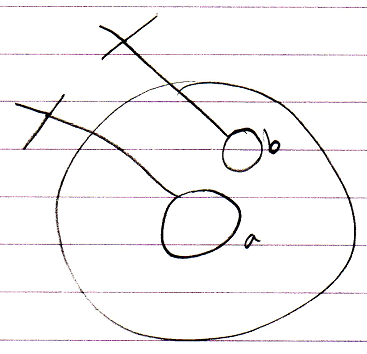
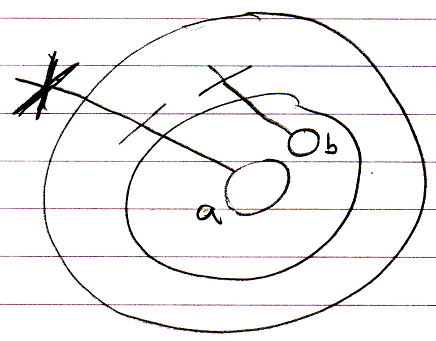
|  |
| --- |
| Circle Language Spec: Inheritance |

## Deeper Specialization

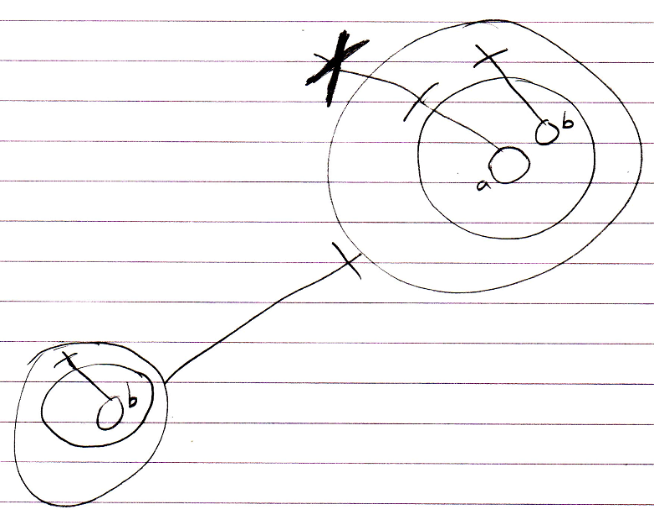
The main concept of deeper specialization is that you can not only specialize direct members of an object, but you can also alter and extend members from deeper objects. A simple example is to exclude a member of a sub-object. Here is a normal object with a couple of public sub-objects:



Here is the same object inside another object which excludes one of its members, making it Private:



This makes it possible to create a simplified set of members for the sub-objects.



It is the higher parent object that imposes restrictions to access to the deeper objects. So the higher parent object will store those restrictions for deeper objects in a tree structure of Deeper Specializations. Even though this is something quite special, it still has a quite simple notation.

### Other Specialization Techniques

Specialization of deeper objects also applies to the other specialization techniques:

- Altering the Member Set:

- Member Addition

- Member Exclusion (already covered)

- Member Inclusion

- Detouring Members:

- Overriding

- Shadowing

- Altering Command Implementations:

- Command Extension

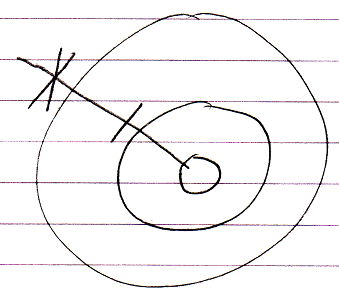
- System Command Extension

- Overriding System Commands

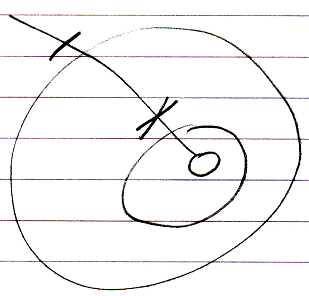
- Shadowing System Commands

These specialization techniques were already explained in the article *Specialization*. Here follow pictures of those situations in case of deeper specialization.

