|  |
| --- |
| Circle Language Spec: Black Boxes |

## Black Box Miscellaneous Issues

In previous projects the main articles about black boxing were finished, but other subjects were left unfinished. This article contains the unfinished material of some of these topics.

This article contains texts and loose ideas about the following topics:

##### Deeper Exclusion

A parent object’s restricting access to specific members of their children.

##### Protected

Accessibility specific to inheritance situations.

##### Internal

Accessibility solely within the module or package itself.

##### Objects Take Over Class Access Control

Objects take over the access control properties of their class.

As said, the texts about these subjects are not finished yet.

#### Deeper Exclusion

Apart from controlling access to the type’s own procedures, a type can also control publicity of procedures *of its chilren*. That way you can publish a child, but make some of its public procedures inaccessible outside the parent. This way you can also exclude members of a base type. A parent can’t make a child’s procedure inaccessible to the child itself; it’s the child’s own procedures.

You can also exclude members of ancestors even further down the hierarchy.

Forms of exclusion other than Private are just called exclusion or privatising.

You can only make a procedure of a child *less* accesible, never *more* accessible. The parent can’t publish a child’s procedure that the child defines as Private. A child can’t put restrictions on how its members are privatised by the parent, so a parent has full control over the exclusion of the procedures its ancestors.

Exclusion access controllers are also called *access* *modifiers*, because they modify formerly set access controllers.

Inheritance, Exclusion,

2008-10-15

This is also a blackboxing thing...

... but I put it in inheritance.

JJ

Inheritance, Exclusion,

2008-10-15

Because it is complexity hiding,

it is blackboxing.

JJ

Exclusion,

20008-10-16

But it is just access control,

so it is blackboxing, not inheritance.

You can apply access control combined with inheritance, to further specialize a class, object or system.

JJ

Exclusion... so a grandparent imposing extra access control onto its grandchildren...

Another advanced topic to cover at the end.

Gee, Public and Private are already a form of exclusion,

because the parent object is imposing restrictions to access

to its sub-objects ( the references it contains. )

When you make it private, it means that it is only accessible to friends...

hmmm...

Deeper exclusion protects members of sub-objects,

but how much can those sub-object members be protected, if they might

also be referenced elsewhere. Somehow they have to be protected from

being referenced elsewhere... fixed logical residence?

What if you reference the object from elsewhere?

Somehow it needs to be controlled, that an object is always referenced

THROUGH the object that protects it...

There is a link with user access control here, but I won't go there yet.

Yeah, there is a discrepancy between controlling access and the fact that

containment is now a volatile thing. Fixed containment needs to be

imposed in order to establish access control...

Multiple objects can impose another access modifier upon the same object,

so where are all these access modifiers stored?

At first it will only be access modification of a direct reference,

which is already under control of the parent object.

But for deeper objects it is different...

so I am not sure yet... are those deeper objects supposed to

be exclusively contained by the grandparent object?

Or can they also be referenced elsewhere,

so that the grandparent has to remember the grandchild access modification

settings and not just delegate that to the child, that holds the grandchild?

I'm not sure.

Perhaps details like that should be worked out when actually turning

this language into an actual programming environment...

and we should keep the focus on the general idea?

Private, Public is exclusion

< Public and Private are an application of exclusion. So then the base of the explanations can be only accessible and inaccessible. >

Exclusion is the primary method for access control. Friends is the method to make exceptions to the privatisation imposed by exclusion. A procedure or type can make any other procedure or type they can reach a Friend, meaning that that procedure or that type has access to all members privately accessible in the befriended.

> And cover deeper access as well.

> Also cover how a fixed logical residence allows parents to impose a permanent, non-volatile access control. Qualified access to an object without it being a fixed logical residence does not guarantee access control imposed by the qualifiers. But if it is a fixed logical residence, the qualifier is mandatory and you can not go around the access control, that it imposes.

#### Protected

There is another basic access controller in addition to Accessible and Inaccessible: *Protected*.



Protected (By the way: it’s not an arrow.)

When a procedure is *Protected*, it means that it’s inaccessible if contained in a circle, but accessible if contained in a triangle.

|  |  |  |
| --- | --- | --- |
|  | ≈ |  |
|  | ≈ |  |

A circular descendant automatically privatises a Protected member:

|  |  |  |
| --- | --- | --- |
|  | ≈ |  |

The member isn’t accessible to any higher symbols than the first circular descendant.

Formally the choice of making an extra Protected access controller seems totally arbitrary. In practice there are situations in which it’s useful to work with members that can be accessible to the interface holder, but not accessible to the outside.

~~Protected is used in exclusion only. It can not be used for ‘outcommenting’ code.~~

A triangular mark:



Is used to indicate that a symbol that owns the line is Protected, meaning, that it is only accessible when the symbol that owns the line is a triangle. A triangular mark is not an arrow.

And then there is Protected. A funny one there. It should be an extra issue, explained after everything else is explained, because it works the same way, except that the befriended object should be referenced with a triangle / should be the friend object’s base object.

Private & Public,

2008-06-10

Protected:

Something needs to be done with protected.

JJ

Being a friend grants full access to the private and protected members of the befriended.

Interfaces,

2009-06-28

do not forget that the protected access modifier can not only

be used for object access, but also for class access and interface access.

(Class Get, Interface Get, etc.)

JJ

#### Internal

< Also called Friend in VB6 >

< better term: internal >

<< I have to note the application of the Visual Basic Friend access modifier. The basic access controller Friend isn’t required (Public Friend and Protected Friend). Friend Basic Access Controller means that you make a procedure’s own globality a Friend globality.

Yes, but for the effect of the Visual Basic Friend access modifier you also need to deprive global members from being accessed outside the globality. And that is done with globality level restriction.

So the effect of the Visual Basic Friend access controller is replaced by • making the globality a friend of the procedure and level restricting global procedures.

NO! It’s just making it Globality Up 1! Nothing Else! Do not use a Friend Globality for specific procedures!

>>

Friend for reference

Means access only anywhere within the globality, but not inside embedded globalities.

< 2008-10-10 That was the VB meaning of Friend. >

A procedure can contain variable objects. The public ones are parameters. The friend ones are parameters within the current globality.

< Old VB meaning of Friend >

< What place the VB6 keyword Friend has in black boxing, I do not know. this is a detail even less prominent, than Protected, which is also a detail later discussed. VB6 keyword friend has to do with access from within globalities. >

#### Objects Take Over Class Access Control

< And references take over object-access control. >

… is objects taking over friend relations from classes as straightforeward as you’d think?

When it is classes, the friendship counts for all the objects derived from that class.

Usually, access to procedures is controlled in a *class* and each of its objects adopts those same access settings.