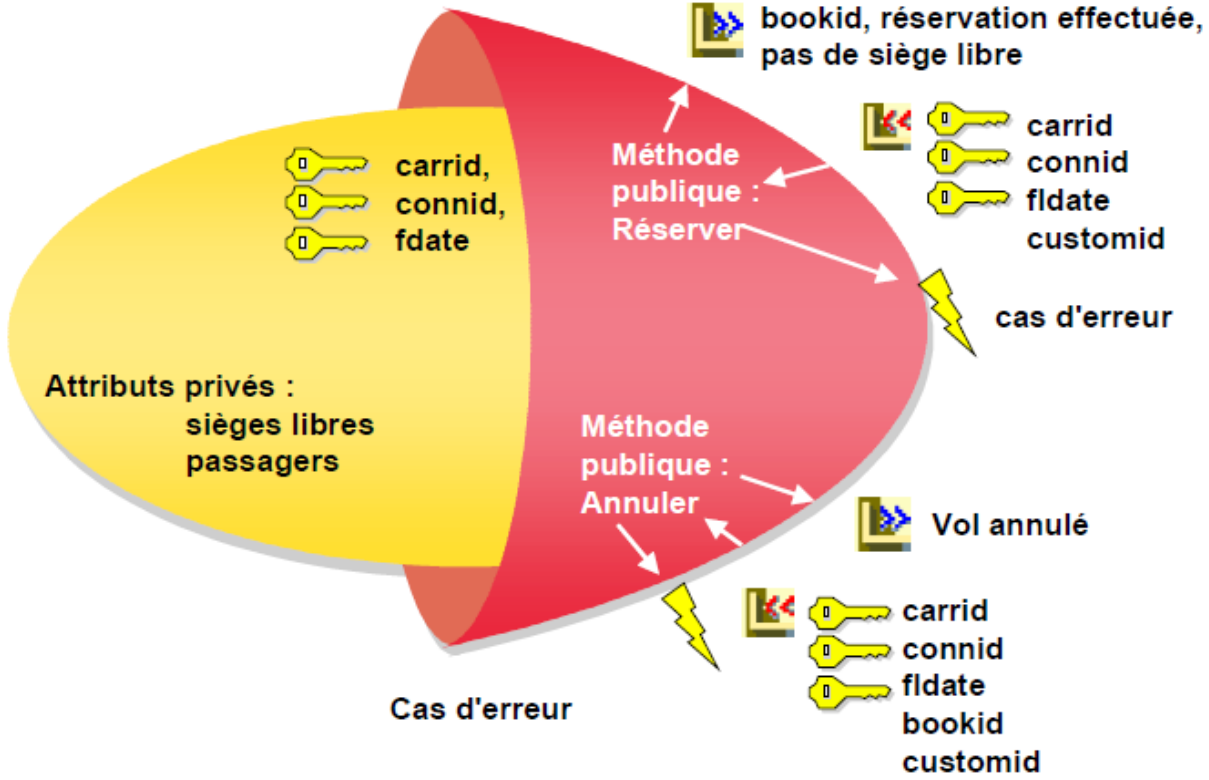
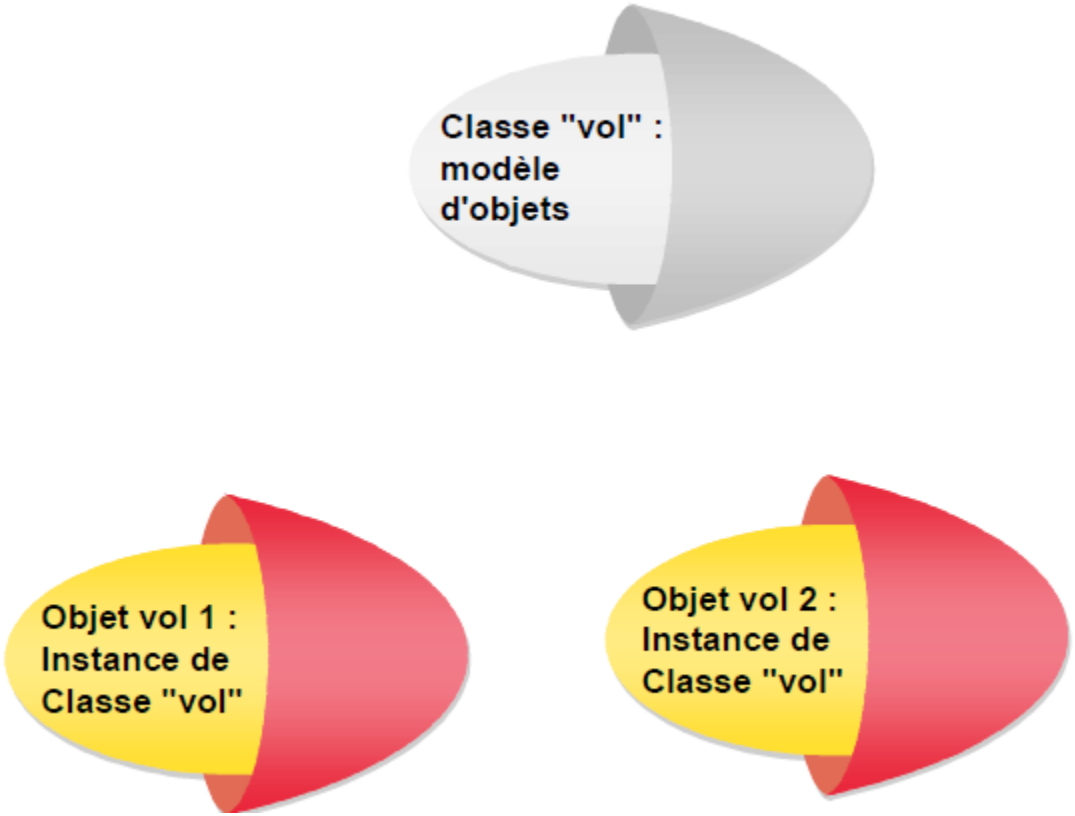
SAP
L'Orienté Objet

Zineb BOUGROUN

Objets ABAP : objet d'exemple "Vol"



Les objets sont des instances d'une classe



Classe syntaxe

```
CLASS <nom de classe> DEFINITION.

ENDCLASS.



CLASS <nom de classe>
IMPLEMENTATION.

ENDCLASS.
```

Partie Définition
Cette partie définit les composantes de la classe (par exemple, les attributs et les méthodes)

Partie Implémentation
Cette partie contient uniquement l'implémentation de méthodes

Déclaration d'attributs

```
CLASS <nom de classe> DEFINITION.
    ...

    TYPES: <définition de type normale>.
    CONSTANTS: constant TYPE <type> VALUE <valeur>.

    DATA: variable1 TYPE <type>,
          variable2 TYPE <ddic_type>,
          variable3 LIKE variable1,
          variable4 TYPE <type> VALUE <valeur>,
          variable5 TYPE <type> READ-ONLY,
          variable6 TYPE REF TO <nom de classe>,
          variable7 TYPE REF TO <interface>.

    CLASS-DATA: ...

ENDCLASS.
```

exemple

- Attributs publics
  - Affichage et modification par tous les utilisateurs et dans toutes les méthodes
  - Accès direct
- Attributs privés
  - Affichage et modification uniquement à partir de la classe
  - Aucun accès direct depuis l'extérieur de la classe

```
CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.
    DATA: name TYPE string.

  PRIVATE SECTION.
    DATA: weight TYPE saplane-weight.

ENDCLASS.
```

mieux

```
CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.
    ...

  PRIVATE SECTION.
    DATA: weight   TYPE saplane-weight,
          name TYPE string.

ENDCLASS.
```

- Attributs d'instance
  - Un attribut par instance
  - Instruction : DATA
- Attributs statiques
  - Un seul attribut par classe
  - Instruction : CLASS-DATA
  - Également appelés attributs de classe

```
CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.

  PRIVATE SECTION.
    DATA: weight TYPE saplane-weight,
          name   TYPE string.


    CLASS-DATA: count TYPE I.

ENDCLASS.
```

Déclaration de méthodes

```
CLASS <nom de classe> DEFINITION.
  ...
    METHODS: <nom_méthode>
               [ IMPORTING <im_var> TYPE <type>
                 EXPORTING <ex_var> TYPE <type>
                 CHANGING  <ch_var> TYPE <type>
                 RETURNING  VALUE(<re_var>) TYPE <type>
                 EXCEPTIONS <exception> ].
ENDCLASS.
```

```
CLASS <nom de classe> IMPLEMENTATION.
  METHOD <nom_méthode>.
    ...
  ENDMETHOD.
ENDCLASS.
```

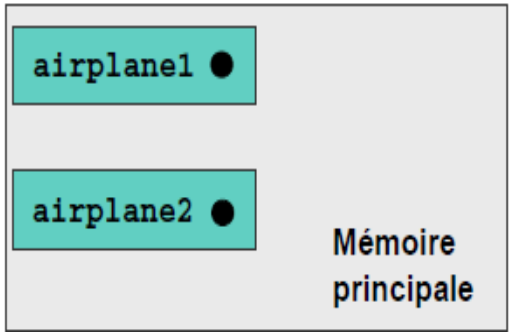
exemple

```
CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.
    METHODS:       set_name IMPORTING im_name TYPE string.
    CLASS-METHODS: get_count RETURNING VALUE(re_count) TYPE I.

  PRIVATE SECTION.
    DATA:       name TYPE string.
    CLASS-DATA: count TYPE I.

ENDCLASS.
```

```
CLASS lcl_airplane IMPLEMENTATION.
  ...
  METHOD get_count.
     re_count = count.
  ENDMETHOD

ENDCLASS.
```

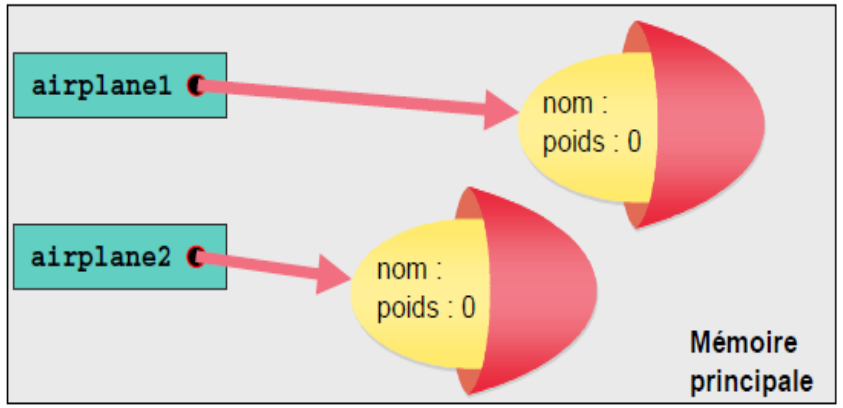
Création d'objet

```
CLASS lcl_airplane DEFINITION.
  PUBLIC SECTION.
    ...
  PRIVATE SECTION.
    ...
ENDCLASS.


CLASS lcl_airplane IMPLEMENTATION.
  ...
ENDCLASS.


DATA: airplane1 TYPE REF TO cl_airplane,
      airplane2 TYPE REF TO cl_airplane.
```




```
CREATE OBJECT <référence>.
```

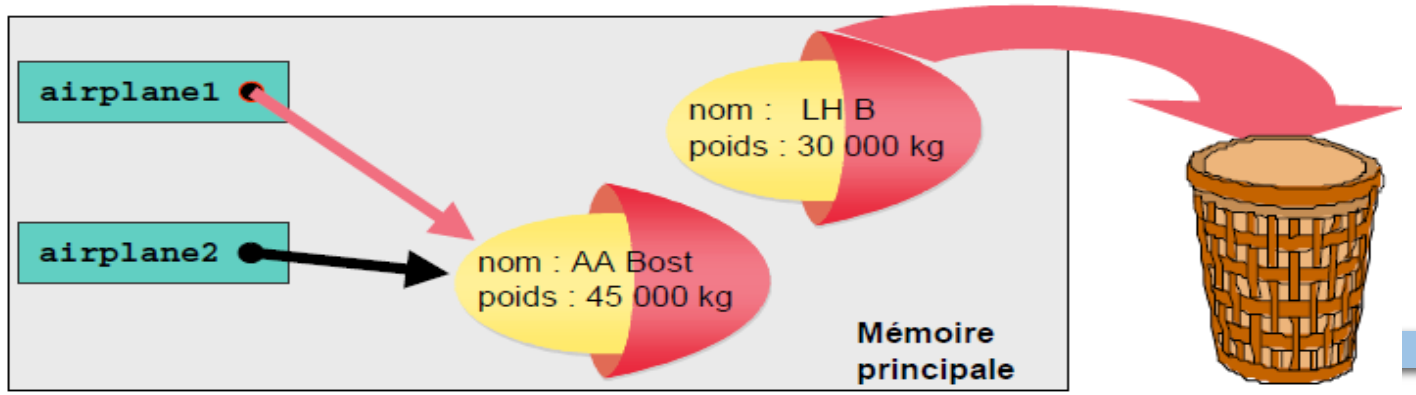
```
DATA: airplane1 TYPE REF TO lcl_airplane,
      airplane2 TYPE REF TO lcl_airplane.

CREATE OBJECT airplane1.
CREATE OBJECT airplane2.
```



Garbage Collector

- Le Garbage Collector est une routine système supprimant automatiquement les objets ne pouvant plus être adressés à partir de la mémoire principale, puis libérant l'espace mémoire qu'ils occupaient.
- Le Garbage Collector supprime de la mémoire tout objet dès que plus aucune référence ne pointe.

```
...
DATA: airplane1 TYPE REF TO lcl_airplane,
      airplane2 TYPE REF TO lcl_airplane.

CREATE OBJECT airplane1 EXPORTING ... .
CREATE OBJECT airplane2 EXPORTING ... .

airplane1 = airplane2.
```