### Deeper Exclusion

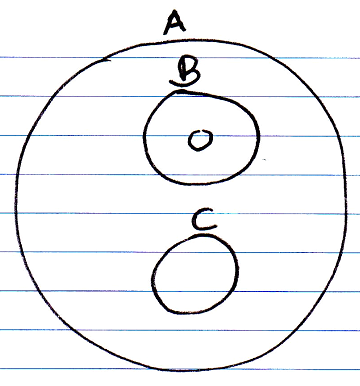


### Deeper Inclusion

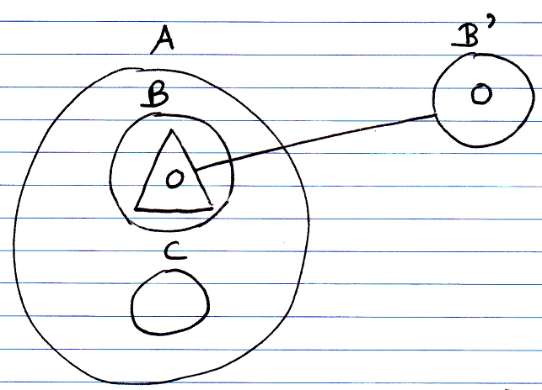


### Deeper Member Addition

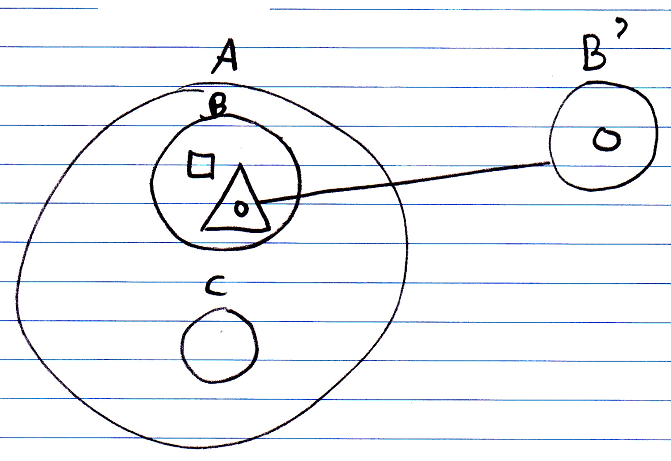
Given the following situation:



If you want to add a member to B that is only there when you access it through A, you replace B by a new object, and let it object-inherit from the original B:



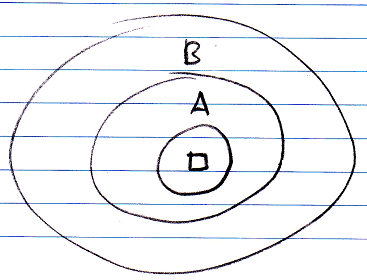
Now you can add any additional members to A . B:



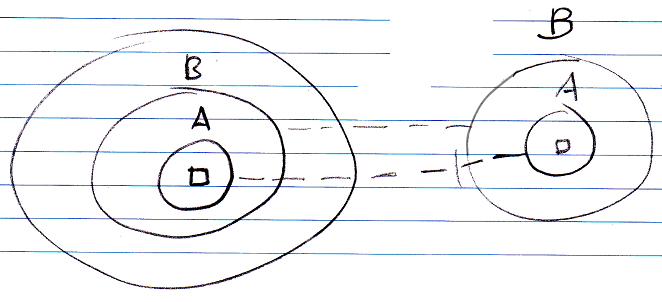
### Even Deeper Member Addition

However, if the additional member must be added to an even deeper level, each level must be given a specialized version.

