Attribute Protection

Simula Standard sier dette:

The protection specification makes it possible to restrict the visibility of class attribute identifiers.

A class attribute, X, which is specified ***protected*** in class C is only visible:

1 - within the body of C or its subclasses
2 - within blocks prefixed by C or any subclass of C.

In any other context the meaning of the identifier X is as if the attribute definition of X were absent.

Access to a protected attribute is, subject to the restriction above, legal by remote accessing. (hva betyr dette ?)

A class attribute may be specified protected only at the prefix level of its definition. Note that a virtual attribute may only be specified protected in the class heading in which the virtual specification occurs.

Attributes of the classes Simset and Simulation are protected. (Alle ?)

A visible class attribute, X, specified ***hidden*** in class C is not visible within subclasses of C or blocks prefixed by C or any subclass of C. In this context the meaning of the identifier X is as if the attribute definition of X were absent.

Only a protected attribute may be specified hidden. However, this specification may occur at a prefix level inner to the protected specification.

The effect of specifying an attribute hidden protected or protected hidden is identical to that of specifying it as both protected and hidden.

Conflicting or illegal hidden and/or protected specifications constitute a error.

Note: Specifying a virtual quantity hidden effectively disables further matching at inner levels.

If in the prefix sequence there are several attributes with the same identifier as that of a hidden specification, and these are previously protected, but not hidden, the innermost accessible attribute is hidden.

Attribute Protection (Hidden Protected)

Jeg fant dette eksempelet i TestBatchen til Simula as.

```
01:   Class A;
02:   Begin
03:      integer i;
04:   end;
05:
06:   A Class B; Protected i;
07:   Begin
08:      integer i;
09:      procedure p; xb.i := 17;
10:   end;
11:
12:   ref(B) xb;
13:   xb :- new B;
14:
15:   xb.p;
16
17:   xb.i := 5;
18:
19:   inspect xb do i := 7;
```

Siden 'i' er Protected i klassen B vil den være usynlig ved remote access og ved inspection. Jeg tolker ihvertfall slik.

Alle tre eksemplene (linje 15-19) vil da aksessere 'i'en til A siden 'i'en til B er usynlig gjennom remote access.

Imidlertid; et av testprogrammene til Simula as. tolker linje 09 annerledes. Der forventes det at det er 'i'en til B som aksesseres. Er dette årsaken til den rare setningen ?

Er det noe jeg har misforstått ?

Algoritme: Find remote attribute

```
//*****************************************************************************
// *** Utility: findRemoteAttributeMeaning
//*****************************************************************************
public Meaning findRemoteAttributeMeaning(String ident)
{return(findRemoteAttributeMeaning(ident,false)); }

public Meaning findRemoteAttributeMeaning(String ident,boolean behindProtected)
{ boolean prtected=false;
  for(String prct:protectedList)
     if(ident.equalsIgnoreCase(prct)) { behindProtected=prtected=true; break; }
  if(!prtected)
  { for(Parameter parameter:parameterList)
      if(ident.equalsIgnoreCase(parameter.identifier))
          return(new Meaning(VariableKind.parameter,parameter,this,this,behindProtected));
    for(Declaration declaration:declarationList)
        if(ident.equalsIgnoreCase(declaration.identifier))
          return(new Meaning(VariableKind.attribute,declaration,this,this,behindProtected));
    for(LabelDeclaration label:labelList)
        if(ident.equalsIgnoreCase(label.identifier))
          return(new Meaning(VariableKind.label,label,this,this,behindProtected));
    for(Virtual virtual:virtualList)
        if(ident.equalsIgnoreCase(virtual.identifier))
          return(new Meaning(VariableKind.virtual,virtual,this,this,behindProtected));
  }
  BlockDeclaration prfx=getPrefix();
  if(prfx!=null)
  { Meaning meaning=prfx.findRemoteAttributeMeaning(ident,behindProtected);
    if(meaning!=null) meaning.declaredIn=this;
    return(meaning);
  }
  return(null);
}
```

Hvor klassen Meaning ser slik ut:

```
public class Meaning {
       public VariableKind variableKind;
       public boolean foundBehindInvisible;
       public Declaration declaredAs;
       public DeclarationScope declaredIn; // Search started here
       public DeclarationScope foundIn;    // Search ended here

        ... ...
}
```

I tilfellet Connection Block blir det litt annerledes. Vi må rette opp 'meaning' for å reflektere en Connection Block.

```
public Meaning findMeaning(String identifier)
{    Meaning result=classDeclaration.findRemoteAttributeMeaning(identifier);
     if(result!=null)
         result=new Meaning(VariableKind.connectedAttribute
                            ,result.declaredAs,this,result.foundIn
                            ,result.foundBehindProtected);
     else if(declaredIn!=null) result=declaredIn.findMeaning(identifier);
     if(result==null) Util.error("Undefined variable: "+identifier);
     return(result);
}
```

En mulig tolkning av den rare setningen om protection er at protection er unntatt hvis vi er inne i klassen det gjelder.

Jeg har definert en funksjon som kan avgjøre slikt:

```
private static boolean withinScope(DeclarationScope otherScope)
{ DeclarationScope scope=Global.currentScope;
    while(scope!=null)
    { if(scope==otherScope) return(true);
      BlockDeclaration prfx=scope.getPrefix();
      while(prfx!=null)
      { if(prfx==otherScope) return(true);
        prfx=prfx.getPrefix();
      }
      scope=scope.declaredIn;
    }
    return(false);
}
```

Vi går oppover i block-strukturen og sjekker om vi finner 'otherScope' - hvis så er vi 'inne i' 'otherScope'

Da kan vi gjøre denne endringen i starten av funksjonen

```
public Meaning findRemoteAttributeMeaning(String ident,boolean behindProtected)
{   boolean prtected=false;
    for(String prct:protectedList)
     if(ident.equalsIgnoreCase(prct))
         { if(!withinScope(this)) behindProtected=prtected=true;
           break;
     }
    if(!prtected)

    ...

}
```

Dermed virket alle testeksemplene til Simula as.

Se nå på dette eksemplet:

```
01:   Class A;
02:   begin
03:     integer i;
04:   end;
05:
06:   A Class B; Protected i;
07:   begin
08:     integer i;
09:   end;
10:
11:   B Class C; Hidden i;
12:   begin
13:     ...
14:   end;
15:
16:   C begin
17:
18:     i := 34;
19:
20:   end;
```

'i'en på linje 18 refererer seg til 'i'en i klassen A.

Fordi 'i' er Hidden i blokker prefixed med C.

Det er 'i'en i B som blir Hidden i C.

Altså er 'i'en i A synlig bak en usynlig 'i'.

Jeg har definert to hjelpefunksjoner:

1) DeclarationScope getHidden(String ident)

2) DeclarationScope getScopeBehindHidden(String ident)

Begge lokale til DeclarationScope.

GetHidden Search backwards through prefix chain
- find hidden then return the scope where it was specified hidden.
- If no HIDDEN was found - return null

GetScopeBehindHidden Search for protected backwards through prefix chain starting with this scope
- find corresponding protected, then return the scope prefix to where it was specified protected.
- If no PROTECTED was found - Runtime Error

Full Definition:

```
public DeclarationScope getHidden(String ident) {
    BlockDeclaration prfx=getPrefix();
    if(prfx==null) return(null);
    for(String hdn:prfx.hiddenList)
        if(ident.equalsIgnoreCase(hdn))
            return(prfx);
    return(prfx.getHidden(ident));
}


private DeclarationScope getScopeBehindHidden(String ident) {
    DeclarationScope scope=this;
    while(scope!=null) {
        BlockDeclaration prfx=scope.getPrefix();
        for(String ptc:scope.protectedList)
            if(ident.equalsIgnoreCase(ptc)) return(prfx);
        scope=prfx;
    }
    throw new RuntimeException(ident+
                " is specified HIDDEN without being PROTECTED");
}
```

Da får vi .... søker innvendig

```
public Meaning findVisibleAttributeMeaning(String ident) {
      boolean searchBehindHidden=false;
      DeclarationScope scope=this;
      SEARCH:while(scope!=null) {

         for(Parameter parameter:scope.parameterList) {
            if(ident.equalsIgnoreCase(parameter.identifier))
                DeclarationScope hiddenScope=scope.getHidden(ident);
                if(hiddenScope==null) return(new
                         Meaning(VariableKind.parameter,parameter
                                         ,this,scope,searchBehindHidden));
                scope=scope.getScopeBehindHidden(ident); continue SEARCH;
            }

         for(Declaration declaration:scope.declarationList)
            if(ident.equalsIgnoreCase(declaration.identifier)) {
                DeclarationScope hiddenScope=scope.getHidden(ident);
                if(hiddenScope==null) return(new
                         Meaning(VariableKind.attribute,declaration
                                         ,this,scope,searchBehindHidden));
                scope=scope.getScopeBehindHidden(ident); continue SEARCH;
            }

         for(LabelDeclaration label:scope.labelList)
            if(ident.equalsIgnoreCase(label.identifier)) {
                DeclarationScope hiddenScope=scope.getHidden(ident);
                if(hiddenScope==null)  return(new
                         Meaning(VariableKind.label,label
                                         ,this,scope,searchBehindHidden));
                scope=scope.getScopeBehindHidden(ident); continue SEARCH;
            }

         for(Virtual virtual:scope.virtualList)
            if(ident.equalsIgnoreCase(virtual.identifier)) {
                DeclarationScope hiddenScope=scope.getHidden(ident);
                if(hiddenScope==null)  return(new
                         Meaning(VariableKind.virtual,virtual
                                         ,this,scope,searchBehindHidden));
                scope=scope.getScopeBehindHidden(ident); continue SEARCH;
            }

            if(scope.getHidden(ident)==null) scope=scope.getPrefix();
            else {
                scope=scope.getScopeBehindHidden(ident);
                searchBehindHidden=true;
            }
      }
       return(null);
}
```