|  |
| --- |
| Circle Language Spec: Inheritance |

## System Inheritance

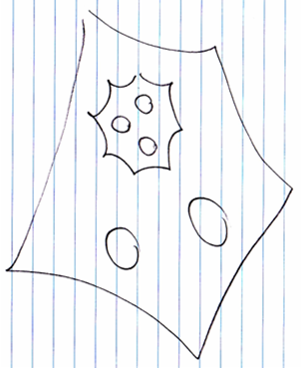
System inheritance introduces the concept of having one module with objects defined inside of it, and inheriting from that module, allowing the derived module to redefine any object defined in the base module.

You can be selective about which objects inside the derived module get specialized. This allows you to both specialize objects with a fixed logical residence inside the base module, as well as specialize objects that are mere references to objects defined totally elsewhere outside the module.

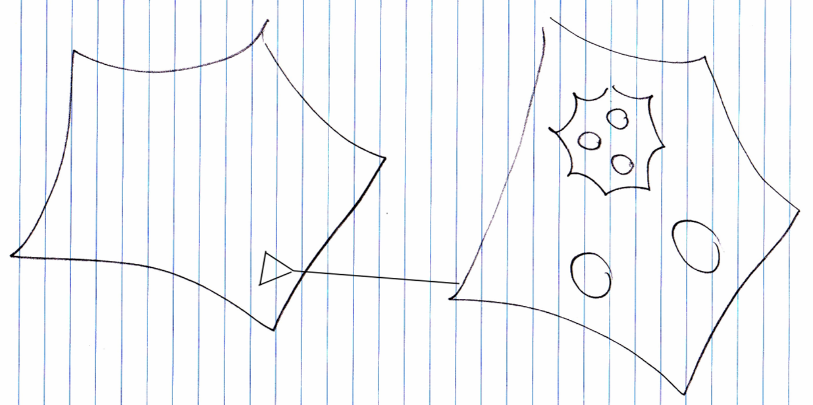
Objects in the derived module will inherit from their counterpart object in the base module. This will all go automatically.

Synonyms for system inheritance are module inheritance or a parallel structure.

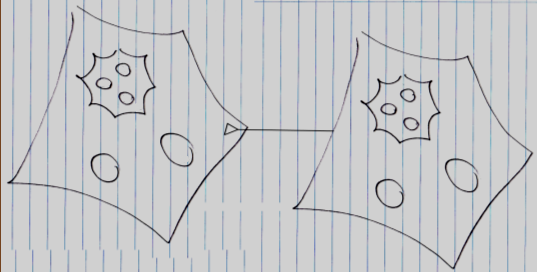
Here is an example system:



Here is another system inheriting from that system:



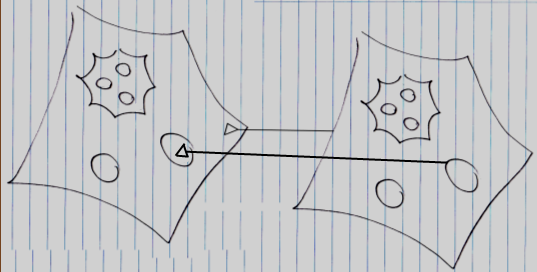
At first the derived system would just contain all objects from the base system



Already you can see a significant difference with normal inheritance: the derived members are not defined inside the triangle, but are defined outside the triangle.

At this point the exact same objects are present in both base system and derived system. This may have been pointed out in the diagram by lines connecting the references to the same objects together, but in the picture above this is left out of the picture.

In system inheritance you can simply pick out the sub-object you are going to specialize and create a derived object right then and there:

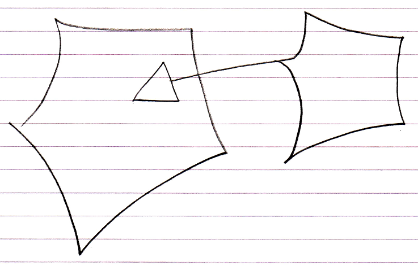


Now the derived object is not the same object as the object in the base system anymore, but the derived object is a new object, which has a triangular reference to the base object.

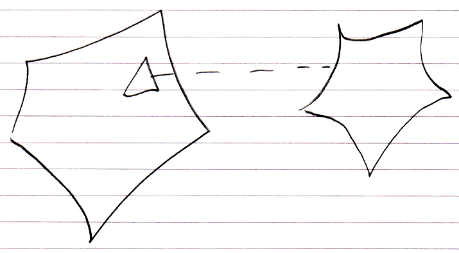
Something to consider though, is that when a much deeper object is specialized, all the levels that lead to the deeper object will also get a specialized version (for more information see: *Even Deeper Member Addition*).

Systems can inherit with either object inheritance or class inheritance.

*System object inheritance:*

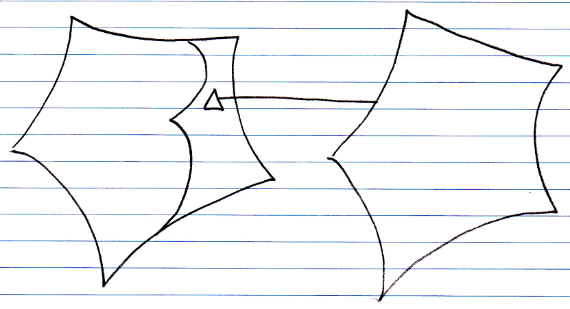


*System class inheritance:*



The difference between the two is completely analogus to normal object inheritance and class inheritance. In system *object* inheritance changing the values of base members will directly change the base system. You are working directly on the base system. In system *class* inheritance you can freely change the values of the base members without affecting the base system.

System inheritance is added behavior compared to normal inheritance, because any of the sub-objects can be redefined. Modules can also inherit from eachother like *normal* objects, but you have to do this using the system interface notation as follows:



You can then add or specialize members from the base system in a normal inheritance situation.