|  |
| --- |
| Circle Language Spec: Interfaces |

## Interfaces Loose Ideas

### Loose Ideas about Relations Through Interfaces

Relations Through Interfaces,

2010-05-07

There is one thing I don’t get: if a class relates to an interface, how does the interface know it is a class? The class is no more than an object, really. That it is a class depends on its being used as a prototype for other objects.

I bet that if you think about it long enough or if you see it in practice in a prototype version the solution will present itself.

JJ

Relations Through Interfaces,

2010-05-08

The solution is that a relation between the a class and an interface is actually no class or interface relation between a class and an interface at all: it is an object relation between two objects.

Wait, no, that is not the solution, they may be just two objects, but they do have a class and interface relation with each other to fix which *kinds* of objects instances can link to.

JJ

### Loose Ideas about Target Interfaces

*The texts below are loose ideas yet to be turned into good documentation.*

Targets,

2010-05

> Now my mind says: follow all redirections, including multiple interface redirections… but in the Target Class story I stopped doing that. Maybe it is just what you want the term Target Interface to define.

> Perhaps there should be a distinction between interface definition and target interface. I do feel that both the ‘follow only one class or interface step’ version is a concept to be aware of, but the target interface concept would actually be following all redirections to find the object that actually determines the publics.

> Yes. What is now called Target Interface should probably be called the *Interface Object* and the *Target Interface* is the object after following all types of redirections in any order.

JJ

#### Out of the original Symbol documentation

##### Tracing Object Aspects

…

The same way, type lines can redirect the interface. Object lines can redirect the type and the type can redirect the interface, so an object line can also direct the interface.

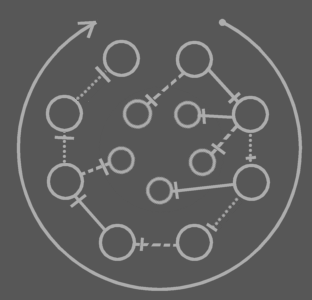
###### Interface Trace

You’ll use all three line types to trace the interface. Follow the interface line if it exists, else follow the object line, else the type line. When you run into a symbol with no lines, then that’s the interface.

A special thing about an interface line is that you can select a single interface of a type and not use the whole type.

When there’s no interface line and the type line is followed, then the type’s *whole* interface is used.

The reason that the object line is preferred over the type line is that if there is no interface line, the *object* can determine *which* interface of the typeis used.



This picture shows any combination of lines an object symbol can have and which of the lines to follow in the interface trace.

In other words: follow interface line, otherwise object line, otherwise type line. The final symbol altogether is the interface.

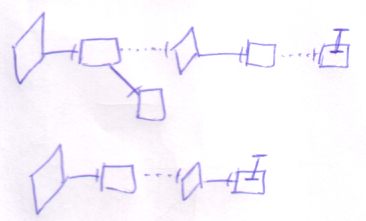
###### Object-Type-Interface Trace

It happens frequently that you need a symbol’s target object, type and interface in one blow. That requires two traces: an object-type trace and an interface trace. The full operation is an *object-type-interface trace*. It deserves a separate name for it’s a very common operation.

##### Tracing Procedure Aspects

###### Procedure Interface Trace

In the *procedure interface trace* you follow both line types, but you prefer following an interface line over a reference line.



The last symbol of the trace is the interface.

Following any of the two line types will eventually lead to the same symbol. In that sense it doesn’t matter what line type you prefer to follow. The interface line is preferred, though, both because it’s more logical to follow, and because following them you’ll probably hit the interface sooner: probably there will be less interface redirections than reference redirections.

###### Execution-Definition-Interface Trace

It happens frequently that you need target execution, definition and interface in one blow. That requires two traces: an execution-definition trace and an interface trace. The full operation is called an e*xecution-definition-interface trace*. It deserves a separate name for it’s a very common operation.

###### Type Line Being called Interface Line

The term *interface target* can mean the target of the *interface line*, but there’s also a second meaning. When the interface line is missing, the interface can also be redirected by an object line or type line. The term *interface target* can also denote the symbol to which the interface is redirected, *be it* with an object line or type line. Sometimes the word *interface line* is even used where I really talk about the line that *redirects* the interface.

The same thing applies to the term *type target*. *Type target* can mean the target of the type line, but it can also mean the symbol to which the type is redirected, be it with an object line, be it with a type line. Sometimes the word *type line* is even used where I really talk about the line that *redirects* the type.

It’s the same thing with the redirection of other system aspects.

### Loose Ideas about Interfaces

The interfaces an object supports are interesting. It’s also interesting to know if an interface is supported by an object. Of course you could derive that from the first data, but that’s not the fastest.

You should be able to group symbols inside a special symbol that controls the access for all the directly contained symbols. <2010-05-07 Just use a triangle and access control the triangle.>