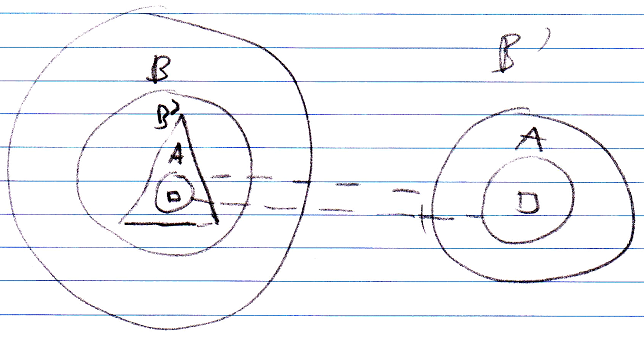
Take the following structure:



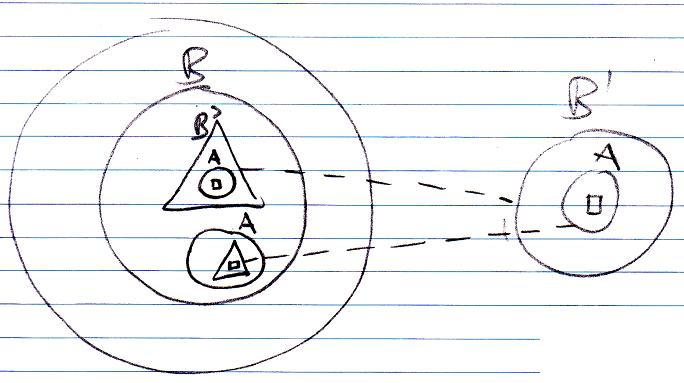
We want to add an extra member to A. However, we can not add them to object A itself, because the members A are defined inside an external class:



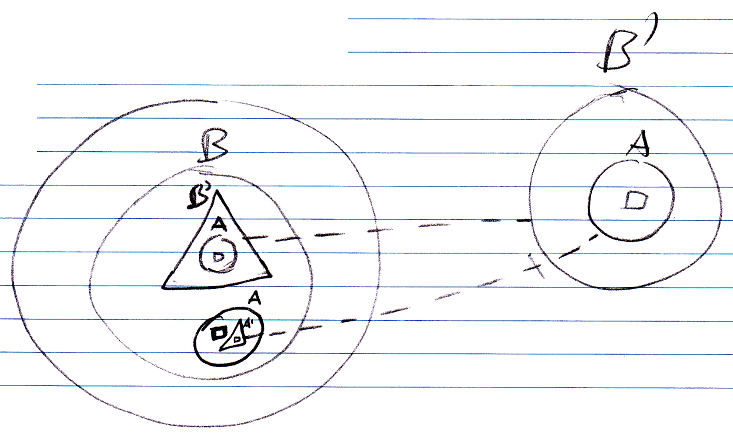
We have to specialize each level that leads to object A, so first we specialize object B by adding a new object B giving it the original object B as a base:



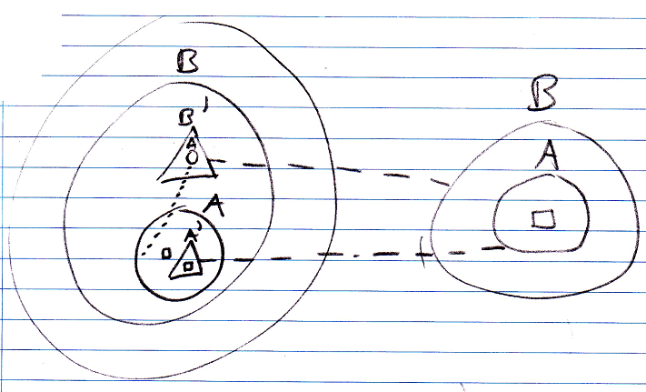
Then we do the same for object A: we add a new object A to B, that has the original object A as a base:



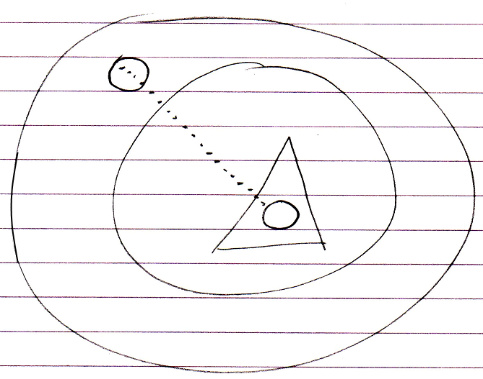
Now we can add members to A without changing the configuration of base members:



There is one more thing we have to do. There are now two members **A** inside object **B**. We point out that the new object **A** shadows the old object **A** by means of a shading line:

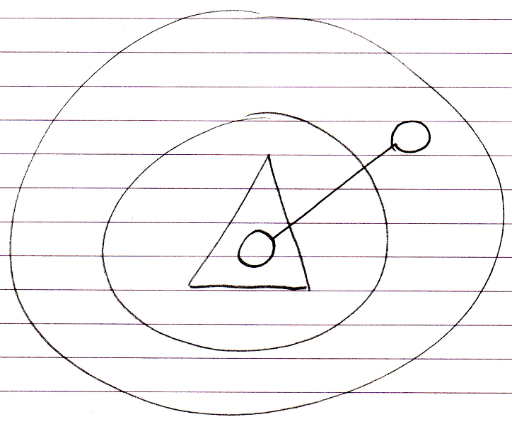


### Deeper Shadowing

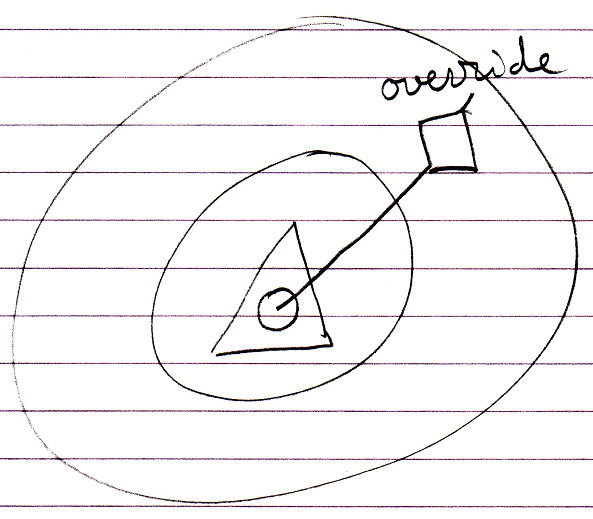


### Deeper Overriding

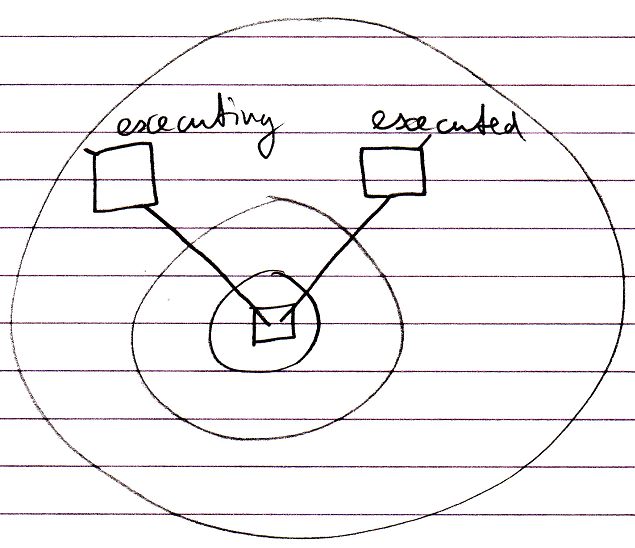
*Object reference notation:*

**

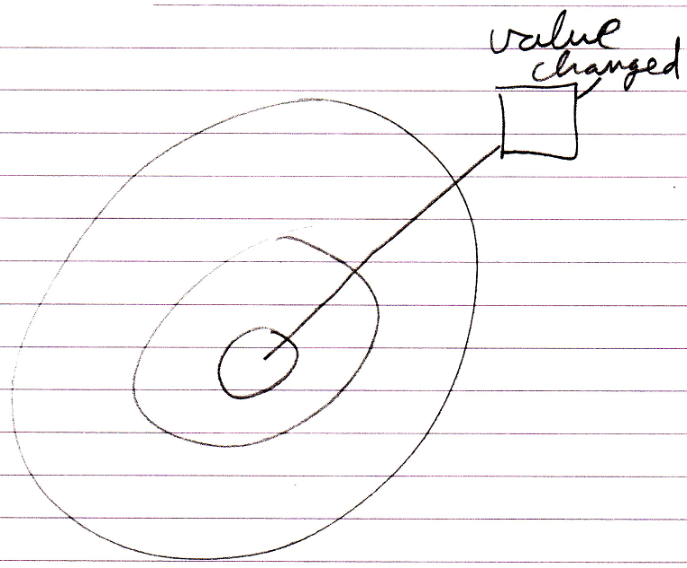
*Event notation:*

**

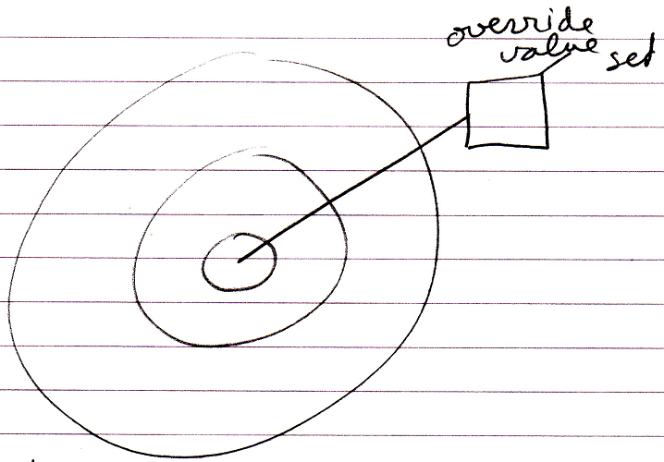
### Deeper Command Extension



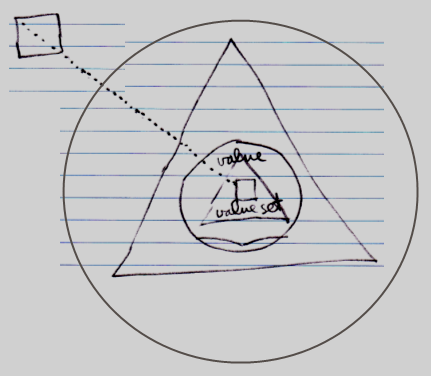