Notion d’égalité

```
DATA: airplane1 TYPE REF TO lcl_airplane,
      airplane2 TYPE REF TO lcl_airplane.

CREATE OBJECT airplane1 EXPORTING im_name = 'LH Berlin' ...
CREATE OBJECT airplane2 EXPORTING im_name = 'LH Berlin' ...

IF airplane1 = airplane2.        "ils ne sont pas égaux
...
ENDIF.
```

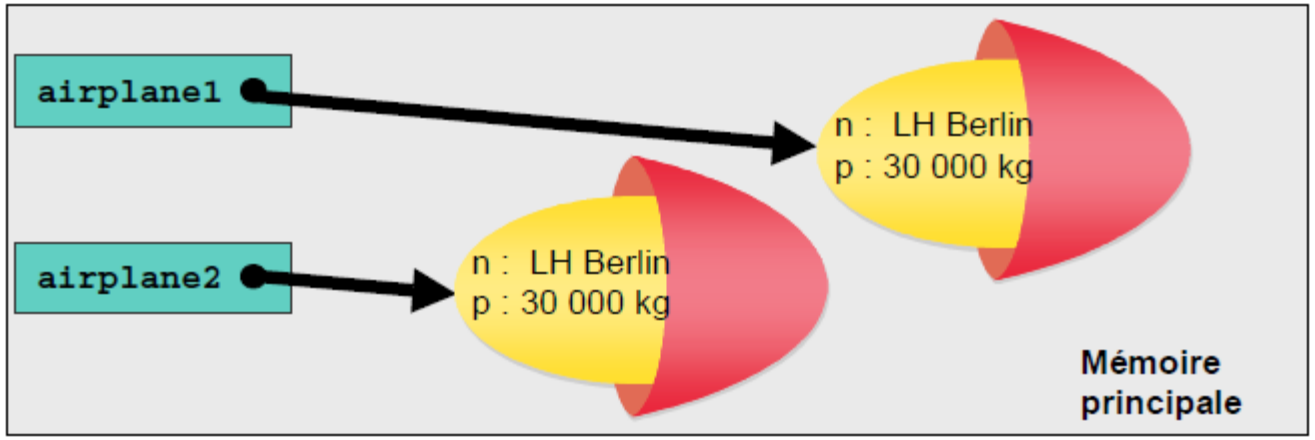
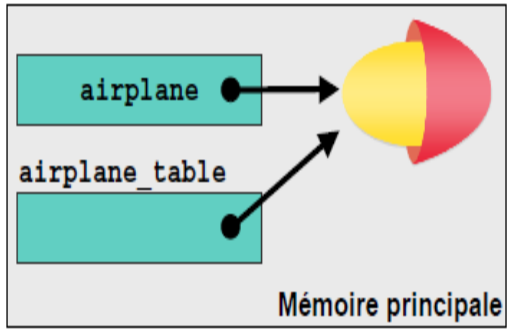


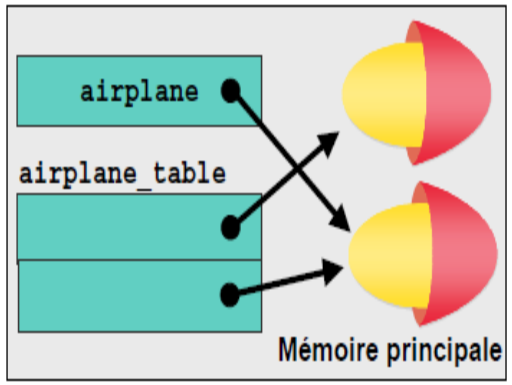
Table d’objets

```
DATA: airplane          TYPE REF TO cl_airplane,
      airplane_table    TYPE TABLE OF REF TO cl_airplane.
```

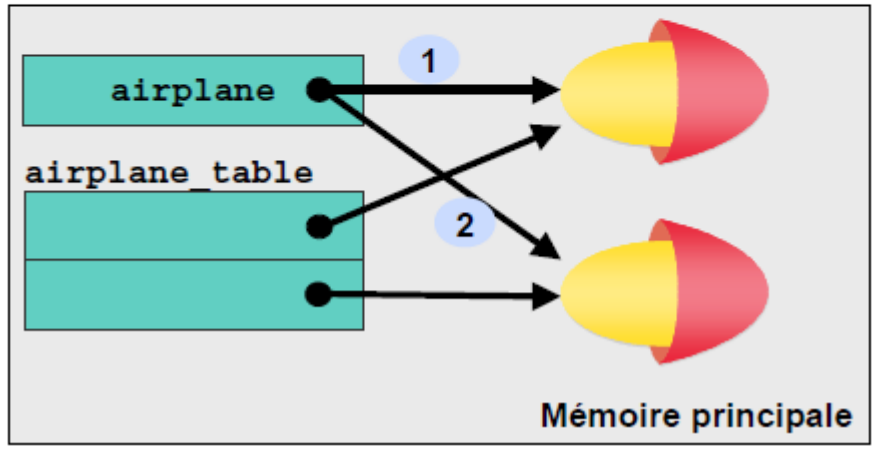
```
CREATE OBJECT airplane.
APPEND airplane TO airplane_table.
```




```
CREATE OBJECT airplane.
APPEND airplane TO airplane_table.
```



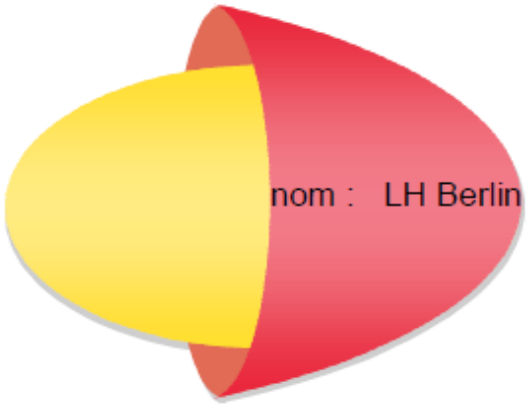

```
LOOP AT TO airplane_table INTO airplane.

*  utilisation de l'instance courante

ENDLOOP.
```



Appel d’attributs

Attribut d'instance :
<référence>-><attribut_instance>

Attribut de classe :
<nom de classe>=><attribut_classe>




```
CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.
    DATA: name    TYPE string READ-ONLY.
    CLASS-DATA: count TYPE I READ-ONLY.
    ...
ENDCLASS.
...

DATA: airplane1 TYPE REF TO lcl_airplane.
DATA: airplane_name TYPE STRING,
      n_o_airplanes TYPE i.
...

airplane_name = airplane1->name.
n_o_airplanes = lcl_airplane=>count.
```

Appel de méthode

Méthodes d'instance :

```
CALL METHOD <instance>-><méthode_instance>
            EXPORTING <im_var> = <variable>
            IMPORTING <ex_var> = <variable>
            CHANGING  <ch_var> = <variable>
            RECEIVING <re_var> = <variable>
            EXCEPTIONS <exception> = <nr>.
```

Méthodes statiques:

```
CALL METHOD <nom_de_classe>=><méthode_classe>
             EXPORTING ...  .
```

```
DATA: airplane TYPE REF TO lcl_airplane.
DATA: name TYPE string.
DATA: count_planes TYPE I.

CREATE OBJECT airplane.

CALL METHOD airplane->set_name EXPORTING im_name = name.
CALL METHOD lcl_airplane=>get_count RECEIVING re_count = count_planes.
```

Méthodes fonctionnelles

- Les méthodes comportant un paramètre RETURNING sont décrites comme des méthodes fonctionnelles. Elles ne peuvent pas comporter de paramètres EXPORTING ou CHANGING, mais plusieurs (ou autant que nécessaire) paramètres IMPORTING et EXCEPTIONS, selon les besoins.
- Les méthodes fonctionnelles peuvent être utilisées directement dans diverses expressions :
  - dans les expressions logiques (IF, ELSEIF, WHILE, CHECK, WAIT) ;
  - dans l'instruction CASE (CASE, WHEN) ;
  - dans l'instruction LOOP ;
  - dans les expressions arithmétiques;
  - dans les expressions binaires;
  - dans l'instruction MOVE.

Méthodes fonctionnelles : exemples

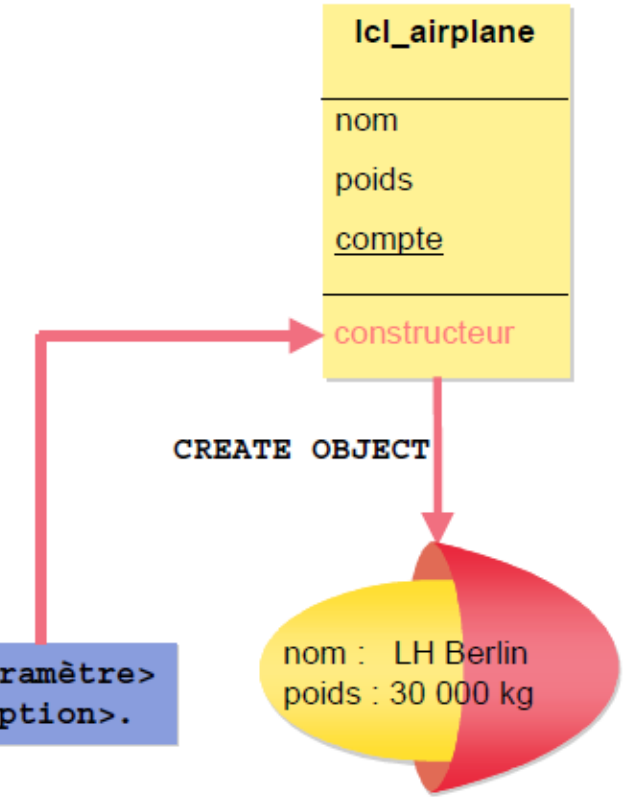
- La syntaxe des méthodes est la suivante, en fonction du nombre de paramètres IMPORTING :
  - aucun paramètre IMPORTING : ref->func_method( )
  - 1 paramètre IMPORTING : ref->func_method(p1) ou ref->func_method(im_1 = p1)
  - plusieurs paramètres IMPORTING : ref->func_method(im_1 = p1 im_2 = p2)

```
CLASS lcl_airplane DEFINITION.
  PUBLIC SECTION.
    METHODS: estimated_fuel_consumption
                      IMPORTING im_distance    TYPE ty_distance
                      RETURNING VALUE(re_fuel) TYPE ty_fuel,
    CLASS-METHODS: get_count RETURNING VALUE(re_count) TYPE i.
ENDCLASS.
```

```
DATA: plane1           TYPE REF TO lcl_airplane,
      plane2           TYPE REF TO lcl_airplane,
      fuel_consumption TYPE ty_fuel,
      count_planes     TYPE i.

* Instanciation omise

* CALL METHOD plane1->get_count RECEIVING re_count = count_planes.
count_planes = lcl_airplane=>get_count( ).

fuel_consumption =    plane1->estimated_fuel_consumption( 1000 )
      + plane2->estimated_fuel_consumption( im_distance = 1500 ).
```

Constructeur

- **Méthode spéciale pour la création d'objets avec un état initial défini**
- **Dispose uniquement des paramètres IMPORTING et des EXCEPTIONS**
- **Un seul constructeur est défini par classe (de manière explicite ou implicite)**
- **Une seule exécution par instance**



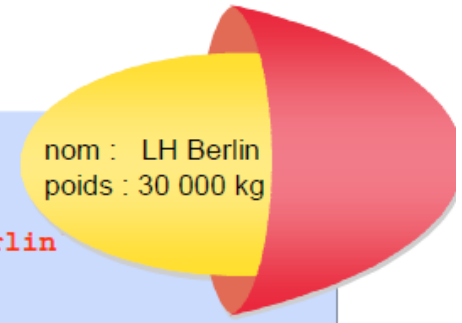

```
METHODS CONSTRUCTOR IMPORTING <im_paramètre>
                    EXCEPTIONS <exception>.
```

Constructeur exemple

```
CLASS lcl_airplane DEFINITION.
  PUBLIC SECTION.
    METHODS CONSTRUCTOR IMPORTING im_name   TYPE string
                                  im_weight TYPE I.
  PRIVATE SECTION.
    DATA: name  TYPE string, weight TYPE I.
    CLASS-DATA count TYPE I.
ENDCLASS.
```

```
CLASS lcl_airplane IMPLEMENTATION.
  METHOD CONSTRUCTOR.
    name   = im_name.
    weight = im_weight.
    count = count + 1.
  ENDMETHOD.
ENDCLASS.
```

```
DATA airplane TYPE REF TO lcl_airplane.
...
CREATE OBJECT airplane
              EXPORTING im_name   = `LH Berlin
                        im_weight = 30000.
```



Appel d’attribut et méthode statique

- Méthode statique spéciale
- Appel automatique avant le premier accès à la classe
- Une seule exécution par programme

```
* Exemple 1 :

DATA airplane TYPE REF TO lcl_airplane.

CREATE OBJECT airplane.
```

```
* Exemple 2 :

DATA class_id TYPE string.

class_id = lcl_airplane=>count.
```

```
* Exemple 3 :

DATA count_airplane TYPE I.

CALL METHOD lcl_airplane=>get_count
  RECEIVING re_count = count_airplane.
```

