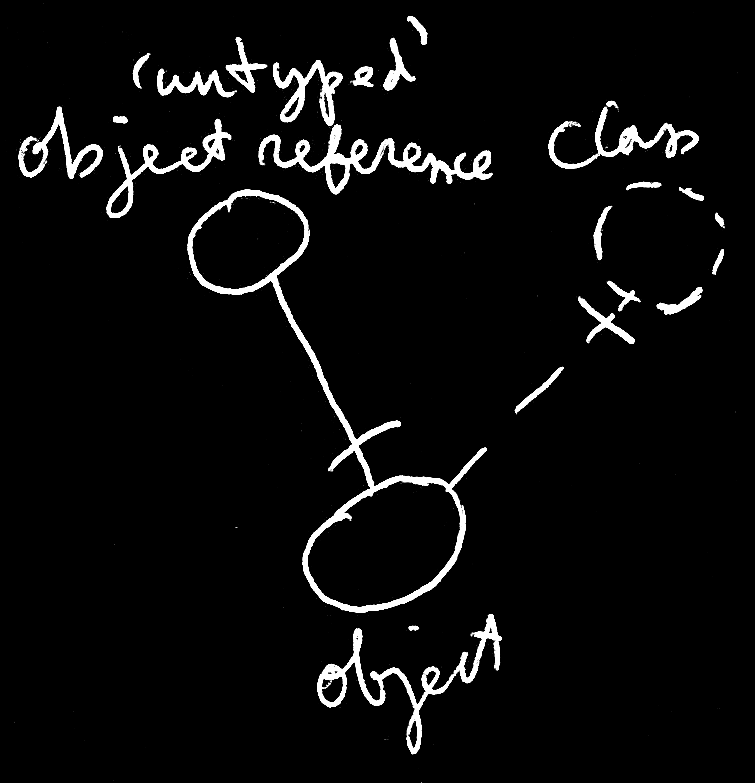
There may be examples where this might be a helpful concept.

### Object Reference would define its Own Class?

There may be other examples, 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?

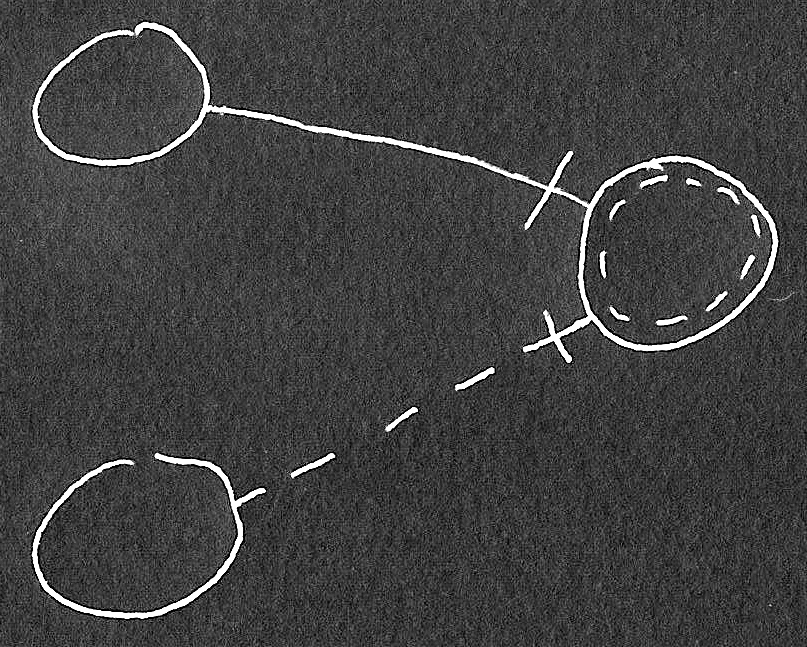


So instead of a 'stuck on itself' situation, it might be seen as an *arbitrariness* situation.

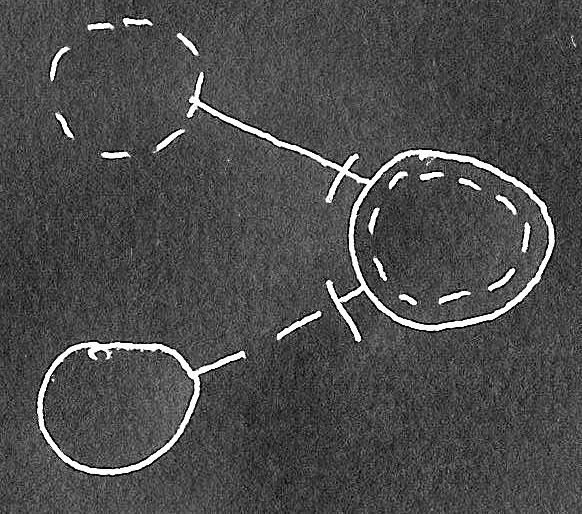


### Shape Both Dashed and Solid

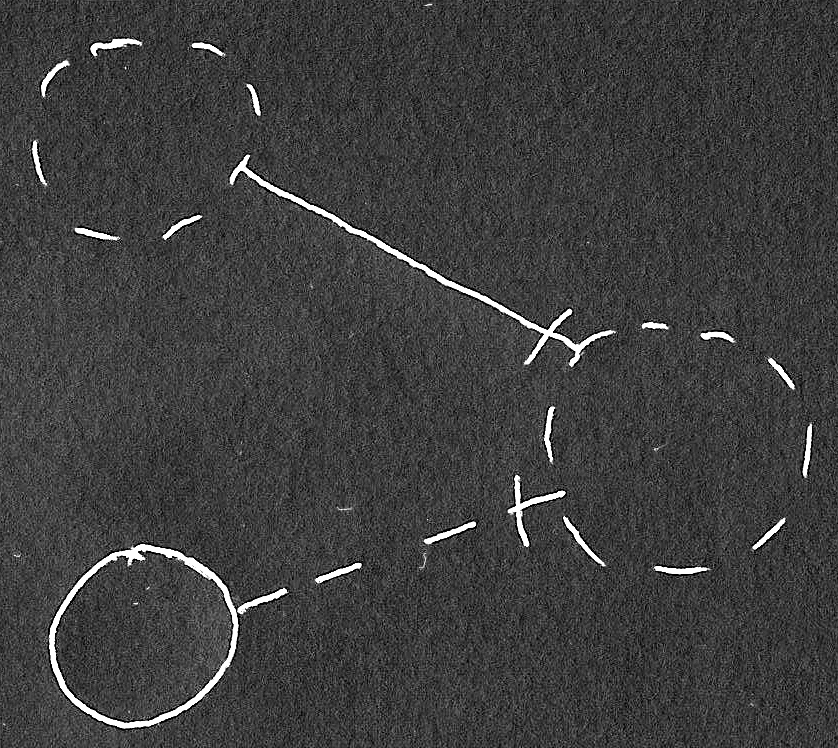
It may be an idea that if a symbol serves as another object’s class, but also would be referenced like an object, the symbol might get a double border to maybe indicate its dual role as both an object and a class.



But there might be different ideas about this. The top reference in the previous picture seems to be a reference to a class, so it might be replaced by a dashed symbol:



But perhaps just a dashed border for the shape on the right would be appropriate after all then, since its primary role seems to be a class.



Then it looks like it might make sense to draw the double-bordered shape with just a dashed border.

The optional nature of the dashed shapes seems to leave things open to discussion.

### Conclusion

Hopefully this would give an impression of how class notation might be used in the Circle language.