### Deeper System Command Extension



### Deeper System Command Overriding

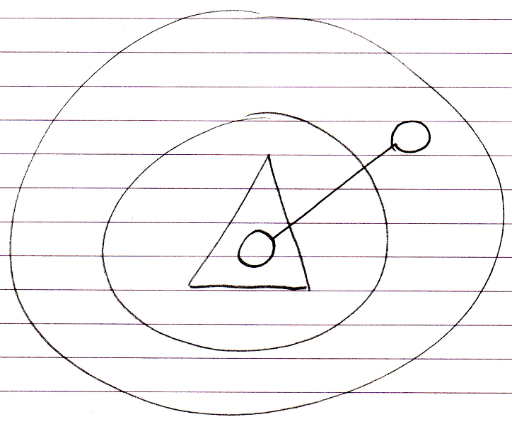


### Deeper System Command Shadowing



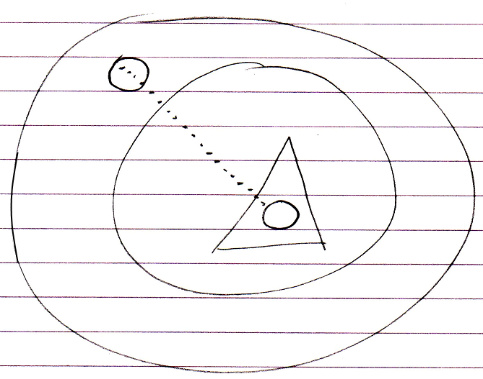
### The Deeper Specializations Structure

Some deeper specializations are registered inside the deeper objects themselves, such as command extensions and overriding. Here is overriding as an example:

**

The override is an object situated in the outer parent. The overridden member is a reference to the override, that resides resides inside the deeper object.