Héritage : syntaxe

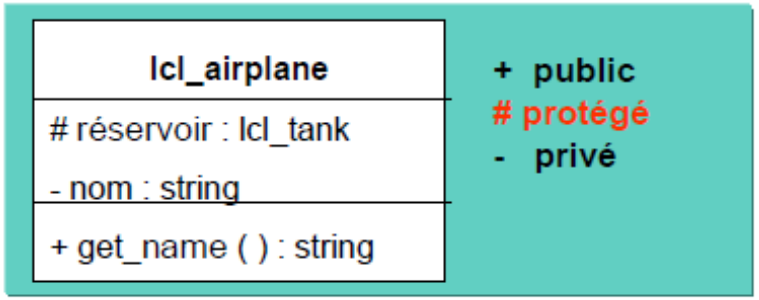
```
CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.
    METHODS: get_fuel_level RETURNING VALUE(re_level) TYPE ty_level.

  PRIVATE SECTION.
    DATA: name  TYPE string,
          weight TYPE I.
ENDCLASS.
```

```
CLASS lcl_cargo_airplane DEFINITION INHERITING FROM lcl_airplane.

  PUBLIC SECTION.
    METHODS: get_cargo RETURNING VALUE(re_cargo) TYPE ty_cargo.

  PRIVATE SECTION.
    DATA: cargo TYPE ty_cargo.

ENDCLASS.
```

Visibilité

- Composantes publiques
  - Visibles par tous
  - Accès direct
- **Composantes protégées**
  - **Visibles uniquement dans leur classe et dans les sous-classes**
- Composantes privées
  - Visibles uniquement dans la classe
  - Aucun accès de l'extérieur, **même à partir de la sous-classe**

```
CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.
    METHODS get_name RETURNING
      VALUE(re_name) TYPE string.

  PROTECTED SECTION.
    DATA tank TYPE REF TO lcl_tank.

  PRIVATE SECTION.
    DATA name TYPE string.

ENDCLASS.
```



Constructeur

- Le constructeur de la super-classe **doit être appelé dans le constructeur de la sous-classe, du fait de la** tâche spécifique du constructeur : s'assurer que les objets sont correctement initialisés.

```
CLASS lcl_airplane DEFINITION.
 PUBLIC SECTION.
  METHODS: CONSTRUCTOR IMPORTING
                im_name TYPE string.
ENDCLASS.
```

```
CLASS lcl_airplane IMPLEMENTATION.
  METHOD CONSTRUCTOR.
    name = im_name.
  ENDMETHOD.
ENDCLASS.
```

```
CLASS lcl_cargo_airplane DEFINITION INHERITING FROM lcl_airplane.
  PUBLIC SECTION.
    METHODS: CONSTRUCTOR IMPORTING im_name  TYPE string
                                   im_cargo TYPE ty_cargo.
  PRIVATE SECTION.
    DATA: cargo TYPE ty_cargo.
ENDCLASS.
```

```
CLASS lcl_cargo_airplane IMPLEMENTATION.
  METHOD CONSTRUCTOR.
    CALL METHOD SUPER->CONSTRUCTOR EXPORTING im_name = im_name.
    cargo = im_cargo.
  ENDMETHOD.
ENDCLASS.
```

Méthodes redéfinies

- **Vous pouvez uniquement redéfinir des méthodes d'instance (publiques ou protégées).**
- Les autres composantes (méthodes statiques, attributs, etc.) ne peuvent pas être redéfinies. En outre, l'implémentation est limitée à une méthode héritée. Vous ne pouvez pas modifier les paramètres de méthode (modification de l 'interface). Il est également impossible de redéfinir le constructeur (d'instance) d'une classe.
- Les méthodes héritées, mais non redéfinies ne sont pas reprises dans la sous-classe, puisque leur présence dans la sous-classe est parfaitement claire sur la base de la relation de spécialisation.
- Ne confondez pas redéfinition et "surcharge". La surcharge décrit la capacité d'une classe à comporter des méthodes portant le même nom, mais une interface différente (nombre et type de paramètres). Cette option n'est pas disponible dans ABAP Objects.

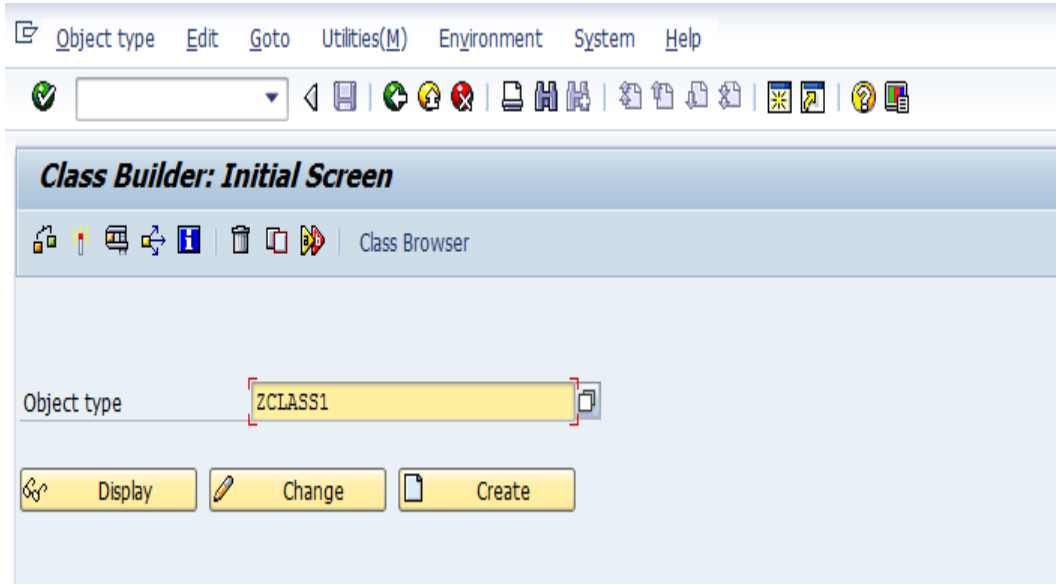
Si vous redéfinissez une méthode, vous n'avez pas à saisir à nouveau son interface dans la sous classe,
mais uniquement son nom, car ABAP Objects ne supporte pas la surcharge (référez-vous aux remarques de la diapositive précédente).

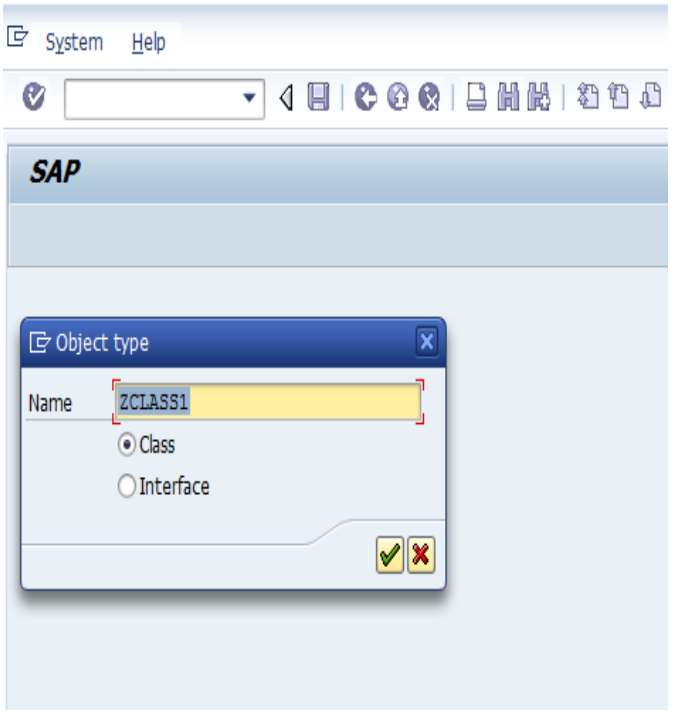
```
CLASS lcl_airplane DEFINITION.
  PUBLIC SECTION.
    METHODS estimate_fuel_consumption
            IMPORTING im_distance    TYPE ty_distance
            RETURNING VALUE(re_fuel) TYPE ty_fuel.
ENDCLASS.
```

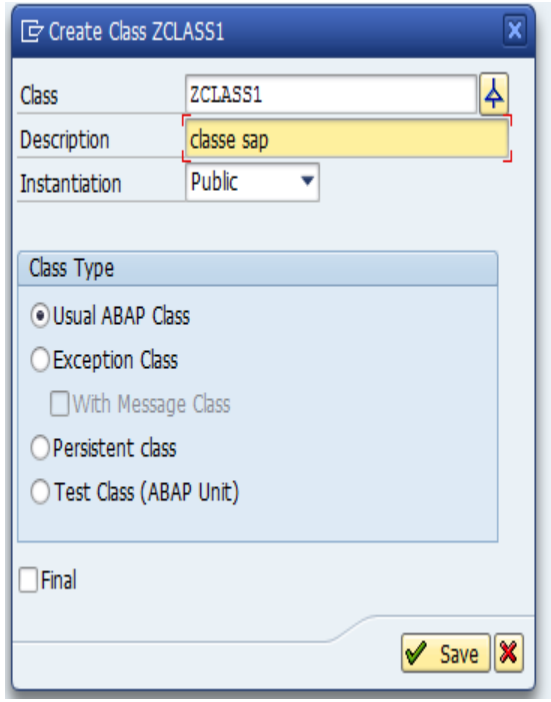
```
CLASS lcl_passenger_airplane DEFINITION INHERITING FROM lcl_airplane.
  PUBLIC SECTION.
    METHODS estimate_fuel_consumption REDEFINITION.
      ...
ENDCLASS.

CLASS lcl_passenger_airplane IMPLEMENTATION.
  METHOD estimate_fuel_consumption.
     ...
  ENDMETHOD.
ENDCLASS.
```

Création d'une classe public

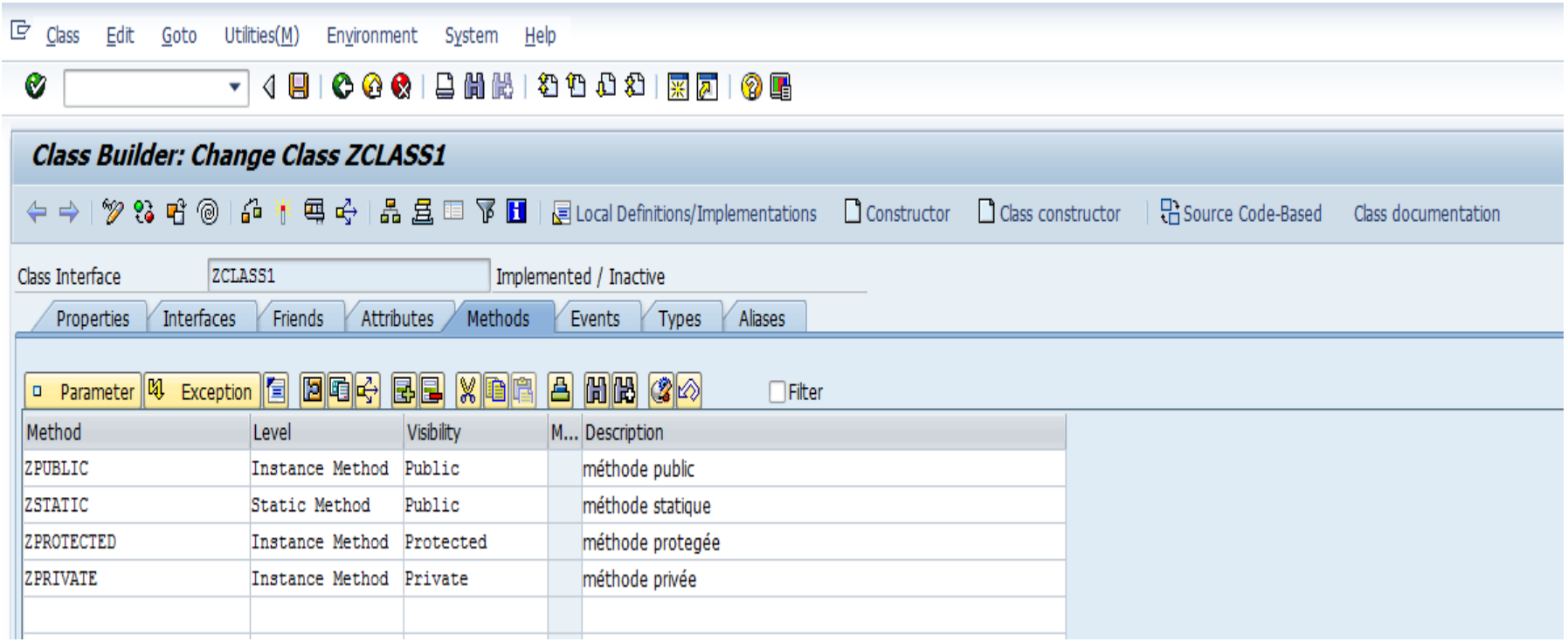
- Utiliser la transaction se24







Création d'une classe



Class Builder: Class ZCLASS1 Change

Pattern | Pretty Printer | Signature

Method ZPUBLIC Inactive

```
1  method ZPUBLIC.
2
3    write : 'je suis dans le début de la méthode public'.
4
5    call METHOD me->ZPROTECTED.
6    call METHOD  me->zprivate.
7
8    write : 'je suis dans la fin de la méthode public'.
9  endmethod.
```

Method Edit Goto Utilities(M) Environment System Help

Class Builder: Class ZCLASS1 Change

Pattern Pret

Method ZPROTECTED

```
1  method ZPROTECTED.
2
3    write : 'je suis dans protected'.
4  endmethod.
```

