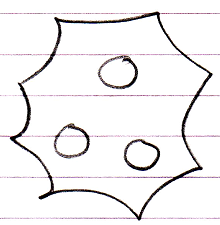
|  |
| --- |
| Circle Language Spec: System Objects |

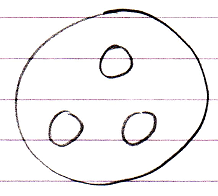
## List Concept

This article explains the list concept. A list is a special type of object, that contains zero, one, or more other objects. This concept collides with the concept of a normal object, because a normal object can also contain zero, one, or more other objects.

If you consider this list:



The following object seems to be equivalent:

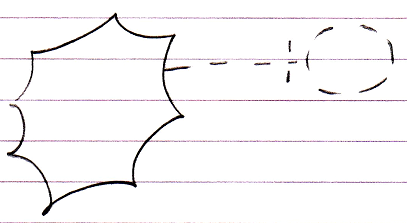


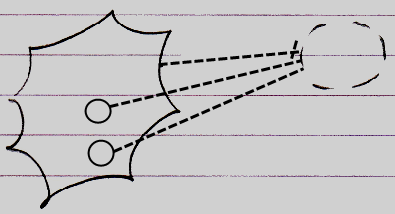
The differences between the two is in the way the sub-objects are controlled.

First of all the intention with the normal object is that the amount of members is constant. The intention with the list is that the amount of objects in it is variable. But if nothing is controlled, the object’s amount of members can also freely change.

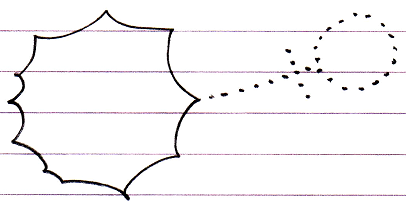
What is special about a list is that you can control the aspects of multiple objects at the same time: you set the rules that all the objects inside the list must comply with.

This basically means that you can collectively set the aspects of the objects. The most basic example is that you set the class for the list, and then all the objects inside the list must be of that class.



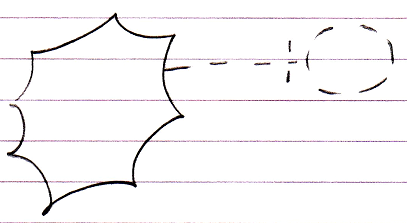


You can also set the interface for the list, and then all objects inside the list must either have that exact interface or support that interface.



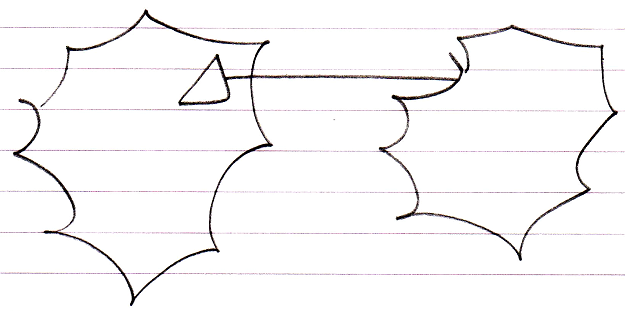
The set of aspects, system commands, connectors, connections, etcetera of a list is the same as that of a normal item, so you can control all aspects of the list items.

A list is an object like any other object. There can several classes of lists: a stack, a linked list, a flat list, etcetera. List objects can also for instance inherit from eachother. To set the list class of the list object itself can not have the same notation as setting which class the list *items* must have. So you can not use this notation:

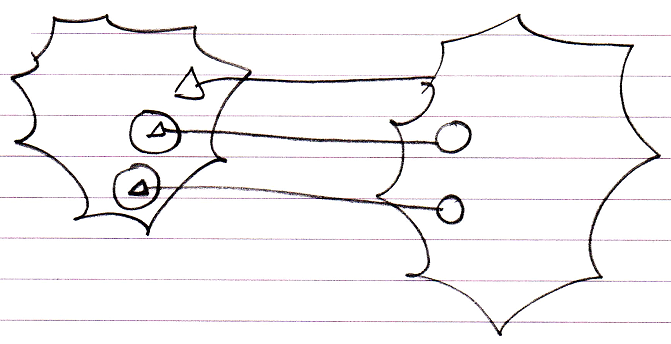


Because it conflicts with specifying the class that the list items should have, so this can not specify the class of the list object itself.