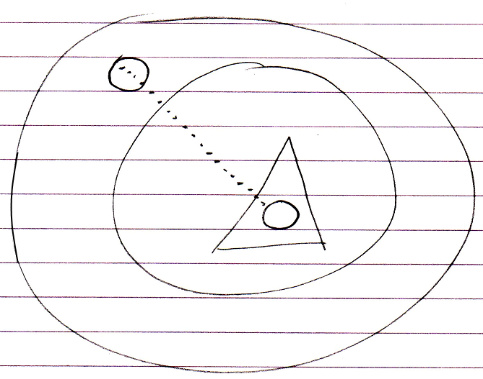
Some deeper specializations though, are not registered inside the deeper objects themselves. This is the case for member exclusions, member inclusion and shadowing. Here is shadowing as an example:



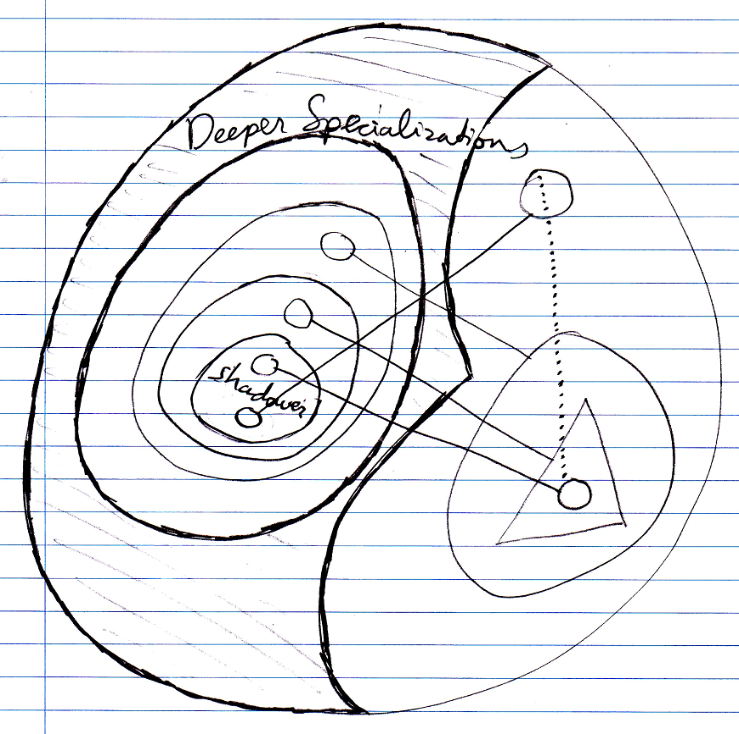
The shadowing of the deeper member is registered inside the outer parent. It would seem easier to just register a shadower inside the deeper object itself, but the problem is that the shadowing is specific to accessing the deeper object through the outer parent that defines the shadow. Through any other path, the deeper member is not shadowed.

To store shadowings, exclusions and inclusions, the outer parent contains a Deeper Specializations structure, which is a parallel structure, parallel to the object structure inside the parent.

You can not *see* the Deeper Specializations object in the diagram, but you do see its effect, because the specializations types each have a specific diagram notation.



The Deeper Specializations structure is part of the system object, so the inner workings of an object. You get to see the inner workings of an object by opening up the interface:



Not all sub-objects are represented inside the Deeper Specializations structure. Only as soon as a member gets specialized in a way that requires the Deeper Specializations structure, the member gets a representation inside the Deeper Specializations object. The exact implementation however, will depend on what offers best performance and efficiency.