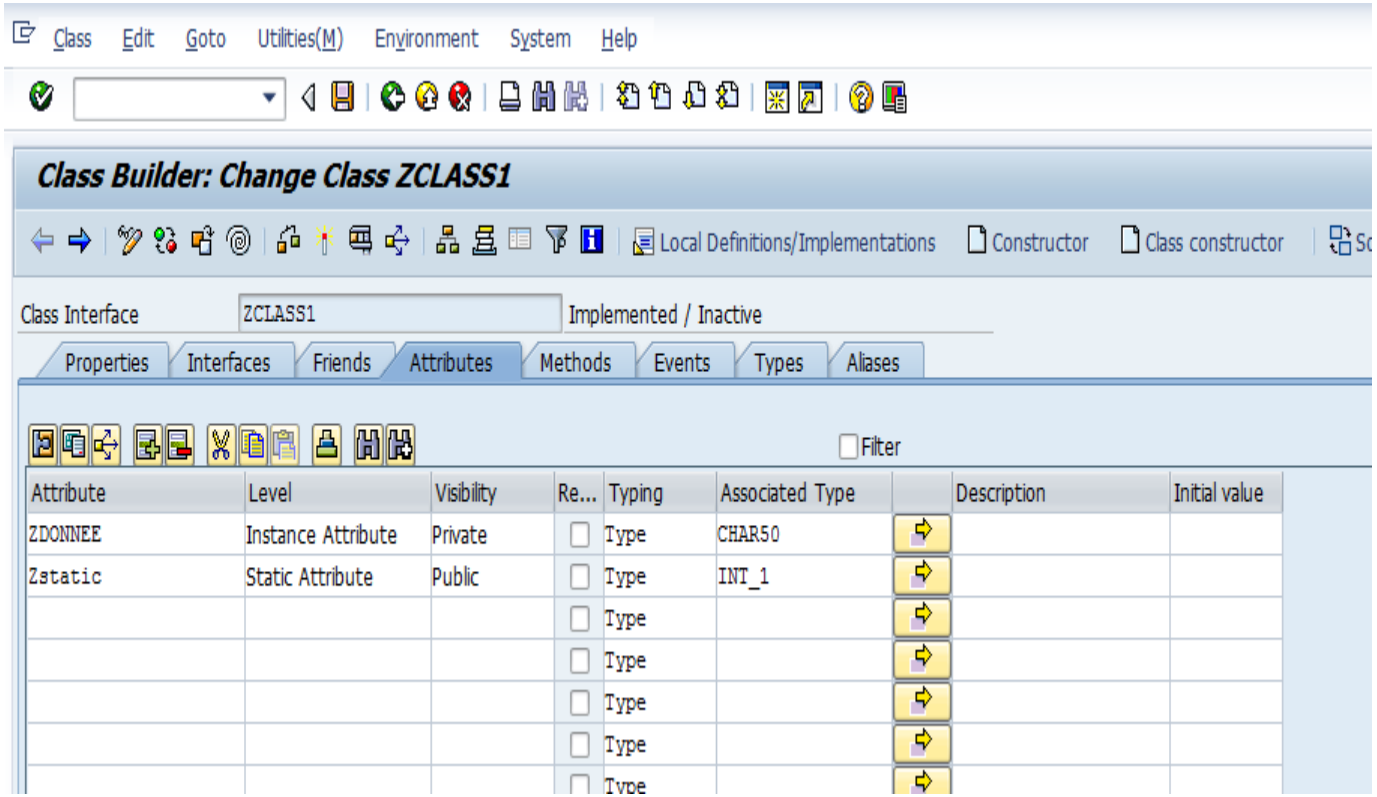
Method Edit Goto Utilities(M) Environment System Help

Class Builder: Class ZCLASS1 Change

Pattern | Pretty Printer | Signat

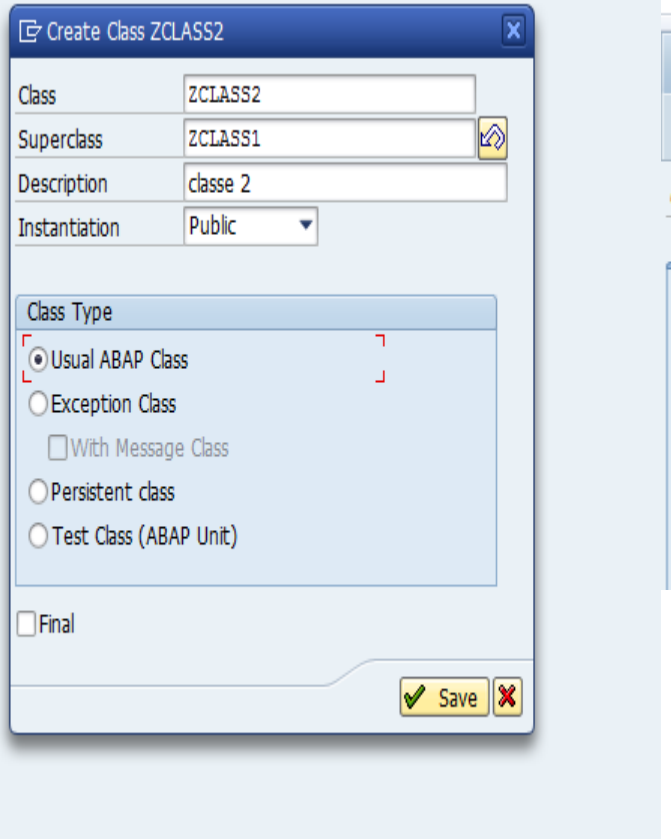
Method ZPRIVATE Active

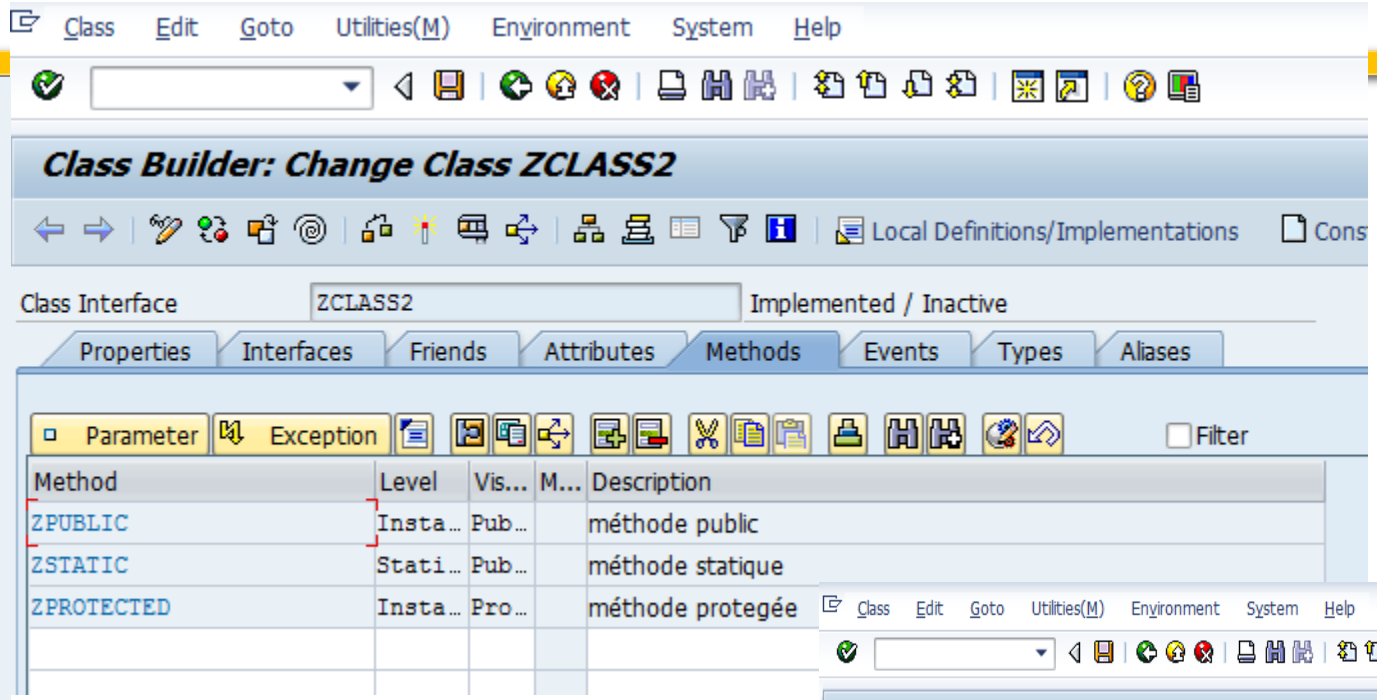
```
1  method ZPRIVATE.
2      write : 'je suis dans méthode privée'.
3  endmethod.
```

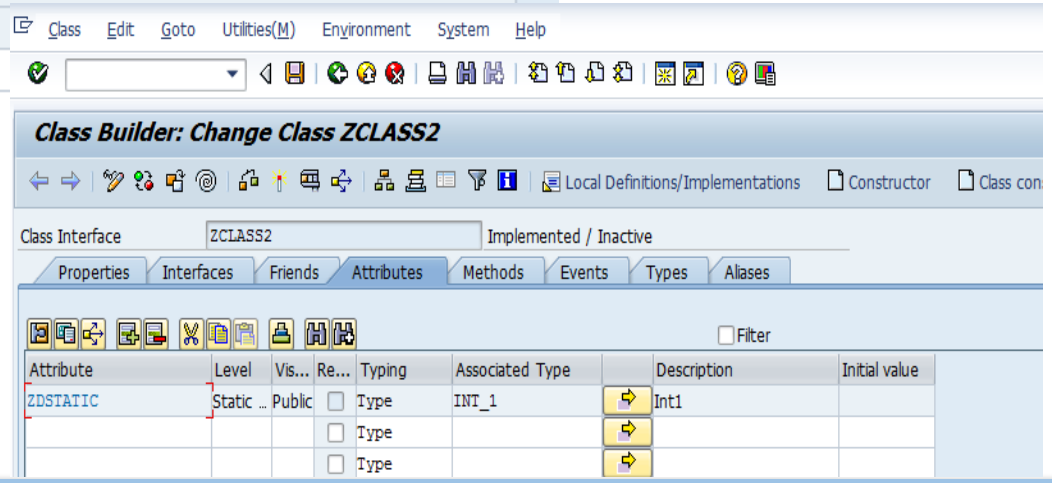

Class Edit Goto Utilities(M) Environment System Help
Class Builder: Change Class ZCLASS1
Local Definitions/Implementations
Constructor
Class constructor
Class Interface
ZCLASS1
Implemented / Inactive
Properties
Interfaces
Friends
Attributes
Methods
Events
Types
Aliases
Filter
Attribute
Level
Visibility
Re...
Typing
Associated Type
Description
Initial value
ZDONNEE
Instance Attribute
Private
Type
CHAR50
Zstatic
Static Attribute
Public
Type
INT_1
Type
Type
Type
Type
Type

```
class ZCLASS1 definition
  public
  create public .

public section.
*"* public components of class ZCLASS1
*"* do not include other source files here!!!

  class-data ZDSTATIC type I .

  methods ZPUBLIC .
  class-methods ZSTATIC .
protected section.
*"* protected components of class ZCLASS1
*"* do not include other source files here!!!

  methods ZPROTECTED .
private section.
*"* private components of class ZCLASS1
*"* do not include other source files here!!!

  data ZDONNEE type CHAR50 .

  methods ZPRIVATE .
ENDCLASS.



CLASS ZCLASS1 IMPLEMENTATION.


* <SIGNATURE>---------------------------------------------------------------------------------------+
* | Instance Private Method ZCLASS1->ZPRIVATE
* +---------------------------------------------------------------------------------------------------+
* +-------------------------------------------------------------------------------------</SIGNATURE>
method ZPRIVATE.
    write : /'je suis dans méthode privée'.
endmethod.


* <SIGNATURE>---------------------------------------------------------------------------------------+
* | Instance Protected Method ZCLASS1->ZPROTECTED
* +---------------------------------------------------------------------------------------------------+
* +-------------------------------------------------------------------------------------</SIGNATURE>
method ZPROTECTED.

  write :/ 'je suis dans protected'.
endmethod.