A triangle in the nonagon connected to another nonagon:

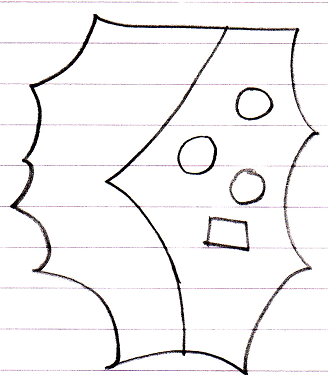


expresses that the list *items* inheriting from the list items in the other list, not that the list object itself inherits from another object.



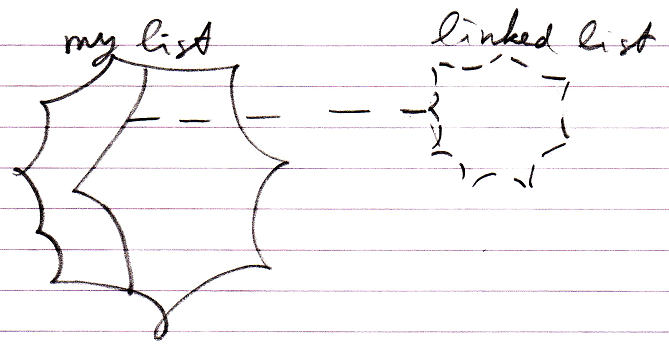
A triangle inside a nonagon will never express anything other than list inheritance as such.

The solution to setting aspects of the list object itself is as follows. The contents you normally see in the list symbol, in the nonagon, are not the actual members of the list object. They are an alternate view on the list object. You do not see the list object members, you see the list items. The actual members of the list object are hidden away as part of the *system* *object*. The system object members are visible inside the system interface:



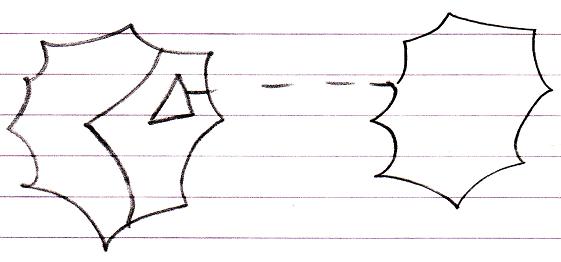
To specify the type of list you use is something special, which should get an alternate notation. Controlling aspects of the list object itself are actually considered the internal workings of the list system object. And internal workings are exposed through the system interface.

This means that when you want to control the list object itself, you have to do this by using the system interface:



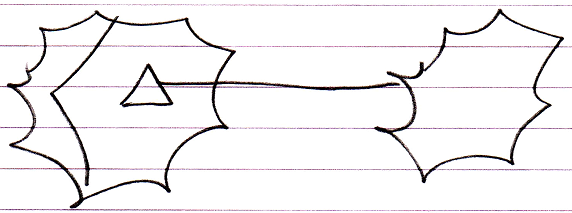
Below you will find other examples.

Here is a list class inheriting from another list class:



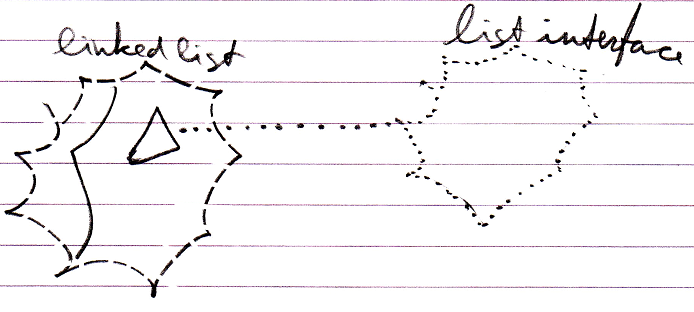
This is called list class inheritance.

A list object can also inherit from another list object:



This is called list object inheritance.

A list object can also take over the interface of another list, and specify its own implementation of that interface:



This is called list interface inheritance.

So the ‘normal’ appearance of the nonagon would be what is the most intuitive behavior of a list: not to express primarily that the list is also just an object: that is internal behavior, hidden away in the system interface. So the class line of a nonagon should express the class that the items should have.

Lastly, it is also possible to have a list of lists:

