

METHOD

```
METHOD method-name  
OF [INTERFACE] interface-name  
IS subprogram-name  
END-METHOD
```

This chapter covers the following topics:

- Function
- Syntax Description
- Example

For an explanation of the symbols used in the syntax diagram, see *Syntax Symbols*.

Related Statements: CREATE OBJECT | DEFINE CLASS | INTERFACE | PROPERTY | SEND METHOD

Belongs to Function Group: *Component Based Programming*

Function

The METHOD statement assigns a subprogram as the implementation to a method, *outside* an interface definition. It is used if the interface definition in question is included from a copycode and is to be implemented in a class-specific way.

The METHOD statement may only be used within the DEFINE CLASS statement and after the interface definition. The interface and method names specified must be defined in the INTERFACE clause of the DEFINE CLASS statement.

Syntax Description

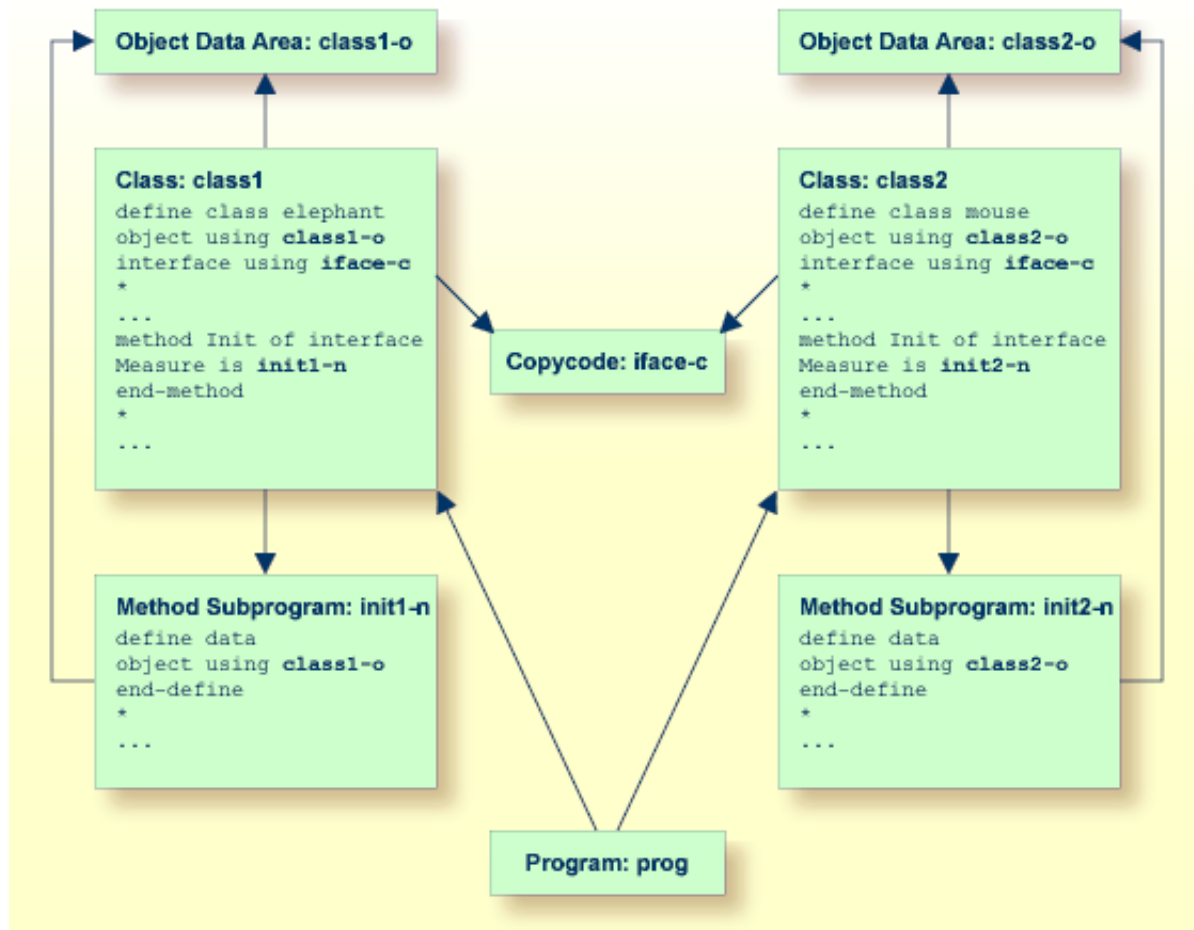
Syntax Element	Description
<i>method-name</i>	Method Name: This is the name assigned to the <i>method</i> .
OF <i>interface-name</i>	Interface Name: This is the name assigned to the <i>interface</i> .
IS <i>subprogram-name</i>	IS Clause: This clause can be used to specify the name of the subprogram that implements the method. The name of the subprogram consists of up to 8 characters. The default is <i>method-name</i> (if the IS clause is not specified).
END-METHOD	End of Method Statement: The Natural reserved word END-METHOD must be used to end the METHOD statement.

Example

The following example shows how the same interface is implemented differently in two classes and how the PROPERTY statement and the METHOD statement are used to achieve this.

The interface `Measure` is defined in the copycode `iface-c`. The classes `Elephant` and `Mouse` implement both the interface `Measure`. Therefore, they both include the copycode `iface-c`. But the classes implement the property `Height` using different variables from their respective object data areas, and they implement the method `Init` with different subprograms. They use the PROPERTY statement to assign the selected data area variable to the property and the METHOD statement to assign the selected subprogram to the method.

Now the program `prog` can create objects of both classes and initialize them using the same method `Init`, leaving the specifics of the initialization to the respective class implementation.



The following shows the complete contents of the Natural modules used in the example above:

Copycode: iface-c

```
interface Measure
*
property Height(p5.2)
end-property
*
property Weight(i4)
end-property
*
method Init
end-method
*
end-interface
```

Class: class1

```
define class elephant
object using class1-o
interface using iface-c
*
property Height of interface Measure is height
end-property
*
property Weight of interface Measure is weight
```

```

end-property
*
method Init of interface Measure is init1-n
end-method
*
end-class
end

```

LDA Object Data: class1-o

```

*   *** Top of Data Area ***
  1 HEIGHT                P 5.2
  1 WEIGHT                 I 2
*   *** End of Data Area ***

```

Method Subprogram: init1-n

```

define data
object using class1-o
end-define
*
height := 17.3
weight := 120
*
end

```

Class: class2

```

define class mouse
object using class2-o
interface using iface-c
*
property Height of interface Measure is size
end-property
*
property Weight of interface Measure is weight
end-property
*
method Init of interface Measure is init2-n
end-method
*
end-class
end

```

LDA Object Data: class2-o

```

*   *** Top of Data Area ***
  1 SIZE                   P 3.2
  1 WEIGHT                  I 1
*   *** End of Data Area ***

```

Method Subprogram: init2-n

```

define data
object using class2-o
end-define
*
size := 1.24
weight := 2
*
end

```

Program: prog

```
define data local
  1 #o handle of object
  1 #height(p5.2)
  1 #weight(i4)
end-define
*
create object #o of class 'Elephant'
send "Init" to #o
#height := #o.Height
#weight := #o.Weight
write #height #weight
*
create object #o of class 'Mouse'
send "Init" to #o
#height := #o.Height
#weight := #o.Weight
write #height #weight
*
end
```