```

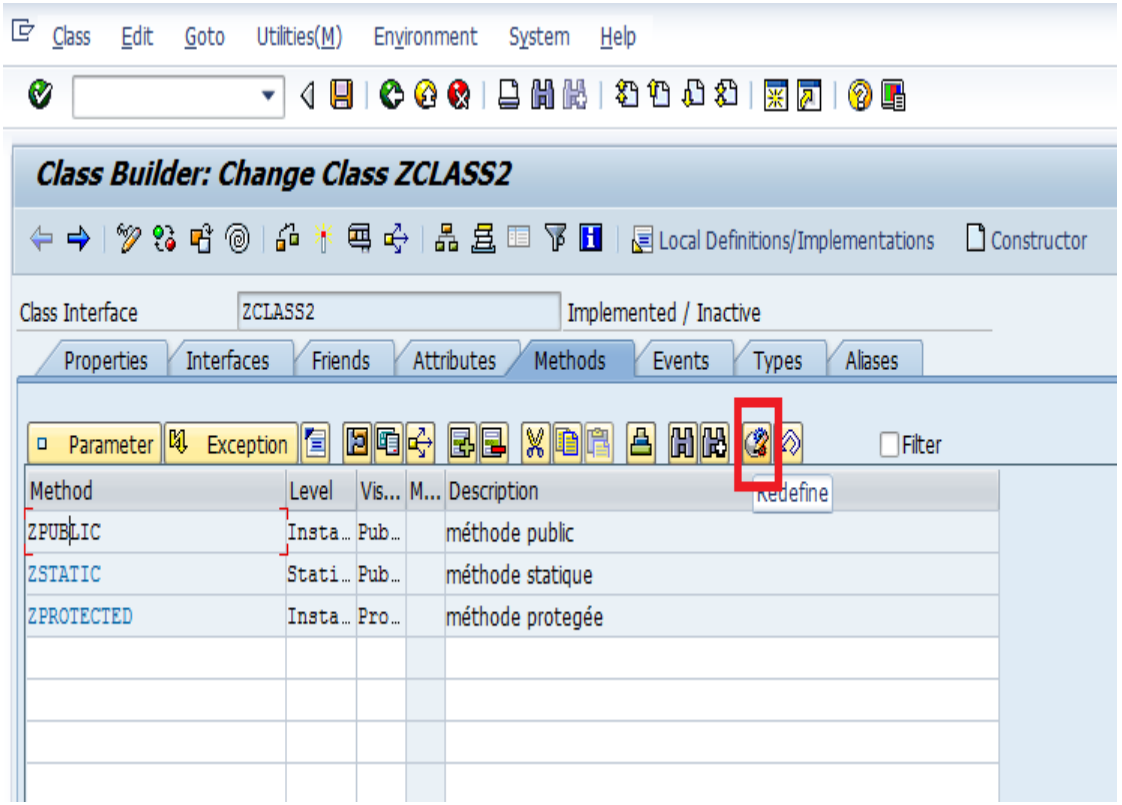
Création d'une classe fille

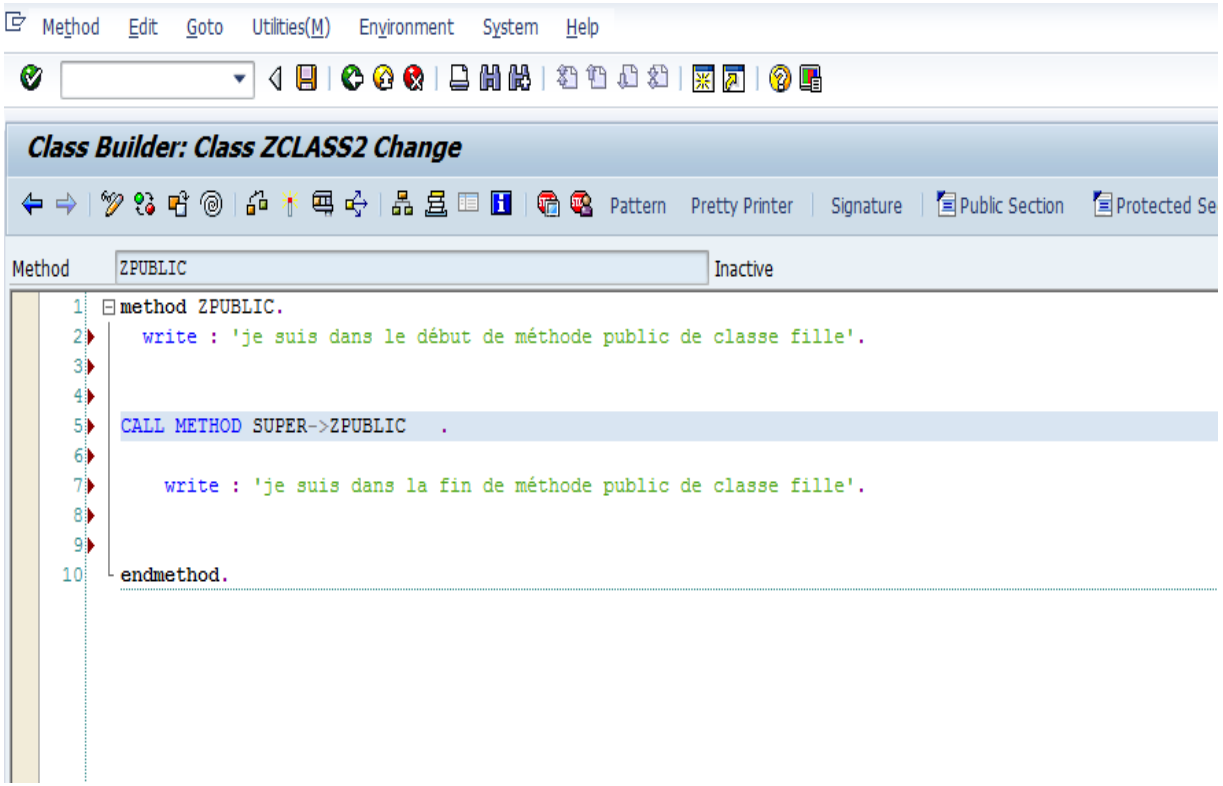






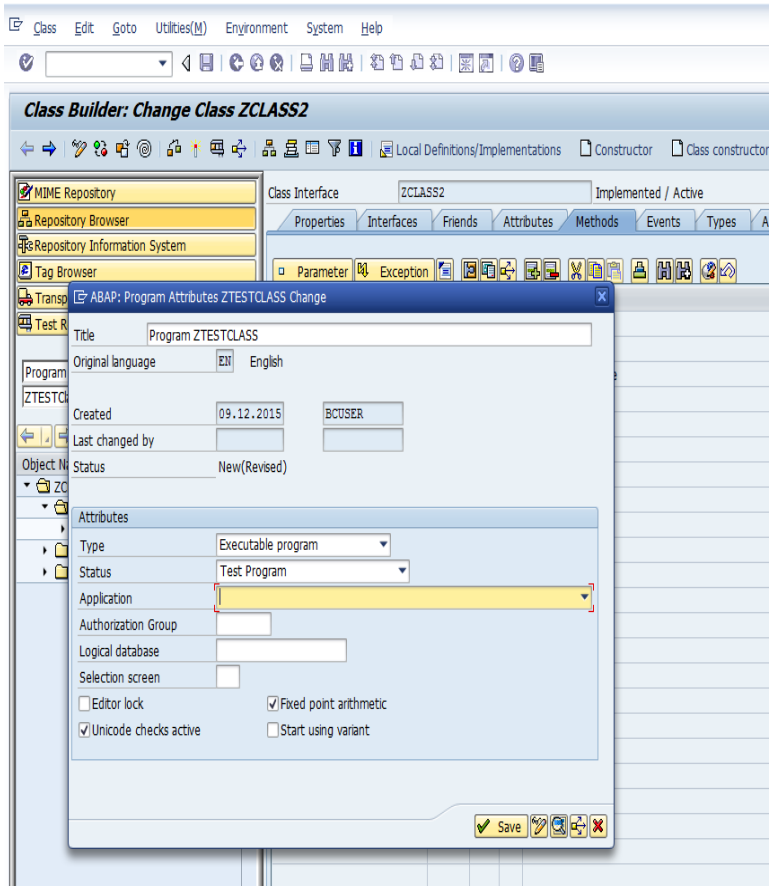
polymorphisme

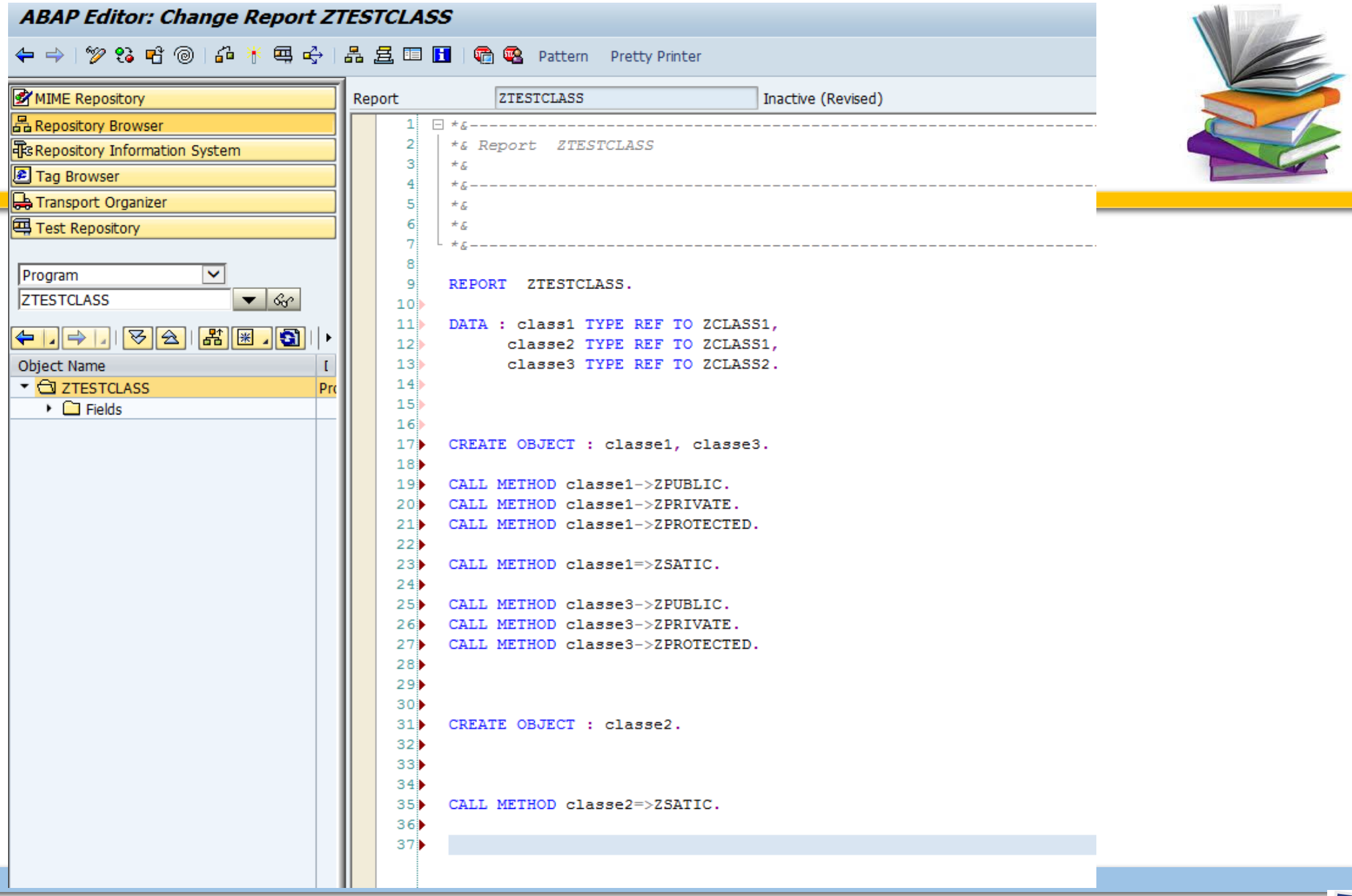




Création de programme

- Utiliser se80 pour créer un programme




ABAP Editor: Change Report ZTESTCLASS
Pattern
Pretty Printer
MIME Repository
Repository Browser
Repository Information System
Tag Browser
Transport Organizer
Test Repository
Program
ZTESTCLASS
Object Name
ZTESTCLASS
Fields
Report
ZTESTCLASS
Inactive (Revised)
&---
*& Report ZTESTCLASS
*&
&---
*&
*&
&---
REPORT ZTESTCLASS.
DATA : class1 TYPE REF TO ZCLASS1,
classe2 TYPE REF TO ZCLASS1,
classe3 TYPE REF TO ZCLASS2.
CREATE OBJECT : classe1, classe3.
CALL METHOD classe1->ZPUBLIC.
CALL METHOD classe1->ZPRIVATE.
CALL METHOD classe1->ZPROTECTED.
CALL METHOD classe1=>ZSATIC.
CALL METHOD classe3->ZPUBLIC.
CALL METHOD classe3->ZPRIVATE.
CALL METHOD classe3->ZPROTECTED.
CREATE OBJECT : classe2.
CALL METHOD classe2=>ZSATIC.

```
*&---------------------------------------------------------------------*
*& Report  ZTESTCLASS
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  ZTESTCLASS.

DATA : classe1 TYPE REF TO ZCLASS1,
       classe2 TYPE REF TO ZCLASS1,
       classe3 TYPE REF TO ZCLASS2.



CREATE OBJECT : classe1, classe3.

CALL METHOD classe1->ZPUBLIC.
*CALL METHOD classe1->ZPRIVATE.
*CALL METHOD classe1->ZPROTECTED.

CALL METHOD ZCLASS1=>ZSTATIC.

CALL METHOD classe3->ZPUBLIC.
*CALL METHOD classe3->ZPRIVATE.
*CALL METHOD classe3->ZPROTECTED.



CREATE OBJECT : classe2.


CALL METHOD ZCLASS1=>ZSTATIC.
```

List Edit Goto System Help

SAP

```
je suis dans le début de la méthode public
je suis dans protected
je suis dans méthode privée
je suis dans la fin de la méthode public
ZDSTATIC         1  je suis dans le début de méthode public de classe fille
je suis dans le début de la méthode public
je suis dans protected
je suis dans méthode privée
je suis dans la fin de la méthode public je suis dans la fin de méthode public de classe fille
ZDSTATIC         2
```

```
REPORT  ZTEST.

CLASS animal DEFINITION ABSTRACT.

  PUBLIC SECTION.
    DATA : noise TYPE char10.
    METHODS make_noise.

ENDCLASS.                    "animal DEFINITION


CLASS animal IMPLEMENTATION.

  METHOD make_noise.
  ENDMETHOD.                    "make_noise

ENDCLASS.                    "animal IMPLEMENTATION


*---------------------------------------------------------------------*
*       CLASS dog DEFINITION
*---------------------------------------------------------------------*
*
*---------------------------------------------------------------------*
CLASS dog DEFINITION INHERITING FROM animal.

  PUBLIC SECTION.
    DATA play TYPE char10.
    METHODS make_noise REDEFINITION.

ENDCLASS.                    "dog DEFINITION

CLASS dog IMPLEMENTATION.

  METHOD make_noise.
    noise  = 'noise dog'.
    play  = 'play dog'.
    WRITE / : noise,
              play.
    SKIP.
  ENDMETHOD.                    "make_noise

ENDCLASS.                    "dog IMPLEMENTATION


*---------------------------------------------------------------------*
*       CLASS cat DEFINITION
*---------------------------------------------------------------------*
*
*---------------------------------------------------------------------*
CLASS cat DEFINITION INHERITING FROM animal.

  PUBLIC SECTION.

    METHODS make_noise REDEFINITION.

ENDCLASS.                    "cat DEFINITION

CLASS cat IMPLEMENTATION.

  METHOD make_noise.
    noise = 'Meow'.
    WRITE / noise.
    SKIP.
  ENDMETHOD.                    "make_noise

ENDCLASS.                    "cat IMPLEMENTATION


START-OF-SELECTION.

  DATA : ref_animal TYPE TABLE OF REF TO animal,
         ref_dog    TYPE REF TO dog,
         ref_cat    TYPE REF TO cat.


  DATA wa_animal LIKE LINE OF ref_animal.

  CREATE OBJECT ref_dog.
  CREATE OBJECT ref_cat.

  wa_animal = ref_dog.
  APPEND wa_animal TO ref_animal.

  wa_animal = ref_cat.
  APPEND wa_animal TO ref_animal.

  wa_animal = ref_dog.
  APPEND wa_animal TO ref_animal.

  LOOP AT ref_animal
    INTO wa_animal.
    CALL METHOD wa_animal->make_noise.
```

List Edit Goto System Help

Program ZTEST

```
Program ZTEST

noise dog
play dog

Meow

noise dog
play dog
```

Création d'une classe dans un programme

Report ZTESTCLASS Inactive

```
10
11
12  CLASS CL_LC DEFINITION.
13    PUBLIC SECTION.
14    DATA: A TYPE I,
15          B TYPE I,
16          C TYPE I.
17    METHODS: DISPLAY,
18             MM1.
19    CLASS-METHODS: MM2.
20    ENDCLASS.
21
22
23  CLASS CL_LC IMPLEMENTATION.
24  METHOD DISPLAY.
25    WRITE:/ 'THIS IS SUPER CLASS' COLOR 7.
26    ENDMETHOD.
27  METHOD MM1.
28    WRITE:/ 'THIS IS MM1 METHOD IN SUPER CLASS'.
29    ENDMETHOD.
30  METHOD MM2.
31    WRITE:/ 'THIS IS THE STATIC METHOD' COLOR 2.
32    WRITE:/ 'THIS IS MM2 METHOD IN SUPER CLASS' COLOR 2.
33    ENDMETHOD.
34    ENDCLASS.
35
36  CLASS CL_SUB DEFINITION INHERITING FROM CL_LC. "HOW WE CAN INHERIT
37    PUBLIC SECTION.
38    DATA: A1 TYPE I,
39          B1 TYPE I,
40          C1 TYPE I.
41    METHODS: DISPLAY REDEFINITION,     "REDEFINE THE SUPER CLASS METHOD
42             SUB.
43    ENDCLASS.
44
45  CLASS CL_SUB IMPLEMENTATION.
46  METHOD DISPLAY.
47    WRITE:/ 'THIS IS THE SUB CLASS OVERWRITE METHOD' COLOR 3.
48    ENDMETHOD.
49  METHOD SUB.
50    WRITE:/ 'THIS IS THE SUB CLASS METHOD' COLOR 3.
51    ENDMETHOD.
52    ENDCLASS.
```

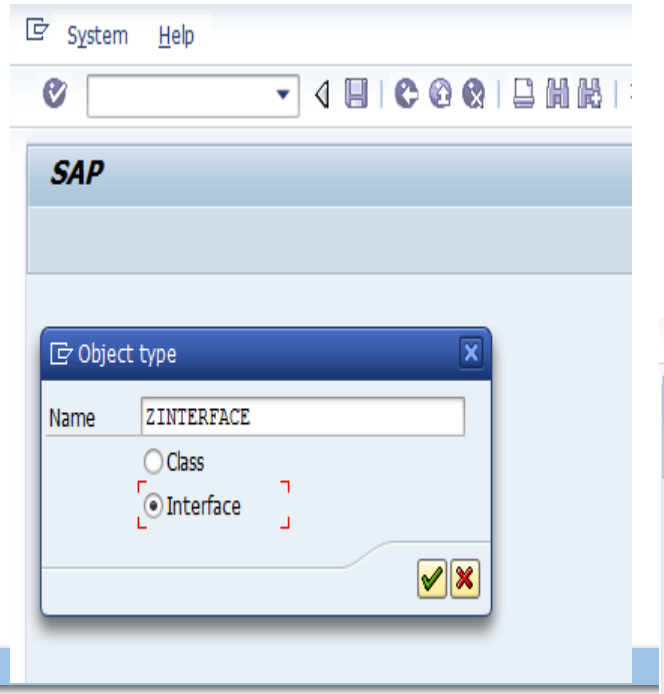
```
DATA: OBJ TYPE REF TO CL_SUB.
DATA: OBJ1 TYPE REF TO CL_LC.

START-OF-SELECTION.

CREATE OBJECT OBJ.
CALL METHOD OBJ->DISPLAY. "THIS IS SUB CLASS METHOD
CALL METHOD OBJ->SUB.


SKIP 1.
CALL METHOD OBJ->MM1.     "THIS IS SUPER CLASS METHOD
CALL METHOD OBJ->MM2.



CREATE OBJECT OBJ1.
SKIP 3.
CALL METHOD OBJ1->DISPLAY. "THIS IS SUPER CLASS METHOD
CALL METHOD OBJ1->MM1.
CALL METHOD OBJ1->MM2.
```

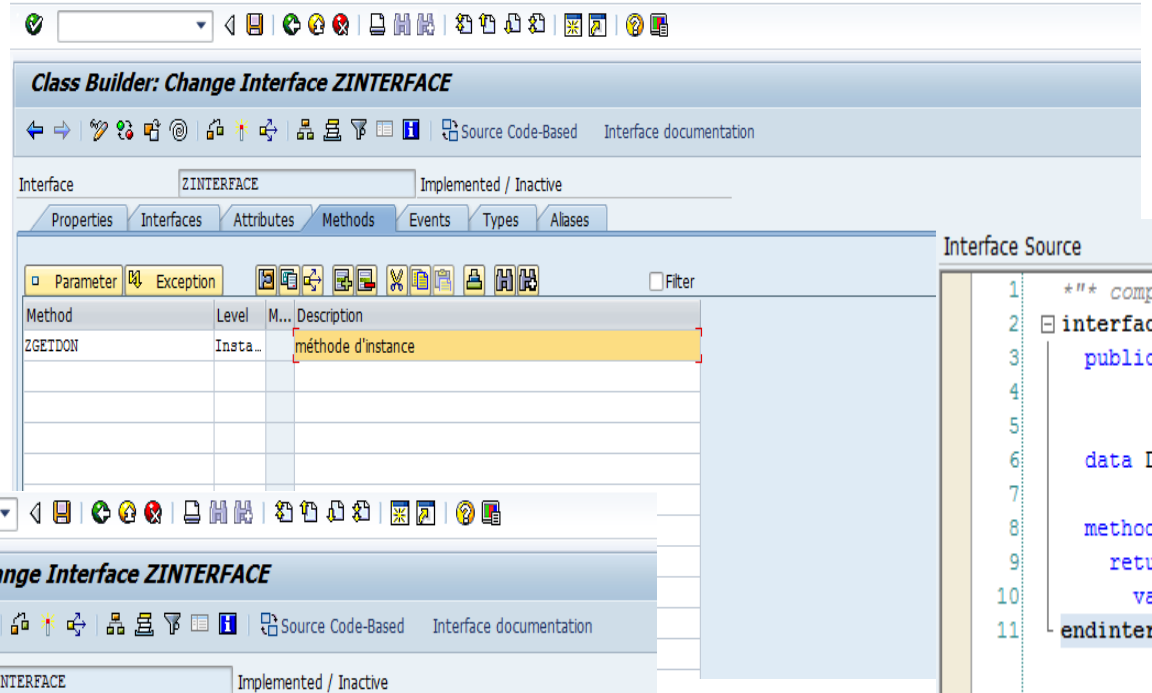
SAP

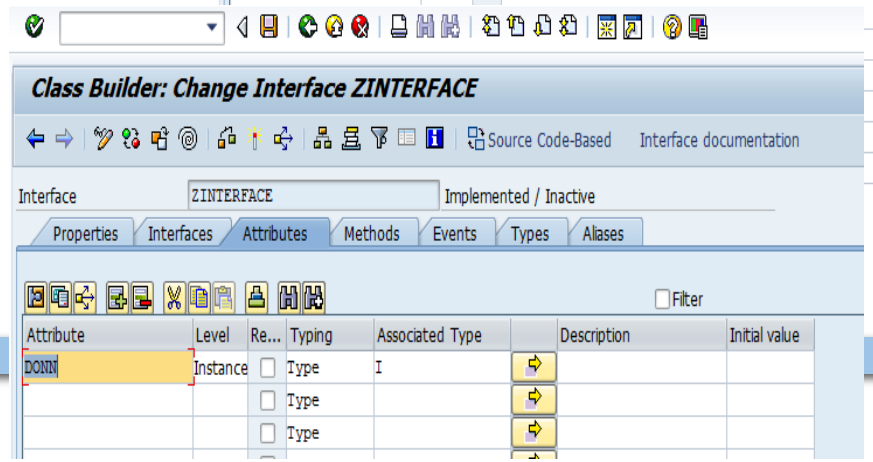
```
THIS IS THE SUB CLASS OVERWRITE METHOD
THIS IS THE SUB CLASS METHOD

THIS IS MM1 METHOD IN SUPER CLASS
THIS IS THE STATIC METHOD
THIS IS MM2 METHOD IN SUPER CLASS



THIS IS SUPER CLASS
THIS IS MM1 METHOD IN SUPER CLASS
THIS IS THE STATIC METHOD
THIS IS MM2 METHOD IN SUPER CLASS
```

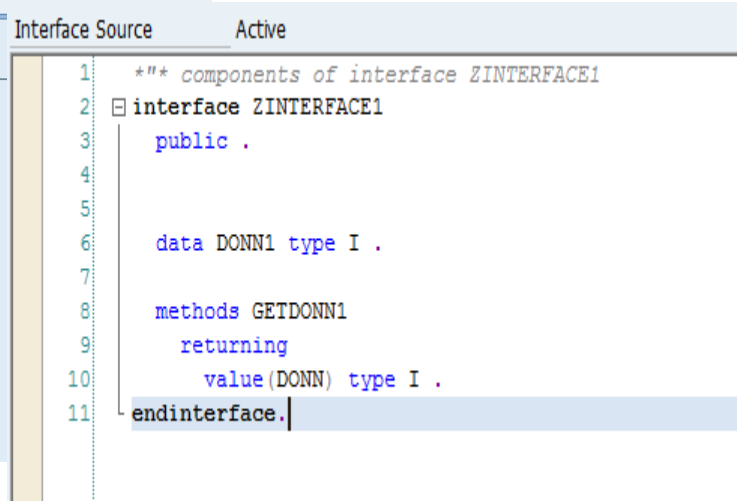
Héritage multiple

- Pas d'héritage multiple => utilisation des interfaces
- Transaction : se24









Class Source Active

```
1  class ZCLASS definition
2    public
3    create public .
4
5  public section.
6  *"* public components of class ZCLASS
7  *"* do not include other source files here!!!
8
9    interfaces ZINTERFACE2 .
10   interfaces ZINTERFACE1 .
11 protected section.
12 *"* protected components of class ZCLASS
13 *"* do not include other source files here!!!
14 private section.
15 *"* private components of class ZCLASS
16 *"* do not include other source files here!!!
17 ENDCLASS.
18
19
20
21 CLASS ZCLASS IMPLEMENTATION.
22
23
24 * <SIGNATURE>---------------------------------------------------------------
25 * | Instance Public Method ZCLASS->ZINTERFACE1~GETDONN1
26 * +--------------------------------------------------------------------------
27 * | [<-()] DONN                                 TYPE        I
28 * +--------------------------------------------------------------------------
29 method ZINTERFACE1~GETDONN1.
30
31 ZINTERFACE1~DONN1 = 1.
32 DONN = ZINTERFACE1~DONN1.
33
34 endmethod.
35
36
37 * <SIGNATURE>---------------------------------------------------------------
38 * | Instance Public Method ZCLASS->ZINTERFACE2~GETDONN2
39 * +--------------------------------------------------------------------------
40 * | [<-()] DONN2                                TYPE        I
41 * +--------------------------------------------------------------------------
42 method ZINTERFACE2~GETDONN2.
43   ZINTERFACE2~DONN2 = 2.
44   DONN2 = ZINTERFACE2~DONN2.
45
46 endmethod.
47 ENDCLASS.
```

Report ZTEST Active

```
4  *&---------------------------------------------------------------------*
5  *&
6  *&
7  *&---------------------------------------------------------------------*
8
9  REPORT  ZTEST.
10
11 DATA : class TYPE REF TO ZCLASS,
12
13       DON1 TYPE I,
14       DON2 TYPE I.
15
16 CREATE OBJECT class .
17
18
19 DON1 = class->ZINTERFACE1~GETDONN1( ).
20
21 DON2 = class->ZINTERFACE2~GETDONN2( ).
22
23 WRITE : / 'donnée 1 d interface 1',DON1
24         ,/ 'donnée 2 d interface 2',DON2 .
```

List Edit Goto System Help

Program ZTEST

```
Program ZTEST

donnée 1 d interface 1          1
donnée 2 d interface 2          2
```

Class Interface ZCLASS Implemented / Active

Properties | Interfaces | Friends | Attributes | Methods | Events | Types | Aliases

Filter

Interface	Abstract	Final	Model...	Description
ZINTERFACE2	☐	☐	☐	interface2
ZINTERFACE1	☐	☐	☐	interface 1

Class Interface ZCLASS Implemented / Active

Properties | Interfaces | Friends | Attributes | Methods | Events | Types | Aliases

Parameter | Exception | Filter

Method	Level	Vis...	M...	Description
ZINTERFACE1~GETDONN1	Insta...	Pub...		
ZINTERFACE2~GETDONN2	Insta...	Pub...		

Local Definitions/Implementations | Source Code-Based | Class documentation

Class Interface ZCLASS Implemented / Active

Properties | Interfaces | Friends | Attributes | Methods | Events | Types | Aliases

Filter

Attribute	Level	Vis...	Re...	Typing	Associated Type		Description	Initial value
ZINTERFACE1~DONN1	Instance	Public	☐	Type	I	⇨		
ZINTERFACE2~DONN2	Instance	Public	☐	Type	I	⇨		
			☐	Type		⇨		
			☐	Type		⇨		
			☐	Type		⇨		

- INTERFACE ZINTERFACE.

  DATA donn TYPE I .

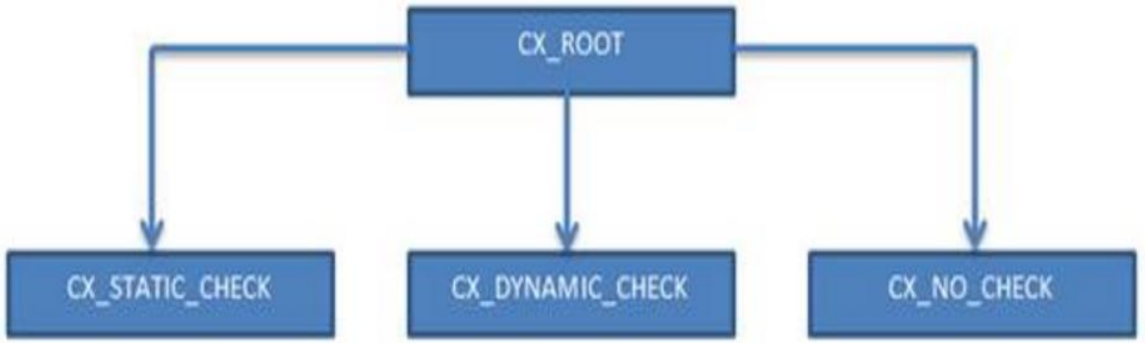
  METHODS zgetdon importing ...

  Exporting …

  returning value (donn1) TYPE type1 .

ENDINTERFACE.

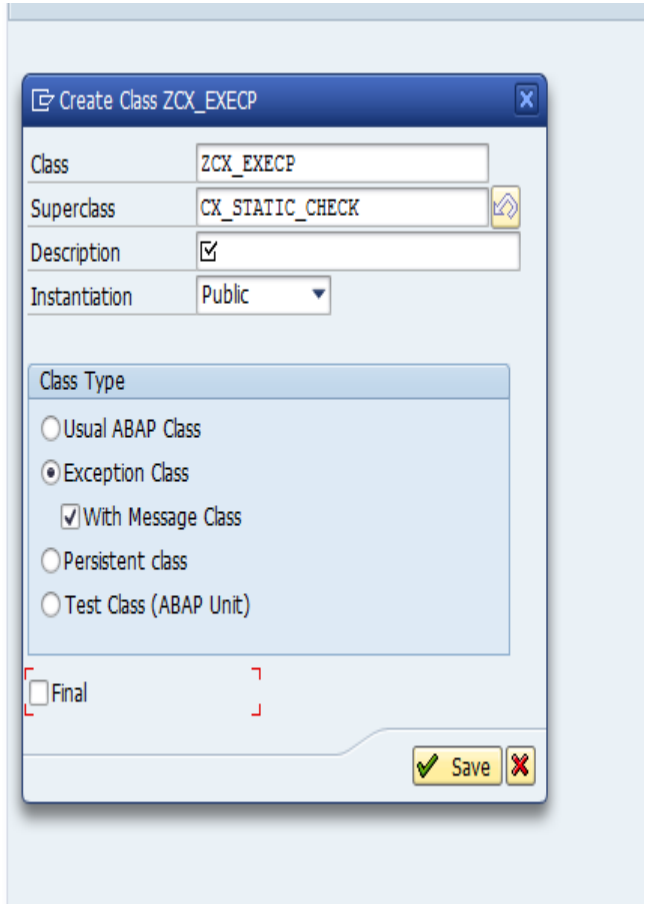
Exception

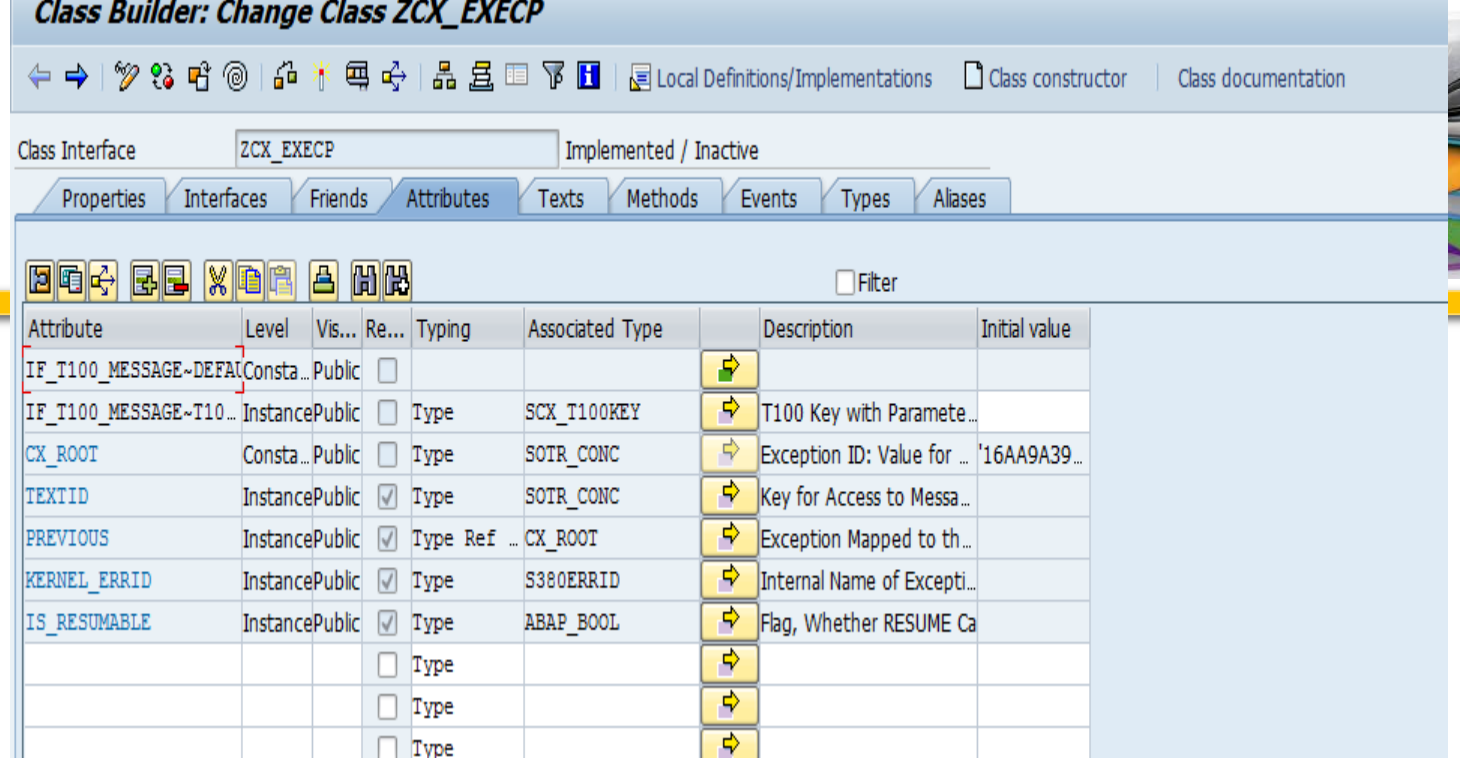


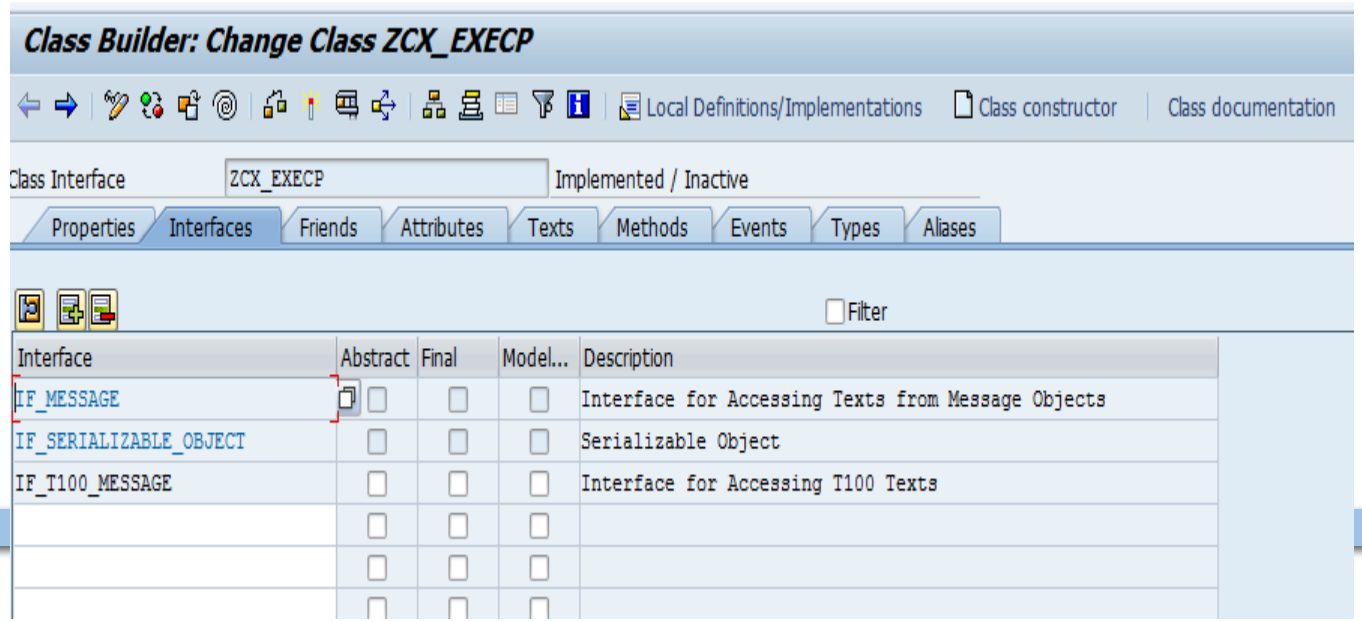
- CX_ROOT : classe d'exception de haut niveau, toute les autres classes d'exceptions héritent de cette classe.
- CX_STATIC_CHECK : englobe les exceptions attrapées directement dans la procédure ou elles sont soulevées , exemple CX_SQL_EXCEPTION
- CX_DYNAMIC_CHECK : est détecté dynamiquement par l'environnement ; exemple CX_SY_ARITHMITIC_CHECK
- CX_NO_CHECK: concerne les ressources de système.

```
DATA: result TYPE p LENGTH 10 DECIMALS 3,
      ex_ref TYPE REF TO cx_root,
      msg_text TYPE string.
PARAMETERS: value1  TYPE char40,
            value2  TYPE char40.
TRY .
      * divide v1 by v2
        result = value1 / value2.

      CATCH cx_sy_zerodivide INTO ex_ref.
        msg_text = ex_ref->get_text( ).
      CATCH cx_sy_conversion_no_number INTO ex_ref.
        msg_text = ex_ref->get_text( ).
      CATCH cx_root INTO ex_ref.
        msg_text = ex_ref->get_text( ).
ENDTRY.


IF NOT msg_text IS INITIAL.
  WRITE / msg_text.
ELSE.
  WRITE: / 'Result :', result.
ENDIF.
```

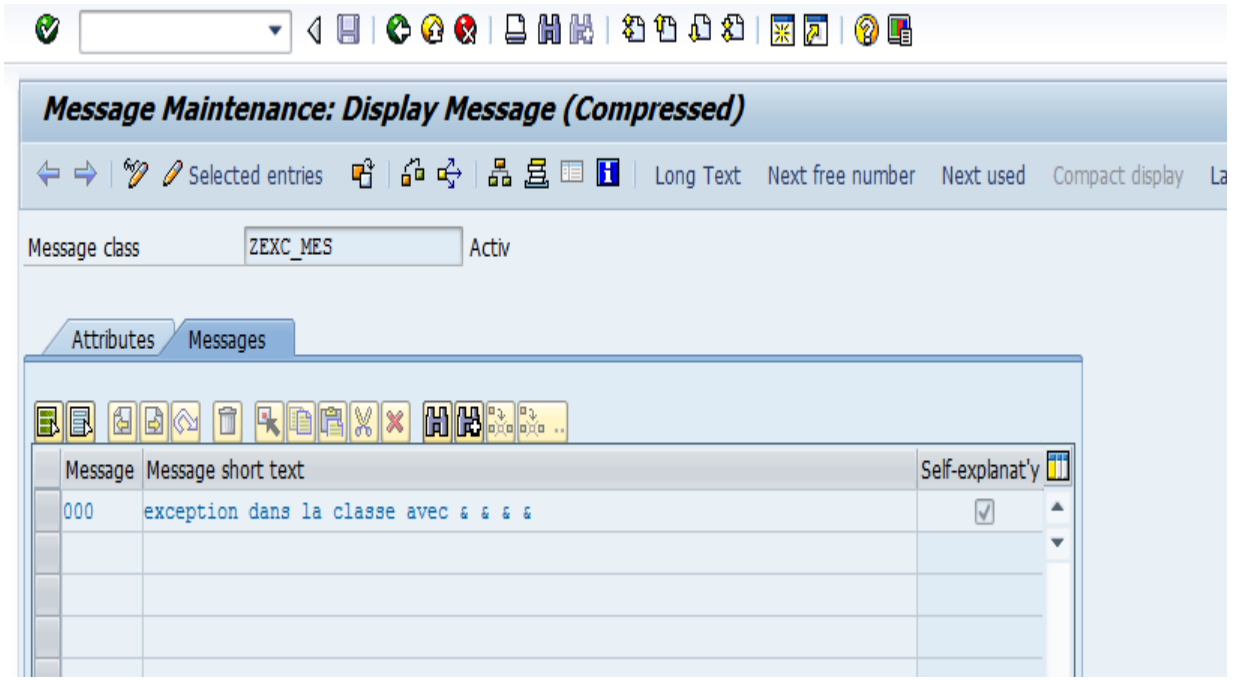
Création d'une exception

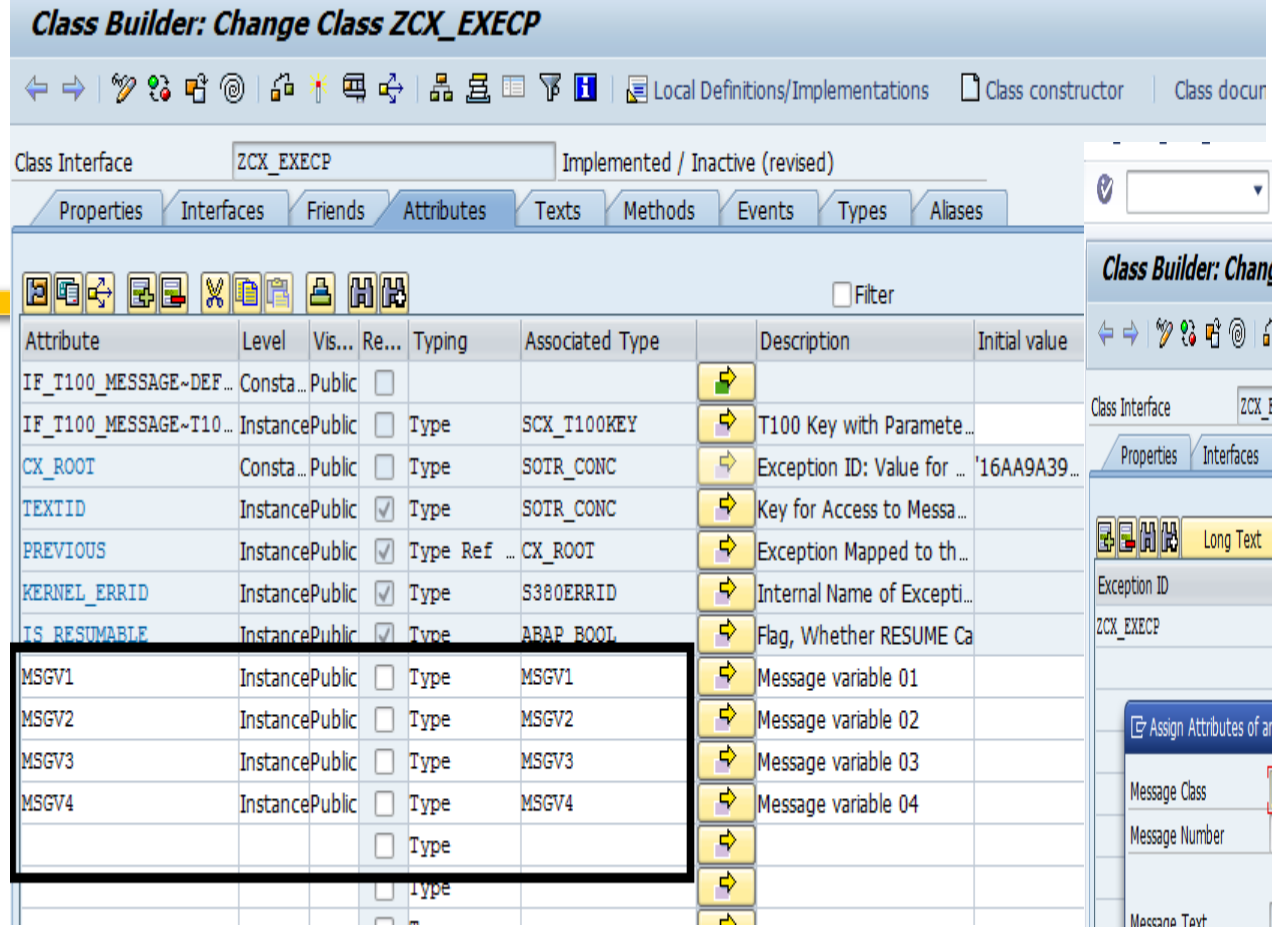


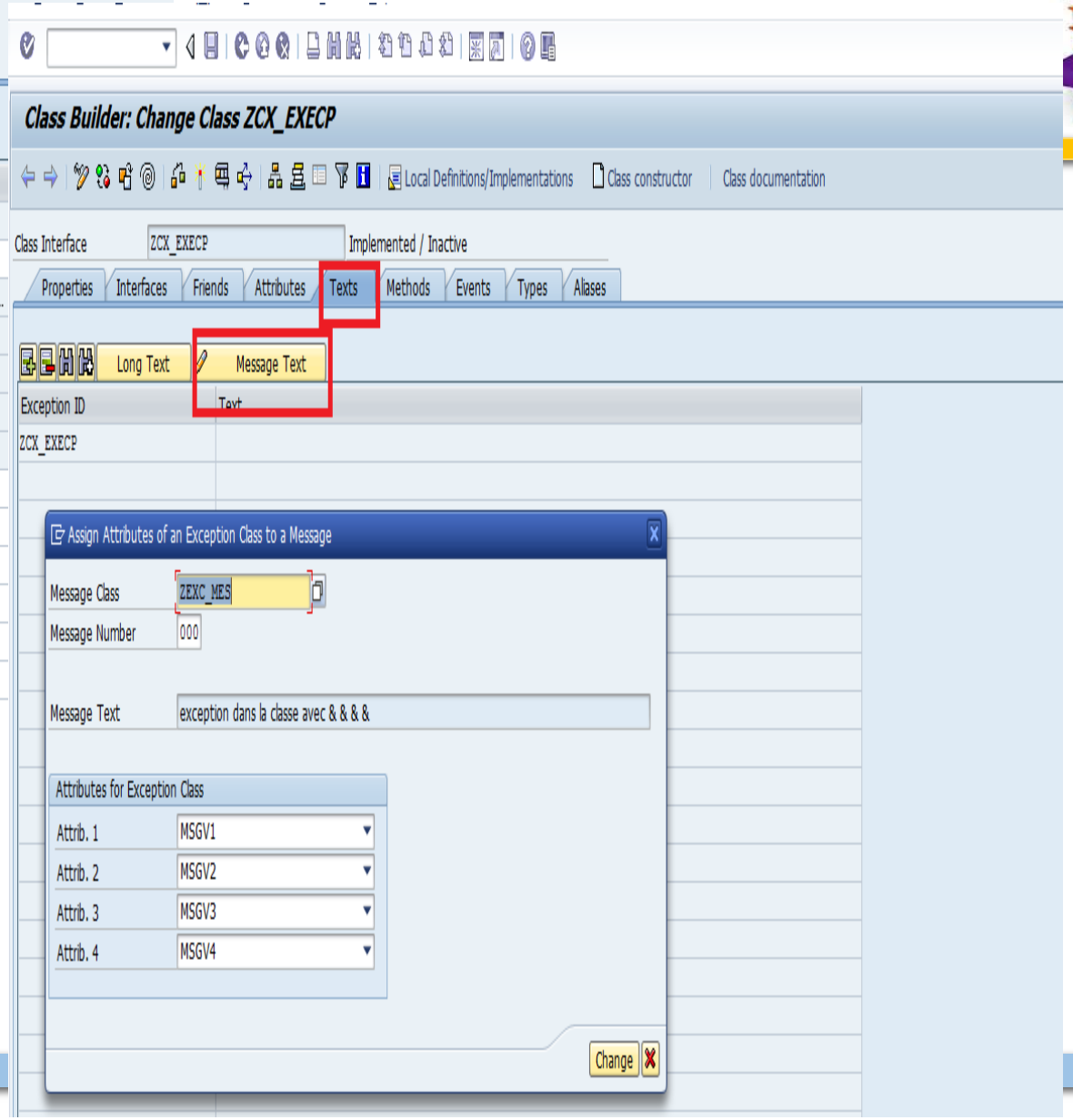




Message T-CODE se91




Class Builder: Change Class ZCX_EXECP
Local Definitions/Implementations
Class constructor
Class docur
Class Interface
ZCX_EXECP
Implemented / Inactive (revised)
Properties
Interfaces
Friends
Attributes
Texts
Methods
Events
Types
Aliases
Filter
Attribute
Level
Vis...
Re...
Typing
Associated Type
Description
Initial value
IF_T100_MESSAGE~DEF...
Consta...
Public
IF_T100_MESSAGE~T10...
Instance
Type
SCX_T100KEY
T100 Key with Paramete...
CX_ROOT
SOTR_CONC
Exception ID: Value for ...
'16AA9A39...
TEXTID
Key for Access to Messa...
PREVIOUS
Type Ref ...
CX_ROOT
Exception Mapped to th...
KERNEL_ERRID
S380ERRID
Internal Name of Excepti...
IS_RESUMABLE
ABAP_BOOL
Flag, Whether RESUME Ca
MSGV1
Message variable 01
MSGV2
Message variable 02
MSGV3
Message variable 03
MSGV4
Message variable 04


Class Builder: Change Class ZCX_EXECP
Local Definitions/Implementations
Class constructor
Class documentation
Class Interface
ZCX_EXECP
Implemented / Inactive
Properties
Interfaces
Friends
Attributes
Texts
Methods
Events
Types
Aliases
Long Text
Message Text
Exception ID
Text
ZCX_EXECP
Assign Attributes of an Exception Class to a Message
Message Class
ZEXC_MES
Message Number
000
Message Text
exception dans la classe avec & & & &
Attributes for Exception Class
Attrib. 1
MSGV1
Attrib. 2
MSGV2
Attrib. 3
MSGV3
Attrib. 4
MSGV4
Change

ABAP Editor: Change Report ZTEST

Pattern Pretty Printer

MIME Repository
Repository Browser
Repository Information System
Tag Browser
Transport Organizer
Test Repository

Program
ZTEST

Object Name	D...
ZTEST	Progr
Fields	

Report ZTEST Inactive (Revised)

```
4   *&---------------------------------------------------------------------*
5   *&
6   *&
7   *&---------------------------------------------------------------------*
8
9   REPORT  ZTEST.
10  DATA: ex_ref TYPE REF TO cx_root.
11  DATA: message_text TYPE string.
12
13  TRY.
14      RAISE EXCEPTION TYPE zcx_execp
15                  EXPORTING msgv1 = 'message'
16                            msgv2 = ' pour'
17                            msgv3 = ' Exceptions '
18                            msgv4 = 'dans le programme '.
19
20    CATCH zcx_execp INTO ex_ref.
21
22    message_text = ex_ref->get_text( ).
23    WRITE message_text.
24  ENDTRY.
```

Program ZTEST

Program ZTEST

exception dans la classe avec message pour Exceptions dans le programme

Fields symbols

- Fields symbols : sont des noms symboliques, ils ne réservent pas physiquement l'espace pour un champ, mais pointent à son contenu.
- Chaque fois que vous vous adressez à un Fields symbols dans un programme, vous adressez au domaine qui est attribué au Fields symbols .

- Fields symbols sont similaires aux pointeurs déréférencé dans C

Fields symbols

- Déclaration

- `field-symbols :` <NOM_FS> TYPE typeFS.

- Exemple
- TYPES: BEGIN OF fint_frange_t,
- fieldname LIKE rsdstabs-prim_fname,
- fieldtype(1) TYPE c,
- selopt_t TYPE fint_selopt_t,

END OF fint_frange_t.

- `field-symbols :` <NOM_FS> TYPE fint_frange_t.

Déclaration

- TYPES:

```
 tt_mara TYPE STANDARD TABLE OF spfli.
DATA: t_mara TYPE tt_mara.
FIELD-SYMBOLS: <lfs_mara> LIKE LINE OF t_mara.
FIELD-SYMBOLS: <lfs_any_tab> TYPE ANY TABLE,
                             <lfs_any> TYPE ANY.
```

Utilisation

- READ TABLE t_mara ASSIGNING <lfs_mara>
  WITH KEY matnr = '123456'.

IF sy-subrc EQ 0.

WRITE: <lfs_mara>-matnr.

ENDIF.

LOOP AT t_mara ASSIGNING <lfs_mara>.

WRITE: <lfs_mara>-matnr.

ENDLOOP.

Utilisation

- IF <lfs_mara> IS ASSIGNED.
  WRITE: 'Assigned'.

ELSE.
  WRITE: 'Unassigned'.

ENDIF.

UNASSIGN <lfs_mara>.